## INSTALLATION

### 1 Installation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.1.1 Ways to Install</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 References</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Instructions</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Spack</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Docker</td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 From Source</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Dependencies</td>
<td>8</td>
</tr>
<tr>
<td>1.3.1 Overview</td>
<td>8</td>
</tr>
<tr>
<td>1.3.2 Requirements</td>
<td>8</td>
</tr>
<tr>
<td>1.4 picongpu.profile</td>
<td>16</td>
</tr>
<tr>
<td>1.4.1 Hemera (HZDR)</td>
<td>16</td>
</tr>
<tr>
<td>1.4.2 Hypnos (HZDR)</td>
<td>24</td>
</tr>
<tr>
<td>1.4.3 Hydra (HZDR)</td>
<td>26</td>
</tr>
<tr>
<td>1.4.4 Summit (ORNL)</td>
<td>27</td>
</tr>
<tr>
<td>1.4.5 Piz Daint (CSCS)</td>
<td>29</td>
</tr>
<tr>
<td>1.4.6 Taurus (TU Dresden)</td>
<td>31</td>
</tr>
<tr>
<td>1.4.7 Lawrencium (LBNL)</td>
<td>38</td>
</tr>
<tr>
<td>1.4.8 Cori (NERSC)</td>
<td>39</td>
</tr>
<tr>
<td>1.4.9 Draco (MPCDF)</td>
<td>41</td>
</tr>
<tr>
<td>1.4.10 D.A.V.I.D.E (CINECA)</td>
<td>42</td>
</tr>
<tr>
<td>1.4.11 JURECA (JSC)</td>
<td>44</td>
</tr>
<tr>
<td>1.4.12 JUWELS (JSC)</td>
<td>51</td>
</tr>
<tr>
<td>1.4.13 ARIS (GRNET)</td>
<td>55</td>
</tr>
<tr>
<td>1.5 Changelog</td>
<td>57</td>
</tr>
<tr>
<td>1.5.1 0.5.0</td>
<td>57</td>
</tr>
<tr>
<td>1.5.2 0.4.3</td>
<td>64</td>
</tr>
<tr>
<td>1.5.3 0.4.2</td>
<td>65</td>
</tr>
<tr>
<td>1.5.4 0.4.1</td>
<td>66</td>
</tr>
<tr>
<td>1.5.5 0.4.0</td>
<td>66</td>
</tr>
<tr>
<td>1.5.6 0.3.2</td>
<td>80</td>
</tr>
<tr>
<td>1.5.7 0.3.1</td>
<td>81</td>
</tr>
<tr>
<td>1.5.8 0.3.0</td>
<td>82</td>
</tr>
<tr>
<td>1.5.9 0.2.5</td>
<td>89</td>
</tr>
<tr>
<td>1.5.10 0.2.4</td>
<td>89</td>
</tr>
<tr>
<td>1.5.11 0.2.3</td>
<td>90</td>
</tr>
<tr>
<td>1.5.12 0.2.2</td>
<td>91</td>
</tr>
<tr>
<td>1.5.13 0.2.1</td>
<td>91</td>
</tr>
<tr>
<td>1.5.14 0.2.0 “Beta”</td>
<td>92</td>
</tr>
<tr>
<td>1.5.15 0.1.0</td>
<td>101</td>
</tr>
<tr>
<td>1.5.16 Open Beta RC6</td>
<td>104</td>
</tr>
<tr>
<td>1.5.17 Open Beta RC5</td>
<td>107</td>
</tr>
<tr>
<td>1.5.18 Open Beta RC4</td>
<td>109</td>
</tr>
</tbody>
</table>
Particle-in-Cell Simulations for the Exascale Era

PIConGPU is a fully relativistic, manycore, 3D3V and 2D3V particle-in-cell (PIC) code. The PIC algorithm is a central tool in plasma physics. It describes the dynamics of a plasma by computing the motion of electrons and ions in the plasma based on the Vlasov-Maxwell system of equations.

Generally, to get started follow the manual pages in order. Individual chapters are based on the information in the chapters before. In case you are already fluent in compiling C++ projects and HPC, running PIC simulations or scientific data analysis, feel free to jump the respective sections.

**Note:** We are migrating our wiki to this manual, but some pages might still be missing. We also have an official homepage.

**Note:** Are you looking for our latest Doxygen docs for the API?

See [http://computationalradiationphysics.github.io/picongpu](http://computationalradiationphysics.github.io/picongpu)
1.1 Introduction

Section author: Axel Huebl

Installing PIConGPU means installing C++ libraries that PIConGPU depends on and setting environment variables to find those dependencies. The first part is usually the job of a system administrator while the second part needs to be configured on the user side.

Depending on your experience, role, computing environment and expectations for optimal hardware utilization, you have several ways to install and select PIConGPU’s dependencies. Choose your favorite install and environment management method below, young padawan, and follow the corresponding sections of the next chapters.

1.1.1 Ways to Install

Choose one of the installation methods below to get started.

Load Modules

On HPC systems and clusters, software is usually provided by system administrators via a module system (e.g. [modules], [Lmod]). In case our software dependencies are available, we usually create a file in our $HOME named <queueName>_picongpu.profile. It loads according modules and sets helper environment variables.

**Important:** For many HPC systems we have already prepared and maintain an environment which will run out of the box. See if your system is in the list so you can skip the installation completely!

Spack

[Spack] is a flexible package manager that can build and organize software dependencies. It can be configured once for your hardware architecture to create optimally tuned binaries and provides modulefile support (e.g. [modules], [Lmod]). Those auto-build modules manage your environment variables and allow easy switching between versions, configurations and compilers.

Build from Source

You choose a supported C++ compiler and configure, compile and install all missing dependencies from source. You are responsible to manage the right versions and configurations. Performance will be ideal if architecture is chosen correctly (and/or if built directly on your hardware). You then set environment variables to find those installs.
Conda

We currently do not have an official conda install (yet). Due to pre-build binaries, performance could be not ideal and HPC cluster support (e.g. MPI) might be very limited. Useful for small desktop or single-node runs.

Nvidia-Docker

Not yet officially supported [nvidia-docker], but we already provide a Dockerfile to get started. Performance might be not ideal if the image is not built for the specific local hardware again. Useful for small desktop or single-node runs. We are also working on Singularity images.

1.1.2 References

1.2 Instructions

Section author: Axel Huebl

As explained in the previous section, select and follow exactly one of the following install options.

See also:

You will need to understand how to use the terminal.

**Warning:** Our spack package is still in beta state and is continuously improved. Please feel free to report any issues that you might encounter.

1.2.1 Spack

Section author: Axel Huebl

Preparation

First install spack itself via:

```bash
# get spack
git clone https://github.com/spack/spack.git $HOME/src/spack

# activate the spack environment
source $HOME/src/spack/share/spack/setup-env.sh

# install a supported compiler
spack compiler list | grep -q gcc@7.3.0 || spack install gcc@7.3.0 && spack load gcc@7.3.0 && spack compiler add

# add the PIConGPU repository
git clone https://github.com/ComputationalRadiationPhysics/spack-repo.git $HOME/src/spack-repo
spack repo add $HOME/src/spack-repo

Note: When you open a terminal next time or log back into the machine, make sure to activate the spack environment again via:

source $HOME/src/spack/share/spack/setup-env.sh
```
Install

The installation of the latest version of PIConGPU is now as easy as:

```
spack install picongpu %gcc@7.3.0
```

Use PIConGPU

PIConGPU can now be loaded with

```
spack load picongpu
```

For more information on variants of the `picongpu` package in `spack`, run `spack info picongpu` and refer to the official `spack` documentation.

**Note:** PIConGPU can also run without a GPU! For example, to use our OpenMP backend, just add `backend=omp2b` to the two commands above:

```
spack install picongpu backend=omp2b
spack load picongpu backend=omp2b
```

**Note:** If the install fails or you want to compile for CUDA 9.2, try using GCC 5.5.0:

```
spack compiler list | grep gcc@5.5.0 | spack install gcc@5.5.0 %gcc@5.5.0
spack compiler add
spack install picongpu %gcc@5.5.0
spack load picongpu %gcc@5.5.0
```

See also:

You will need to understand how to use the terminal.

**Warning:** Docker images are experimental and not yet fully automated or integrated.

### 1.2.2 Docker

*Section author: Axel Huebl*

**Preparation**

First install `nvidia-docker` for your distribution. Use version 2 or newer.

**Install**

The download of a pre-configured image with the latest version of PIConGPU is now as easy as:

```
docker pull ax3l/picongpu
```
Use PIConGPU

Start a pre-configured LWFA live-simulation with

```
docker run --runtime=nvidia -p 2459:2459 -t ax3l/picongpu /bin/bash -lc start_lwfa
```

# open firefox and isaac client

or just open the container and run your own:

```
docker run --runtime=nvidia -it ax3l/picongpu
```

**Note:** PIConGPU can also run without a GPU! We will provide more image variants in the future.

See also:

You will need to understand how to use the terminal.

**Note:** This section is a short introduction in case you are missing a few software packages, want to try out a cutting edge development version of a software or have no system administrator or software package manager to build and install software for you.

### 1.2.3 From Source

*Section author: Axel Huebl*

Don’t be afraid, young physicist, self-compiling C/C++ projects is easy, fun and profitable!

Building a project from source essentially requires three steps:

1. configure the project and find its dependencies
2. compile the project
3. install the project

All of the above steps can be performed without administrative rights (“root” or “superuser”) as long as the install is not targeted at a system directory (such as /usr) but inside a user-writable directory (such as $HOME or a project directory).

**Preparation**

In order to compile projects from source, we assume you have individual directories created to store source code, build temporary files and install the projects to:

```
# source code
mkdir $HOME/src
# temporary build directory
mkdir $HOME/build
# install target for dependencies
mkdir $HOME/lib
```

Note that on some supercomputing systems, you might need to install the final software outside of your home to make dependencies available during run-time (when the simulation runs). Use a different path for the last directory then.
What is Compiling?

**Note:** This section is not yet the installation of PIConGPU from source. It just introduces in general how one compiles projects.

If you like to skip this introduction, *jump straight to the dependency install section.*

Compiling can differ in two principle ways: building *inside* the source directory (“in-source”) and in a *temporary directory* (“out-of-source”). Modern projects prefer the latter and use a build system such as [CMake].

An example could look like this

```bash
# go to an empty, temporary build directory
cd $HOME/build
rm -rf ../build/*

# configure, build and install into $HOME/lib/project
cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/project $HOME/src/project_to_compile
make
make install
```

Often, you want to pass further options to CMake with `-D OPTION=VALUE` or modify them interactively with `ccmake` after running the initial `cmake` command. The second step which compiles the project can in many cases be parallelized by `make -j`. In the final install step, you might need to prefix it with `sudo` in case `CMAKE_INSTALL_PREFIX` is pointing to a system directory.

Some older projects often build *in-source* and use a build system called *autotools*. The syntax is still very similar:

```bash
# go to the source directory of the project
cd $HOME/src/project_to_compile

# configure, build and install into $HOME/lib/project
configure --prefix=$HOME/lib/project
make
make install
```

One can usually pass further options with `--with-something=VALUE` or `--enable-thing` to `configure`. See `configure --help` when installing an *autotools* project.

That is all on the theory of building projects from source!

**Now Start**

You now know all the basics to install from source. Continue with the following section to *build our dependencies*.

**References**

If anything goes wrong, an overview of the full list of PIConGPU dependencies is provided in *section Dependencies*.

After installing, the last step is the setup of a *profile*.

**See also:**

You will need to understand how to use *the terminal*, *what are environment variables* and please read our *compiling introduction*.

*Note:* If you are a scientific user at a supercomputing facility we might have already prepared a software setup for you. See the *following chapter* if you can skip this step fully or in part by loading existing modules on those
1.3 Dependencies

Section author: Axel Huebl

1.3.1 Overview

![Diagram showing inter-library dependencies for parallel execution of PIConGPU on a typical HPC system.]

Due to common binary incompatibilities between compilers, MPI and boost versions, we recommend to organize software with a version-aware package manager such as `spack` and to deploy a hierarchical module system such as `lmod`. An Lmod example setup can be found here.

1.3.2 Requirements

Mandatory

**gcc**

- 4.9 - 7 (if you want to build for Nvidia GPUs, supported compilers depend on your current CUDA version)
  - CUDA 9.2 - 10.0: Use gcc 4.9 - 7
  - CUDA 10.1/10.2: Use gcc 4.9 - 8
- note: be sure to build all libraries/dependencies with the same gcc version; GCC 5 or newer is recommended
- Debian/Ubuntu:
- sudo apt-get install gcc-5.3 g++-5.3 build-essential
- sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5.3 3 60 --slave /usr/bin/g++ g++ /usr/bin/g++-5.3

• Arch Linux:
  - sudo pacman --sync base-devel
  - if the installed version of gcc is too new, compile an older gcc

• Spack:
  - spack install gcc@5.3.0
  - make it the default in your packages.yaml or suffix all following spack install commands with a space and %gcc@5.3.0

CMake

• 3.11.4 or higher
• Debian/Ubuntu: sudo apt-get install cmake file cmake-curses-gui
• Arch Linux: sudo pacman --sync cmake
• Spack: spack install cmake

MPI 2.3+

• OpenMPI 1.7+ / MVAPICH2 1.8+ or similar
• for running on Nvidia GPUs, perform a GPU aware MPI install after installing CUDA
• Debian/Ubuntu: sudo apt-get install libopenmpi-dev
• Arch Linux: sudo pacman --sync openmpi
• Spack:
  - GPU support: spack install openmpi+cuda
  - CPU only: spack install openmpi
• environment:
  - export MPI_ROOT=<MPI_INSTALL>
  - as long as CUDA awareness (openmpi+cuda) is missing: export OMPI_MCA_mpi_leave_pinned=0

zlib

• Debian/Ubuntu: sudo apt-get install zlib1g-dev
• Arch Linux: sudo pacman --sync zlib
• Spack: spack install zlib
• from source:
  - ./configure --prefix=$HOME/lib/zlib
  - make && make install
• environment: (assumes install from source in $HOME/lib/zlib)
  - export ZLIB_ROOT=$HOME/lib/zlib

1.3. Dependencies
boost

- 1.65.1 - 1.70.0 (program_options, filesystem, system, math, serialization and header-only libs, optional: fiber with context, thread, chrono, atomic, date_time)

  - Debian/Ubuntu: sudo apt-get install libboost-program-options-dev
  - libboost-filesystem-dev
  - libboost-system-dev
  - libboost-thread-dev
  - libboost-chrono-dev
  - libboost-atomic-dev
  - libboost-date-time-dev
  - libboost-math-dev
  - libboost-serialization-dev
  - libboost-fiber-dev
  - libboost-context-dev

- Arch Linux: sudo pacman --sync boost

- Spack: spack install boost

- from source:
  - curl -Lo boost_1_65_1.tar.gz https://dl.bintray.com/boostorg/release/1.65.1/source/boost_1_65_1.tar.gz
  - tar -xzf boost_1_65_1.tar.gz
  - cd boost_1_65_1
  - ./bootstrap.sh --with-libraries=atomic,chrono,context,date_time,fiber,filesystem,math,program_options,serialization,system,thread --prefix=$HOME/lib/boost
  - ./b2 cxxflags="-std=c++11" -j4 && ./b2 install

- environment: (assumes install from source in $HOME/lib/boost)
  - export BOOST_ROOT=$HOME/lib/boost
  - export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH

git

- 1.7.9.5 or higher

  - Debian/Ubuntu: sudo apt-get install git

  - Arch Linux: sudo pacman --sync git

  - Spack: spack install git

rsync

- Debian/Ubuntu: sudo apt-get install rsync

  - Arch Linux: sudo pacman --sync rsync

  - Spack: spack install rsync

alpaka 0.4.0

- alpaka is included in the PIConGPU source code
cupla 0.2.0

- cupla is included in the PIConGPU source code

mallocMC 2.3.0crp

- only required for CUDA backend
- mallocMC is included in the PIConGPU source code

PIConGPU Source Code

- `git clone https://github.com/ComputationalRadiationPhysics/picongpu.git $HOME/src/picongpu`
  - _optional:_ update the source code with `cd $HOME/src/picongpu && git fetch && git pull`
  - _optional:_ change to a different branch with `git branch (show)` and `git checkout <BranchName> (switch)`

- _environment:_
  - `export PICSRC=$PICHOME/src/picongpu`
  - `export PIC_EXAMPLES=$PICSRC/share/picongpu/examples`
  - `export PATH=$PICSRC:$PATH`
  - `export PATH=$PICSRC/bin:$PATH`
  - `export PATH=$PICSRC/src/tools/bin:$PATH`
  - `export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH`

Optional Libraries

CUDA

- _9.2 - 10.2_
- required if you want to run on Nvidia GPUs
- _Debian/Ubuntu:_ `sudo apt-get install nvidia-cuda-toolkit`
- _Arch Linux:_ `sudo pacman --sync cuda`
- _Spack:_ `spack install cuda`
- at least one CUDA capable GPU
- _compute capability:_ sm_30 or higher
- _full list_ of CUDA GPUs and their _compute capability_
- _More is always better._ Especially, if we are talking GPUs :-)
- _environment:_
  - `export CUDA_ROOT=<CUDA_INSTALL>`

If you do not install the following libraries, you will not have the full amount of PIConGPU plugins. We recommend to install at least pngwriter and either libSplash (+ HDF5) or ADIOS.
libpng

- 1.2.9+ (requires zlib)
- Debian/Ubuntu dependencies: sudo apt-get install libpng-dev
- Arch Linux dependencies: sudo pacman --sync libpng
- Spack: spack install libpng
- from source:
  - mkdir -p ~/src ~/lib
  - cd ~/src
  - curl -Lo libpng-1.6.34.tar.gz ftp://ftp-osl.osuosl.org/pub/libpng/src/libpng16/libpng-1.6.34.tar.gz
  - tar -xf libpng-1.6.34.tar.gz
  - cd libpng-1.6.34
  - CPPFLAGS=-I$HOME/lib/zlib/include LDFLAGS=-L$HOME/lib/zlib/lib ./
    configure --enable-static --enable-shared --prefix=$HOME/lib/libpng
  - make
  - make install
- environment: (assumes install from source in $HOME/lib/libpng)
  - export PNG_ROOT=$HOME/lib/libpng
  - export CMAKE_PREFIX_PATH=$PNG_ROOT:$CMAKE_PREFIX_PATH
  - export LD_LIBRARY_PATH=$PNG_ROOT/lib:$LD_LIBRARY_PATH

pngwriter

- 0.7.0+ (requires libpng, zlib, and optional freetype)
- Spack: spack install pngwriter
- from source:
  - mkdir -p ~/src ~/build ~/lib
  - git clone https://github.com/pngwriter/pngwriter.git ~/src/pngwriter/
  - cd ~/build
  - cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/pngwriter ~/src/pngwriter
  - make install
- environment: (assumes install from source in $HOME/lib/pngwriter)
  - export CMAKE_PREFIX_PATH=$HOME/lib/pngwriter:$CMAKE_PREFIX_PATH
  - export LD_LIBRARY_PATH=$HOME/lib/pngwriter/lib:$LD_LIBRARY_PATH

libSplash

- 1.7.0+ (requires HDF5, boost program-options)
- Debian/Ubuntu dependencies: sudo apt-get install libhdf5-openmpi-dev
  libboost-program-options-dev
• **Arch Linux dependencies**: sudo pacman --sync hdf5-openmpi boost

• **Spack**: spack install libsplash ^hdf5~fortran

• **from source**:
  - mkdir -p ~/src ~/build ~/lib
  - git clone https://github.com/ComputationalRadiationPhysics/libSplash.git ~/src/splash/
  - cd ~/build && rm -rf ../build/*
  - cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/splash -DSplash_USE_MPI=ON -DSplash_USE_PARALLEL=ON ~/src/splash
  - make install

• **environment**: (assumes install from source in $HOME/lib/splash)
  - export CMAKE_PREFIX_PATH=$HOME/lib/splash:$CMAKE_PREFIX_PATH
  - export LD_LIBRARY_PATH=$HOME/lib/splash/lib:$LD_LIBRARY_PATH

---

**HDF5**

• 1.8.13+

• standard shared version (no C++, enable parallel)

• **Debian/Ubuntu**: sudo apt-get install libhdf5-openmpi-dev

• **Arch Linux**: sudo pacman --sync hdf5-openmpi

• **Spack**: spack install hdf5~fortran

• **from source**:
  - mkdir -p ~/src ~/lib
  - cd ~/src
  - download hdf5 source code from release list of the HDF5 group, for example:
  - curl -Lo hdf5-1.8.20.tar.gz https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8/hdf5-1.8.20/src/hdf5-1.8.20.tar.gz
  - tar -xzf hdf5-1.8.20.tar.gz
  - cd hdf5-1.8.20
  - ./configure --enable-parallel --enable-shared --prefix $HOME/lib/hdf5/
  - make
  - optional: make test
  - make install
  - If you encounter errors related to linking MPI during ./configure, you might try setting the compiler manually via ./configure --enable-parallel --enable-shared --prefix $HOME/lib/hdf5/ CC=mpicc CXX=mpic++.

• **environment**: (assumes install from source in $HOME/lib/hdf5)
  - export HDF5_ROOT=$HOME/lib/hdf5
  - export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH
**splash2txt**

- requires *libSplash* and *boost* `program_options`
- converts slices in dumped hdf5 files to plain txt matrices
- assume you [downloaded](#requirements) PIConGPU to `PICSRC=$HOME/src/picongpu`
- `mkdir -p ~/build && cd ~/build`
- `cmake -DCMAKE_INSTALL_PREFIX=$PICSRC/src/tools/bin $PICSRC/src/tools/splash2txt`
- `make`
- `make install`

**environment:**

- `export PATH=$PATH:$PICSRC/src/splash2txt/build`

**options:**

- `splash2txt --help`
- list all available datasets: `splash2txt --list <FILE_PREFIX>`

**png2gas**

- requires *libSplash, pngwriter* and *boost* `program_options`)
- converts png files to hdf5 files that can be used as an input for species initial density profiles
- compile and install exactly as `splash2txt` above

**c-blosc**

- general purpose compressor, used in ADIOS for in situ data reduction
- *Debian/Ubuntu:* `sudo apt-get install libblosc-dev`
- *Arch Linux:* `sudo pacman --sync blosc`
- *Spack:* `spack install c-blosc`

**from source:**

- `mkdir -p ~/src ~/build ~/lib`
- `cd ~/src`
- `curl -Lo c-blosc-1.15.0.tar.gz https://github.com/Blosc/c-blosc/archive/v1.15.0.tar.gz`
- `tar -xzvf c-blosc-1.15.0.tar.gz`
- `cd ~/build && rm -rf ../build/*`
- `cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/c-blosc -DPREFER_EXTERNAL_ZLIB=ON ~/src/c-blosc-1.15.0/`
- `make`
- `make install`

**environment:** (assumes install from source in `$HOME/lib/c-blosc`)

- `export BLOSC_ROOT=$HOME/lib/c-blosc`
- `export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH`
- export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

ADIOS

• 1.13.1+ (requires MPI, zlib and c-blosc)
• Debian/Ubuntu: sudo apt-get install libadios-dev libadios-bin
• Arch Linux using an AUR helper: pacaur --sync libadios
• Arch Linux using the AUR manually:
  - sudo pacman --sync --needed base-devel
  - git clone https://aur.archlinux.org/libadios.git
  - cd libadios
  - makepkg -sri
• Spack: spack install adios
• from source:
  - mkdir -p ~/src ~/lib
  - cd ~/src
  - curl -Lo adios-1.13.1.tar.gz http://users.nccs.gov/~pnorbert/
    adios-1.13.1.tar.gz
  - tar -xzf adios-1.13.1.tar.gz
  - cd adios-1.13.1
  - CFLAGS="-fPIC" ./configure --enable-static --enable-shared
  --prefix=$HOME/lib/adios --with-mpi=$MPI_ROOT --with-zlib=$HOME/
    lib/zlib --with-blosc=$HOME/lib/c-blosc
  - make
  - make install
• environment: (assumes install from source in $HOME/lib/adios)
  - export ADIOS_ROOT=$HOME/lib/adios
  - export LD_LIBRARY_PATH=$ADIOS_ROOT/lib:$LD_LIBRARY_PATH

ISAAC

• 1.4.0+
• requires boost (header only), IceT, Jansson, libjpeg (preferably libjpeg-turbo), libwebsockets (only for the ISAAC server, but not the plugin itself)
• enables live in situ visualization, see more here Plugin description
• Spack: spack install isaac
• from source: build the in situ library and its dependencies as described in ISAAC’s INSTALL.md
• environment: set environment variable CMAKE_PREFIX_PATH for each dependency and the ISAAC in situ library
VampirTrace

- for developers: performance tracing support
- download 5.14.4 or higher, e.g. from www.tu-dresden.de
- from source:
  - mkdir -p ~/src ~/build ~/lib
  - cd ~/src
  - tar -xzf VampirTrace-5.14.4.tar.gz
  - cd VampirTrace-5.14.4
  - ./configure --prefix=$HOME/lib/vampirtrace --with-cuda-dir=<CUDA_ROOT>
  - make all -j
  - make install
- environment: (assumes install from source in $HOME/lib/vampirtrace)
  - export VT_ROOT=$HOME/lib/vampirtrace
  - export PATH=$VT_ROOT/bin:$PATH

See also:
You need to have all dependencies installed to complete this chapter.

1.4 picongpu.profile

Section author: Axel Huebl

Use a picongpu.profile file to set up your software environment without colliding with other software. Ideally, store that file directly in your $HOME/ and source it after connecting to the machine:

```bash
source $HOME/picongpu.profile
```

We listed some example picongpu.profile files below which can be used to set up PIConGPU’s dependencies on various HPC systems.

1.4.1 Hemera (HZDR)

System overview: link (internal)

User guide: None

Production directory: /bigdata/hplsim/ with external/, scratch/, development/ and production/

For this profile to work, you need to download the PIConGPU source code manually.

Queue: defq (2x Intel Xeon Gold 6148, 20 Cores + 20 HyperThreads/CPU)
# Name and Path of this Script #################################### (DO NOT change!
export PIC_PROFILE="$(cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules ####################################
module purge
module load gcc/7.3.0
module load cmake/3.15.2
module load openmpi/2.1.2
module load boost/1.68.0

# Other Software ################################
module load zlib/1.2.11
module load c-blosc/1.14.4
module load adios/1.13.1
module load hdf5-parallel/1.8.20
module load libbash/1.7.0
module load libpng/1.6.35
module load pngwriter/0.7.0

# Environment ################################
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:skylake-avx512"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ################################
# - SLURM (sbatch)
# - "defq" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/defq.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
}

(continues on next page)
fi
    srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --cpus-per-task=20 --mem=360000 -p defq --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ]; then
        numDevices=1
    else
        if [ "$1" -gt 2 ]; then
            echo "The maximal number of devices per node is 2." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    srun --time=1:00:00 --ntasks-per-node=$(($numDevices)) --cpus-per-task=$(($20 * $numDevices)) --mem=$(($180000 * $numDevices)) -p defq --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." 1>&2
    fi

Queue: gpu (4x NVIDIA P100 16GB)

# Name and Path of this Script #####################################
# (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"/$(basename $BASH_SOURCE)

# User Information ##################################################
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$ (whoami) <$MY_MAIL>"

# Text Editor for Tools #############################################
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
# export EDITOR="nano"

# General modules ###############################################
# module purge
module load gcc/7.3.0
module load cmake/3.15.2
module load cuda/10.0
module load openmpi/2.1.2-cuda100
module load boost/1.68.0

# Other Software ############################################
# module load zlib/1.2.11

(continues on next page)
module load c-blosc/1.14.4
module load adios/1.13.1-cuda100
module load hdf5-parallel/1.8.20-cuda100
module load libsplash/1.7.0-cuda100
module load libpng/1.6.35
module load pngwriter/0.7.0

# Environment #################################################################
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #################################################################
# - SLURM (sbatch)
# - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/gpu.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ]; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --cpus-per-task=6 --gres=gpu:4 --mem=378000 -p gpu --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ]; then
    numGPUs=1
  else
    if [ "$1" -gt 4 ]; then
      echo "The maximal number of devices per node is 4." 1>&2
      return 1
    else
      numGPUs=$1
    fi
  fi
  srun --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=6 --gres=gpu:$numGPUs --mem=$((94500 * numGPUs)) -p gpu --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then

(continues on next page)
source $BASH_COMP_FILE

else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

Queue: fwkt_v100 (4x NVIDIA V100 32GB)

# Name and Path of this Script ############################################### (DO NOT change!)
export PIC_PROFILE="$(cd $(dirname $BASH_SOURCE) && pwd)/""/
export PIC_PROFILE="/""/
export PIC_PROFILE="$(basename $BASH_SOURCE)

# User Information ########################################################## (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$WHOAMI <$MY_MAIL>"

# Text Editor for Tools #################################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
export EDITOR="nano"

# General modules ###########################################################
#
module purge
module load gcc/7.3.0
module load cmake/3.15.2
module load cuda/10.0
module load openmpi/2.1.2-cuda100
module load boost/1.68.0

# Other Software ############################################################
#
module load zlib/1.2.11
module load c-blosc/1.14.4
module load adios/1.13.1-cuda100
module load hdf5-parallel/1.8.20-cuda100
module load libsplash/1.7.0-cuda100
module load libpng/1.6.35
module load pngwriter/0.7.0

# Environment #################################################################
#
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ####################################################
# - SLURM (sbatch)
**# - "fwkt_v100" queue**

export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/fwkt_v100.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)

function getNode() {
  if [ -z "$1" ]; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --cpus-per-task=6 --gres=gpu:4 --mem=378000 -p fwkt_v100 -A fwkt_v100 --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)

function getDevice() {
  if [ -z "$1" ]; then
    numGPUs=1
  else
    if [ "$1" -gt 4 ]; then
      echo "The maximal number of devices per node is 4." 1>&2
      return 1
    else
      numGPUs=$1
    fi
  fi
  srun --time=1:00:00 --ntasks-per-node=$((numGPUs)) --cpus-per-task=6 --gres=gpu:numGPUs --mem=$((94500 * numGPUs)) -p fwkt_v100 -A fwkt_v100 --pty bash
}

# Load autocompletion for PIConGPU commands

if [ -f $BASH_COMP_FILE ]; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

---

**Queue: k20 (4x Nvidia K20m GPUs 4.7GB)**

---

1.4. picongpu.profile
module purge
module load gcc/7.3.0
module load cmake/3.15.2
module load cuda/10.0
module load openmpi/2.1.2-cuda100
module load boost/1.68.0

# Other Software ################################################################
#
module load zlib/1.2.11
module load c-blosc/1.14.4
module load adios/1.13.1-cuda100
module load hdf5-parallel/1.8.20-cuda100
module load libsplash/1.7.0-cuda100
module load libpng/1.6.35
module load pngwriter/0.7.0

# Environment ###################################################################
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ##########################################################
# - SLURM (sbatch)
# - "k20" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/k20.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --cpus-per-task=2 --gres=gpu:4 -A k20 --mem=62000 -p k20 --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numGPUs=1
  else
    numGPUs=$1
  fi
  echo "The maximal number of devices per node is 4." 1>&2
  return 1
else
    numGPUs=$1
fi
if	srun --time=1:00:00 --ntasks-per-node=\$((numGPUs)) --cpus-per-task=2 --gres=gpu:numGPUs -A k20 --mem=\$((15500 * numGPUs)) -p k20 --pty bash
# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file 'BASH_COMP_FILE' not found." >&2
fi
Queue: k80 (8x NVIDIA K80 12GB)
# Name and Path of this Script ################################################################# (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)
# User Information #################################################################### (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"
# Text Editor for Tools #################################################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"
# General modules ####################################################################
# module purge
module load gcc/7.3.0
module load cmake/3.15.2
module load cuda/10.0
module load openmpi/2.1.2-cuda100
module load boost/1.68.0
# Other Software ####################################################################
# module load zlib/1.2.11
module load c-blosc/1.14.4
module load adios/1.13.1-cuda100
module load hdf5-parallel/1.8.20-cuda100
module load libisplash/1.7.0-cuda100
module load libpng/1.6.35
module load pngwriter/0.7.0
# Environment ###################################################################
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
(continues on next page)
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options
# SLURM (sbatch)
# "k80" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/k80.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=8 --cpus-per-task=2 --gres=gpu:8 -A k80 --mem=238000 -p k80 --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numGPUs=1
  else
    if [ "$1" -gt 8 ] ; then
      echo "The maximal number of devices per node is 8." 1>&2
      return 1
    else
      numGPUs=$1
    fi
  fi
  srun --time=1:00:00 --nodes=$numGPUs --ntasks-per-node=$numGPUs --cpus-per-task=2 --gres=gpu:$numGPUs -A k80 --mem=$((29750 * $numGPUs)) -p k80 --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.2 Hypnos (HZDR)

System overview: [link (internal)]

User guide: [link (internal)]

Production directory: /bigdata/hplsim/ with external/, scratch/, development/ and production/
For these profiles to work, you need to download the **PIConGPU source code** manually.

**Queue: laser (AMD Opteron 6276 CPUs)**

```bash
# Name and Path of this Script ######################## (DO NOT change!)
export PIC_PROFILE="$($cd $$($dirname $$BASH_SOURCE) $$pwd)/"$$($basename $$BASH_SOURCE)

# User Information ################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me mails on batch system job (b)egin, (e)nd, (a)bortion or (n)o mail
export MY_MAILNOTIFY="n"
export MY_MAIL="someone@example.com"
export MY_NAME="$$($whoami) <$$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules #####################################################################

if [ -f /etc/profile.modules ]; then
  . /etc/profile.modules
  module purge
  # export MODULES_NO_OUTPUT=1

  # Core Dependencies
  module load gcc/5.3.0
  module load cmake/3.13.4
  module load boost/1.65.1
  module load openmpi/1.8.6
  module load numactl

  # Plugins (optional)
  module load zlib/1.2.8
  module load pngwriter/0.7.0
  module load hdf5-parallel/1.8.15 libsplash/1.7.0

  # either use libSplash or ADIOS for file I/O
  #module load adios/1.13.1

  # Debug Tools
  #module load gdb
  #module load valgrind/3.8.1

  # unset MODULES_NO_OUTPUT
fi

# Environment #################################################################

alias getNode='qsub -I -q laser -lwalltime=00:30:00 -lnodes=1:ppn=64'
export PICSRC=/home/$($whoami)/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:bdver1"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/splash2txt/build
export PATH=$PATH:$PICSRC/src/tools/bin

(continues on next page)
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options
# - PBS/Torque (qsub)
# - "laser" queue
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="etc/picongpu/hypnos-hzdr/laser.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.3 Hydra (HZDR)

System overview: link (internal)
User guide: link (internal)
Production directory: /bigdata/hplsim/ with external/, scratch/, development/ and production/

For this profile to work, you need to download the PIConGPU source code manually.

```bash
export PIC_PROFILE="$(cd $(dirname $BASH_SOURCE) && pwd)"/$(basename $BASH_SOURCE)

# User Information
# - automatically add your name and contact to output file meta data
# - send me mails on batch system job (b)egin, (e)nd, (a)bortion or (n)o mail
export MY_MAILNOTIFY="n"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules
if [ -f /etc/profile.modules ]
then
  . /etc/profile.modules
  module purge
  export MODULES_NO_OUTPUT=1

  # Core Dependencies
  module load gcc/5.3.0
  module load cmake/3.13.4
  module load boost/1.65.1
  module load openmpi/1.8.6
  module load numactl

  # Plugins (optional)
  module load pngwriter/0.7.0
  module load hdf5-parallel/1.8.15 libsplash/1.7.0
```

(continues on next page)
# either use libSplash or ADIOS for file I/O
#module load adios/1.13.1

# Debug Tools
#module load gdb
#module load valgrind/3.8.1

# unset MODULES_NO_OUTPUT
fi

# Environment #################################################################

alias getNode='qsub -I -q default -lwalltime=00:30:00 -lnodes=1:ppn=32'

export PICSRC=/home/$(whoami)/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:ivybridge"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/splash2txt/build
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/src/tools/lib/python:$PYTHONPATH

# "tbg" default options #################################################################
# - PBS/Torque (qsub)
# - "default" queue
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="etc/picongpu/hydra-hzdr/default.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.4 Summit (ORNL)

System overview: link

User guide: link

Production directory: usually $PROJWORK/$proj/ (link). Note that $HOME is mounted on compute nodes as read-only.

For this profile to work, you need to download the PIConGPU source code and install libSplash and PNGwriter manually.

V100 GPUs (recommended)

# Name and Path of this Script ################################################################# (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/""$(basename $BASH_SOURCE)

# User Information ##################################################################### (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on job (-B)egin, Fi(-N)ish

(continues on next page)
export MY_MAILNOTIFY=""
export MY_MAIL="someone@example.com"
export MY_NAME="$\$(whoami) <MY_MAIL>"

# Project Information ########################################## (edit this line)
# - project account for computing time
export proj=<yourProject>

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#module load nano
#export EDITOR="nano"

# basic environment ###########################################
module load gcc/6.4.0
export CC=$\$(which gcc)
export CXX=$\$(which g++)

# required tools and libs
module load git
module load cmake/3.14.2
module load cuda/10.1.168
module load boost/1.66.0

# plugins (optional) ##########################################
module load hdf5/1.10.3
module load adios/1.13.1-py2 c-blosc zfp sz lz4

# optionally download libSplash and compile it yourself from
# https://github.com/ComputationalRadiationPhysics/libSplash/
# export Splash_ROOT=<your libSplash install directory> # e.g., ${HOME}/sw/
# export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$Splash_ROOT/lib
#export T3PIO_ROOT=$PROJWORK/$proj/lib/t3pio
#export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$T3PIO_ROOT/lib

module load zlib/1.2.11
module load libpng/1.6.34 freetype/2.9.1

# optionally install pngwriter yourself:
# https://github.com/pngwriter/pngwriter#install
# export PNGwriter_ROOT=<your pngwriter install directory> # e.g., ${HOME}/sw/
# export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$PNGwriter_ROOT/lib

# helper variables and tools ###################################
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

alias getNode="bsub -P $proj -W 2:00 -nnodes 1 -Is /bin/bash"

# "tbg" default options ########################################
export TBG_SUBMIT="bsub"
export TBGTplFILE="etc/picongpu/summit-ornl/gpu_batch.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
else
    echo "bash completion file \"$BASH_COMP_FILE\" not found." >&2
fi

1.4.5 Piz Daint (CSCS)

System overview: [link]
User guide: [link]
Production directory: $SCRATCH (link).

For this profile to work, you need to download the PIConGPU source code and install boost, zlib, libpng, c-blosc, PNGwriter, libSplash and ADIOS manually.

Note: The MPI libraries are lacking Fortran bindings (which we do not need anyway). During the install of ADIOS, make sure to add the --disable-fortran flag.

Note: Please find a Piz Daint quick start from August 2018 here.

# Name and Path of this Script ################################################### (DO NOT change!)
export PIC_PROFILE=$(
    cd $(dirname $BASH_SOURCE) && pwd)="/$(basename $BASH_SOURCE)

# User Information # (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAILED, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$whoami <$MY_MAIL>"

# Text Editor for Tools ######################################################### (edit those lines)
# examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
# module load nano
#export EDITOR="nano"

# Programming Environment ########################################################
# # if the wrong environment is loaded we switch to the gnu environment
# note: this loads gcc/5.3.0 (6.0.4 is the version of the programming env!)
CRAYENV_FOUND=$(module li 2>&1 | grep "PrgEnv-cray" > /dev/null && { echo 0; } ||
    { echo 1; })
if [ $CRAYENV_FOUND -eq 0 ]; then
    module swap PrgEnv-cray PrgEnv-gnu/6.0.4
else
    module load PrgEnv-gnu/6.0.4
fi
module load daint-gpu
# currently loads CUDA 8.0
module load craype-accel-nvidia60

(continues on next page)
# Compile for cluster nodes
# (CMake likes to unwrap the Cray wrappers)
export CC=$(which cc)
export CXX=$(which CC)

# define cray compiler target architecture
# if not defined the linker crashed because wrong from */lib instead
# of */lib64 are used
export CRAY_CPU_TARGET=x86-64

# Libraries ###################################################################
module load CMake/3.11.4
module load cray-mpich/7.6.0
module load cray-hdf5-parallel/1.10.0.3

# Self-Build Software ###################################################################
#
# needs to be compiled by the user
export PIC_LIBS=$HOME/lib
export BOOST_ROOT=$PIC_LIBS/boost-1.65.1
export ZLIB_ROOT=$PIC_LIBS/zlib-1.2.11
export PNG_ROOT=$PIC_LIBS/libpng-1.6.34
export BLOSC_ROOT=$PIC_LIBS/blosc-1.12.1
export PNGwriter_DIR=$PIC_LIBS/pngwriter-0.7.0
export ADIOS_ROOT=$PIC_LIBS/adios-1.13.1
export Splash_DIR=$PIC_LIBS/splash-1.7.0

export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$ZLIB_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$PNG_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$PNGwriter_DIR/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$ADIOS_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$Splash_DIR/lib:$LD_LIBRARY_PATH

export PATH=$PNG_ROOT/bin:$PATH
export PATH=$ADIOS_ROOT/bin:$PATH
export PATH=$Splash_DIR/bin:$PATH

export CMAKE_PREFIX_PATH=$ZLIB_ROOT:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$PNG_ROOT:$CMAKE_PREFIX_PATH

export MPI_ROOT=$MPICH_DIR
export HDF5_ROOT=$HDF5_DIR

# Environment ###################################################################
#
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ###################################################################
# - SLURM (sbatch)
# - "normal" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/pizdaint-cscs/normal.tpl"

# allocate tools

# allocate an interactive shell for one hour
getNode 2 # allocates two interactive nodes (default: 1)
getNode() {
    if [ -z "$1" ]; then
        numNodes=1
    else
        numNodes=$1
    fi
    # --ntasks-per-core=2 # activates intel hyper threading
    salloc --time=1:00:00 --nodes="$numNodes" --ntasks-per-node=12 --ntasks-per-
        core=2 --partition normal --gres=gpu:1 --constraint=gpu
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.6 Taurus (TU Dresden)

System overview: link

User guide: link

Production directory: /scratch/$USER/ and /scratch/$proj/

For these profiles to work, you need to download the PIConGPU source code and install PNGwriter and libSplash manually.

Queue: gpu1 (Nvidia K20x GPUs)

# Name and Path of this Script (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"$basename $BASH_SOURCE"

# User Information (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $1}')

# Text Editor for Tools (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
export EDITOR="nano"

# Modules

(continues on next page)
module load modenv/scs5
module load foss/2018a
module load GCC/6.4.0-2.28
module load CMake/3.11.4-GCCcore-6.4.0
module load CUDA/9.2.88 # gcc <= 7, intel 15-17
module load OpenMPI/2.1.2-GCC-6.4.0-2.28
module load git/2.18.0-GCCcore-6.4.0
module load gnuplot/5.2.4-foss-2018a
module load Boost/1.66.0-foss-2018a
# currently not linking correctly:
#module load HDF5/1.10.1-foss-2018a
module load zlib/1.2.11-GCCcore-6.4.0
# module system does not export cmake prefix path:
export CMAKE_PREFIX_PATH=$EBROOTLIBPNG:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$CMAKE_PREFIX_PATH
# Environment #################################################################
#
# path to own libraries:
export ownLibs=$HOME
# workaround HDF5:
export HDF5_ROOT=$ownLibs/lib/hdf5
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH
# pngwriter needs to be built by the user:
export PNGwriter_DIR=$ownLibs/lib/pngwriter
export CMAKE_PREFIX_PATH=$PNGwriter_DIR:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PNGwriter_DIR/lib/
# splash needs to be built by the user:
export Splash_DIR=$ownLibs/lib/splashModule2
export CMAKE_PREFIX_PATH=$Splash_DIR:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$Splash_DIR/lib/
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH
# "tbg" default options #################################################################
# - SLURM (sbatch)
# - "gpu1" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/k20x.tpl"
# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
(continues on next page)
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

Queue: gpu2 (Nvidia K80 GPUs)

```bash
# Name and Path of this Script ########################################## (DO NOT change!)
ex
port PIC_PROFILE=$
d(irname $BASH_SOURCE) $& pxr)"/"$(basename $BASH_SOURCE)

# User Information ###################################################### (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_90 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
ex
port MY_MAIL="someone@example.com"
ex
port MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information #################################################### (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $1}')

# Text Editor for Tools ##################################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
ex
port EDITOR="nano"

# Modules ####################################################################
#
module load modenv/scs5
module load foss/2018a
module load GCC/6.4.0-2.28
module load CMake/3.11.4-GCCcore-6.4.0
module load CUDA/9.2.88 # gcc <= 7, intel 15-17
module load OpenMPI/2.1.2-GCC-6.4.0-2.28
module load git/2.18.0-GCCcore-6.4.0
module load gnuplot/5.2.4-foss-2018a
module load Boost/1.66.0-foss-2018a
# currently not linking correctly:
#module load HDF5/1.10.1-foss-2018a
module load zlib/1.2.11-GCCcore-6.4.0

# module system does not export cmake prefix path:
ex
port CMAKE_PREFIX_PATH=$EBROOTGZIP:$CMAKE_PREFIX_PATH
ex
port CMAKE_PREFIX_PATH=$EBROOTZLIB:$CMAKE_PREFIX_PATH

# Environment #################################################################
#
# path to own libraries:
ex
port ownLibs=$HOME

# workaround HDF5:
ex
port HDF5_ROOT=$ownLibs/lib/hdf5
ex
port LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH
ex
port CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH

# pngwriter needs to be built by the user:
ex
port PNGwriter_DIR=$ownLibs/lib/pngwriter

(continues on next page)
export CMAKE_PREFIX_PATH=${PNGwriter_DIR}:${CMAKE_PREFIX_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${PNGwriter_DIR}/lib/

# splash needs to be built by the user:
export Splash_DIR=${ownLibs}/lib/splashModule2
export CMAKE_PREFIX_PATH=${Splash_DIR}:${CMAKE_PREFIX_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${Splash_DIR}/lib/

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ####################################################
# - SLURM (sbatch)
# - "gpu2" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/k80.tpl"

alias getNode='srun -p gpu2-interactive --gres=gpu:4 -n 1 --pty --mem=0 -t 2:00:00 bash'

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

Queue: knl (Intel Xeon Phi - Knights Landing)

For this profile, you additionally need to install your own *boost*.

 exporting PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/$(basename $BASH_SOURCE)

# User Information ################################################################ (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ################################################################ (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $1}')

# Text Editor for Tools ################################################################ (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
export EDITOR="nano"

# Modules ###################################################################
# module load modenv/scs5
module load impi/2018a
module load git/2.18.0-GCCcore-6.4.0
module load CMake/3.11.4-GCCcore-7.3.0
module load Boost/1.66.0-intel-2018a
module load HDF5/1.10.1-intel-2018a
module load libpng/1.6.34-GCCcore-7.3.0

# module system does not export cmake prefix path:
export CMAKE_PREFIX_PATH=$EBROOTLIBPNG:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$CMAKE_PREFIX_PATH

# Environment ###################################################################
#
# compilers are not set correctly by the module system:
export CC=`which icc`
export CXX=$CC
# path to own libraries:
export ownLibs=$HOME
export PNGwriter_DIR=$ownLibs/lib/pngwriter
export CMAKE_PREFIX_PATH=$PNGwriter_DIR:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PNGwriter_DIR/lib/
export Splash_DIR=$ownLibs/lib/splash
export CMAKE_PREFIX_PATH=$Splash_DIR:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$Splash_DIR/lib/
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:MIC-AVX512"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ###################################################################
# - SLURM (sbatch)
# - "knl" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/knl.tpl"

alias getNode='srun -p knl -N 1 -c 64 --mem=90000 --constraint="Quadrant&Cache" --pty bash'

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
Queue: ml (NVIDIA V100 GPUs on Power9 nodes)

For this profile, you additionally need to compile and install everything for the power9-architecture including your own boost, HDF5, c-blosc and ADIOS.

Note: Please find a Taurus ml quick start here.

Note: You need to compile the libraries and PICongPU on an ml node since only nodes in the ml queue are Power9 systems.

```bash
# Name and Path of this Script ################################ (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)

# User Information ################################ (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) "$MY_MAIL"

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules ####################################################################
module switch modenv/ml
module load CUDa/9.2.88-GCC-7.3.0-2.30, also loads GCC/7.3.0-2.30, zlib, OpenMPI and others
module load fosscuda/2018b
module load CMake/3.11.4-GCCcore-7.3.0
module load libpng/1.6.34-GCCcore-7.3.0
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
export LD_LIBRARY_PATH=$ADIOS_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$Splash_DIR/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$Splash_DIR:$CMAKE_PREFIX_PATH

export LD_LIBRARY_PATH=$PICSRC:$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

# python not included yet
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# This is necessary in order to make alpaka compile.
# The workaround is from Axel Huebl according to alpaka PR #702.
export CXXFLAGS="-Dlinux"

# "tbg" default options
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/V100.tpl"

# allocate an interactive shell for two hours
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  export OMP_NUM_THREADS=7
  srun --time=2:00:00 --nodes=$numNodes --ntasks=$((6 * $numNodes)) --ntasks-per-node=6 --cpus-per-task=7 --mem=0 --exclusive --gres=gpu:6 -p ml --pty bash
}

# allocate an interactive shell for two hours
# getDevice 2 # allocates 2 interactive devices on one node (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numDevices=1
  else
    if [ "$1" -gt 6 ] ; then
      echo "The maximal number of devices per node is 6." 1>&2
      return 1
    else
      numDevices=$1
    fi
  fi
  export OMP_NUM_THREADS=7
  srun --time=2:00:00 --nodes=1 --ntasks=$numDevices --ntasks-per-node=$((--$numDevices)) --cpus-per-task=7 --mem=$((254000 / $numDevices)) --gres=gpu:--$numDevices -p ml --pty bash
}
# Load autocompletion for PIConGPU commands

```
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

1.4.7 Lawrencium (LBNL)

System overview: link

User guide: link

Production directory: /global/scratch/$USER/

For this profile to work, you need to download the PIConGPU source code and install boost, PNGwriter and libSplash manually. Additionally, you need to make the rsync command available as written below.

```
# Name and Path of this Script ######################################### (DO NOT change!)
export PIC_PROFILE="$(cd $(dirname $BASH_SOURCE) && pwd)/$(basename $BASH_SOURCE)"

# User Information ################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL, TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules #####################################################################
# if [ -f /etc/profile.d/modules.sh ]
# then
  . /etc/profile.d/modules.sh
  module purge
  # Core Dependencies
  module load gcc
  module load cuda
  echo "WARNING: Boost version is too old! (Need: 1.65.1+)" >&2
  # module load boost/1.65.1-gcc
  module load openmpi/1.6.5-gcc
  # Core tools
  module load git
  module load cmake
  module load python/2.6.6
  module load ipython/0.12 matplotlib/1.1.0 numpy/1.6.1 scipy/0.10.0
  # Plugins (optional)
  module load hdf5/1.8.11-gcc-p
  export CMAKE_PREFIX_PATH=$HOME/lib/pngwriter:$CMAKE_PREFIX_PATH
  export CMAKE_PREFIX_PATH=$HOME/lib/libSplash:$CMAKE_PREFIX_PATH
```

(continues on next page)
export LD_LIBRARY_PATH=$HOME/lib/pngwriter/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HOME/lib/libSplash/lib:$LD_LIBRARY_PATH

# Debug Tools
#module load valgrind/3.10.1
#module load totalview/8.10.0-0

fi

# Environment #################################################################
#
alias allocK20='salloc --time=0:30:00 --nodes=1 --ntasks-per-node=1 --cpus-per-
-->task=8 --partition lr_manycore'
alias allocFermi='salloc --time=0:30:00 --nodes=1 --ntasks-per-node=2 --cpus-per-
-->task=6 --partition mako_manycore'

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:20"

# fix pic-create: re-enable rsync
# ssh lrc-xfer.scs00
# -> cp /usr/bin/rsync $HOME/bin/
export PATH=$HOME/bin:$PATH
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/splash2txt/build
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #################################################################
# - SLURM (sbatch)
# - fermi queue (also available: 2 K20 via k20.tpl)
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/lawrencium-lbnl/fermi.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file "$BASH_COMP_FILE" not found." >&2
fi

## 1.4.8 Cori (NERSC)

System overview: link
User guide: link
Production directory: $SCRATCH (link).

For these profiles to work, you need to download the PIConGPU source code and install PNGwriter and libSplash manually.
Queue: regular (Intel Xeon Phi - Knights Landing)

```
# Name and Path of this Script ########################### (DO NOT change!)
export PIC_PROFILE="$(cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) "<MY_MAIL>"

# Project Information ################################ (edit this line)
# - project account for computing time
export proj="<yourProject>"

# Text Editor for Tools ################################ (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #################################################################
# module swap craype-haswell craype-mic-knl
module swap PrgEnv-intel PrgEnv-gnu  # GCC 8.2.0
module load cmake/3.14.4
module load boost/1.70.0
# Other Software ##################################################################
# module load adios/1.13.1
module load cray-hdf5-parallel/1.10.2.0
module load png/1.6.34

export Splash_ROOT="${HOME}/sw/libSplash-1.7.0-8-gb9421ba"
export PNGwriter_ROOT="${HOME}/sw/pngwriter-0.7.0-21-g9dc58ed"

# Environment########################################################################
# export CC="$(which cc)"
export CXX="$(which CC)"
export CRAYPE_LINK_TYPE=dynamic

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b"  # usually ":MIC-AVX512" but we use PrgEnv wrappers

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ############################################################
# - SLURM (sbatch)
# - "defq" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/cori-nersc/knl.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
```

(continues on next page)
if [ -z "$1" ] ; then
  numNodes=1
else
  numNodes=$1
fi
srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=1 --cpus-per-task=64 -C "knl,quad,cache" -p regular --pty bash

# Load autocompletion for PICoGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.9 Draco (MPCDF)

System overview: link
User guide: link
Production directory: /ptmp/$USER/

For this profile to work, you need to download the PICoGPU source code and install libpng, PNGwriter and libSplash manually.

# Name and Path of this Script ################################ (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General Modules #############################################################
# module purge
module load git/2.14
module load gcc/6.3
module load cmake/3.11.4
module load boost/gcc/1.64
module load impi/2017.3
module load hdf5-mpi/gcc/1.8.18

# Other Software #############################################################
#
# needs to be compiled by the user
export PNGWRITER_ROOT=$HOME/lib/pngwriter-0.7.0
export SPLASH_ROOT=$HOME/lib/splash-1.7.0

(continues on next page)
export LD_LIBRARY_PATH=$PNGWRITER_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$SPLASH_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BOOST_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HDF5_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$I_MPI_ROOT/lib64:$LD_LIBRARY_PATH
export HDF5_ROOT=$HDF5_HOME
export CXX=$(which g++)
export CC=$(which gcc)

# PIConGPU Helper Variables ##############################################
# export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:haswell"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ###################################################
# - SLURM (sbatch)
# - "normal" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/draco-mpcdf/general.tpl"

# helper tools #################################################################
# allocate an interactive shell for one hour
alias getNode='salloc --time=1:00:00 --nodes=1 --exclusive --ntasks-per-node=2 --cpus-per-task=32 --partition general'

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.10 D.A.V.I.D.E (CINECA)

System overview: link

User guide: link

Production directory: $CINECA_SCRATCH/ (link)

For this profile to work, you need to download the PIConGPU source code manually.

Queue: dvd_usr_prod (Nvidia P100 GPUs)

# Name and Path of this Script (DO NOT change!)  
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) 2>/dev/null)/"$"(basename $BASH_SOURCE)

# User Information (edit the following lines)  
(continues on next page)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>

# Project Information ######################################## (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $2}')

# Text Editor for Tools ###################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################

module purge
module load gnu/6.4.0
module load cmake/3.11.4
module load cuda/9.2.88
module load openmpi/3.1.0--gnu--6.4.0
module load boost/1.68.0--openmpi--3.1.0--gnu--6.4.0
export CMAKE_PREFIX_PATH=$CUDA_HOME:$OPENMPI_HOME:$CMAKE_PREFIX_PATH

# Other Software ##############################################################

module load zlib/1.2.11--gnu--6.4.0
module load szip/2.1.1--gnu--6.4.0
module load blosc/1.12.1--gnu--6.4.0
module load hdf5/1.10.4--openmpi--3.1.0--gnu--6.4.0
module load libsplash/1.7.0--openmpi--3.1.0--gnu--6.4.0
module load adios/1.13.1--openmpi--3.1.0--gnu--6.4.0
module load libpng/1.6.35--gnu--6.4.0
module load freetype/2.9.1--gnu--6.4.0
module load pngwriter/0.7.0--gnu--6.4.0
export CMAKE_PREFIX_PATH=$ZLIB_HOME:$SZIP_HOME:$BLOSC_HOME:$CMAKE_PREFIX_PATH

# Work-Arounds ################################################################

# fix for Nvidia NVCC bug id 2448610
# see https://github.com/ComputationalRadiationPhysics/alpaka/issues/701
export CXXFLAGS="--Dlinux"

# Environment #################################################################

#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"

export PATH=$PATH:$PICSRC

(continues on next page)
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options
# SLURM (sbatch)
# "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/davide-cineca/gpu.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates two interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=0:30:00 --nodes=$numNodes --ntasks-per-socket=8 --ntasks-per-node=16 --mem=252000 --gres=gpu:4 -A $proj -p dvd_usr_prod --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numGPUs=1
  else
    if [ "$1" -gt 4 ] ; then
      echo "The maximal number of devices per node is 4." 1>&2
      return 1
    else
      numGPUs=$1
    fi
  fi
  srun --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=$((4 * $numGPUs)) --gres=gpu:$numGPUs --mem=$((63000 * $numGPUs)) -A $proj -p dvd_usr_prod --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

1.4.11 JURECA (JSC)

System overview: link
User guide: link
Production directory: $SCRATCH (link)

For these profiles to work, you need to download the PIConGPU source code and install PNGwriter, c-blosc, adios and libSplash, for the gpus partition also Boost and HDF5, manually.
Queue: batch (2 x Intel Xeon E5-2680 v3 CPUs, 12 Cores + 12 Hyperthreads/CPU)

```
# Name and Path of this Script ############################################ (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)

# User Information ############################################################ (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#   TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ########################################################## (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $5}"

# Text Editor for Tools ######################################################## (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules ################################################################
# module purge
module load Intel/2019.0.117-GCC-7.3.0
module load CMake/3.13.0
module load IntelIMPI/2018.4.274
module load Python/3.6.6
module load Boost/1.68.0-Python-3.6.6

# Other Software ################################################################
# module load zlib/.1.2.11
module load HDF5/1.10.1
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_batch
LIBSPLASH_ROOT=$PARTITION_LIB/libSplash
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$LIBSPLASH_ROOT:$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

ADIOS_ROOT=$PARTITION_LIB/adios
export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$ADIOS_ROOT:$CMAKE_PREFIX_PATH

# Environment ####################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:haswell"
export PATH=$PATH:$PICSRC

(continues on next page)
```
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export CC=$(which icc)
export CXX=$(which icpc)
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options
# - SLURM (sbatch)
# - "batch" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/batch.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  if [ $numNodes -gt 8 ] ; then
    echo "The maximal number of interactive nodes is 8." 1>&2
    return 1
  fi
  echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple processes in the interactive mode"
  export OMP_NUM_THREADS=24
  salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --mem=126000 -A $proj -p devel bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates 2 interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numDevices=1
  else
    if [ "$1" -gt 2 ] ; then
      echo "The maximal number of devices per node is 2." 1>&2
      return 1
    else
      numDevices=$1
    fi
  fi
  echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple processes in the interactive mode"
  export OMP_NUM_THREADS=24
  salloc --time=1:00:00 --ntasks-per-node=$((numDevices)) --mem=126000 -A $proj -p devel bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
Queue: gpus (2 x Nvidia Tesla K80 GPUs)

```bash
# Name and Path of this Script ################################### (DO NOT change!)
export PIC_PROFILE="$($cd $(dirname $BASH_SOURCE) && pwd)/"$(basename $BASH_SOURCE)

# User Information ############################################ (edit the following lines)
# - automatically add your name and contact to output file metadata
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$whoami <$MY_MAIL>

# Project Information ######################################## (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $5}')

# Text Editor for Tools ##################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules ###############################################
# module purge
module load GCC/7.3.0
module load CUDA/9.2.88
module load CMake/3.13.0
module load MVAPICH2/2.3-GDR
module load Python/3.6.6

# Other Software ###############################################
# module load zlib/.1.2.11
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_gpus
BOOST_ROOT=$PARTITION_LIB/boost
export CMAKE_PREFIX_PATH=$BOOST_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH

HDF5_ROOT=$PARTITION_LIB/hdf5
export PATH=$HDF5_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH

LIBSPLASH_ROOT=$PARTITION_LIB/libSplash
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$LIBSPLASH_ROOT:$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

ADIOS_ROOT=$PARTITION_LIB/adios
export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$ADIOS_ROOT:$CMAKE_PREFIX_PATH

# Environment #####################################################
```

(continues on next page)
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37" # Nvidia K80 architecture
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ################################################################################
# - SLURM (sbatch)
# - "gpus" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/gpus.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ] ; then
    numNodes=1
  else
    numNodes=$1
  fi
  if [ $numNodes -gt 8 ] ; then
    echo "The maximal number of interactive nodes is 8." 1>&2
    return 1
  fi
  echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple processes in the interactive mode"
  salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --gres=gpu:4 --mem=126000 -A $proj -p develgpus bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates 2 interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ] ; then
    numDevices=1
  else
    if [ "$1" -gt 4 ] ; then
      echo "The maximal number of devices per node is 4." 1>&2
      return 1
    else
      numDevices=$1
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple processes in the interactive mode"
    salloc --time=1:00:00 --ntasks-per-node=\{\$numDevices\} --gres=gpu:4 --mem=126000 -A $proj -p develgpus bash
  fi
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
Queue: booster (Intel Xeon Phi 7250-F, 68 cores + Hyperthreads)

```bash
# Name and Path of this Script ############################################# (DO NOT change!)
export PIC_PROFILE=$(
    cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ######################################################### (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>

# Project Information ######################################################## (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $5}"

# Text Editor for Tools ####################################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules ################################################################
# module purge
module load Architecture/KNL
module load Intel/2019.0.117-GCC-7.3.0
module load CMake/3.12.3
module load IntelMPI/2018.4.274
module load Python/3.6.6
module load Boost/1.68.0-Python-3.6.6

# Other Software ################################################################
# module load szip/.1.2.11
module load HDF5/1.10.1
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH
PARTITION_LIB=$PROJECT/lib_booster
LIBSPLASH_ROOT=$PARTITION_LIB/libSplash
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$LIBSPLASH_ROOT:$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH
BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH
ADIOS_ROOT=$PARTITION_LIB/adios
export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$ADIOS_ROOT:$CMAKE_PREFIX_PATH

# Environment ###################################################################
```

(continues on next page)
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:MIC-AVX512"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export CC=$(which icc)
export CXX=$(which icpc)

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ####################################################################
# - SLURM (sbatch)
# - "booster" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/booster.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
if [ -z "$1" ] ; then
    numNodes=1
else
    numNodes=$1
fi
if [ $numNodes -gt 8 ] ; then
    echo "The maximal number of interactive nodes is 8." 1>&2
    return 1
fi
export OMP_NUM_THREADS=34
salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --mem=94000 -A $proj -p develbooster bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates 2 interactive devices (default: 1)
function getDevice() {
if [ -z "$1" ] ; then
    numDevices=1
else
    if [ "$1" -gt 1 ] ; then
        echo "The maximal number of devices per node is 4." 1>&2
        return 1
    else
        numDevices=$1
    fi
    export OMP_NUM_THREADS=34
    overclock --time=1:00:00 --ntasks-per-node=${$numDevices} --mem=94000 -A $proj -p develbooster bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ ! -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    (continues on next page)
1.4.12 JUWELS (JSC)

**System overview:** [link]

**User guide:** [link]

**Production directory:** `$SCRATCH` ([link])

For these profiles to work, you need to download the *PIConGPU source code* and install *PNGwriter, c-blosc, adios and libSplash*, for the gpus partition also *Boost and HDF5*, manually.

**Queue:** batch (2 x Intel Xeon Platinum 8168 CPUs, 24 Cores + 24 Hyperthreads/CPU)

```
# Name and Path of this Script ########################################## (DO NOT change!)
export PIC_PROFILE=
$(
  cd
  $(dirname $BASH_SOURCE)
  && pwd
)
"/"

# User Information ################################################# (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $4}')</g>

# Text Editor for Tools ################################### (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules #################################################################
#
module purge
module load Intel/2019.0.117-GCC-7.3.0
module load CMake/3.13.0
module load IntelMPI/2018.4.274
module load Python/3.6.6
module load Boost/1.68.0-Python-3.6.6

# Other Software ##################################################################
#
module load zlib/.1.2.11
module load HDF5/1.10.1
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH
PARTITION_LIB=$PROJECT/lib_batch
LIBSPLASH_ROOT=$PARTITION_LIB/libSplash
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$LIBSPLASH_ROOT:$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH
```

(continues on next page)
BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

ADIOS_ROOT=$PARTITION_LIB/adios
export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$ADIOS_ROOT:$CMAKE_PREFIX_PATH

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:skylake"

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
export CC=$(which icc)
export CXX=$(which icpc)

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #################################################################
# - SLURM (sbatch)
# - "batch" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/juwels-jsc/batch.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ]; then
    numNodes=1
  else
    numNodes=$1
  fi
  if [ $numNodes -gt 8 ]; then
    echo "The maximal number of interactive nodes is 8." 1>&2
    return 1
  fi
  echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching
multiple processes in the interactive mode"
  export OMP_NUM_THREADS=48
  salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --mem=94000 -A
  --$proj -p batch bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates 2 interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ]; then
    numDevices=1
  else
    if [ "$1" -gt 2 ]; then
      echo "The maximal number of devices per node is 2." 1>&2
      return 1
    else
      numDevices=$1
    fi
  fi
}
(continues on next page)
fi

fi

export OMP_NUM_THREADS=48
salloc --time=1:00:00 --ntasks-per-node=\$(\$numDevices) --mem=94000 -A $proj -p batch bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
  source $BASH_COMP_FILE
else
  echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi

Queue: gpus (4 x Nvidia V100 GPUs)

# Name and Path of this Script **************************** (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)/"$\$(basename $BASH_SOURCE)

# User Information *************************************** (edit the following lines)
# - automatically add your name and contact to output file meta data
# - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$\$(whoami) <$MY_MAIL>"

# Project Information #################################### (edit this line)
# - project account for computing time
export proj=$(groups | awk '{print $4}')

# Text Editor for Tools **************************** (edit this line)
# - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules ****************************
# module purge
module load GCC/7.3.0
module load CUDA/9.2.88
module load CMake/3.13.0
module load MVAPICH2/2.3-GDR
module load Python/3.6.6

# Other Software ****************************
# module load zlib/.1.2.11
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

# This is required for Boost to have correct dynamic library dependencies
module load ICU/61.1
export LD_LIBRARY_PATH=$EBROOTICU/lib:$LD_LIBRARY_PATH

(continues on next page)
PARTITION_LIB=$PROJECT/lib_gpus
BOOST_ROOT=$PARTITION_LIB/boost
export CMAKE_PREFIX_PATH=$BOOST_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH

HDF5_ROOT=$PARTITION_LIB/hdf5
export PATH=$HDF5_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH

LIBSPLASH_ROOT=$PARTITION_LIB/libSplash
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$LIBSPLASH_ROOT:$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

ADIOS_ROOT=$PARTITION_LIB/adios
export PATH=$ADIOS_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$ADIOS_ROOT:$CMAKE_PREFIX_PATH

export LD_LIBRARY_PATH=$EBROOTICU/lib:$LD_LIBRARY_PATH

# Environment #################################################################
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70" # Nvidia V100 architecture

export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
# - SLURM (sbatch)
# - "gpus" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/juwels-jsc/gpus.tpl"

# allocate an interactive shell for one hour
# getNode 2 # allocates 2 interactive nodes (default: 1)
function getNode() {
  if [ -z "$1" ]; then
    numNodes=1
  else
    numNodes=$1
  fi
  if [ $numNodes -gt 8 ]; then
    echo "The maximal number of interactive nodes is 8." 1>&2
    return 1
  fi
  echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching
  multiple processes in the interactive mode"
salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --gres=gpu:4 --
  mem=180000 -A $proj --gpu bash
}
# allocate an interactive shell for one hour
# getDevice 2 # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching
    multiple processes in the interactive mode"
    salloc --time=1:00:00 --ntasks-per-node=\$(\$numDevices) --gres=gpu:4 --
    mem=180000 -A $proj -p gpus bash
}

1.4.13 ARIS (GRNET)

System overview: link

User guide: link

Production directory: $WORKDIR (link)

For these profiles to work, you need to download the PICongPU source code.

Queue: gpu (2 x NVIDIA Tesla k40m GPUs)
# General modules #################################################################
#
module purge
module load gnu/6.4.0
module load cmake
module load cuda/9.2.148
module load make
module load utils
module load python/2.7.13
module load git
module load picongpu
#module load boost/1.62.0
#module load hdf5/1.8.17/gnu
# Other Software #################################################################
#
# module load zlib/1.2.8
# module load pngwriter/0.7.0
# module load hdf5-parallel/1.8.20 libsplash/1.7.0
# Work-Arounds ####################################################################
#
# fix for Nvidia NVCC bug id 2448610
# see https://github.com/ComputationalRadiationPhysics/alpaka/issues/701
#export CXXFLAGS="-Dlinux"
# Environment ####################################################################
#
export CMAKE_PREFIX_PATH=$PICONGPUROOT
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"
export PATH=$PATH:$PICSRC
export PATH=$PATH:$PICSRC/bin
export PATH=$PATH:$PICSRC/src/tools/bin
#export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH
# "tbg" default options #################################################################
#
# - SLURM (sbatch)
# - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/aris-grnet/gpu.tpl"

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
  if [ -z "$1" ]; then
    numNodes=1
  else
    numNodes=$1
  fi
  srun --time=0:30:00 --nodes=$numNodes --ntasks-per-socket=8 --ntasks-per-
  node=16 --mem=252000 --gres=gpu:4 -A $proj -p dvd_usr_prod --pty bash
}

# allocate an interactive shell for one hour
# getDevice 2 # allocates two interactive devices (default: 1)
function getDevice() {
if [ -z "$1" ]; then
    numGPUs=1
else
    if [ "$1" -gt 4 ]; then
        echo "The maximal number of devices per node is 4." 1>&2
        return 1
    else
        numGPUs=$1
    fi
fi

srun --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=$(($4 * $numGPUs)) --gres=gpu:$numGPUs --mem=$(($43000 * $numGPUs)) -A $proj -p dvd_usr__prod --pty bash

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ]; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." 1>&2
fi

1.5 Changelog

1.5.1 0.5.0

Date: 2020-06-03

Perfectly Matched Layer (PML) and Bug Fixes

This release adds a new field absorber for the Yee solver, convolutional perfectly matched layer (PML). Compared to the still supported exponential damping absorber, PML provides better absorption rate and much less spurious reflections.

We added new plugins for computing emittance and transition radiation, particle rendering with the ISAAC plugin, Python tools for reading and visualizing output of a few plugins.

The release also adds a few quality-of-life features, including a new memory calculator, better command-line experience with new options and bashcompletion, improved error handling, cleanup of the example setups, and extensions to documentation.

Thanks to Igor Andriyash, Sergei Bastrakov, Xeinia Bastrakova, Andrei Berceanu, Finn-Ole Carstens, Alexander Debus, Jian Fuh Ong, Marco Garten, Axel Huebl, Sophie Rudat (Koßagk), Anton Lebedev, Felix Meyer, Pawel Ordyna, Richard Pausch, Franz Pöschel, Adam Simpson, Sebastian Starke, Klaus Steiniger, René Widera for contributions to this release!

Changes to “0.4.0”

User Input Changes:

- Particle pusher acceleration #2731
- stop moving window after N steps #2792
- Remove unused ABSORBER_FADE_IN_STEPS from .param files in examples #2942
- add namespace “radiation” around code related to radiation plugin #3004
- Add a runtime parameter for window move point #3022
• Ionization: add silicon to pre-defines #3078
• Make dependency between boundElectrons and atomicNumbers more explicit #3076
• openPMD: use particle id naming #3165
• Docs: update species.param #2793 #2795

New Features:

• PIC:
  – Particle pusher acceleration #2731
  – Stop moving window after N steps #2792
  – Auto domain adjustment #2840
  – Add a wrapper around main() to catch and report exceptions #2962
  – Absorber perfectly matched layer PML #2950 #2967
  – Make dependency between boundElectrons and atomicNumbers more explicit #3076

• PMacc:
  – ExchangeTypeNames Verify Parameter for Access #2926
  – Name directions in species buffer warnings #2925
  – Add an implementation of exp for pmacc vectors #2956
  – SimulationFieldHelper: getter method to access cell description #2986

• plugins:
  – PhaseSpaceData: allow multiple iterations #2754
  – Python MPL Visualizer: plot for several simulations #2762
  – Emittance Plugin #2588
  – DataReader: Emittance & PlotMPL: Emittance, SliceEmittance, EnergyWaterfall #2737
  – Isaac: updated for particle rendering #2940
  – Resource Monitor Plugin: Warnings #3013
  – Transition radiation plugin #3003
  – Add output and python module doc for radiation plugin #3052
  – Add reference to thesis for emittance plugin doc #3101
  – Plugins: ADIOS & PhaseSpace Wterminate #2817
  – Calorimeter Plugin: Document File Suffix #2800
  – Fix returning a stringstream by value #3251

• tools:
  – Support alpaka accelerator threads #2701
  – Add getter for omega and n to python module #2776
  – Python Tools: Incorporate sim_time into readers and visualizers #2779
  – Add PICoGPU memory calculator #2806
  – Python visualizers as jupyter widgets #2691
  – pic-configure: add --force/-f option #2901
  – Correct target thickness in memory calculator #2873
  – CMake: Warning in 3.14+ Cache List #3008
– Add an option to account for PML in the memory calculator #3029
– Update profile hemera-hzdr: CMake version #3059
– Travis CI: OSX sed Support #3073
– CMake: mark cuda 10.2 as tested #3118
– Avoid bash completion file path repetition #3136
– Bash completion #3069
– Jupyter widgets output capture #3149
– Docs: Add ionization prediction plot #2870
– pic-edit: clean cmake file cache if new param added #2904
– CMake: Honor _ROOT Env Hints #2891
– Slurm: Link stdout live #2839

Bug Fixes:

• PIC:
  – fix EveryNthCellImpl #2768
  – Split ParserGridDistribution into hpp/cpp file #2899
  – Add missing inline qualifiers potentially causing multiple definitions #3006
  – fix wrong used method prefix #3114
  – fix wrong constructor call #3117
  – Fix calculation of omega_p for logging #3163
  – Fix laser bug in case focus position is at the init plane #2922
  – Fix binomial current interpolation #2838
  – Fix particle creation if density zero #2831
  – Avoid two slides #2774
  – Fix warning: comparison of unsigned integer #2987

• PMacc:
  – Typo fix in Send/receive buffer warning #2924
  – Explicitly specify template argument for std::forward #2902
  – Fix signed int overflow in particle migration between supercells #2989
  – Boost 1.67.0+ Template Aliases #2908
  – Fix multiple definitions of PMacc identifiers and aliases #3036
  – Fix a compilation issue with ForEach lookup #2985

• plugins:
  – Fix misspelled words in plugin documentation #2705
  – Fix particle merging #2753
  – OpenMPI: Use ROMIO for IO #2857
  – Radiation Plugin: fix bool conditions for hdf5 output #3021
  – CMake Modules: Update ADIOS FindModule #3116
  – ADIOS Particle Writer: Fix timeOffset #3120
  – openPMD: use particle id naming #3165
– Include int16 and uint16 types as traits for ADIOS #2929
– Fix observation direction of transition radiation plugin #3091
– Fix doc transition radiation plugin #3089
– Fix doc rad plugin units and factors #3113
– Fix wrong underline in TransRad plugin doc #3102
– Fix docs for radiation in 2D #2772
– Fix radiation plugin misleading filename #3019

• tools:
  – Update cuda_memtest: NVML Noise #2785
  – Dockerfile: No SSH Deamon & Keys, Fix Flex Build #2970
  – Fix hemera k80_restart.tpl #2938
  – Templates/profile for hemera k20 queue #2935
  – Splash2txt Build: Update deps #2914
  – splash2txt: fix file name trimming #2913
  – Fix compile splash2txt #2912
  – Docker CUDA Image: Hwloc Default #2906
  – Fix Python EnergyHistogramData: skip of first iteration #2799

• Spack: Fix Compiler Docs #2997
• Singularity: Workaround Chmod Issue, No UCX #3017
• Fix examples particle filters #3065
• Fix CUDA device selection #3084
• Fix 8.cfg for Bremsstrahlung example #3097
• Fix taurus profile #3152
• Fix a typo in density ratio value of the KHI example #3162
• Fix GCC constexpr lambda bug #3188
• CFL Static Assert: new grid.param #2804
• Fix missing exponent in fieldIonization.rst #2790
• Spack: Improve Bootstrap #2773
• Fix python requirements: remove sys and getopt #3172

Misc:

• refactoring:
  – PIC:
    * Eliminate M_PI (again) #2833
    * Fix MappingDesc name hiding #2835
    * More fixes for MSVC capturing constexpr in lambdas #2834
    * Core Particles: C++11 Using for Typedef #2859
    * Remove unused getCommTag() in FieldE, FieldB, FieldJ #2947
    * Add a using declaration for Difference type to yee::Curl #2955
    * Separate the code processing currents from MySimulation #2964
* Add DataConnector::consume(), which shares and consumes the input #2951
* Move picongpu/simulationControl to picongpu/simulation/control #2971
* Separate the code processing particles from MySimulation #2974
* Refactor cell types #2972
* Rename compileTime into meta #2983
* Move fields/FieldManipulator to fields/absorber/ExponentialDamping #2995
* Add picongpu::particles::manipulate() as a high-level interface to particle manipulation #2993
* particles::forEach #2991
* Refactor and modernize implementation of fields #3005
* Modernize ArgsParser::ArgsErrorCode #3023
* Allow constructor for density free formular functor #3024
* Reduce PML memory consumption #3122
* Bremsstrahlung: use more constexpr #3176
* Pass mapping description by value instead of pointer from simulation stages #3014
* Add missing inline specifiers for functions defined in header files #3051
* Remove ZigZag current deposition #2837
* Fix style issues with particlePusherAcceleration #2781

- **PMacc:**
  * Supercell particle counter #2637
  * ForEachIdx::operator(): Use Universal Reference #2881
  * Remove duplicated definition of BOOST_MPL_LIMIT_VECTOR_SIZE #2883
  * Cleanup pmacc/types.hpp #2927
  * Add pmacc::memory::makeUnique similar to std::make_unique #2949
  * PMacc Vector: C++11 Using #2957
  * Remove pmacc::forward and pmacc::RefWrapper #2963
  * Add const getters to ParticleBox #2941
  * Remove unused pmacc::traits::GetEmptyDefaultConstructibleType #2976
  * Remove pmacc::traits::IsSameType which is no longer used #2979
  * Remove template parameter for initialization method of Pointer and FramePointer #2977
  * Remove pmacc::expressions which is no longer used #2978
  * Remove unused pmacc::IDataSorter #3030
  * Change PMACC_C_STRING to produce a static constexpr member #3050
  * Refactor internals of pmacc::traits::GetUniqueTypeId #3049
  * rename “counterParticles” to “numParticles” #3062
  * Make pmacc::DataSpace conversions explicit #3124

- **Plugins:**
  * Small update for python visualizers #2882
  * Add namespace “radiation” around code related to radiation plugin #3004
  * Remove unused includes of pthread #3040

1.5. Changelog
* SpeciesEligibleForSolver for radiation plugin #3061
* ADIOS: Avoid unsafe temporary strings #2946

- tools:
  * Update cuda_memtest: CMake CUDA_ROOT Env #2892
  * Update hemera tpl after SLURM update #3123
- Add pillow as dependency #3180
- Params: remove boost::vector<> usage #2769
- Use _X syntax in OnceIonized manipulator #2745
- Add missing const to some GridController getters #3154

- documentation:
  * Containers: Update 0.4.0 #2750
  * Merge 0.4.0 Changelog #2748
  * Update Readme & License: People #2749
  * Add .zenodo.json #2747
  * Fix species.param docu (in all examples too) #2795
  * Fix species.param example doc and grammar #2793
  * Further improve wording in docs #2710
  * MemoryCalculator: fix example output for documentation #2822
  * Manual: Plugin & Particle Sections, Map #2820
  * System: D.A.V.I.D.E #2821
  * License Header: Update 2019 #2845
  * Docs: Memory per Device Spelling #2868
  * CMake 3.11.0+ #2959
  * CUDA 9.0+, GCC 5.1+, Boost 1.65.1+ #2961
  * CMake: CUDA 9.0+ #2965
  * Docs: Update Sphinx #2969
  * CMake: CUDA 9.2-10.1, Boost <= 1.70.0 #2975
  * Badge: Commits Since Release & Good First #2980
  * Update info on maintainers in README.md #2984
  * Fix grammar in all .profile.example #2930
  * Docs: Dr.s #3009
  * Fix old file name in radiation doc #3018
  * System: ARIS #3039
  * fix typo in getNode and getDevice #3046
  * Window move point clean up #3045
  * Docs: Cori’s KNL Nodes (NERSC) #3043
  * Fix various sphinx issues not related to doxygen #3056
  * Extend the particle merger plugin documentation #3057
  * Fix docs using outdated ManipulateDeriveSpecies #3068
- Adjust cores per gpu on taurus after multicore update #3071
- Docs: create conda env for building docs #3074
- Docs: add missing checkpoint options #3080
- Remove titan ornl setup and doc #3086
- Summit: Profile & Templates #3007
- Update URL to ADIOS #3099
- License Header: Update 2020 #3138
- Add PhD thesis reference in radiation plugin #3151
- Spack: w/o Modules by Default #3182
- Add a brief description of simulation output to basics #3183
- Fix a typo in exchange communication tag status output #3141
- Add a link to PoGit to the docs #3115
- fix optional install instructions in the Summit profile #3094
- Update the form factor documentation #3083
- Docs: Add New References #3072
- Add information about submit.cfg and submit.tpl files to docs. #3070
- Fix style (underline length) in profile.rst #2936
- Profiles: Section Title Length #2934
- Contributor name typo in LICENSE.md #2880
- Update modules and memory in gpu_picongpu.profile #2923
- Add k80_picongpu.profile and k80.tpl #2919
- Update taurus-tud profiles for the ml partition #2903
- Hypnos: CMake 3.13.4 #2887
- Docs: Install Blosc #2829
- Docs: Source Intro Details #2828
- Taurus Profile: Project #2819
- Doc: Add System Links #2818
- remove grep file redirect #2788
- Correct jupyter widget example #3191
- fix typo: UNIT_LENGHT to UNIT_LENGTH #3194
- Change link to CRP group @ HZDR #2814

• Examples: Unify .cfg #2826
• Remove unused ABSORBER_FADE_IN_STEPS from .param files in examples #2942
• Field absorber test example #2948
• Singularity: Avoid Dotfiles in Home #2981
• Boost: No std::auto_ptr #3012
• Add YeePML to comments for field solver selection #3042
• Add a runtime parameter for window move point #3022
• Ionization: add silicon to pre-defines #3078
• Add 1.cfg to Bremsstrahlung example #3098
• Fix cmake flags for MSVS #3126
• Fix missing override flags #3156
• Fix warning #222-D: floating-point operation result is out of range #3170
• Update alpaka to 0.4.0 and cupla to 0.2.0 #3175
• Slurm update taurus: workdir to chdir #3181
• Adjust profiles for taurus-tud #2990
• Update mallocMC to 2.3.1crp #2893
• Change imread import from scipy.misc to imageio #3192

1.5.2 0.4.3

Date: 2019-02-14

System Updates and Bug Fixes

This release adds updates and new HPC system templates. Important bug fixes include I/O work-arounds for issues in OpenMPI 2.0-4.0 (mainly with HDF5), guards for particle creation with user-defined profiles, a fixed binomial current smoothing, checks for the number of devices in grid distributions and container (Docker & Singularity) modernizations.

Thanks to Axel Huebl, Alexander Debus, Igor Andriyash, Marco Garten, Sergei Bastrakov, Adam Simpson, Richard Pausch, Juncheng E, Klaus Steiniger, and René Widera for contributions to this release!

Changes to “0.4.2”

Bug Fixes:

• fix particle creation if density ≤ zero #2831
• fix binomial current interpolation #2838
• Docker & Singularity updates #2847
• OpenMPI: use ROMIO for IO #2841 #2857
• --gridDist: verify devices and blocks #2876
• Phase space plugin: unit of colorbar in 2D3V #2878

Misc:

• ionizer.param: fix typo in “Aluminium” #2865
• System Template Updates:
  – Add system links #2818
  – Taurus:
    * add project #2819
    * add Power9 V100 nodes #2856
  – add D.A.V.I.D.E (CINECA) #2821
  – add JURECA (JSC) #2869
  – add JUWELS (JSC) #2874
  – Hypnos (HZDR): CMake update #2887
  – Slurm systems: link stdout to simOutput/output #2839
CPU Plugin Performance

This release fixes a performance regression for energy histograms and phase space plugins on CPU with our OpenMP backend on CPU. At least OpenMP 3.1 is needed to benefit from this. Additionally, several small documentation issues have been fixed and the energy histogram python tool forgot to return the first iteration.

Thanks to Axel Huebl, René Widera, Sebastian Starke, and Marco Garten for contributions to this release!

Changes to “0.4.1”

Bug Fixes:

• Plugin performance regression:
  – Speed of plugins EnergyHistogram and PhaseSpace on CPU (omp2b) #2802
  • Tools:
    – Python EnergyHistogramData: skip of first iteration #2799

Misc:

• update Alpaka to 0.3.5 to fix #2802
• Docs:
  – CFL Static Assert: new grid.param #2804
  – missing exponent in fieldIonization.rst #2790
  – remove grep file redirect #2788
  – Calorimeter Plugin: Document File Suffix #2800
Minor Bugs and Example Updates

This release fixes minor bugs found after the 0.4.0 release. Some examples were slightly outdated in syntax, the new “probe particle” EveryNthCell initialization functor was broken when not used with equal spacing per dimension. In some rare cases, sliding could occur twice in moving window simulations.

Thanks to Axel Huebl, René Widera, Richard Pausch and Andrei Berceanu for contributions to this release!

Changes to “0.4.0”

Bug Fixes:

- PIconGPU:
  - avoid sliding twice in some corner-cases #2774
  - EveryNthCell: broken if not used with same spacing #2768
  - broken compile with particle merging #2753
- Examples:
  - fix outdated derive species #2756
  - remove current deposition in bunch example #2758
  - fix 2D case of single electron init (via density) #2766
- Tools:
  - Python Regex: r Literals #2767
  - cuda_memtest: avoid noisy output if NVML is not found #2785

Misc:

- .param files: refactor boost::vector<> usage #2769
- Docs:
  - Spack: Improve Bootstrap #2773
  - Fix docs for radiation in 2D #2772
  - Containers: Update 0.4.0 #2750
  - Update Readme & License: People #2749
  - Add .zenodo.json #2747

1.5.5 0.4.0

Date: 2018-10-19

CPU Support, Particle Filter, Probes & Merging

This release adds CPU support, making PIconGPU a many-core, single-source, performance portable PIC code for all kinds of supercomputers. We added particle filters to initialization routines and plugins, allowing fine-grained in situ control of physical observables. All particle plugins now support those filters and can be called multiple times with different settings.

Particle probes and more particle initialization manipulators have been added. A particle merging plugin has been added. The Thomas-Fermi model has been improved, allowing to set empirical cut-offs. PIconGPU input and output (plugins) received initial Python bindings for efficient control and analysis.
User input files have been dramatically simplified. For example, creating the PIConGPU binary from input files for GPU or CPU is now as easy as `pic-build -b cuda` or `pic-build -b omp2b` respectively.

Thanks to Axel Huebl, René Widera, Benjamin Worpitz, Sebastian Starke, Marco Garten, Richard Pausch, Alexander Matthes, Sergei Bastrakov, Heiko Burau, Alexander Debus, Ilja Göthel, Sophie Rudat, Jeffrey Kelling, Klaus Steiniger, and Sebastian Hahn for contributing to this release!

### Changes to “0.3.0”

#### User Input Changes:

- (re)move directory `simulation_defines/` #2331
- add new param file `particleFilters.param` #2385
- components.param: remove define `ENABLE_CURRENT` #2678
- laser.param: refactor Laser Profiles to Functors #2587 #2652
- visualization.param: renamed to `png.param` #2530
- speciesAttributes.param: format #2087
- fieldSolver.param: doxygen, refactored #2534 #2632
- mallocMC.param: file doxygen #2594
- precision.param: file doxygen #2593
- memory.param:
  - `GUARD_SIZE` docs #2591
  - exchange buffer size per species #2290
  - guard size per dimension #2621
- density.param:
  - Gaussian density #2214
  - Free density: fix `float_X` #2555
- ionizer.param: fixed excess 5p shell entry in gold effective Z #2558
- seed.param:
  - renamed to `random.param` #2605
  - expose random number method #2605
- isaac.param: doxygen documentation #2260
- unit.param:
  - doxygen documentation #2467
  - move conversion units #2457
  - earlier normalized speed of light in `physicalConstants.param` #2663
- `float_X` constants to literals #2625
- refactor particle manipulators #2125
- new tools:
  - `pic-edit`: adjust .param files #2219
  - `pic-build`: combine pic-configure and make install #2204
- `pic-configure`:
  - select CPU/GPU backend and architecture with -b #2243
• default backend: CUDA #2248

• tbg:
  – .tpl no _profile suffix #2244
  – refactor .cfg files: devices #2543
  – adjust LWFA setup for 8GPUs #2480

• SliceField plugin: Option .frequency to .period #2034

• particle filters:
  – add filter support to phase space plugin #2425
  – multi plugin energy histogram with filter #2424
  – add particle filter to EnergyParticles #2386

• Default Inputs: C++11 using for typedef #2315

• Examples: C++11 using for typedef #2314

• Python: Parameter Ranges for Param Files (LWFA) #2289

• FieldTmp: SpeciesEligibleForSolver Traits #2377

• Particle Init Methods: Unify API & Docs #2442
  – get species by name #2464
  – remove template dimension from current interpolator’s #2491
  – compile time string #2532

New Features:

• PIC:
  – particle merging #1959
  – check cells needed for stencils #2257
  – exchange buffer size per species #2290
  – push with currentStep #2318
  – InitController: unphysical particles #2365
  – New Trait: SpeciesEligibleForSolver #2364
  – Add upper energy cut-off to ThomasFermi model #2330
  – Particle Pusher: Probe #2371
  – Add lower ion density cut-off to ThomasFermi model #2361
  – CT Factory: GenerateSolversIfSpeciesEligible #2380
  – add new param file particleFilters.param #2385
  – Probe Particle Usage #2384
  – Add lower electron temperature cut-off to ThomasFermi model #2376
  – new particle filters #2418 #2659 #2660 #2682
  – Derived Attribute: Bound Electron Density #2453
  – get species by name #2464
  – New Laser Profile: Exp. Ramps with Prepulse #2352
  – Manipulator: UnboundElectronsTimesWeighting #2398
  – Manipulator: unary::FreeTotalCellOffset #2498
- expose random number method to the user #2605
- seed generator for RNG #2607
- FLYlite: initial interface & helper fields #2075

**PMacc:**
- cupla compatible RNG #2226
- generic min() and max() implementation #2173
- Array: store elements without a default constructor #1973
- add array to hold context variables #1978
- add ForEachIdx #1977
- add trait GetNumWorker #1985
- add index pool #1958
- Vector float1_X to float_X cast #2020
- extend particle handle #2114
- add worker config class #2116
- add interfaces for functor and filter #2117
- Add complex logarithm to math #2157
- remove unused file BitData.hpp #2174
- Add Bessel functions to math library #2156
- Travis: Test PMacc Unit Tests #2207
- rename CUDA index names in ConcatListOfFrames #2235
- cuSTL Foreach with lockstep support #2233
- Add complex sin() and cos() functions. #2298
- Complex BesselJ0 and BesselJ1 functions #2161
- CUDA9 default constructor warnings #2347
- New Trait: HasIdentifiers #2363
- RNG with reduced state #2410
- PMacc RNG 64bit support #2451
- PhaseSpace: add lockstep support #2454
- signed and unsigned comparison #2509
- add a workaround for MSVC bug with capturing constexpr #2522
- compile time string #2532
- Vector: add method remove...() #2602
- add support for more cpu alpaka accelerators #2603 #2701
- Vector::sumOfComponents #2609
- math::CT::max improvement #2612

**plugins:**
- ADIOS: allow usage with accelerator omp2b #2236
- ISAAC:
  - alpaka support #2268 #2349

---

**1.5. Changelog**
* require version 1.4.0+ #2630
  – InSituVolumeRenderer: removed (use ISAAC instead) #2238
  – HDF5: Allow Unphysical Particle Dump #2366
  – SpeciesEligibleForSolver Traits #2367

* PNG:
  * lockstep kernel refactoring Visualisation.hpp #2225
  * require PNGwriter version 0.7.0+ #2468

* ParticleCalorimeter:
  * add particle filter #2569
  * fix usage of uninitialized variable #2320

* Python:
  * Energy Histogram Reader #2209 #2658
  * Phase Space Reader #2334 #2634 #2679
  * Move SliceField Module & add Python3 support #2354 #2718
  * Multi-Iteration Energy Histogram #2508
  * MPL Visualization modules #2484 #2728
  * migrated documentation to Sphinx manual #2172 #2726 #2738
  * shorter python imports for postprocessing tools #2727
  * fix energy histogram deprecation warning #2729
  * data: base class for readers #2730
  * param_parser for JSON parameter files #2719

• tools:
  – Tool: New Version #2080
  – Changelog & Left-Overs from 0.3.0 #2120
  – TBG: Check Modified Input #2123
  – Hypnos (HZDR) templates:
    * mpiexec and LD_LIBRARY_PATH #2149
    * K20 restart #2627
    * restart .tpl files: new checkpoints.period syntax #2650
  – Travis: Enforce PEP8 #2145
  – New Tool: pic-build #2204
  – Docker:
    * Dockerfile introduced #2115 #2286
    * spack clean & load #2208
    * update ISAAC client URL #2565
  – add HZDR cluster hydra #2242
  – pic-configure: default backend CUDA #2248
  – New Tool: pic-edit #2219
  – FoilLCT: Plot Densities #2259
- tbg: Add -f|--force #2266
- Improved the cpuNumaStarter.sh script to support not using all hw threads #2269
- Removed libm dependency for Intel compiler... #2278
- CMake: Same Boost Min for Tools #2293
- HZDR tpl: killall return #2295
- PMacc: Set CPU Architecture #2296
- ThermalTest: Flake Dispersion #2297
- Python: Parameter Ranges for Param Files (LWFA) #2289
- LWFA: GUI .cfg & Additional Parameters #2336
- Move mpiInfo to new location #2355
- bracket test for external libraries includes #2399
- Clang-Tidy #2303
- tbg -f: mkdir -p submitAction #2413
- FIx initial setting of Parameter values #2422
- Move TBG to bin/ #2537
- Tools: Move pic-* to bin/ #2539
- Simpler Python Parameter class #2550

**Bug Fixes:**

- **PIC:**
  - fix restart with background fields enabled #2113
  - wrong border with current background field #2326
  - remove usage of pure float with float_X #2606
  - fix stencil conditions #2613
  - fix that guard size must be one #2614
  - fix dead code #2301
  - fix memory leaks #2669

- **PMacc:**
  - event system:
    * fix illegal memory access #2151
    * fix possible deadlock in blocking MPI ops #2683
  - cuSTL:
    * missing #include in ForEach #2406
    * HostBuffer 1D Support #2657
  - fix warning concerning forward declarations of pmacc::detail::Environment #2489
    pmacc::math::Size_t<0>::create() in Visual Studio #2513
  - fix V100 deadlock #2600
  - fix missing include #2608
  - fix gameOfLife #2700
  - Boost template aliases: fix older CUDA workaround #2706
• plugins:
  – energy fields: fix reduce #2112
  – background fields: fix restart GUARD #2139
  – Phase Space:
    * fix weighted particles #2428
    * fix momentum meta information #2651
  – ADIOS:
    * fix 1 particle dumps #2437
    * fix zero size transform writes #2561
    * remove adios_set_max_buffer_size #2670
    * require 1.13.1+ #2583
  – IO fields as source #2461
  – ISAAC: fix gcc compile #2680
  – Calorimeter: Validate minEnergy #2512
• tools:
  – fix possible linker error #2107
  – cmakeFlags: Escape Lists #2183
  – splash2txt: C++98 #2136
  – png2gas: C++98 #2162
  – tbg env variables escape \ and & #2262
  – XDMF Scripts: Fix Replacements & Offset #2309
  – pic-configure: cmakeFlags return code #2323
  – tbg: fix wrong quoting of ' #2419
  – CMake in-source builds: too strict #2407
• --help to stdout #2148
• Density: Param Gaussian Density #2214
• Fixed excess 5p shell entry in gold effective Z #2558
• Hypnos: Zlib #2570
• Limit Supported GCC with nvcc 8.0-9.1 #2628
• Syntax Highlighting: Fix RTD Theme #2596
• remove extra typename in documentation of manipulators #2044

Misc:
• new example: Foil (LCT) TNSA #2008
• adjust LWFA setup for 8 GPUs #2480
• picongpu --version #2147
• add internal Alpaka & cupla #2179 #2345
• add alpaka dependency #2205 #2328 #2346 #2590 #2501 #2626 #2648 #2684 #2717
• Update mallocMC to 2.3.0crp #2350 #2629
• cuda_memtest:
1.5. Changelog

- update #2356 #2724
- usage on hypnos #2722

• Examples:
  - remove unused loaders #2247
  - update species.param #2474

• Bunch: no precision.param #2329

• Travis:
  - stages #2341
  - static code analysis #2404

• Visual Studio: ERROR macro defined in wingdi.h #2503

• Compile Suite: update plugins #2595

• refactoring:
  - PIC:
    * const POD Default Constructor #2300
    * FieldE: Fix Unreachable Code Warning #2332
    * Yee solver lockstep refactoring #2027
    * lockstep refactoring of KernelComputeCurrent #2025
    * FieldJ bash/insert lockstep refactoring #2054
    * lockstep refactoring of KernelFillGridWithParticles #2059
    * lockstep refactoring KernelLaserE #2056
    * lockstep refactoring of KernelBinEnergyParticles #2067
    * remove empty init() methods #2082
    * remove ParticlesBuffer::createParticleBuffer() #2081
    * remove init method in FieldE and FieldB #2088
    * move folder fields/tasks to libPMacc #2090
    * add AddExchangeToBorder, CopyGuardToExchange #2091
    * lockstep refactoring of KernelDeriveParticles #2097
    * lockstep refactoring of ThreadCollective #2101
    * lockstep refactoring of KernelMoveAndMarkParticles #2104
    * Esirkepov: reorder code order #2121
    * refactor particle manipulators #2125
    * Restructure Repository Structure #2135
    * lockstep refactoring KernelManipulateAllParticles #2140
    * remove all lambda expressions. #2150
    * remove usage of native CUDA function prefix #2153
    * use nvidia::atomicAdd instead of our old wrapper #2152
    * lockstep refactoring KernelAbsorbBorder #2160
    * functor interface refactoring #2167
    * lockstep kernel refactoring KernelAddCurrentToEMF #2170
* lockstep kernel refactoring KernelComputeSupercells #2171
* lockstep kernel refactoring CopySpecies #2177
* Marriage of PIConGPU and cupla/alpaka #2178
* Ionization: make use of generalized particle creation #2189
* use fast atomicAllExch in KernelFillGridWithParticles #2230
* enable ionization for CPU backend #2234
* ionization: speedup particle creation #2258
* lockstep kernel refactoring KernelCellwiseOperation #2246
* optimize particle shape implementation #2275
* improve speed to calculate number of ppc #2274
* refactor picongpu::particles::startPosition #2168
* Particle Pusher: Clean-Up Interface #2359
* create separate plugin for checkpointing #2362
* Start Pos: OnePosition w/o Weighting #2378
* rename filter: IsHandleValid -> All #2381
* FieldTmp: SpeciesEligibleForSolver Traits #2377
* use lower case begin for filter names #2389
* refactor PMacc functor interface #2395
* PIConGPU: C++11 using #2402
* refactor particle manipulators/filter/startPosition #2408
* rename GuardHandlerCallPlugins #2441
* activate synchrotron for CPU back-end #2284
* DifferenceToLower/Upper forward declaration #2478
* Replace usage of M_PI in picongpu with Pi #2492
* remove template dimension from current interpolator’s #2491
* Fix issues with name hiding in Particles #2506
* refactor: field solvers #2534
* optimize stride size for update FieldJ #2615
* guard size per dimension #2621
* Lasers: float_X Constants to Literals #2624
  * float_X: C++11 Literal #2622
* log: per “device” instead of “GPU” #2662 #2677
* earlier normalized speed of light #2663
* fix GCC 7 fallthrough warning #2665 #2671
* png.unitless: static asserts clang compatible #2676
  * remove define ENABLE_CURRENT #2678

  **PMacc:**
  * refactor ThreadCollective #2021
  * refactor reduce #2015
- lock step kernel KernelShiftParticles #2014
- lockstep refactoring of KernelCountParticles #2061
- lockstep refactoring KernelFillGapsLastFrame #2055
- lockstep refactoring of KernelFillGaps #2083
- lockstep refactoring of KernelDeleteParticles #2084
- lockstep refactoring of KernelInsertParticles #2089
- lockstep refactoring of KernelBashParticles #2086
- call KernelFillGaps* from device #2098
- lockstep refactoring of KernelSetSetValue #2099
- Game of Life lockstep refactoring #2142
- HostDeviceBuffer rename conflicting type defines #2154
- use c++11 move semantic in cuSTL #2155
- lockstep kernel refactoring SplitIntoListOfFrames #2163
- lockstep kernel refactoring Reduce #2169
- enable cuSTL CartBuffer on CPU #2271
- allow update of a particle handle #2382
- add support for particle filters #2397
- RNG: Normal distribution #2415
- RNG: use non generic place holder #2440
- extended period syntax #2452
- Fix buffer cursor dim #2488
- Get rid of <sys/time.h> #2495
- Add a workaround for PMACC_STRUCT to work in Visual Studio #2502
- Fix type of index in OpenMP-parallelized loop #2505
- add support for CUDA9 __shfl_sync, __ballot_sync #2348
- Partially replace compound literals in PMacc #2494
- fix type cast in pmacc::exec::KernelStarter::operator() #2518
- remove modulo in 1D to ND index transformation #2542
- Add Missing Namespaces #2579
- Tests: Add Missing Namespaces #2580
- refactor RNG method interface #2604
- eliminate M_PI from PMacc #2486
- remove empty last frame #2649
- no throw in destructors #2666
- check minimum GCC & Clang versions #2675

- plugins:
  - SliceField Plugin: Option .frequency to .period #2034
  - change notifyFrequency(s) to notifyPeriod #2039
  - lockstep refactoring KernelEnergyParticles #2164
remove LiveViewPlugin #2237
* Png Plugin: Boost to std Thread #2197
* lockstep kernel refactoring KernelRadiationParticles #2240
* generic multi plugin #2375
* add particle filter to EnergyParticles #2386
* PluginController: Eligible Species #2368
* IO with filtered particles #2403
* multi plugin energy histogram with filter #2424
* lockstep kernel refactoring ParticleCalorimeter #2291
* Splash: 1.7.0 #2520
* multi plugin ParticleCalorimeter #2563
* Radiation Plugin: Namespace #2576
* Misc Plugins: Namespace #2578
* EnergyHistogram: Remove Detector Filter #2465
* ISAAC: unify the usage of period #2455
* add filter support to phase space plugin #2425
* Resource Plugin: fix boost::core::swap #2721

– tools:
  * Python: Fix Scripts PEP8 #2028
  * Prepare for Python Modules #2058
  * pic-compile: fix internal typo #2186
  * Tools: All C++11 #2194
  * CMake: Use Imported Targets Zlib, Boost #2193
  * Python Tools: Move lib to / #2217
  * pic-configure: backend #2243
  * tbg: Fix existing-folder error message to stderr #2288
  * Docs: Fix Flake8 Errors #2340
  * Group parameters in LWFA example #2417
  * Python Tools (PS, Histo): Filter Aware #2431
  * Clearer conversion functions for Parameter values between UI scale and internal scale #2432
  * tbg:
    - add content of -o arg to env #2499
    - better handling of missing egetopt error message #2712
  – Format speciesAttributes.param #2087
  – Reduce # photons in Bremsstrahlung example #1979
  – TBG: .tpl no _profile suffix #2244
  – Default Inputs: C++11 Using for Typedef #2315
  – Examples: C++11 Using for Typedef #2314
  – LWFA Example: Restore a0=8.0 #2324
- add support for CUDA9 __shfl_sync #2333
- add support for CUDA10 #2732
- Update cuda_memtest: no cuBLAS #2401
- Examples: Init of Particles per Cell #2412
- Travis: Image Updates #2435
- Particle Init Methods: Unify API & Docs #2442
- PIConGPU use tiny RNG #2447
- move conversion units to unit.param #2457
- (Re)Move simulation_defines/ #2331
- CMake: Project Vars & Fix Memtest #2538
- Refactor .cfg files: devices #2543
- Free Density: Fix float_X #2555
- Boost: Format String Version #2566
- Refactor Laser Profiles to Functors #2587
- Params: float_X Constants to Literals #2625

• documentation:
  - new subtitle #2734
  - Lockstep Programming Model #2026 #2064
  - IdxConfig append documentation #2022
  - multiMask: Refactor Documentation #2119
  - CtxArray #2390
  - Update openPMD Post-Processing #2322 #2733
  - Checkpoints Backends #2387
  - Plugins:
    * HDF5: fix links, lists & MPI hints #2313 #2711
    * typo in libSplash install #2735
    * External dependencies #2175
    * Multi & CPU #2423
    * Update PS & Energy Histo #2427
    * Memory Complexity #2434
  - Image Particle Calorimeter #2470
  - Update EnergyFields #2559
  - Note on Energy Reduce #2584
  - ADIOS: More Transport & Compression Doc #2640
  - ADIOS Metafile #2633
  - radiation parameters #1986
  - CPU Compile #2185
  - pic-configure help #2191
  - Python yt 3.4 #2273
- Namespace ComputeGridValuePerFrame #2567
- Document ionization param files for issue #1982 #1983
- Remove ToDo from ionizationEnergies.param #1989
- Parameter Order in Manual #1991
- Sphinx:
  * Document Laser Cutoff #2000
  * Move Author Macros #2005
  * PDF Radiation #2184
  * Changelog in Manual #2527
- PBS usage example #2006
- add missing linestyle to ionization plot for documentation #2032
- fix unit ionization rate plot #2033
- fix mathmode issue in ionization plot #2036
- fix spelling of guard #2644
- param: extended description #2041
- fix typos found in param files and associated files #2047
- Link New Coding Style #2074
- Install: Rsync Missing #2079
- Dev Version: 0.4.0-dev #2085
- Fix typo in ADK documentation #2096
- Profile Preparations #2095
- SuperConfig: Header Fix #2108
- Extended $SCRATCH Info #2093
- Doxygen: Fix Headers #2118
- Doxygen: How to Build HTML #2134
- Badge: Docs #2144
- CMake 3.7.0 #2181
- Boost (1.62.0-) 1.65.1 - 1.68.0 #2182 #2707 #2713
- Bash Subshells: cmd to $(cmd) #2187
- Boost Transient Deps: date_time, chrono, atomic #2195
- Install Docs: CUDA is optional #2199
- Fix broken links #2200
- PIConGPU Logo: More Platforms #2190
- Repo Structure #2218
- Document KNL GCC -march #2252
- Streamline Install #2256
- Added doxygen documentation for isaac.param file #2260
- License Docs: Update #2282
- Heiko to Former Members #2294
– Added an example profile and tpl file for taurus’ KNL #2270
– Profile: Draco (MPCDF) #2308
– $PIC_EXAMPLES #2327
– Profiles for Titan & Taurus #2201
– Taurus:
  * CUDA 8.0.61 #2337
  * Link KNL Profile #2339
  * SCS5 Update #2667
– Move ParaView Profile #2353
– Spack: Own GitHub Org #2358
– LWFA Example: Improve Ranges #2360
– fix spelling mistake in checkpoint #2372
– Spack Install: Clarify #2373 #2720
– Probe Pusher #2379
– CI/Deps: CUDA 8.0 #2420
– Piz Daint (CSCS):
  * Update Profiles #2306 #2655
  * ADIOS Build #2343
  * ADIOS 1.13.0 #2416
  * Update CMake #2436
  * Module Update #2536
  * avoid pmi_alps warnings #2581
– Hypnos (HZDR): New Modules #2521 #2661
– Hypnos: PNGwriter 0.6.0 #2166
– Hypnos & Taurus: Profile Examples Per Queue #2249
– Hemera: tbg templates #2723
– Community Map #2445
– License Header: Update 2018 #2448
– Docker: Nvidia-Docker 2.0 #2462 #2557
– Hide Double ToC #2463
– Param Docs: Title Only #2466
– New Developers #2487
– Fix Docs: FreeTotalCellOffset Filter #2493
– Stream-line Intro #2519
– Fix HDF5 Release Link #2544
– Minor Formatting #2553
– PIC Model #2560
– Doxygen: Publish As Well #2575
– Limit Filters to Eligible Species #2574
- Doxygen: Less XML #2641
- NVCC 8.0 GCC <= 5.3 && 9.0/9.1: GCC <= 5.5 #2639
- typo: element-wise #2638
- fieldSolver.param doxygen #2632
- memory.param: GUARD_SIZE docs #2591
- changelog script updated to python3 #2646

  • not yet supported on CPU (Alpaka): #2180
    - core:
      * Bremsstrahlung
    - plugins:
      * PositionsParticles
      * ChargeConservation
      * ParticleMerging
      * count per supercell (macro particles)
      * field intensity

1.5.6 0.3.2

Date: 2018-02-16

Phase Space Momentum, ADIOS One-Particle Dumps & Field Names

This release fixes a bug in the phase space plugin which derived a too-low momentum bin for particles below
the typical weighting (and too-high for above it). ADIOS dumps crashed on one-particle dumps and in the name
of on-the-fly particle-derived fields species name and field name were in the wrong order. The plugins libSplash
(1.6.0) and PNGwriter (0.6.0) need exact versions, later releases will require a newer version of PIConGPU.

Changes to “0.3.1”

Bug Fixes:

  • PIConGPU:
    - wrong border with current background field #2326

  • libPMacc:
    - cuSTL: missing include in ForEach #2406
    - warning concerning forward declarations of pmacc::detail::Environment #2489
    - pmacc::math::Size_t<0>::create() in Visual Studio #2513

  • plugins:
    - phase space plugin: weighted particles’ momentum #2428
    - calorimeter: validate minEnergy #2512

  • ADIOS:
    - one-particle dumps #2437
      - FieldTmp: derived field name #2461
    - exact versions of libSplash 1.6.0 & PNGwriter 0.6.0

  • tools:
PIConGPU Documentation, Release 0.5.0

– tbg: wrong quoting of ' #2419
– CMake: false-positive on in-source build check #2407
– pic-configure: cmakeFlags return code #2323

Misc:
• Hypnos (HZDR): new modules #2521 #2524

Thanks to Axel Huebl, René Widera, Sergei Bastrakov and Sebastian Hahn for contributing to this release!

1.5.7 0.3.1

Date: 2017-10-20
Field Energy Plugin, Gaussian Density Profile and Restarts
This release fixes the energy field plugin diagnostics and the “downramp” parameter of the pre-defined Gaussian density profile. Restarts with enabled background fields were fixed. Numerous improvements to our build system were added to deal more gracefully with co-existing system-wide default libraries. A stability issue due to an illegal memory access in the PMacc event system was fixed.

Changes to “0.3.0”

.param file changes:
• density.param in Gaussian profile, the parameter gasSigmaRight was not properly honored but gasCenterRight was taken instead #2214
• fieldBackground.param: remove micro meters usage in default file #2138

Bug Fixes:
• PIConGPU:
  – gasSigmaRight of Gaussian density profile was broken since 0.2.0 release #2214
  – restart with enabled background fields #2113 #2139
  – KHI example: missing constexpr in input #2309
• libPMacc:
  – event system: illegal memory access #2151
• plugins:
  – energy field reduce #2112
• tools:
  – CMake:
    * Boost dependency:
      · same minimal version for tools #2293
    · transient dependencies: date_time, chrono, atomic #2195
    * use targets of boost & zlib #2193 #2292
    * possible linker error #2107
  – XDMF script: positionOffset for openPMD #2309
  – cmakeFlags: escape lists #2183
  – tbg:
    * --help exit with 0 return code #2213
env variables: proper handling of \ and & #2262

Misc:

• PIConGPU: --help to stdout #2148
• tools: all to C++11 #2194
• documentation:
  – Hypnos .tpl files: remove passing LD_LIBRARY_PATH to avoid warning #2149
  – fix plasma frequency and remove German comment #2110
  – remove micro meters usage in default background field #2138
  – README: update links of docs badge #2144

Thanks to Axel Huebl, Richard Pausch and René Widera for contributions to this release!

1.5.8 0.3.0

Date: 2017-06-16

C++11: Bremsstrahlung, EmZ, Thomas-Fermi, Improved Lasers

This is the first release of PIConGPU requiring C++11. We added a newly developed current solver (EmZ), support for the generation of Bremsstrahlung, Thomas-Fermi Ionization, Laguerre-modes in the Gaussian-Beam laser, in-simulation plane for laser initialization, new plugins for in situ visualization (ISAAC), a generalized particle calorimeter and a GPU resource monitor. Initial support for clang (host and device) has been added and our documentation has been streamlined to use Sphinx from now on.

Changes to “0.2.0”

.param & .unitless file changes:

• use C++11 constexpr where possible and update arrays #1799 #1909
• use C++11 using instead of typedef
• removed Config suffix in file names #1965
• gasConfig is now density
• speciesDefinition:
  – simplified Particles<> interface #1711 #1942
  – ionizer< ... > became a sequence of ionizers< ... > #1999
• radiation: replace #defines with clean C++ #1877 #1930 #1931 #1937

Basic Usage:

We renamed the default tools to create, setup and build a simulation. Please make sure to update your picongpu.profile with the latest syntax (e.g. new entries in PATH) and use from now on:

• $PICSRC/createParameterSet -> pic-create
• $PICSRC/configure -> pic-configure
• $PICSRC/compile -> pic-compile

See the Installation and Usage chapters in our new documentation on https://picongpu.readthedocs.io for detailed instructions.

New Features:

• PIConGPU:
  – laser:
• allow to define the initialization plane #1796
• add transverse Laguerre-modes to standard Gaussian Beam #1580

– ionization:
  * Thomas-Fermi impact ionization model #1754 #2003 #2007 #2037 #2046
  * $Z_{\text{eff}}$, energies, isotope: Ag, He, C, O, Al, Cu #1804 #1860
  * BSI models restructured #2013
  * multiple ionization algorithms can be applied per species, e.g., cut-off barrier suppression ionization (BSI), probabilistic field ionization (ADK) and collisional ionization #1999

– Add EmZ current deposition solver #1582

– FieldTmp:
  * Multiple slots #1703
  * Gather support to fill GUARD #2009

– Particle StartPosition: OnePosition #1753

– Add Bremsstrahlung #1504

– Add kinetic energy algorithm #1744

– Added species manipulators:
  * CopyAttribute #1861
  * FreeRngImpl #1866

– Clang compatible static assert usage #1911

– Use PMACC_ASSERT and PMACC_VERIFY #1662

• PMacc:

  – Improve PMacc tests system #1589
  – Add test for IdProvider #1590
  – Specialize HasFlag and GetFlagType for Particle #1604
  – Add generic atomicAdd #1606
  – Add tests for all RNG generators #1494
  – Extent function twistVectorFieldAxes<>() #1568
  – Expression validation/assertion #1578
  – Use PMacc assert and verify #1661
  – GetNComponents: improve error message #1670
  – Define MakeSeq_t #1708
  – Add Array<> with static size #1725
  – Add shared memory allocator #1726
  – Explicit cast blockIdx and threadIdx to dim3 #1742
  – CMake: allow definition of multiple architectures #1729
  – Add trait FilterByIdentifier #1859
  – Add CompileTime Accessor: Type #1998

• plugins:

  – HDF5/ADIOS:
* MacroParticleCounter #1788
* Restart: Allow disabling of moving window #1668
* FieldTmp: MidCurrentDensityComponent #1561

– Radiation:
  * Add pow compile time using c++11 #1653
  * Add radiation form factor for spherical Gaussian charge distribution #1641

– Calorimeter: generalize (charged & uncharged) #1746

– PNG: help message if dependency is not compiled #1702

– Added:
  * In situ: ISAAC Plugin #1474 #1630
  * Resource log plugin #1457

• tools:
  – Add a tpl file for k80 hypnos that automatically restarts #1567
  – Python3 compatibility for plotNumericalHeating #1747
  – Tpl: Variable Profile #1975
  – Plot heating & charge conservation: file export #1637

• Support for clang as host && device compiler #1933

**Bug Fixes:**

• PIConGPU:
  – 3D3V: missing absorber in z #2042
  – Add missing minus sign wavepacket laser transversal #1722
  – RatioWeighting (DensityWeighting) manipulator #1759
  – MovingWindow: slide_point now can be set to zero. #1783
  – boundElectrons: non-weighted attribute #1808
  – Verify number of ionization energy levels == proton number #1809
  – Ionization:
    * charge of ionized ions #1844
    * ADK: fix effective principal quantum number nEff #2011
  – Particle manipulators: position offset #1852

• PMacc:
  – Avoid CUDA local memory usage of Particle<> #1579
  – Event system deadlock on MPI_Barrier #1659
  – ICC: AllCombinations #1646
  – Device selection: guard valid range #1665
  – MapTuple: broken compile with icc #1648
  – Missing ‘%%’ to use ptx special register #1737
  – ConstVector: check arguments init full length #1803
  – CudaEvent: cyclic include #1836
  – Add missing HDINLINE #1825
– Remove `BOOST_BIND_NO_PLACEHOLDERS` #1849
– Remove CUDA native static shared memory #1929

**plugins:**
– Write openPMD meta data without species #1718
– openPMD: iterationFormat only Basename #1751
– ADIOS trait for `bool` #1756
– Adjust `radAmplitude` python module after openPMD changes #1885
– HDF5/ADIOS: ill-placed helper `#include` #1846
– `#include`: never inside namespace #1835

**work-around for bug in boost 1.64.0 (odeint) + CUDA NVCC 7.5 & 8.0 #2053 #2076**

**Misc:**

**refactoring:**
– PIConGPU:
  * Switch to C++11 only #1649
  * Begin kernel names with upper case letter #1691
  * Maxwell solver, use curl instance #1714
  * Lehe solver: optimize performance #1715
  * Simplify species definition #1711
  * Add missing `math::` namespace to `tan()` #1740
  * Remove usage of pmacc and boost auto #1743
  * Add missing `typename` #1741
  * Change ternary if operator to `if` condition #1748
  * Remove usage of `BOOST_AUTO` and `PMACC_AUTO` #1749
  * mallocMC: organize setting #1779
  * `ParticlesBase` allocate member memory #1791
  * `Particle` constructor interface #1792
  * Species can omit a current solver #1794
  * Use constexpr for arrays in `gridConfig.param` #1799
  * Update mallocMC #1798
  * `DataConnector`: `#includes` #1800
  * Improve Esirkepov speed #1797
  * Ionization Methods: Const-Ness #1824
  * Missing/wrong includes #1858
  * Move functor `Manipulate` to separate file #1863
  * Manipulator `FreeImpl` #1815
  * Ionization: clean up params #1855
  * MySimulation: remove particleStorage #1881
  * New `DataConnector` for fields (& species) #1887 #2045
  * Radiation filter functor: remove macros #1877
* Topic use remove shared keyword #1727
* Remove define ENABLE_RADIATION #1931
* Optimize AssignedTrilinearInterpolation #1936
* Particles<> interface #1942
* Param/Unitless files: remove “config” suffix #1965
* Kernels: Refactor Functions to Functors #1669
* Gamma calculation #1857
* Include order in default loader #1864
* Remove ENABLE_ELECTRONS/IONS #1935
* Add Line<> default constructor #1588

– PMacc:
  * Particles exchange: avoid message spamming #1581
  * Change minimum CMake version #1591
  * CMake: handle PMacc as separate library #1692
  * ForEach: remove boost preprocessor #1719
  * Refactor InheritLinearly #1647
  * Add missing HDINLINE prefix #1739
  * Refactor .h files to .hpp files #1785
  * Log: make events own level #1812
  * float to int cast warnings #1819
  * DataSpaceOperations: Simplify Formula #1805
  * DataConnector: Shared Pointer Storage #1801
  * Refactor MPIReduce #1888
  * Environment refactoring #1890
  * Refactor MallocMCBuffer share #1964
  * Rename typedefs inside ParticleBuffer #1577
  * Add typedefs for Host/DeviceBuffer #1595
  * DeviceBufferIntern: fix shadowed member variable #2051

– plugins:
  * Source files: remove non-ASCII chars #1684
  * replace old analyzer naming #1924
  * Radiation:
    - Remove Nyquist limit switch #1930
    - Remove precompiler flag for form factor #1937
  * compile-time warning in 2D live plugin #2063

– tools:
  * Automatically restart from ADIOS output #1882
  * Workflow: rename tools to set up a sim #1971
  * Check if binary cuda_memtest exists #1897
- C++11 constexpr: remove boost macros #1655
- Cleanup: remove EOL white spaces #1682
- .cfg files: remove EOL white spaces #1690
- Style: more EOL #1695
- Test: remove more EOL white spaces #1685
- Style: replace all tabs with spaces #1698
- Pre-compiler spaces #1693
- Param: Type List Syntax #1709
- Refactor Density Profiles #1762
- Bunch Example: Add Single e- Setup #1755
- Use Travis TRAVIS_PULL_REQUEST_SLUG #1773
- ManipulateDeriveSpecies: Refactor Functors & Tests #1761
- Source Files: Move to Headers #1781
- Single Particle Tests: Use Standard MySimulation #1716
- Replace NULL with C++11 nullptr #1790

**documentation:**
- Wrong comment random->quiet #1633
- Remove sm_20 Comments #1664
- Empty Example & TBG_macros.cfg #1724
- License Header: Update 2017 #1733
- speciesInitialization: remove extra typename in doc #2044
- INSTALL.md:
  - List Spack Packages #1764
  - Update Hypnos Example #1807
  - grammar error #1941
- TBG: Outdated Header #1806
- Wrong sign of delta_angle in radiation observer direction #1811
- Hypnos: Use CMake 3.7 #1823
- Piz Daint: Update example environment #2030
- Doxygen:
  - Warnings Radiation #1840
  - Warnings Ionization #1839
  - Warnings PMacc #1838
  - Warnings Core #1837
  - Floating Docstrings #1856
  - Update struct.hpp #1879
  - Update FieldTmp Operations #1789
  - File Comments in Ionization #1842
  - Copyright Header is no Doxygen #1841

1.5. Changelog
- Sphinx:
  * Introduce Sphinx + Breathe + Doxygen #1843
  * PDF, Link rst/md, png #1944 #1948
  * Examples #1851 #1870 #1878
  * Models, PostProcessing #1921 #1923
  * PMacc Kernel Start #1920
  * Local Build Instructions #1922
  * Python Tutorials #1872
  * Core Param Files #1869
  * Important Classes #1871
  * .md files, tbg, profiles #1883
  * ForEach & Identifier #1889
  * References & Citation #1895
  * Slurm #1896 #1952
  * Restructure Install Instructions #1943
  * Start a User Workflows Section #1955
- ReadTheDocs:
  * Build PDF & EPUB #1947
  * remove linenumbers #1974
- Changelog & Version 0.2.3 (master) #1847
- Comments and definition of radiationObserver default setup #1829
- Typos plot radiation tool #1853
- doc/ -> docs/ #1862
- Particles Init & Manipulators #1880
- INSTALL: Remove gimli #1884
- BibTex: Change ShortHand #1902
- Rename slide_point to movePoint #1917
- Shared memory allocator documentation #1928
- Add documentation on slurm job control #1945
- Typos, modules #1949
- Mention current solver EmZ and compile tests #1966
- Remove assert.hpp in radiation plugin #1667
- Checker script for __global__ keyword #1672
- Compile suite: GCC 4.9.4 chain #1689
- Add TSC and PCS rad form factor shapes #1671
- Add amend option for tee in k80 autorestart tpl #1681
- Test: EOL and suggest solution #1696
- Test: check & remove pre-compiler spaces #1694
- Test: check & remove tabs #1697
• Travis: check PR destination #1732
• Travis: simple style checks #1675
• PositionFilter: remove (virtual) Destructor #1778
• Remove namespace workaround #1640
• Add Bremsstrahlung example #1818
• WarmCopper example: FLYlite benchmark #1821
• Add compile tests for radiation methods #1932
• Add visual studio code files to gitignore #1946
• Remove old QT in situ volume visualization #1735

Thanks to Axel Huebl, René Widera, Alexander Matthes, Richard Pausch, Alexander Grund, Heiko Burau, Marco Garten, Alexander Debus, Erik Zenker, Bifeng Lei and Klaus Steiniger for contributions to this release!

1.5.9 0.2.5

Date: 2017-05-27

Absorber in z in 3D3V, effective charge in ADK ionization

The absorbing boundary conditions for fields in 3D3V simulations were not enabled in z direction. This caused unintended reflections of electro-magnetic fields in z since the 0.1.0 (beta) release. ADK ionization was fixed to the correct charge state (principal quantum number) which caused wrong ionization rates for all elements but Hydrogen.

Changes to “0.2.5”

Bug Fixes:

• ADK ionization: effective principal quantum number nEff #2011
• 3D3V: missing absorber in z #2042

Misc:

• compile-time warning in 2D live plugin #2063
• DeviceBufferIntern: fix shadowed member variable #2051
• speciesInitialization: remove extra typename in doc #2044

Thanks to Marco Garten, Richard Pausch, René Widera and Axel Huebl for spotting the issues and providing fixes!

1.5.10 0.2.4

Date: 2017-03-06

Charge of Bound Electrons, openPMD Axis Range, Manipulate by Position

This release fixes a severe bug overestimating the charge of ions when used with the boundElectrons attribute for field ionization. For HDF5 & ADIOS output, the openPMD axis annotation for fields in simulations with non-cubic cells or moving window was interchanged. Assigning particle manipulators within a position selection was rounded to the closest supercell (IfRelativeGlobalPositionImpl).
Changes to “0.2.3”

**Bug Fixes:**

- ionization: charge of ions with `boundElectrons` attribute #1844
- particle manipulators: position offset, e.g. in `IfRelativeGlobalPositionImpl` rounded to super-cell #1852 #1910
- PMacc:
  - remove `BOOST_BIND_NO_PLACEHOLDERS` #1849
  - add missing `HDINLINE` #1825
  - `CudaEvent`: cyclic include #1836
- plugins:
  - std includes: never inside namespaces #1835
  - HDF5/ADIOS openPMD:
    - GridSpacing, GlobalOffset #1900
    - ill-places helper includes #1846

Thanks to Axel Huebl, René Widera, Thomas Kluge, Richard Pausch and Rémi Lehe for spotting the issues and providing fixes!

1.5.11 0.2.3

**Date:** 2017-02-14

Energy Density, Ionization NaNs and openPMD

This release fixes energy density output, minor openPMD issues, corrects a broken species manipulator to derive density weighted particle distributions, fixes a rounding issue in ionization routines that can cause simulation corruption for very small particle weightings and allows the moving window to start immediately with timestep zero. For ionization input, we now verify that the number of arguments in the input table matches the ion species’ proton number.

Changes to “0.2.2”

**Bug Fixes:**

- openPMD:
  - iterationFormat only basename #1751
  - ADIOS trait for bool #1756
  - `boundElectrons`: non-weighted attribute #1808
- RatioWeighting (DensityWeighting) manipulator #1759
- MovingWindow: slide_point now can be set to zero #1783
- energy density #1750 #1744 (partial)
- possible NAN momenta in ionization #1817
- `tbg` bash templates were outdated/broken #1831

**Misc:**

- ConstVector:
  - check arguments init full length #1803
– float to int cast warnings #1819
  • verify number of ionization energy levels == proton number #1809

Thanks to Axel Huebl, René Widera, Richard Pausch, Alexander Debus, Marco Garten, Heiko Burau and Thomas Kluge for spotting the issues and providing fixes!

1.5.12 0.2.2

Date: 2017-01-04

Laser wavepacket, vacuum openPMD & icc

This release fixes a broken laser profile (wavepacket), allows to use icc as the host compiler, fixes a bug when writing openPMD files in simulations without particle species (“vacuum”) and a problem with GPU device selection on shared node usage via CUDA_VISIBLE_DEVICES.

Changes to “0.2.1”

Bug Fixes:
  • add missing minus sign wavepacket laser transversal #1722
  • write openPMD meta data without species #1718
  • device selection: guard valid range #1665
  • PMacc icc compatibility:
    – MapTuple #1648
    – AllCombinations #1646

Misc:
  • refactor InheritLinearly #1647

Thanks to René Widera and Richard Pausch for spotting the issues and providing fixes!

1.5.13 0.2.1

Date: 2016-11-29

QED synchrotron photon & fix potential deadlock in checkpoints

This releases fixes a potential deadlock encountered during checkpoints and initialization. Furthermore, we forgot to highlight that the 0.2.0 release also included a QED synchrotron emission scheme (based on the review in A. Gonoskov et al., PRE 92, 2015).

Changes to “0.2.0”

Bug Fixes:
  • potential event system deadlock init/checkpoints #1659

Thank you to René Widera for spotting & fixing and Heiko Burau for the QED synchrotron photon emission implementation!
1.5.14 0.2.0 “Beta”

Date: 2016-11-24

Beta release: full multiple species support & openPMD

This release of PIConGPU, providing “beta” status for users, implements full multi-species support for an arbitrary number of particle species and refactors our main I/O to be formatted as openPMD (see http://openPMD.org). Several major features have been implemented and stabilized, including checkpoints, a classical radiation reaction pusher (based on the work of M. Vranic/IST), parallel particle-IDs, generalized on-the-fly particle creation, advanced field ionization schemes and unification of plugin and file names.

This is our last C++98 compatible release (for CUDA 5.5-7.0). Upcoming releases will be C++11 only (CUDA 7.5+), which is already supported in this release, too.

Thank you to Axel Huebl, René Widera, Alexander Grund, Richard Pausch, Heiko Burau, Alexander Debus, Marco Garten, Benjamin Worpitz, Erik Zenker, Frank Winkler, Carline Christian Eckert, Stefan Tietze, Benjamin Schneider, Maximilian Knespel and Michael Bussmann for contributions to this release!

Changes to “0.1.0”

Input file changes: the generalized versions of input files are as always in src/picongpu/include/simulation_defines/.

.param file changes:

- all const parameters are now BOOST_CONSTEXPR_OR_CONST
- add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
- add manipulator for setting boundElectrons<> attribute #768
- add PMACC_CONST_VECTOR for ionization energies #768 #1022
- ionizationEnergies.param #865
- speciesAttributes.param: add ionization model ADK (Ammosov-Delone-Krainov) for lin. pol. and circ. pol cases #922 #1541
- laserConfig.param: documentation fixed and clarified #1043 #1232 #1312 #1477
- speciesAttributes.param: new required traits for for each attribute #1483
- species*.param: refactor species mass/charge definition (relative to base mass/charge) #948
- seed.param: added for random number generator seeds #951
- remove use of native double and float #984 #991
- speciesConstants.param: move magic gamma cutoff value from radiation plugin here #713
- remove invalid typename #926 #944

.unitless file changes:

- add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
- pusher traits simplified #1515
- fieldSolver: numericalCellType is now a namespace not a class #1319
- remove usage of native double and float #983 #991
- remove invalid typename #926
- add new param file: synchrotronPhotons.param #1354
• improve the CFL condition depending on dimension in KHI example #774
• add laserPolynom as option to componentsConfig.param #772

tbg: template syntax

Please be aware that templates (.tpl) used by tbg for job submission changed slightly. Simply use the new system-wise templates from src/picongpu/submit/.

#695 #1609 #1618

Due to unifications in our command line options (plugins) and multi-species support, please update your .cfg files with the new namings. Please visit doc/TBG_macros.cfg and our wiki for examples.

New Features:

• description of 2D3V simulations is now scaled to a user-defined “dZ” depth looking like a one-z-cell 3D simulation #249 #1569 #1601
• current interpolation/smoothing added #888
• add synchrotron radiation of photons from QED- and classical spectrum #1354 #1299 #1398
• species attributes:
  – particle ids for tracking #1410
  – self-describing units and dimensionality #1261
  – add trait GetDensityRatio, add attribute densityRatio
  – current solver is now a optional for a species #1228
  – interpolation is now a optional attribute for a species #1229
  – particle pusher is now a optional attribute for a species #1226
  – add species shape piecewise biquadratic spline P4S #781
• species initialization:
  – add general particle creation module #1353
  – new manipulators to clone electrons from ions #1018
  – add manipulator to change the in cell position after gas creation #947 #959
  – documentation #961
• species pushers:
  – enable the way for substepping particle pushers as RLL
    * add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
    * enable substepping in pushers #1201 #1215 #1339 #1210 #1202 #1221
    * add Runge Kutta solver #1177
    * enable use of macro-particle weighting in pushers #1213
  – support 2D for all pushers #1126
• refactor gas profile definitions #730 #980 #1265
• extend FieldToParticleInterpolation to 1D- and 2D-valued fields #1452
• command line options:
  – parameter validation #863
  – support for --softRestarts <n> to loop simulations #1305
  – a simulation --author can be specified (I/O, etc.) #1296 #1297
  – calling ./picongpu without arguments triggers --help #1294
• FieldTmp:
– scalar fields renamed #1259 #1387 #1523
– momentum over component #1481

• new traits:
  – GetStringProperties for all solvers and species flags #1514 #1519
  – MacroWeighted and WeightingPower #1445

• speedup current deposition solver ZigZag #927
• speedup particle operations with collective atomics #1016
• refactor particle update call #1377
• enable 2D for single particle test #1203

• laser implementations:
  – add phase to all laser implementations #708
  – add in-plane polarization to TWTS laser #852
  – refactor specific float use in laser polynom #782
  – refactored TWTS laser #704

• checkpoints: now self-test if any errors occurred before them #897

• plugins:
  – add 2D support for SliceFieldPrinter plugin #845
  – notify plugins on particles leaving simulation #1394
  – png: threaded, less memory hungry in 2D3V, with author information #995 #1076 #1086 #1251 #1281 #1292 #1298 #1311 #1464 #1465
  – openPMD support in I/O
    * HDF5 and ADIOS plugin refactored #1427 #1428 #1430 #1478 #1517 #1520 #1522 #1529
    * more helpers added #1321 #1323 #1518
    * both write now in a sub-directory in simOutput: h5/ and bp/ #1530
    * getUnit and getUnitDimension in all fields & attributes #1429

• ADIOS:
  * prepare particles on host side before dumping #907
  * speedup with OpenMP #908
  * options to control striping & meta file creation #1062
  * update to 1.10.0+ #1063 #1557
  * checkpoints & restarts implemented #679 #828 #900
  – speedup radiation #996
  – add charge conservation plugin #790
  – add calorimeter plugin #1376

• radiation:
  * ease restart on command line #866
  * output is now openPMD compatible #737 #1053
  * enable compression for hdf5 output #803
  * refactor specific float use #778
refactor radiation window function for 2D/3D #799

- tools:
  - add error when trying to compile picongpu with CUDA 7.5 w/o C++11 #1384
  - add tool to load hdf5 radiation data into python #1332
  - add uncrustify tool (format the code) #767
  - live visualisation client: set fps panel always visible #1240
  - tbg:
    * simplify usage of -p|--project #1267
    * transfers UNIX-permisions from *.tplt to submit.start #1140
  - new charge conservation tools #1102, #1118, #1132, #1178
  - improve heating tool to support unfinished and single simulations #729
  - support for python3 #1134
  - improve graphics of numerical heating tool #742
  - speed up sliceFieldReader.py #1399

- ionization models:
  - add possibility for starting simulation with neutral atoms #768
  - generalize BSI: rename BSI to BSIHydrogenLike, add BSIStarkShifted and BSIEffectiveZ #1423
  - add ADK (Ammosov-Delone-Krainov) for lin. pol. and circ. pol cases #922 #1490 #1541 #1542
  - add Keldysh #1543
  - make use of faster RNG for Monte-Carlo with ionization #1542 #1543

- support radiation + ionization in LWFA example #868

- PMacc:
  - running with synchronized (blocking) kernels now adds more useful output #725
  - add RNGProvider for persistent PRNG states #1236, #1493
  - add MRG32k3a RNG generator #1487
  - move readCheckpointMasterFile to PMacc #1498
  - unify cuda error printing #1484
  - add particle ID provider #1409 #1373
  - split off HostDeviceBuffer from GridBuffer #1370
  - add a policy to GetKeyFromAlias #1252
  - Add border mapping #1133, #1169 #1224
  - make cuSTL gather accept CartBuffers and handle pitches #1196
  - add reference accessors to complex type #1198
  - add more rounding functions #1099
  - add conversion operator from uint3 to Dataspace #1145
  - add more specializations to GetMPI_StructAsArray #1088
  - implement cartBuffer conversion for HostBuffer #1092
  - add a policy for async communication #1079
  - add policies for handling particles in guard cells #1077
– support more types in atomicAddInc and warpBroadcast #1078
– calculate better seeds #1040 #1046
– move MallocMCBuffer to PMacc #1034
– move TypeToPointerPair to PMacc #1033
– add 1D, 2D and 3D linear interpolation cursor #1217 #1448
– add method ‘getPluginFromType()’ to PluginConnector #1393
– math:
  * add abs, asin, acos, atan, log10, fmod, modf, floor to algorithms::math #837 #1218 #1334 #1362 #1363 #1374 #1473
  * precisionCast<> for PMacc::math::Vector<> #746
  * support for boost::mpl::integral_c<> in math::CT::Vector<> #802
  * add complex support #664
– add cuSTL/MapTo1DNavigator #940
– add 2D support for cuSTL::algorithm::mpi::Gather #844
– names for exchanges #1511
– rename EnvMemoryInfo to MemoryInfo #1301
– mallocMC (Memory Allocator for Many Core Architectures) #640 #747 #903 #977 #1171 #1148
  * remove HeapDataBox, RingDataBox, HeapBuffer, RingBuffer #640
  * out of heap memory detection #756
  * support to read mallocMC heap on host side #905
– add multi species support for plugins #794
– add traits:
  * GetDataType #728
  * FilterByFlag #1219
  * GetUniqueTypeId #957 #962
  * GetDefaultConstructibleType #1045
  * GetInitializedInstance #1447
  * ResolveAliasFromSpecies #1451
  * GetStringProperties #1507
– add pointer class for particles FramePointer #1055
– independent sizes on device for GridBuffer::addExchange
– Communicator: query periodic directions #1510
– add host side support for kernel index mapper #902
– optimize size of particle frame for border frames #949
– add pre-processor macro for struct generation #972
– add warp collective atomic function #1013
– speedup particle operations with collective atomics #1014
– add support to deselect unknown attributes in a particle #1524
– add boost.test #1245
* test for HostBufferIntern #1258
* test for setValue() #1268

- add resource monitor #1456
- add MSVC compatibility #816 #821 #931
- const box’es return const pointer #945
- refactor host/device identifier #946

**Bug Fixes:**

- laser implementations:
  - make math calls more robust & portable #1160
  - amplitude of Gaussian beam in 2D3V simulations #1052 #1090
  - avoid non zero E-field integral in plane wave #851
  - fix length setup of plane wave laser #881
  - few-cycle wavepacket #875
  - fix documentaion of a_0 conversation #1043
- FieldTmp Lamor power calculation #1287
- field solver:
  - stricter condition checks #880
  - 2D3V NoSolver did not compile #1073
  - more experimental methods for DS #894
  - experimental: possible out of memory access in directional splitting #890
- moving window moved not exactly with c #1273 #1337 #1549
- 2D3V: possible race conditions for very small, non-default super-cells in current deposition (StrideMapping) #1405
- experimental: 2D3V zigzag current deposition fix for v_z != 0 #823
- vacuumuum: division by zero in Quiet particle start #1527
- remove variable length arrays #932
- gas (density) profiles:
  - gasFreeFormula #988 #899
  - gaussianCloud #807 #1136 #1265
- C++ should catch by const reference #1295
- fix possible underflow on low memory situations #1188
- C++11 compatibility: use BOOST_STATIC_CONSTEXPR where possible #1165
- avoid CUDA 6.5 int(bool) cast bug #680
- PMacc detection in CMake #808
- PMacc:
  - EventPool could run out of free events, potential deadlock #1631
  - Particle<>: avoid using CUDA lmem #1579
  - possible deadlock in event system could freeze simulation #1326
  - HostBuffer includes & constructor #1255 #1596
– const references in Foreach #1593
– initialize pointers with NULL before cudaMalloc #1180
– report device properties of correct GPU #1115
– rename types.h to pmacc_types.hpp #1367
– add missing const for getter in GridLayout #1492
– Cuda event fix to avoid deadlock #1485
– use Host DataBox in Hostbuffer #1467
– allow 1D in CommunicatorMPI #1412
– use better type for params in vector #1223
– use correct sqrt function for abs(Vector) #1461
– fix CMAKE_PREFIX_PATHs #1391, #1390
– remove unnecessary floating point ops from reduce #1212
– set pointers to NULL before calling cudaMalloc #1180
– do not allocate memory if not gather root #1181
– load plugins in registered order #1174
– C++11 compatibility: use BOOST_STATIC_CONSTEXPR where possible #1176 #1175
– fix usage of boost::result_of #1151
– use correct device number #1115
– fix vector shrink function #1113
– split EventSystem.hpp into hpp and tpp #1068
– fix move operators of CartBuffer #1091
– missing includes in MapTuple #627
– GoL example: fix offset #1023
– remove deprecated throw declarations #1000
– cuSTL:
  * cudaPitchedPtr.xsize used wrong #1234
  * gather for supporting static load balancing #1244
  * reduce #936
  * throw exception on cuda error #1235
  * DeviceBuffer: assign operator #1375, #1308, #1463, #1435, #1401, #1220, #1197
  * Host/DeviceBuffers: Constructors (Pointers) #1094
    * let kernel/runtime/Foreach compute best BlockDim #1309
– compile with CUDA 7.0 #748
– device selection with process exclusive enabled #757
– math::Vector<> assignment #806
– math::Vector<> copy constructor #872
– operator[] in ConstVector #981
– empty AllCombinations<...> #1230
– racecondition in kernelShiftParticles #1049
- warning in FieldManipulator #1254
- memory pitch bug in MultiBox and PitchedBox #1096
- math::abs() for the type double #1470
- invalid kernel call in kernelSetValue<> #1407
- data alignment for kernel parameter #1566
- rsqrt usage on host #967
- invalid namespace qualifier #968
- missing namespace prefix #971

• plugins:
  - radiation:
    * enable multi species for radiation plugin #1454
    * compile issues with math in radiation #1552
    * documentation of radiation observer setup #1422
    * gamma filter in radiation plugin #1421
    * improve vector type name encapsulating #998
    * saveguard restart #716
  - CUDA 7.0+ warning in PhaseSpace #750
  - racecondition in ConcatListOfFrames #1278
  - illegal memory acces in Visualisation #1526
  - HDF5 restart: particle offset overflow fixed #721

• tools:
  - mpiInfo: add missing include #786
  - actually exit when pression no in compilesuite #1411
  - fix incorrect mangling of params #1385
  - remove deprecated throw declarations #1003
  - make tool python3 compatible #1416
  - trace generating tool #1264
  - png2gas memory leak fixed #1222
  - tbg:
    * quoting interpretation #801
    * variable assignments stay in .start files #695 #1609
    * multiple variable use in one line possible #699 #1610
    * failing assignments at template evaluation time keep vars undefined #1611
  - heating tool supports multi species #729
  - fix numerical heating tool normalization #825
  - fix logic behind fill color of numerical heating tool #779

• libSplash minimum version check #1284

Misc:

• 2D3V simulations are now honoring the cell “depth” in z to make density interpretations easier #1569
• update documentation for dependencies and installation #1556, #1557, #1559, #1127
• refactor usage of several math functions #1462, #1468
• FieldJ interface clear() replaced with an explicit assign(x) #1335
• templates for known systems updated:
  – renaming directories into “cluster-insitutition”
  – tbg copies cmakeFlags now #1101
  – tbg aborts if mkdir fails #797
  – *tpl & *.profile.example files updated
  – system updates: #937 #1266 #1297 #1329 #1364 #1426 #1512 #1443 #1493
    * Lawrencium (LBNL)
    * Titan/Rhea (ORNL)
    * Piz Daint (CSCS)
    * Taurus (TUD) #1081 #1130 #1114 #1116 #1111 #1137
• replace deprecated CUDA calls #758
• remove support for CUDA devices with sm_10, sm_11, sm_12 and sm_13 #813
• remove unused/unsupported/broken plugins #773 843
  – IntensityPlugin, LiveViewPlugin(2D), SumCurrents, divJ #843
• refactor value_identifier #964
• remove native type double and float #985 #990
• remove __startAtomicTransaction() #1233
• remove __syncthreads() after shared memory allocation #1082
• refactor ParticleBox interface #1243
• rotating root in GatherSlice (reduce load of master node) #992
• reduce GatherSlice memory footprint #1282
• remove None type of ionize, pusher #1238 #1227
• remove math function implementations from Vector.hpp
• remove unused defines #921
• remove deprecated thow declaration #918
• remove invalid typename #917 #933
• rename particle algorithms from ...clone... to ...derive... #1525
• remove math functions from Vector.hpp #1472
• radiation plugin remove uint with uint32_t #1007
• GoL example: CMake modernized #1138
• INSTALL.md
  – moved from /doc/ to /
  – now in root of the repo #1521
  – add environment variable $PICHOME #1162
  – more portable #1164
  – arch linux instructions #1065
• refactor ionization towards independence from Particle class #874
• update submit templates for hypnos #860 #861 #862
• doxygen config and code modernized #1371 #1388
• cleanup of stdlib includes #1342 #1346 #1347 #1348 #1368 #1389
• boost 1.60.0 only builds in C++11 mode #1315 #1324 #1325
• update minimal CMake version to 3.1.0 #1289
• simplify HostMemAssigner #1320
• add asserts to cuSTL containers #1248
• rename TwistVectorAxes -> TwistComponents (cuSTL) #893
• add more robust namespace qualifiers #839 #969 #847 #974
• cleanup code #885 #814 #815 #915 #920 #1027 #1011 #1009
• correct spelling #934 #938 #941
• add compile test for ALL pushers #1205
• tools:
  – adjust executable rights and shebang #1110 #1107 #1104 #1085 #1143
  – live visualization client added #681 #835 #1408
• CMake
  – modernized #1139
  – only allow out-of-source builds #1119
  – cleanup score-p section #1413
  – add OpenMP support #904
• shipped third party updates:
  – restructured #717
  – cuda_memtest #770 #1159
  – CMake modules #1087 #1310 #1533
• removed several -Wshadow warnings #1039 #1061 #1070 #1071

1.5.15 0.1.0

Date: 2015-05-21

This is version 0.1.0 of PIConGPU, a pre-beta version.

Initial field ionization support was added, including the first model for BSI. The code-base was substantially hardened, fixing several minor and major issues. Especially, several restart related issues, an issue with 2D3V zigzack current calculation and a memory issue with Jetson TK1 boards were fixed. A work-around for a critical CUDA 6.5 compiler bug was applied to all affected parts of the code.

Changes to “Open Beta RC6”

.param file changes: See full syntax for each file at https://github.com/ComputationalRadiationPhysics/picongpu/tree/0.1.0/src/picongpu/include/simulation_defines/param

• componentsConfig.param & gasConfig.param fix typo gasHomogeneous #577
• physicalConstants.param: new variable GAMMA_THRES #669
• speciesAttributes.param: new identifier boundElectrons and new aliases ionizer, atomicNumbers
• ionizationEnergies.param, ionizerConfig.param: added

.unitless file changes: See full syntax for each file at https://github.com/ComputationalRadiationPhysics/picongpu/tree/0.1.0/src/picongpu/include/simulation_defines/unitless
• gasConfig.unitless: typo in gasHomogeneous #577
• speciesAttributes.unitless: new unit for boundElectrons identifier
• speciesDefinition.unitless: new traits GetCharge, GetMass, GetChargeState and added ionizers
• ionizerConfig.unitless: added

New Features:
• initial support for field ionization:
  – basic framework and BSI #595
  – attribute (constant flag) for proton and neutron number #687 #731
  – attribute boundElectrons #706
• tools:
  – python scripts:
    * new reader for SliceFieldPrinter plugin #578
    * new analyzer tool for numerical heating #672 #692
  – cuda_memtest:
    * 32bit host system support (Jetson TK1) #583
    * works without nvidia-smi, grep or gawk - optional with NVML for GPU serial number detection (Jetson TK1) #626
  – splash2txt:
    * removed build option S2T_RELEASE and uses CMAKE_BUILD_TYPE #591
  – tbg:
    * allows for defaults for -s, -t, -c via env vars #613 #622
    – 3D live visualization: server tool that collects clients and simulations was published #641
• new/updated particle traits and attributes:
  – getCharge, getMass #596
  – attributes are now automatically initialized to their generic defaults #607 #615
• PMacc:
  – machine-dependent UInt vector class is now split in explicit UInt32 and UInt64 classes #665
  – nvidia random number generators (RNG) refactored #711
• plugins:
  – background fields do now affect plugins/outputs #600
  – Radiation uses/requires HDF5 output #419 #610 #628 #646 #716
  – SliceFieldPrinter supports FieldJ, output in one file, updated command-line syntax #548
  – CountParticles, EnergyFields, EnergyParticles support restarts without overwriting their previous output #636 #703

Bug Fixes:
• CUDA 6.5: \( \text{int(bool)} \) casts were broken (affects plugins BinEnergyParticles, PhaseSpace and might had an effect on methods of the basic PIC cycle) \#570 \#651 \#656 \#657 \#678 \#680

• the ZigZag current solver was broken for 2D3V if non-zero momentum-components in z direction were used (e.g. warm plasmas or purely transversal KHI) \#823

• host-device-shared memory (SoC) support was broken (Jetson TK1) \#633

• boost 1.56.0+ support via \texttt{Resolve<T>} trait \#588 \#593 \#594

• potential race condition in field update and pusher \#604

• using \texttt{--gridDist} could cause a segfault when adding additional arguments, e.g., in 2D3V setups \#638

• MessageHeader (used in \texttt{png} and 2D live visualization) leaked memory \#683

• restarts with HDF5:
  - static load-balancing via \texttt{--gridDist} in y-direction was broken \#639
  - parallel setups with particle-empty GPUs hung with HDF5 \#609 \#611 \#642
  - 2D3V field reads were broken (each field’s z-component was not initialized with the checkpointed values again, e.g., \( B_z \)) \#688 \#689
  - loading more than 4 billion global particles was potentially broken \#721

• plugins:
  - Visualization (png & 2D live sim) memory bug in double precision runs \#621
  - ADIOS
    * storing more than 4 billion particles was broken \#666
    * default of \texttt{adios.aggregators} was broken (now = MPI\_Size) \#662
    * parallel setups with particle-empty GPUs did hang \#661
  - HDF5/ADIOS output of grid-mapped particle energy for non-relativistic particles was zero \#669

• PMacc:
  - CMake: path detection could fail \#796 \#808
  - \texttt{DeviceBuffer<*,DIM3>::getPointer()} was broken (does not affect PIConGPU) \#647
  - empty super-cell memory footprint reduced \#648
  - \texttt{float2int} return type should be \texttt{int} \#623
  - CUDA 7:
    * cuSTL prefixed templates with _ are not allowed; usage of static dim member \#630
    * explicit call to template-ed \texttt{operator()} to avoid warning \#750
    * \texttt{EnvironmentController} caused a warning about extendend friend syntax \#644
  - multi-GPU nodes might fail to start up when not using default compute mode with CUDA 7 drivers \#643

Misc:

• HDF5 support requires libSplash 1.2.4+ \#642 \#715

• various code clean-up for MSVC \#563 \#564 \#566 \#624 \#625

• plugins:
  - removed \texttt{LineSliceFields} \#590
  - \texttt{png} plugin write speedup 2.3x by increasing file size about 12% \#698

• updated contribution guidelines, install, cfg examples \#601 \#598 \#617 \#620 \#673 \#700 \#714

1.5. Changelog 103
updated module examples and cfg files for:
- lawrencium (LBL) #612
- titan (ORNL) #618
- hypnos (HZDR) #670

- an Empty example was added, which defaults to the setup given by all .param files in default mode (a standard PIC cycle without lasers nor particles), see src/picongpu/include/simulation_defines/ #634
- some source files had wrong file permissions #668

1.5.16 Open Beta RC6

Date: 2014-11-25

This is the 6th release candidate, a pre-beta version.

Initial “multiple species” support was added for flexible particles, but is yet still limited to two species. The checkpoint system was refactored and unified, also incorporating extreme high file I/O bandwidth with ADIOS 1.7+ support. The JetsonTK1 development kit (32bit ARM host side) is now experimentally supported by PMacc/PIConGPU. The ZigZag current deposition scheme was implemented providing 40% to 50% speedup over our optimized Esirkepov implementation.

Changes to “Open Beta RC5”

.param file changes:
- Restructured file output control (HDF5/ADIOS), new fileOutput.param #495
- componentsConfig.param: particle pushers and current solvers moved to new files:
  - species.param: general definitions to change all species at once (pusher, current solver)
  - pusherConfig.param: special tweaks for individual particle pushers, forward declarations restructured
  - particleConfig.param: shapes moved to species.param, still defines initial momentum/temperature
  - speciesAttributes.param: defines unique attributes that can be used across all particle species
  - speciesDefinition.param: finally, assign common attributes from speciesAttributes.param and methods from species.param to define individual species, also defines a general compile time “list” of all available species
- currentConfig.param: removed (contained only forward declarations)
- particleDefinition.param: removed, now in speciesAttributes.param
- laserConfig.param: new polarization/focus sections for plane wave and wave-packet: git
diff --ignore-space-change beta-rc5..beta-rc6 src/picongpu/include/simulation_defines/param/laserConfig.param
- memory.param: remove TILE_ globals and define general SuperCellSize and MappingDesc instead #435

.unitless file changes:
- fileOutput.unitless: restructured and moved to fileOutput.param
- checkpoint.unitless: removed some includes
- currentConfig.unitless: removed
• `gasConfig.unitless`: calculate 3D gas density (per volume) and 2D surface charge density (per area) #445
• `gridConfig.unitless`: include changed
• `laserConfig.unitless`: added ellipsoid for wave packet
• `physicalConstatns.unitless`: `GAS_DENSITY_NORMED` fixed for 2D #445
• `pusherConfig.unitless`: restructured, according to `pusherConfig.param`
• `memory.unitless`: removed #435
• `particleDefinition.unitless`: removed
• `speciesAttributes.unitless`: added, contains traits to access species attributes (e.g., position)
• `speciesDefinition.unitless`: added, contains traits to access quasi-fixed attributes (e.g., charge/mass)

**New Features:**

• ZigZag current deposition scheme #436 #476
• initial multi/generic particle species support #457 #474 #516
• plugins
  – BinEnergy supports clean restarts without loosing old files #540
  – phase space now works in 2D3V, with arbitrary super cells and with multiple species #463 #470 #480
  – radiation: 2D support #527 #530
• tools
  – splash2txt now supports ADIOS files #531 #545
• `tools` now support user-defined polarization #534 #535
• wave packet lasers can be ellipses #434 #446
• central restart file to store available checkpoints #455
• PMacc
  – added `math::erf` #525
  – experimental 32bit host-side support (JetsonTK1 dev kits) #571
  – `CT::Vector` refactored and new methods added #473
  – `cuSTL`: better 2D container support #461

**Bug Fixes:**

• esirkepov + CIC current deposition could cause a deadlock in some situations #475
• initialization for kernelSetDrift was broken (traversal of frame lists, CUDA 5.5+) #538 #539
• the particleToField deposition (e.g., in FieldTmp solvers for analysis) forgot a small fraction of the particle #559
• PMacc
  – no `static` keyword for non-storage class functions/members (CUDA 6.5+) #483 #484
  – fix a game-of-life compile error #550
  – `ParticleBox::setAsLastFrame/setAsFirstFrame` race condition (PIConGPU was not affected) #514
• tools
  – `tbg` caused errors on empty variables, tabs, ampersands, comments #485 #488 #528 #529
• dt/CFL ratio in stdout corrected #512
• 2D live view: fix out-of-mem access #439 #452

Misc:
• updated module examples and cfg files for:
  – hypnos (HZDR) #573 #575
  – taurus (ZIH/TUDD) #558
  – titan (ORNL) #489 #490 #492
• Esirkepov register usage (stack frames) reduced #533
• plugins
  – EnergyFields output refactored and clarified #447 #502
  – warnings fixed #479
  – ADIOS
    * upgraded to 1.7+ support #450 #494
    * meta attributes synchronized with HDF5 output #499
• tools
  – splash2txt updates
    * requires libSplash 1.2.3+ #565
    * handle exceptions more transparently #556
    * fix listing of data sets #549 #555
    * fix warnings #553
  – BinEnergyPlot: refactored #542
  – memtest: warnings fixed #521
  – pic2xdmf: refactor XDMF output format #503 #504 #505 #506 #507 #508 #509
  – paraview config updated for hypnos #493
• compile suite
  – reduce verbosity #467
  – remove virtual machine and add access-control list #456 #465
• upgraded to ADIOS 1.7+ support #450 #494
• boost 1.55.0 / nvcc <6.5 work around only applied for affected versions #560
• boost::mkdir is now used where necessary to increase portability #460
• PMacc
  – ForEach refactored #427
  – plugins: notify() is now called before checkpoint() and a getter method was added to retrieve the last call’s time step #541
  – DomainInfo and SubGrid refactored and redefined #416 #537
  – event system overhead reduced by 3-5% #536
  – warnings fixed #487 #515
  – cudaSetDeviceFlags: uses cudaDeviceScheduleSpin now #481 #482
  – __delete makro used more consequently #443
1.5.17 Open Beta RC5

Date: 2014-06-04

This is the 5th release candidate, a pre-beta version.

We rebuild our complete plugin and restart scheme, most of these changes are not backwards compatible and you will have to upgrade to libSplash 1.2+ for HDF5 output (this just means: you can not restart from a beta-rc4 checkpoint with this release).

HDF5 output with libSplash does not contain ghost/guard data any more. These information are just necessary for checkpoints (which are now separated from the regular output).

Changes to “Open Beta RC4”

.param file changes:

- Added selection of optional window functions in radiationConfig.param #286
- Added more window functions in radiationConfig.param #320
- removed double #define __COHERENTINCOHERENTWEIGHTING__ 1 in some examples radiationConfig.param #323
- new file: seed.param allows to vary the starting conditions of “identical” runs #353
- Updated a huge amount of .param files to remove outdated comments #384
- Update gasConfig.param/gasConfig.unitless and doc string in componentsConfig.param with new gasFromHdf5 profile #280

.unitless file changes:

- update fileOutput.unitless and add new file checkpoints.unitless #387
- update fieldSolver.unitless #314
- Update radiationConfig.unitless: adjust to new supercell size naming #394
- Corrected CFL criteria (to be less conservative) in gridConfig.unitless #371

New Features:

- Radiation plugin: add optional window functions to reduce ringing effects caused by sharp boundaries #286 #323 #320
- load gas profiles from png #280
- restart mechanism rebuild #326 #375 #358 #387 #376 #417
- new unified naming scheme for domain and window sizes/offsets #128 #334 #396 #403 #413 #421
- base seed for binary identical simulations now exposed in seed.param #351 #353
- particle kernels without “early returns” #359 #360
- lowered memory foot print during shiftParticles #367
- ShiftCoordinateSystem refactored #414

1.5. Changelog
• tools:
  – tbg warns about broken line continuations in tpl files #259
  – new CMake modules for: ADIOS, libSplash, PNGwriter #271 #304 #307 #308 #406
  – pic2xdmf
    * supports information tags #290 #294
    * one xdmf for grids and one for particles #318 #345
  – Vampir and Score-P support updated/addded #293 #291 #399 #422
  – ParaView remote server description for Hypnos (HZDR) added #355 #397

• plugins
  – former name: “modules” #283
  – completely refactored #287 #336 #342 #344
  – restart capabilities added (partially) #315 #326 #425
  – new 2D phase space analysis added (for 3D sims and one species at a time) #347 #364 #391 #407
  – libSplash 1.2+ upgrade (incompatible output to previous versions) #388 #402

• PMacc
  – new Environment class provides all singletons #254 #276 #404 #405
  – new particle traits, methods and flags #279 #306 #311 #314 #312
  – cuSTL ForEach on 1-3D data sets #335
  – cuSTL twistVectorAxes refactored #370
  – NumberOfExchanges replaced numberOfNeighbors implementation #362
  – new math functions: tan, float2int_rd (host) #374 #410
  – CT::Vector now supports ::shrink #392

Bug fixes:

• CUDA 5.5 and 6.0 support was broken #401
• command line argument parser messages were broken #281 #270 #309
• avoid deadlock in computeCurrent, remove early returns #359
• particles that move in the absorbing GUARD are now cleaned up #363
• CFL criteria fixed (the old one was too conservative) #165 #371 #379
• non-GPU-aware (old-stable) MPI versions could malform host-side pinned/page-locked buffers for subsequent cudaMalloc/cudaFree calls (core routines not affected) #438

• ADIOS
  – particle output was broken #296
  – CMake build was broken #260 #268

• libSplash
  – output performance drastically improved #297

• PMacc
  – GameOfLife example was broken #295
  – log compile broken for high log level #372
  – global reduce did not work for references/const #448
• cuSTL assign was broken for big data sets #431
• cuSTL reduce minor memory leak fixed #433

• compile suite updated and messages escaped #301 #385

• plugins
  • BinEnergyParticles header corrected #317 #319
  • PNG undefined buffer values fixed #339
  • PNG in 2D did not ignore invalid slides #432

• examples
  • Kelvin-Helmholtz example box size corrected #352
  • Bunch/SingleParticleRadiationWithLaser observation angle fixed #424

Misc:
• more generic 2 vs 3D algorithms #255
• experimental PGI support removed #257
• gcc 4.3 support dropped #264
• various gcc warnings fixed #266 #284
• CMake 3.8.12-2 warnings fixed #366
• picongpu.profile example added for
  • Titan (ORNL) #263
  • Hypnos (HZDR) #415
• documentation updated #275 #337 #338 #357 #409
• wiki started: plugins, developer hints, simulation control, examples #288 #321 #328
• particle interfaces cleaned up #278
• ParticleToGrid kernels refactored #329
• slide log is now part of the SIMULATION_STATE level #354
• additional NGP current implementation removed #429

• PMacc
  • GameOfLife example documented #305
  • compile time vector refactored #349
  • shortened compile time template error messages #277
  • cuSTL inline documentation added #365
  • compile time operators and ForEach refactored #380
  • TVec removed in preference of CT::Vector #394
• new developers added #331 #373
• Attribution text updated and BibTex added #428

1.5.18 Open Beta RC4

Date: 2014-03-07
This is the 4th release candidate, a pre-beta version.
Changes to “Open Beta RC3”

{.param file changes:}

- Removed unnecessary includes #234 from: observer.hpp, physicalConstants.param, visColorScales.param, visualization.param, particleConfig.param, gasConfig.param, fieldBackground.param, particleDefinition.param see the lines that should be removed in #234
- Renamed observer.hpp -> radiationObserver.param #237 #241 Changed variable name N_theta to N_observer https://github.com/ComputationalRadiationPhysics/picongpu/commit/9e487ec30ade10ece44fc19fd7a815b8dfe58f61
- Added background FieldJ (current) capability #245 Add the following lines to your fieldBackground.param: https://github.com/ComputationalRadiationPhysics/picongpu/commit/7b22f37c6a58250d6623cfbc821c4f996145aad9

New Features:

- 2D support for basic PIC cycle #212
- hdf5 output xdmf meta description added: ParaView/VisIt support #219
- background current (FieldJ) can be added now #245

Bug fixes:

- beta-rc3 was broken for some clusters due to an init bug #239
- examples/WeibelTransverse 4 GPU example was broken #221
- smooth script was broken for 1D fields #223
- configure non-existing path did not throw an error #229
- compile time vector “max” was broken #224
- cuda_memtest did throw false negatives on hypnos #231 #236
- plugin “png” did not compile for missing freetype #248

Misc:

- documentation updates
  - radiation post processing scripts #222
  - more meta data in hdf5 output #216
  - tbg help extended and warnings to errors #226
  - doc/PARTICIPATE.md is now GitHub’s CONTRIBUTING.md #247 #252
  - slurm interactive queue one-liner added #250
  - developers updated #251
- clean up / refactoring
  - cell_size -> cellSize #227
  - typeCast -> precisionCast #228
  - param file includes (see above for details) #234
  - DataConnector interface redesign #218 #232
  - Esirkepov implementation “paper-like” #238
1.5.19 Open Beta RC3

Date: 2014-02-14

This is the third release candidate, a pre-beta version.

Changes to “Open Beta RC2”

.param and .cfg file changes:

- componentsConfig.param:
  - remove simDim defines #134 #137 (example how to update your existing componentsConfig.
    param, see https://github.com/ComputationalRadiationPhysics/picongpu/commit/af1f20790ad2aa15e6fc2c9a51d8c870)

- dimension.param: new file with simDim setting #134
  - only add this file to your example/test/config if you want to change it from the default value (3D)

- fieldConfig.param: renamed to fieldSolver.param #131

- fieldBackground.param: new file to add external background fields #131

- cfg files cleaned up #153 #193

New Features:

- background fields for E and B #131
- write parallel hdf5 with libSplash 1.1 #141 #151 #156 #191 #196
- new plugins
  - ADIOS output support #179 #196
  - makroParticleCounter/PerSuperCell #163
- cuda_memtest can check mapped memory now #173
- EnergyDensity works for 2-3D now #175
- new type floatD_X shall be used for position types (2-3D) #184
- PMacc
  - new functors for multiplications and substractions #135
  - opened more interfaces to old functors #197
  - MappedMemoryBuffer added #169 #182
  - unary transformations can be performed on DataBox’es now, allowing for non-commutative operations in reduces #204

Bug fixes:

- PMacc
  - GridBuffer could deadlock if called uninitialized #149
  - TaskSetValue was broken for all arrays with x-size != n*256 #174
  - CUDA 6.0 runs crashed during cudaSetDeviceFlags #200
  - extern shared mem could not be used with templated types #199
- tbg
  - clearify error message if the tpl file does not exist #130
- HDF5Writer did not write ions any more #188
- return type of failing Slurm runs fixed #198 #205
• particles in-cell position fixed with cleaner algorithm #209

Misc:
• documentation improved for
  – cuSTL #116
  – gasConfig.param describe slopes better (no syntax changes) #126
  – agreed on coding guide lines #155 #161 #140
  – example documentation started #160 #162 #176
  – taurus (slurm based HPC cluster) updates #206
• IDE: ignore Code::Blocks files #125
• Esirkepov performance improvement by 30% #139
• MySimulation asserts refactored for nD #187
• Fields.def with field forward declarations added, refactored to provide common ValueType #178
• icc warnings in cuda_memcheck fixed #210
• PMacc
  – refactored math::vector to play with DataSpace #138 #147
  – addLicense script updated #167
  – MPI_CHECK writes to stderr now #168
  – TVec from/to CT::Int conversion #185
  – PositionFilter works for 2-3D now #189 #207
  – DeviceBuffer cudaPitchedPtr handling clean up #186
  – DataBoxDim1Access refactored #202

1.5.20 Open Beta RC2

Date: 2013-11-27
This is the second release candidate, a pre-beta version.

Changes to “Open Beta RC1”

.param file changes:
• gasConfig.param:
  – add gasFreeFormula #96 (example how to update your existing gasConfig.param, see https://github.com/ComputationalRadiationPhysics/picongpu/pull/96/files#diff-1)
  – add inner radius to gasSphereFlanks #66 (example how to update your existing gasConfig.param, see https://github.com/ComputationalRadiationPhysics/picongpu/pull/66/files#diff-0)

New Features:
• A change log was introduced for master releases #93
• new gas profile “gasFreeFormula” for user defined profiles #96
• CMake (config) #79
  – checks for minimal required versions of dependent libraries #92
  – checks for libSplash version #85
– update to v2.8.5+ #52
– implicit plugin selection: enabled if found #52
– throw more warnings #37
– experimental support for icc 12.1 and PGI 13.6 #37

• PMacc
  – full rewrite of the way we build particle frames # 86
  – cuSTL: ForEach works on host 1D and 2D data now #44
  – math::pow added #54
  – compile time ForEach added #50

• libSplash
  – dependency upgraded to beta (v1.0) release #80
  – type traits for types PIConGPU - libSplash #69
  – splash2txt update to beta interfaces #83

• new particle to grid routines calculating the Larmor energy #68
• dumping multiple FieldTmp to hdf5 now possible #50
• new config for SLURM batch system (taurus) #39

Bug fixes:

• PMacc
  – cuSTL
    * assign was broken for deviceBuffers #103
    * lambda expressions were broken #42 #46 #100
    * icc support was broken #100 #101
    * views were broken #62
  – InheritGenerator and deselect: icc fix #101
  – VampirTrace (CUPTI) support: cudaDeviceReset added #90
  – GameOfLife example fixed #53 #55
  – warnings in __cudaKernel fixed #51

• picongpu
  – removed all non-ascii chars from job scripts #95 #98

• CMake
  – keep ptx code was broken #82
  – PGI: string compare broken #75
  – MPI: some libs require to load the C++ dependencies, too #64
  – removed deprecated variables #52
  – Threads: find package was missing #34
  – various libSplash bugs #78 #80 #84
  – current calculation speedup was broken #72
  – Cell2Particle functor missed to provide static methods #49

• tools

1.5. Changelog
– compile: script uses -q now implicit for parallel (-j N) tests
– plotDensity: update to new binary format #47

• libraries
  – boost 1.55 work around, see trac #9392 (nvcc #391854)

Misc:
• new reference: SC13 paper, Gordon Bell Finals #106
• new flavoured logo for alpha
• Compile Suite: GitHub integration #33 #35
• dropped CUDA sm_13 support (now sm_20+ is required) #42

1.5.21 Open Beta RC1

Date: 2013-09-05 07:47:03 -0700

This is the first release candidate, a pre-beta version. We tagged this state since we started to support sm_20+ only.

Changes to “Open Alpha”

n/a

1.5.22 Open Alpha

Date: 2013-08-14 02:25:36 -0700

That’s our our open alpha release. The alpha release is developer and power user release only! Users should wait for our beta release!
2.1 Reference

Section author: Axel Huebl

PIConGPU is an almost decade-long scientific project with many people contributing to it. In order to credit the work of others, we expect you to cite our latest paper describing PIConGPU when publishing and/or presenting scientific results.

In addition to that and out of good scientific practice, you should document the version of PIConGPU that was used and any modifications you applied. A list of releases alongside a DOI to reference it can be found here:

https://github.com/ComputationalRadiationPhysics/picongpu/releases

2.1.1 Citation

BibTeX code:

```latex
@inproceedings{PIConGPU2013,
  title = {Radiative Signatures of the Relativistic Kelvin-Helmholtz Instability},
  booktitle = {Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis},
  series = {SC '13},
  year = {2013},
  isbn = {978-1-4503-2378-9},
  location = {Denver, Colorado},
  pages = {5:1--5:12},
  articleno = {5},
  numpages = {12},
  url = {http://doi.acm.org/10.1145/2503210.2504564},
  doi = {10.1145/2503210.2504564},
  acmid = {2504564},
  publisher = {ACM},
  address = {New York, NY, USA},
}
```

2.1.2 Acknowledgements

In many cases you receive support and code base maintenance from us or the PIConGPU community without directly justifying a full co-authorship. Additional to the citation, please consider adding an acknowledgement of the following form to reflect that:

We acknowledge all contributors to the open-source code PIConGPU for enabling our simulations.
or:

We acknowledge [list of specific persons that helped you] and all further contributors to the open-source code PIConGPU for enabling our simulations.

## 2.1.3 Community Map

PIConGPU comes without a registration-wall, with open and re-distributable licenses and without any strings attached. We therefore rely on you to show our community, diversity and usefulness, e.g. to funding agencies.

Please consider adding yourself to our community map!

Thank you and enjoy PIConGPU and our community!

See also:

You need to have an environment loaded (source $HOME/picongpu.profile) that provides all PIConGPU dependencies to complete this chapter.

## 2.2 Basics

*Section author: Axel Huebl*

### 2.2.1 Preparation

First, decide where to store input files, a good place might be $HOME (~) because it is usually backed up. Second, decide where to store your output of simulations which needs to be placed on a high-bandwidth, large-storage file system which we will refer to as $SCRATCH.

For a first test you can also use your home directory:

```
export SCRATCH=$HOME
```

We need a few directories to structure our workflow:

```bash
# PIConGPU input files
mkdir $HOME/picInputs

# PIConGPU simulation output
mkdir $SCRATCH/runs
```

### 2.2.2 Step-by-Step

1. Create an Input (Parameter) Set

    ```bash
    # clone the LWFA example to $HOME/picInputs/myLWFA
    pic-create $PIC_EXAMPLES/LaserWakefield $HOME/picInputs/myLWFA
    
    # switch to your input directory
    cd $HOME/picInputs/myLWFA
    ```

PIConGPU is controlled via two kinds of textual input sets: compile-time options and runtime options.

Compile-time *param files* reside in include/picongpu/param/ and define the physics case and deployed numerics. After creation and whenever options are changed, PIConGPU requires a re-compile. Feel free to take a look now, but we will later come back on how to edit those files.

Runtime (command line) arguments are set in etc/picongpu/*.cfg files. These options do not require a re-compile when changed (e.g. simulation size, number of devices, plugins, ...).
2. Compile Simulation

In our input, .param files are build directly into the PIConGPU binary for performance reasons. A compile is required after changing or initially adding those files.

In this step you can optimize the simulation for the specific hardware you want to run on. By default, we compile for Nvidia GPUs with the CUDA backend, targeting the oldest compatible architecture.

```
pic-build
```

This step will take a few minutes. Time for a coffee or a sword fight!

We explain in the details section below how to set further options, e.g. CPU targets or tuning for newer GPU architectures.

3. Run Simulation

While you are still in $HOME/picInputs/myLWFA, start your simulation on one CUDA capable GPU:

```
# example run for an interactive simulation on the same machine
$ pic-build -s bash -c etc/picongpu/1.cfg -t etc/picongpu/bash/mpiexec.tpl $SCRATCH/runs/ →lwfa_001
```

This will create the directory $SCRATCH/runs/lwfa_001 where all simulation output will be written to. $pic-build will further create a subfolder input/ in the directory of the run with the same structure as myLWFA to archive your input files. Subfolder simOutput/ has all the simulation results. Particularly, the simulation progress log is in simOutput/output.

2.2.3 Details on the Commands Above

$pic-build

The $pic-build tool is explained in detail in its own section. Its primary purpose is to abstract the options in runtime .cfg files from the technical details on how to run on various supercomputers.

For example, if you want to run on the HPC System "Hypnos" at HZDR, your $pic-build submit command would just change to:

```
# request 1 GPU from the PBS batch system and run on the queue "k20"
$ pic-build -s qsub -c etc/picongpu/1.cfg -t etc/picongpu/bash/mpiexec.tpl $SCRATCH/ →runs/lwfa_002
```

```
# request 16 GPUs
$ pic-build -s qsub -c etc/picongpu/16.cfg -t etc/picongpu/bash/mpiexec.tpl $SCRATCH/ →runs/lwfa_003
```

Note that we can use the same 1.cfg file, your input set is portable.

$pic-create

This tool is just a short-hand to create a new set of input files. It copies from an already existing set of input files (e.g. our examples or a previous simulation) and adds additional helper files.

See $pic-create --help for more options during input set creation:

```
usage: pic-create [OPTION] [src_dir] dest_dir
```

(continues on next page)
If no src_dir is set picongpu a default case is cloned
If src_dir is not in the current directory, pic-create will
look for it in $PIC_EXAMPLES

-\( f \) | --force - merge data if destination already exists
-\( h \) | --help - show this help message

Dependencies: rsync

A run simulation can also be reused to create derived input sets via pic-create:

```
pic-create $SCRATCH/runs/lwfa_001/input $HOME/picInputs/mySecondLWFA
```

**pic-build**

This tool is actually a short-hand for an *out-of-source build with CMake*.

In detail, it does:

```
# go to an empty build directory
mkdir -p .build
cd .build

# configure with CMake
pic-configure $OPTIONS ..

# compile PIConGPU with the current input set (e.g. myLWFA)
# - "make -j install" runs implicitly "make -j" and then "make install"
# - make install copies resulting binaries to input set
make -j install
```

pic-build accepts the same command line flags as *pic-configure*. For example, if you want to build for running on CPUs instead of a GPUs, call:

```
# example for running efficiently on the CPU you are currently compiling on
pic-build --backend omp2b
```

Its full documentation from pic-build --help reads:

```
Build new binaries for a PIConGPU input set

Creates or updates the binaries in an input set. This step needs to be performed every time a .param file is changed.

This tool creates a temporary build directory, configures and compiles current input set in it and installs the resulting binaries.
This is just a short-hand tool for switching to a temporary build directory and running 'pic-configure ..' and 'make install' manually.

You must run this command inside an input directory.

usage: pic-build [OPTIONS]

-\( b \) | --backend - set compute backend and optionally the architecture syntax: backend[:architecture]
supported backends: cuda, omp2b, serial, tbb, threads (e.g.: "cuda:20;35;37;52;60" or "omp2b:native" or "omp2b") default: "cuda" if not set via environment variable PIC_...
```

(continues on next page)
pic-configure

This tool is just a convenient wrapper for a call to CMake. It is executed from an empty build directory.

You will likely not use this tool directly. Instead, pic-build from above calls pic-configure for you, forwarding its arguments.

We strongly recommend to set the appropriate target compute backend via `-b` for optimal performance. For Nvidia CUDA GPUs, set the compute capability of your GPU:

```
# example for running efficiently on a K80 GPU with compute capability 3.7
pic-configure -b "cuda:37" $HOME/picInputs/myLWFA
```

For running on a CPU instead of a GPU, set this:

```
# example for running efficiently on the CPU you are currently compiling on
pic-configure -b "omp2b:native" $HOME/picInputs/myLWFA
```

**Note:** If you are compiling on a cluster, the CPU architecture of the head/login nodes versus the actual compute architecture does likely vary! Compiling a backend for the wrong architecture does in the best case dramatically reduce your performance and in the worst case will not run at all!

During configure, the backend’s architecture is forwarded to the compiler’s `-mtune` and `-march` flags. For example, if you are compiling with GCC for running on AMD Opteron 6276 CPUs set `-b omp2b:bdver1` or for Intel Xeon Phi Knight’s Landing CPUs set `-b omp2b:knl`.

See pic-configure --help for more options during input set configuration:

```
Configure PIConGPU with CMake

Generates a call to CMake and provides short-hand access to selected PIConGPU CMake options.
Advanced users can always run 'ccmake .'' after this call for further compilation options.

usage: pic-configure [OPTIONS] <inputDirectory>

-1 | --install - path were picongpu shall be installed
(default is <inputDirectory>)
-b | --backend - set compute backend and optionally the architecture syntax: backend[:architecture]
supported backends: cuda, omp2b, serial, tbb, threads
(e.g.: "cuda:20;35;37;52;60" or "omp2b:native" or "omp2b")
default: "cuda" if not set via environment variable PIC_BACKEND

-c | --cmake - overwrite options for cmake
(t.e.: "-DPIC_VERBOSE=21 -DCMAKE_BUILD_TYPE=Debug")
-t <presetNumber> - configure this preset from cmakeFlags
-f | --force - clear the cmake file cache and scan for new param files
-h | --help - show this help message
```
After running configure you can run `ccmake` to set additional compile options (optimizations, debug levels, hardware version, etc.). This will influence your build done via `make install`.

You can pass further options to configure PIConGPU directly instead of using `ccmake ..` by passing `-c "-DOPTION1=VALUE1 -DOPTION2=VALUE2"`.

2.3 .param Files

Section author: Axel Huebl

Parameter files, 
`*.param` placed in `include/picongpu/param/` are used to set all compile-time options for a PIConGPU simulation. This includes most fundamental options such as numerical solvers, floating precision, memory usage due to attributes and super-cell based algorithms, density profiles, initial conditions etc.

2.3.1 Editing

For convenience, we provide a tool `pic-edit` to edit the compile-time input by its name. For example, if you want to edit the grid and time step resolution, file output and add a laser to the simulation, open the according files via:

```
# first switch to your input directory
cd $HOME/picInputs/myLWFA
pic-edit grid fileOutput laser
```

See `pic-edit --help` for all available files:

```
# Edit compile-time options for a PIConGPU input set
# Opens .param files in an input set with the default "EDITOR".
# If a .param file is not yet part of the input set but exists in the defaults, it will be transparently added to the input set.
# You must run this command inside an input directory.
# The currently selected editor is: /usr/bin/vim.basic
# You can change it via the "EDITOR" environment variable.
usage: pic-edit <input>
Available <input>s:
  bremsstrahlung components density dimension fieldBackground fieldSolver fileOutput
  flylite grid ionizationEnergies ionizer isaac laser mallocMC memory particle
  particleCalorimeter particleFilters particleMerger physicalConstants pml png
  pngColorScales precision pusher radiation radiationObserver random species
  --starter synchrotronPhotons transitionRadiation unit
```

2.3.2 Rationale

High-performance hardware comes with a lot of restrictions on how to use it, mainly memory, control flow and register limits. In order to create an efficient simulation, PIConGPU compiles to exactly the numerical solvers (kernels) and physical attributes (fields, species) for the setup you need to run, which will furthermore be specialized for a specific hardware.

This comes at a small cost: when even one of those settings is changed, you need to recompile. Nevertheless, wasting about 5 minutes compiling on a single node is nothing compared to the time you save at scale!
All options that are less or non-critical for runtime performance, such as specific ranges observables in plugins or how many nodes shall be used, can be set in runtime configuration files (*.cfg) and do not need a recompile when changed.

### 2.3.3 Files and Their Usage

If you use our pic-configure script wrappers, you do not need to set all available parameter files since we will add the missing ones with sane defaults. Those defaults are:

- a standard, single-precision, well normalized PIC cycle suitable for relativistic plasmas
- no external forces (no laser, no initial density profile, no background fields, etc.)

### 2.3.4 All Files

When setting up a simulation, it is recommended to adjust .param files in the following order:

#### PIC Core

**dimension.param**

The spatial dimensionality of the simulation.

**Defines**

**SIMDIM**

Possible values: DIM3 for 3D3V and DIM2 for 2D3V.

```cpp
namespace picongpu

Variables

constexpr uint32_t simDim = SIMDIM
```

**grid.param**

Definition of cell sizes and time step.

Our cells are defining a regular, cartesian grid. Our explicit FDTD field solvers define an upper bound for the time step value in relation to the cell size for convergence. Make sure to resolve important wavelengths of your simulation, e.g. shortest plasma wavelength and central laser wavelength both spatially and temporarily.

**Units in reduced dimensions**

In 2D3V simulations, the CELL_DEPTH_SI (Z) cell length is still used for normalization of densities, etc..

A 2D3V simulation in a cartesian PIC simulation such ours only changes the degrees of freedom in motion for (macro) particles and all (field) information in z travels instantaneous, making the 2D3V simulation behave like the interaction of infinite “wire particles” in fields with perfect symmetry in Z.

```cpp
namespace picongpu

2.3. .param Files 121
```
**Variables**

```cpp
constexpr uint32_t picongpu::ABSORBER_CELLS[3][2] = {{32, 32}, {32, 32}, {32, 32}};
```

Defines the size of the absorbing zone (in cells)

unit: none

```cpp
constexpr float_X picongpu::ABSORBER_STRENGTH[3][2] = {{1.0e-3, 1.0e-3}, {1.0e-3, 1.0e-3}, {1.0e-3, 1.0e-3}};
```

Define the strength of the absorber for any direction.

unit: none

```cpp
constexpr float_64 movePoint = 0.9
```

When to move the co-moving window.

An initial pseudo particle, flying with the speed of light, is fired at the begin of the simulation. When it reaches movePoint % of the absolute(*) simulation area, the co-moving window starts to move with the speed of light.

(*) Note: beware, that there is one “hidden” row of gpus at the y-front, when you use the co-moving window 0.75 means only 75% of simulation area is used for real simulation

Warning: this variable is deprecated, but currently still required for building purposes. Please keep the variable here. In case a moving window is enabled in your .cfg file, please set the move point using the ‘windowMovePoint’ parameter in that file, its default value is movePoint.

```cpp
namespace SI
```

**Variables**

```cpp
constexpr float_64 DELTA_T_SI = 0.8e-16
```

Duration of one timestep unit: seconds.

```cpp
constexpr float_64 CELL_WIDTH_SI = 0.1772e-6
```

equals X unit: meter

```cpp
constexpr float_64 CELL_HEIGHT_SI = 0.4430e-7
```

equals Y - the laser & moving window propagation direction unit: meter

```cpp
constexpr float_64 CELL_DEPTH_SI
```

equals Z unit: meter

`components.param`

Select a user-defined simulation class here, e.g.

with strongly modified initialization and/or PIC loop beyond the parametrization given in other .param files.

```cpp
namespace simulation_starter
```

Simulation Starter Selection: This value does usually not need to be changed.

Change only if you want to implement your own SimulationHelper (e.g. MySimulation) class.

- defaultPIConGPU : default PIConGPU configuration

```cpp
fieldSolver.param
```

Configure the field solver.

Select the numerical Maxwell solver (e.g. Yee’s method).

Also allows to configure ad hoc mitigations for high frequency noise in some setups via current smoothing.
namespace picongpu

namespace fields

Typedefs

using CurrentInterpolation = currentInterpolation::None
Current Interpolation.

CurrentInterpolation is used to set a method performing the interpolate/assign operation from the
generated currents of particle species to the electro-magnetic fields.

Allowed values are:
• None:
  – default for staggered grids/Yee-scheme
  – updates E
• Binomial: 2nd order Binomial filter
  – smooths the current before assignment in staggered grid
  – updates E & breaks local charge conservation slightly
• NoneDS:
  – experimental assignment for all-centered/directional splitting
  – updates E & B at the same time

using Solver = maxwellSolver::Yee<CurrentInterpolation>
FieldSolver.

Field Solver Selection:
• Yee< CurrentInterpolation > : standard Yee solver
• YeePML< CurrentInterpolation >: standard Yee solver with PML absorber
• Lehe< CurrentInterpolation >: Num. Cherenkov free field solver in a chosen direction
• DirSplitting< CurrentInterpolation >: Sentoku’s Directional Splitting Method
• None< CurrentInterpolation >: disable the vacuum update of E and B

laser.param

Configure laser profiles.

All laser propagate in y direction.

Available profiles:
• None : no laser init
• GaussianBeam : Gaussian beam (focusing)
• PulseFrontTilt : Gaussian beam with a tilted pulse envelope in ‘x’ direction
• PlaneWave : a plane wave (Gaussian in time)
• Wavepacket : wavepacket (Gaussian in time and space, not focusing)
• Polynom : a polynomial laser envelope
• ExpRampWithPrepulse : wavepacket with exponential upramps and prepulse

In the end, this file needs to define a Selected class in namespace
picongpu::fields::laserProfiles. A typical profile consists of a laser profile class and its
parameters. For example:

using Selected = GaussianBeam< GaussianBeamParam >;

namespace picongpu

2.3. .param Files
namespace fields

namespace laserProfiles

Typedefs

using Selected = None<>  
  currently selected laser profile

struct ExpRampWithPrepulseParam  
  Based on a wavepacket with Gaussian spatial envelope.
  
  and the following temporal shape: A Gaussian peak (optionally lengthened by a plateau) is preceded by two pieces of exponential preramps, defined by 3 (time, intensity)-points. The first two points get connected by an exponential, the 2nd and 3rd point are connected by another exponential, which is then extrapolated to the peak. The Gaussian is added everywhere, but typically contributes significantly only near the peak. It is advisable to set the third point far enough from the plateau (approx 3*FWHM), then the contribution from the Gaussian is negligible there, and the intensity can be set as measured from the laser profile. Optionally a Gaussian prepulse can be added, given by the parameters of the relative intensity and time point. The time of the prepulse and the three preramp points are given in SI, the intensities are given as multiples of the peak intensity.

Public Types

enum PolarisationType  
  Available polarisation types.
  
  Values:
  
  LINEAR_X = 1u  
  LINEAR_Z = 2u  
  CIRCULAR = 4u

Public Static Attributes

constexpr float_X INT_RATIO_PREPULSE = 0.  
constexpr float_X INT_RATIO_POINT_1 = 1.e-8  
constexpr float_X INT_RATIO_POINT_2 = 1.e-4  
constexpr float_X INT_RATIO_POINT_3 = 1.e-4  
constexpr float_64 TIME_PREPULSE_SI = -950.0e-15  
constexpr float_64 TIME_PEAKPULSE_SI = 0.0e-15  
constexpr float_64 TIME_POINT_1_SI = -1000.0e-15  
constexpr float_64 TIME_POINT_2_SI = -300.0e-15  
constexpr float_64 TIME_POINT_3_SI = -100.0e-15  
constexpr float_64 WAVE_LENGTH_SI = 0.8e-6  
  unit: meter  
constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::UNITCONV.
constexpr float_64 _A0 = 20.
unit: W / m^2

constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI
unit: Volt / meter

constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 0.0 * WAVE_LENGTH_SI / picongpu::SI::SPEED_OF_LIGHT_SI
unit: Volt / meter

The profile of the test Lasers 0 and 2 can be stretched by a constant area between the up and downramp unit: seconds

constexpr float_64 PULSE_LENGTH_SI = 3.0e-14 / 2.35482
Pulse length: sigma of std. gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt{ 2* ln(2) } ] [ 2.354820045 ] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls “pulse duration” unit: seconds (1 sigma)

constexpr float_64 W0_X_SI = 2.5 * WAVE_LENGTH_SI
beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, W0_X_SI is this distance in x-direction W0_Z_SI is this distance in z-direction if both values are equal, the laser has a circular shape in x-z W0_SI = FWHM_of_Intensity / sqrt{ 2* ln(2) } [ 1.17741 ] unit: meter

constexpr float_64 W0_Z_SI = W0_X_SI

constexpr float_64 RAMP_INIT = 16.0
The laser pulse will be initialized half of PULSE_INIT times of the PULSE_LENGTH before plateau and half at the end of the plateau unit: none.

constexpr uint32_t initPlaneY = 0
cell from top where the laser is initialized

if initPlaneY == 0 than the absorber are disabled, if initPlaneY > absorbercells negative Y the negative absorber in y direction is enabled
valid ranges:
• initPlaneY == 0
• absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

constexpr float_X LASER_PHASE = 0.0
laser phase shift (no shift: 0.0)
sin(omega*time + laser_phase): starts with phase=0 at center > E-field=0 at center
unit: rad, periodic in 2*pi

constexpr PolarisationType Polarisation = LINEAR_X
Polarization selection.

struct GaussianBeamParam

Public Types

enum PolarisationType
Available polarisation types.

Values:

LINEAR_X = 1u
LINEAR_Z = 2u
CIRCULAR = 4u
using LAGUERREMODES_t = gaussianBeam::LAGUERREMODES_t

Public Static Attributes

constexpr float_64 WAVE_LENGTH_SI = 0.8e-6
unit: meter

constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::SI::ELECTRON_MASS_SI * picongpu::SI::SPEED_OF_LIGHT_SI / picongpu::SI::ELECTRON_CHARGE_SI
Convert the normalized laser strength parameter a0 to Volt per meter.

constexpr float_64 AMPLITUDE_SI = 1.738e13
unit: W / m^2
unit: none unit: Volt / meter unit: Volt / meter

constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0
Pulse length: sigma of std. gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt{ 2* ln(2) } ] [ 2.354820045 ] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls “pulse duration”
unit: seconds (1 sigma)

constexpr float_64 W0_SI = 5.0e-6 / 1.17741
beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, at the focus position of the laser W0_SI = FWHM_of_Intensity / sqrt{ 2* ln(2) } [ 1.17741 ]
unit: meter

constexpr float_64 FOCUS_POS_SI = 4.62e-5
the distance to the laser focus in y-direction unit: meter

constexpr float_64 PULSE_INIT = 20.0
The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH.
unit: none

constexpr uint32_t initPlaneY = 0
cell from top where the laser is initialized
if initPlaneY == 0 than the absorber are disabled. if initPlaneY > absorberCells negative Y the negative absorber in y direction is enabled
valid ranges:
• initPlaneY == 0
• absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

constexpr float_X LASER_PHASE = 0.0
laser phase shift (no shift: 0.0)
\sin(\omega*time + laser\_phase): starts with phase=0 at center > E\_field=0 at center
unit: rad, periodic in 2*\pi

constexpr uint32_t MODENUMBER = gaussianBeam::MODENUMBER

constexpr PolarisationType Polarisation = CIRCULAR
Polarization selection.

struct PlaneWaveParam
Public Types

```cpp
enum PolarisationType
    Available polarization types.
    Values:
    LINEAR_X = 1u
    LINEAR_Z = 2u
    CIRCULAR = 4u
```

Public Static Attributes

```cpp
constexpr float_64 WAVE_LENGTH_SI = 0.8e-6
    unit: meter

constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::SI::ELECTRON_MASS_SI * picongpu::SI::SPEED_OF_LIGHT_SI / picongpu::SI::ELECTRON_CHARGE_SI
    Convert the normalized laser strength parameter a0 to Volt per meter.

constexpr float_64 _A0 = 1.5
    unit: W / m^2
    unit: none

constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI
    unit: Volt / meter

constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 13.34e-15
    unit: Volt / meter
    The profile of the test Lasers 0 and 2 can be stretched by a constant area between the up and downramp unit: seconds

constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0
    Pulse length: sigma of std. gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / \left[ \frac{2*sqrt{ 2* ln(2) } }{ 2.354820045 } \right] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls “pulse duration” unit: seconds (1 sigma)

constexpr uint32_t initPlaneY = 0
    cell from top where the laser is initialized
    if initPlaneY == 0 than the absorber are disabled. if initPlaneY > absorbercells negative Y the negative absorber in y direction is enabled
    valid ranges:
    • initPlaneY == 0
    • absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

constexpr float_64 RAMP_INIT = 20.6146
    The laser pulse will be initialized half of PULSE_INIT times of the PULSE_LENGTH before and after the plateau unit: none.

constexpr float_X LASER_PHASE = 0.0
    laser phase shift (no shift: 0.0)
    sin(omega*time + laser_phase): starts with phase=0 at center > E-field=0 at center
    unit: rad, periodic in 2*pi

constexpr PolarisationType Polarisation = LINEAR_X
    Polarization selection.
```
**struct PolynomParam**

Based on a wavepacket with Gaussian spatial envelope.

*Wavepacket* with a polynomial temporal intensity shape.

**Public Types**

*enum PolarisationType*

Available polarization types.

*Values:*

- **LINEAR_X** = 1u
- **LINEAR_Z** = 2u
- **CIRCULAR** = 4u

**Public Static Attributes**

*constexpr float_64 WAVE_LENGTH_SI = 0.8e-6*

unit: meter

*constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::SI::ELECTRON_MASS_SI * picongpu::SI::SPEED_OF_LIGHT_SI / picongpu::SI::ELECTRON_CHARGE_SI*

Convert the normalized laser strength parameter a0 to Volt per meter.

*constexpr float_64 AMPLITUDE_SI = 1.738e13*

unit: W / m^2

unit: none unit: Volt / meter unit: Volt / meter

*constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 13.34e-15*

The profile of the test Lasers 0 and 2 can be stretched by a constant area between the up and downramp unit: seconds.

*constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0*

Pulse length: sigma of std. gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt{ 2* ln(2) } ] [ 2.354820045 ] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls "pulse duration" unit: seconds (1 sigma)

*constexpr float_64 W0_X_SI = 4.246e-6*

beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, at the focus position of the laser unit: meter

*constexpr uint32_t initPlaneY = 0*

cell from top where the laser is initialized

if initPlaneY == 0 than the absorber are disabled. if initPlaneY > absorbercells negative Y the negative absorber in y direction is enabled

valid ranges:
- initPlaneY == 0
- absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

*constexpr float_64 PULSE_INIT = 20.0*

The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH. unit: none

*constexpr float_X LASER_PHASE = 0.0*

laser phase shift (no shift: 0.0)

sin(omega*time + laser_phase): starts with phase=0 at center > E-field=0 at center
unit: rad, periodic in 2*pi

```cpp
constexpr PolarisationType Polarisation = LINEAR_X
```
Polarization selection.

```cpp
struct PulseFrontTiltParam
```

**Public Types**

```cpp
enum PolarisationType
```
Available polarisation types.

*Values:*

- LINEAR_X = 1u
- LINEAR_Z = 2u
- CIRCULAR = 4u

**Public Static Attributes**

```cpp
constexpr float_64 WAVE_LENGTH_SI = 0.8e-6
```
unit: meter

```cpp
constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::SI::ELECTRON_MASS_SI*
```

```cpp
constexpr float_64 AMPLITUDE_SI = 1.738e13
```
unit: none unit: Volt / meter

```cpp
constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0
```
Pulse length: sigma of std.
gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt{ 2* ln(2) } ] [ 2.354820045 ] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls “pulse duration”
unit: seconds (1 sigma)

```cpp
constexpr float_64 W0_SI = 5.0e-6 / 1.17741
```
beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, at the focus position of the laser W0_SI = FWHM_of_Intensity / sqrt{ 2* ln(2) } [ 1.17741 ]
unit: meter

```cpp
constexpr float_64 FOCUS_POS_SI = 4.62e-5
```
the distance to the laser focus in y-direction unit: meter

```cpp
constexpr float_64 TILT_X_SI = 0.0
```
the tilt angle between laser propagation in y-direction and laser axis in x-direction (0 degree == no tilt) unit: degree

```cpp
constexpr float_64 PULSE_INIT = 20.0
```
The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH.
unit: none

```cpp
constexpr uint32_t initPlaneY = 0
```
cell from top where the laser is initialized

if initPlaneY == 0 than the absorber are disabled. if initPlaneY > absorbercells negative Y the negative absorber in y direction is enabled
valid ranges:
- initPlaneY == 0
- absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

`constexpr float_X LASER_PHASE = 0.0`

laser phase shift (no shift: 0.0)

\[ \sin(\omega \times \text{time} + \text{laser phase}) \] starts with phase=0 at center > E-field=0 at center

unit: rad, periodic in 2*pi

`constexpr PolarisationType Polarisation = CIRCULAR`

Polarization selection.

```cpp
struct WavepacketParam
```

**Public Types**

```cpp
enum PolarisationType

Available polarisation types.

Values:
- LINEAR_X = 1u
- LINEAR_Z = 2u
- CIRCULAR = 4u
```

**Public Static Attributes**

```cpp
constexpr float_64 WAVE_LENGTH_SI = 0.8e-6

unit: meter
```

```cpp
constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * picongpu::SI::ELECTRON_MASS_SI * picongpu::SI::SPEED_OF_LIGHT_SI * picongpu::SI::SPEED_OF_LIGHT_SI / picongpu::SI::ELECTRON_CHARGE_SI
```

Convert the normalized laser strength parameter a0 to Volt per meter.

```cpp
constexpr float_64 AMPLITUDE_SI = 1.738e13

unit: W / m^2

unit: none unit: Volt / meter unit: Volt / meter
```

```cpp
constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 7.0 * WAVE_LENGTH_SI / picongpu::SI::SPEED_OF_LIGHT_SI
```

The profile of the test Lasers 0 and 2 can be stretched by a constant area between the up and downramp unit: seconds.

```cpp
constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0

Pulse length: sigma of std. gauss for intensity (E^2) PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt{ 2* ln(2) } ] [ 2.354820045 ] Info: FWHM_of_Intensity = FWHM_Illumination = what a experimentalist calls "pulse duration"

unit: seconds (1 sigma)
```

```cpp
constexpr float_64 W0_X_SI = 4.246e-6

beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, at the focus position of the laser W0_SI = FWHM_of_Intensity / sqrt{ 2* ln(2) } [ 1.17741 ]

unit: meter
```

```cpp
constexpr float_64 W0_Z_SI = W0_X_SI
```

130 Chapter 2. Usage
constexpr float_64 PULSE_INIT = 20.0
The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH.

unit: none

constexpr uint32_t initPlaneY = 0
cell from top where the laser is initialized

if initPlaneY == 0 than the absorber are disabled. if initPlaneY > absorber cells negative Y the negative absorber in y direction is enabled

valid ranges:
• initPlaneY == 0
• absorber cells negative Y < initPlaneY < cells in y direction of the top gpu

constexpr float_X LASER_PHASE = 0.0
laser phase shift (no shift: 0.0)

sin(omega*time + laser_phase): starts with phase=0 at center > E-field=0 at center

unit: rad, periodic in 2*pi

constexpr PolarisationType Polarisation = LINEAR_X
Polarization selection.

namespace gaussianBeam

Functions

picongpu::fields::laserProfiles::gaussianBeam::PMACC_CONST_VECTOR(float_X, MODENUMBER + 1, LAGUERREMODES, 1.0)

Variables

constexpr uint32_t MODENUMBER = 0
Use only the 0th Laguerremode for a standard Gaussian.

List of available laser profiles.

Laser Profiles

Gaussian Beam

template<typename T_Params>
struct GaussianBeam : public picongpu::fields::laserProfiles::gaussianBeam::Unitless<T_Params>
Gaussian Beam laser profile with finite pulse length.

Template Parameters

• T_Params: class parameter to configure the Gaussian Beam profile, see members of gaussianBeam::default::GaussianBeamParam for required members

```cpp
// Use only the 0th Laguerre mode for a standard Gaussian
static constexpr uint32_t MODENUMBER = 0;
PMACC_CONST_VECTOR(float_X, MODENUMBER + 1, LAGUERREMODES, 1.0);
// This is just an example for a more complicated set of Laguerre modes
// constexpr uint32_t MODENUMBER = 12;
// PMACC_CONST_VECTOR(float_X, MODENUMBER + 1, LAGUERREMODES, -1.0, 0.0300519,
 // -0.319461, -0.23783, 0.0954839, 0.0318653, -0.144547, 0.0249208, -0.111989, 0.
 // -0.0434385, -0.030038, -0.00896321, -0.0160788);
struct GaussianBeamParam
```

(continues on next page)
/** unit: meter */
static constexpr float_64 WAVE_LENGTH_SI = 0.8e-6;

/** Convert the normalized laser strength parameter a0 to Volt per meter */
static constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * ::picongpu::SI::ELECTRON_MASS_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI * ::picongpu::SI::ELECTRON_CHARGE_SI;

/** unit: W / m^2 */
// calculate: _A0 = 8.549297e-6 * sqrt( Intensity[W/m^2] ) * wavelength[m], (linearly polarized)
/** unit: none */
//static constexpr float_64 _A0 = 1.5;
/** unit: Volt / meter */
//static constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;
/** unit: Volt / meter */
static constexpr float_64 AMPLITUDE_SI = 1.738e13;

/** Pulse length: sigma of std. gauss for intensity (E^2) */
* PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt( 2* ln(2) ) ]
* Info: FWHM_of_Intensity = FWHM_Illumination
* - what a experimentalist calls "pulse duration"
* unit: seconds (1 sigma) */
static constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0;

/** beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, */
* at the focus position of the laser
* W0_SI = FWHM_of_Intensity / sqrt( 2* ln(2) )
* Info: W0_SI = FWHM_illumination
* - what a experimentalist calls "beam waist"
* unit: meter */
static constexpr float_64 W0_SI = 5.0e-6 / 1.17741;

/** The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH */
* unit: none */
static constexpr float_64 PULSE_INIT = 20.0;

/** cell from top where the laser is initialized */
* if 'initPlaneY == 0' than the absorber are disabled.
* if 'initPlaneY > absorbercells negative Y' the negative absorber in y direction is enabled
* valid ranges:
* - initPlaneY == 0
* - absorber cells negative Y < initPlaneY < cells in y direction of the top gpu */

(continues on next page)
static constexpr uint32_t initPlaneY = 0;

/** laser phase shift (no shift: 0.0)
 * \[ \sin(\omega \cdot t + \text{laser phase}) \]: starts with phase=0 at center \( \rightarrow \) E-field=0 at center 
 * unit: rad, periodic in \( 2\pi \)
 */
static constexpr float_X LASER_PHASE = 0.0;

using LAGUERREMODES_t = defaults::LAGUERREMODES_t;
static constexpr uint32_t MODENUMBER = defaults::MODENUMBER;

/** Available polarisation types */
enum PolarisationType {
    LINEAR_X = 1u,
    LINEAR_Z = 2u,
    CIRCULAR = 4u,
};
/** Polarization selection */
static constexpr PolarisationType Polarisation = CIRCULAR;

---

Gaussian Beam with Pulse Front Tilt

```cpp
template<typename T_Params>
struct PulseFrontTilt : public picongpu::fields::laserProfiles::pulseFrontTilt::Unitless<T_Params>
{
    Gaussian Beam laser profile with titled pulse front.

    **Template Parameters**
    - T_Params: class parameter to configure the Gaussian Beam with pulse front tilt, see members of pulseFrontTilt::defaults::PulseFrontTiltParam for required members

    ```cpp
    struct PulseFrontTiltParam {
        /** unit: meter */
        static constexpr float_64 WAVE_LENGTH_SI = 0.8e-6;

        /** Convert the normalized laser strength parameter \( a_0 \) to Volt per meter */
        static constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * ::picongpu::SI::ELECTRON_MASS_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI / ::picongpu::SI::ELECTRON_CHARGE_SI;

        /** unit: W / m^2 */
        // calculate: \( A_0 = 8.549297e-6 \times \sqrt{\text{Intensity}[W/m^2]} \) \times \text{wavelength}[m] \( \rightarrow \) (linearly polarized)
        static constexpr float_64 _A0 = 1.5;

        /** unit: Volt / meter */
        //static constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;
    }
```
/** unit: Volt / meter */
static constexpr float_64 AMPLITUDE_SI = 1.738e13;

/** Pulse length: sigma of std. gauss for intensity (E^2) */
* PULSE_LENGTH_SI = FWHM_of_Intensity / \{ 2*sqrt( 2* ln(2) ) \}[
* \[ 2.354820045 \]
* Info: FWHM_of_Intensity = FWHM_Illumination
* unit: seconds (1 sigma) */
static constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0;

/** beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part, at the focus position of the laser */
* W0_SI = FWHM_of_Intensity / sqrt( 2* ln(2) )[
* \[ 1.17741 \]
* unit: meter */
static constexpr float_64 W0_SI = 5.0e-6 / 1.17741;

/** the distance to the laser focus in y-direction */
* unit: meter */
static constexpr float_64 FOCUS_POS_SI = 4.62e-5;

/** the tilt angle between laser propagation in y-direction and laser axis */
in x-direction (0 degree -- no tilt)
* unit: degree */
static constexpr float_64 TILT_X_SI = 0.0;

/** The laser pulse will be initialized PULSE_INIT times of the PULSE_LENGTH */
* unit: none */
static constexpr float_64 PULSE_INIT = 20.0;

/** cell from top where the laser is initialized */
* if `initPlaneY == 0` than the absorber are disabled.
* if `initPlaneY > absorbercells negative Y` the negative absorber in y direction is enabled
* valid ranges:
* - `initPlaneY == 0`
* - absorber cells negative Y < initPlaneY < cells in y direction of the top gpu
*/
static constexpr uint32_t initPlaneY = 0;

/** laser phase shift (no shift: 0.0) */
* sin(omega*time + laser_phase): starts with phase=0 at center --> E-field=0 at center
* unit: rad, periodic in 2*pi */
static constexpr float_X LASER_PHASE = 0.0;

//! Available polarisation types
enum PolarisationType
{ 
  LINEAR_X = 1u,
  LINEAR_Z = 2u,
  CIRCULAR = 4u,
};

/** Polarization selection */
static constexpr PolarisationType Polarisation = LINEAR_X;
};

Wavepacket

template<typename T_Params>
struct Wavepacket : public picongpu::fields::laserProfiles::wavepacket::Unitless<T_Params>
Wavepacket with Gaussian spatial and temporal envelope.

Template Parameters

* T_Params: class parameter to configure the Wavepacket profile, see members of wavepacket::defaults::WavepacketParam for required members

struct WavepacketParam
{
  /** unit: meter */
  static constexpr float_64 WAVE_LENGTH_SI = 0.8e-6;

  /** Convert the normalized laser strength parameter a0 to Volt per meter */
  static constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * ::picongpu::SI::ELECTRON_MASS_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI / ::picongpu::SI::ELECTRON_CHARGE_SI;

  /** unit: W / m^2 */
  // calculate: _A0 = 8.549297e-6 * sqrt( Intensity[W/m^2] ) * wavelength[m]
  /** (linearly polarized) */

  /** unit: none */
  //static constexpr float_64 _A0 = 1.5;

  /** unit: Volt / meter */
  //static constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;

  /** unit: Volt / meter */
  static constexpr float_64 AMPLITUDE_SI = 1.738e13;

  /** unit: seconds */
  static constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 7.0 * WAVE_LENGTH_SI / ::picongpu::SI::SPEED_OF_LIGHT_SI;

  /** Pulse length: sigma of std. gauss for intensity (E^2) */
  static constexpr float_64 PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2*sqrt( 2* ln(2) ) ]
  // Info: FWHM_of_Intensity = FWHM_Illumination
  // = what a experimentalist calls "pulse duration"

  /** unit: seconds */
};

(continues on next page)
static constexpr float_64 PULSE_LENGTH_SI = 10.615e-15 / 4.0;

/** beam waist: distance from the axis where the pulse intensity (E\(^2\))
 * decreases to its 1/e\(^2\)-th part,
 * at the focus position of the laser
 * \( W_0\_SI = \text{FWHM\_of\_Intensity} / \sqrt{2\cdot \ln(2)} \)
 * \[ 1.17741 \]
 * unit: meter */
static constexpr float_64 W0_X_SI = 4.246e-6;
static constexpr float_64 W0_Z_SI = W0_X_SI;

/** The laser pulse will be initialized PULSE\_INIT times of the PULSE\_LENGTH
 * unit: none */
static constexpr float_64 PULSE_INIT = 20.0;

/** cell from top where the laser is initialized
 * if `initPlaneY == 0` than the absorber are disabled.
 * if `initPlaneY > absorber\_cells negative Y` the negative absorber in y
 * direction is enabled
 * valid ranges:
 * - initPlaneY == 0
 * - absorber cells negative Y < initPlaneY < cells in y direction of the top gpu
 */
static constexpr uint32_t initPlaneY = 0;

/** laser phase shift (no shift: 0.0)
 * sin(omega\_time + laser\_phase): starts with phase=0 at center --\( \rightarrow \) E-field=0 at center
 * unit: rad, periodic in 2*pi */
static constexpr float_X LASER_PHASE = 0.0;

/** Available polarisation types */
enum PolarisationType
{
    LINEAR_X = 1u,
    LINEAR_Z = 2u,
    CIRCULAR = 4u,
};

/** Polarization selection */
static constexpr PolarisationType Polarisation = LINEAR_X;

Wavepacket with Exponential Ramp and Prepulse

template<typename T_Params>
struct ExpRampWithPrepulse : public picongpu::fields::laserProfiles::expRampWithPrepulse::Unitless<T_Params>

Wavepacket with spatial Gaussian envelope and adjustable temporal shape.
Allows defining a prepulse and two regions of exponential preramp with independent slopes. The definition works by specifying three (t, intensity)-points, where time is counted from the very beginning in SI and the intensity (yes, intensity, not amplitude) is given in multiples of the main peak.

Be careful - problematic for few cycle pulses. Though the rest is cloned from laserWavepacket, the correctionFactor is not included (this made a correction to the laser phase, which is necessary for very short pulses, since otherwise a test particle is, after the laser pulse has passed, not returned to immobility, as it should). Since the analytical solution is only implemented for the Gaussian regime, and we have mostly exponential regimes here, it was not retained here.

A Gaussian peak (optionally lengthened by a plateau) is preceded by two pieces of exponential preramps, defined by 3 (time, intensity)-points.

The first two points get connected by an exponential, the 2nd and 3rd point are connected by another exponential, which is then extrapolated to the peak. The Gaussian is added everywhere, but typically contributes significantly only near the peak. It is advisable to set the third point far enough from the plateau (approx 3*FWHM), then the contribution from the Gaussian is negligible there, and the intensity can be set as measured from the laser profile.

Optionally a Gaussian prepulse can be added, given by the parameters of the relative intensity and time point. The time of the prepulse and the three preramp points are given in SI, the intensities are given as multiples of the peak intensity.

Template Parameters

- T_Params: class parameter to configure the Gaussian Beam profile, see members of expRampWithPrepulse::defaults::ExpRampWithPrepulseParam for required members

```cpp
struct ExpRampWithPrepulseParam
{
    // Intensities of prepulse and exponential preramp
    static constexpr float INT_RATIO_PREPULSE = 0.;
    static constexpr float INT_RATIO_POINT_1 = 1.e-8;
    static constexpr float INT_RATIO_POINT_2 = 1.e-4;
    static constexpr float INT_RATIO_POINT_3 = 1.e-4;

    // time-positions of prepulse and preramps points
    static constexpr float TIME_PREPULSE_SI = -950.0e-15;
    static constexpr float TIME_PEAKPULSE_SI = 0.0e-15;
    static constexpr float TIME_POINT_1_SI = -1000.0e-15;
    static constexpr float TIME_POINT_2_SI = -300.0e-15;
    static constexpr float TIME_POINT_3_SI = -100.0e-15;

    /** unit: meter */
    static constexpr float WAVE_LENGTH_SI = 0.8e-6;

    /** UNITCONV */
    static constexpr float UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * ::picongpu::SI::ELECTRON_MASS_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI / ::picongpu::SI::ELECTRON_CHARGE_SI;

    /** unit: W / m^2 */
    // calculate: _A0 = 8.549297e-6 * sqrt( Intensity[W/m^2] ) * wavelength[m] (linearly polarized)
    static constexpr float _A0 = 20.;

    /** unit: Volt /meter */
    static constexpr float AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;
}
```

(continues on next page)
//constexpr float_64 AMPLITUDE_SI = 1.738e13;

/** Stretch temporal profile by a constant plateau between the up and downramp *
 * unit: seconds */
static constexpr float_64 LASER_NOFOCUS_CONSTANT_SI = 0.0 * WAVE_LENGTH_SI / ::picongpu::SI::SPEED_OF_LIGHT_SI;

/** Pulse length: sigma of std. gauss for intensity (E^2) *
 * PULSE_LENGTH_SI = FWHM_of_Intensity / [ 2 * sqrt( 2 * ln(2) ) ]
 * Info: FWHM_of_Intensity = FWHM_Illumination
 * unit: seconds (1 sigma) */
static constexpr float_64 PULSE_LENGTH_SI = 3.0e-14 / 2.35482; // half of the time in which E falls to half its initial value (then I falls to half its value in 15fs, approx 6 wavelengths). Those are 4.8 wavelengths.

/** beam waist: distance from the axis where the pulse intensity (E^2) decreases to its 1/e^2-th part,
 * W0_X_SI is this distance in x-direction
 * W0_Z_SI is this distance in z-direction
 * if both values are equal, the laser has a circular shape in x-z
 * W0_SI = FWHM_of_Intensity / sqrt{ 2* ln(2) }
 * unit: meter */
static constexpr float_64 W0_X_SI = 2.5 * WAVE_LENGTH_SI;
static constexpr float_64 W0_Z_SI = W0_X_SI;

/** The laser pulse will be initialized half of PULSE_INIT times of the PULSE_LENGTH before plateau
 * and half at the end of the plateau
 * unit: none */
static constexpr float_64 RAMP_INIT = 16.0;

/** cell from top where the laser is initialized
 * if `initPlaneY == 0` than the absorber are disabled.
 * if `initPlaneY > absorbercells negative Y` the negative absorber in y direction is enabled
 * valid ranges:
 * - initPlaneY == 0
 * - absorber cells negative Y < initPlaneY < cells in y direction of the top gpu
 */
static constexpr uint32_t initPlaneY = 0;

/** laser phase shift (no shift: 0.0)
 * sin(omega*time + laser_phase): starts with phase=0 at center --> E-field=0 at center
 * unit: rad, periodic in 2*pi */
static constexpr float X LASER_PHASE = 0.0;

/** Available polarisation types */
enum PolarisationType
Wavepacket with Polynomial Profile

template<typename T_Params>
struct Polynom : public picongpu::fields::laserProfiles::polynom::Unitless<T_Params>

Wavepacket with a polynomial temporal intensity shape.

Based on a wavepacket with Gaussian spatial envelope.

Template Parameters

• T_Params: class parameter to configure the polynomial laser profile, see members of polynom::defaults::PolynomParam for required members

```
struct PolynomParam
{
    /** unit: meter */
    static constexpr float_64 WAVE_LENGTH_SI = 0.8e-6;

    /** Convert the normalized laser strength parameter a0 to Volt per meter */
    static constexpr float_64 UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI * ::picongpu::SI::ELECTRON_MASS_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI * ::picongpu::SI::SPEED_OF_LIGHT_SI / ::picongpu::SI::ELECTRON_CHARGE_SI;

    /** unit: W / m^2 */
    // calculate: \( A_0 = 8.549297e-6 \times \sqrt{ \text{Intensity}[\text{W/m}^2] } \times \text{wavelength}[\text{m}] \) (linearly polarized)
    static constexpr float_64 _A0 = 8.549297e-6 * sqrt( Intensity[W/m^2] ) * wavelength[m];

    /** unit: Volt / meter */
    static constexpr float_64 AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;

    /** Pulse length: sigma of std. gauss for intensity (E^2) */
    static constexpr float_64 FWHM_of_Intensity = 2.354820045;

    /** unit: seconds (1 sigma) */
    static constexpr float_64 PULSE_LENGTH_SI = 4.0e-15;

    /** beam waist: distance from the axis where the pulse intensity (E^2) decreases to its \(1/e^2\)-th part,
```
* at the focus position of the laser
* unit: meter
*/
static constexpr float_64 W0_X_SI = 4.246e-6; // waist in x-direction
static constexpr float_64 W0_Z_SI = W0_X_SI; // waist in z-direction

/** cell from top where the laser is initialized
* if `initPlaneY == 0` than the absorber are disabled.
* if `initPlaneY > absorbercells negative Y` the negative absorber in y
* direction is enabled
* valid ranges:
* - `initPlaneY == 0`
* - `absorber cells negative Y < initPlaneY < cells in y direction of
  the top gpu`
*/
static constexpr uint32_t initPlaneY = 0;

/** laser phase shift (no shift: 0.0)
* `sin(omega*time + laser_phase): starts with phase=0 at center --> E-
* field=0 at center`
* unit: rad, periodic in 2*pi
*/
static constexpr float_X LASER_PHASE = 0.0;

/** Available polarization types
*/
enum PolarisationType
{
  LINEAR_X = 1u,
  LINEAR_Z = 2u,
  CIRCULAR = 4u,
};

/** Polarization selection
*/
static constexpr PolarisationType Polarisation = LINEAR_X;

/** Wavepacket with a polynomial temporal intensity shape.
* Based on a wavepacket with Gaussian spatial envelope.
* @tparam T_Params class parameter to configure the polynomial laser profile,
* see members of polynom::defaults::PolynomParam for
* required members
*/

Plane Wave

```cpp
template<typename T_Params>
struct PlaneWave : public picongpu::fields::laserProfiles::planeWave::Unitless<T_Params>
{
  Plane wave laser profile.

  Defines a plane wave with temporally Gaussian envelope.

  Template Parameters
```
• T_Params: class parameter to configure the plane wave profile, see members of planeWave::defaults::PlaneWaveParam for required members

```cpp
struct PlaneWaveParam {
    // unit: meter
    static constexpr float WAVE_LENGTH_SI = 0.8e-6;

    // Convert the normalized laser strength parameter a0 to Volt per meter
    static constexpr float UNITCONV_A0_to_Amplitude_SI = -2.0 * PI / WAVE_LENGTH_SI;

    // calculate: _A0 = 8.549297e-6 * sqrt( Intensity[W/m^2] ) * wavelength[m]
    // (linearly polarized)
    static constexpr float _A0 = 1.5;

    // unit: Volt / meter
    static constexpr float AMPLITUDE_SI = _A0 * UNITCONV_A0_to_Amplitude_SI;

    // Pulse length: sigma of std. gauss for intensity (E^2)
    static constexpr float PULSE_LENGTH_SI = FWHM_of_Intensity / \left[ 2 \cdot \text{sqrt}(2 \cdot \text{ln}(2)) \right]
    static constexpr float FWHM_of_Intensity = FWHM_Illumination;

    // unit: seconds (1 sigma)
    static constexpr float LASER_NOFOCUS_CONSTANT_SI = 13.34e-15;

    // cell from top where the laser is initialized
    static constexpr uint32_t initPlaneY = 0;

    // The laser pulse will be initialized half of PULSE_INIT times of the PULSE_LENGTH before and after the plateau
    static constexpr float RAMP_INIT = 20.6146;

    // Laser phase shift (no shift: 0.0)
    sin(omega*time + laser_phase): starts with phase=0 at center -> E-field=0 at center
```
(continues on next page)
* unit: rad, periodic in 2*π
 */
static constexpr float X LASER_PHASE = 0.0;
/** Available polarization types 
 */
enum PolarisationType
{
    LINEAR_X = 1u,
    LINEAR_Z = 2u,
    CIRCULAR = 4u,
};
/** Polarization selection 
 */
static constexpr PolarisationType Polarisation = LINEAR_X;
}; // namespace defaults

None

template<typename T_Params>
struct None : public picongpu::fields::laserProfiles::none::Unitless<T_Params>
{
    Empty laser profile.

    Does not define a laser profile but provides some hard-coded constants that are accessed directly in some
    places.

    **Template Parameters**

    - T_Params: class parameter to configure the “no laser” profile, see members of
        none::defaults::NoneParam for required members

pml.param

Configure the perfectly matched layer (PML).

To enable PML use YeePML field solver.

namespace picongpu

    namespace fields

        namespace maxwellSolver

            namespace yeePML

                Variables

                constexpr uint32_t THICKNESS = 8

                constexpr uint32_t picongpu::fields::maxwellSolver::yeePML::NUM CELLS[3][2] = {
                    { THICKNESS, THICKNESS },
                    { THICKNESS, THICKNESS },
                    { THICKNESS, THICKNESS }
                };

                Thickness of the absorbing layer, in number of cells.

                PML is located inside the global simulation area, near the outer borders. Setting size
to 0 results in disabling absorption at the corresponding boundary. Normally thickness is
between 6 and 16 cells, with larger values providing less reflections. 8 cells should be good
enough for most simulations. There are no requirements on thickness being a multiple of
the supercell size. It is only required that PML is small enough to be fully contained in a single layer of local domains near the global simulation area boundary (Note that the domains of this layer might be changing, e.g. due to moving window.) Unit: number of cells.

**constexpr** float\_64 SIGMA\_KAPPA\_GRADING\_ORDER = 4.0
Order of polynomial grading for artificial electric conductivity and stretching coefficient.

The conductivity (sigma) is polynomially scaling from 0 at the internal border of PML to the maximum value (defined below) at the external border. The stretching coefficient (kappa) scales from 1 to the corresponding maximum value (defined below) with the same polynomial. The grading is given in [Taflove, Hagness], eq. (7.60a, b), with the order denoted ‘m’. Must be \(\geq 0\). Normally between 3 and 4, not required to be integer. Unitless.

**constexpr** float\_64 SIGMA\_OPT\_SI[3] = \{0.8 * (SIGMA\_KAPPA\_GRADING\_ORDER + 1.0) / (SI::Z0\_SI * SI::CELL\_WIDTH\_SI), , \}

**constexpr** float\_64 SIGMA\_MAX\_SI[3] = \{SIGMA\_OPT\_SI[0] * SIGMA\_OPT\_MULTIPLIER, , \}
Max value of artificial electric conductivity in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Grading is described in comments for SIGMA\_KAPPA\_GRADING\_ORDER. Too small values lead to significant reflections from the external border, too large - to reflections due to discretization errors. Artificial magnetic permeability will be chosen to perfectly match this. Must be \(\geq 0\). Normally between 0.7 * SIGMA\_OPT\_SI and 1.1 * SIGMA\_OPT\_SI. Unit: siemens / m.

**constexpr** float\_64 KAPPA\_MAX[3] = \{1.0, , \}
Max value of coordinate stretching coefficient in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Grading is described in comments for SIGMA\_KAPPA\_GRADING\_ORDER. Must be \(\geq 1\). For relatively homogeneous domains 1.0 is a reasonable value. Highly elongated domains can have better absorption with values between 7.0 and 20.0, for example, see section 7.11.2 in [Taflove, Hagness]. Unitless.

**constexpr** float\_64 ALPHA\_GRADING\_ORDER = 1.0
Order of polynomial grading for complex frequency shift.

The complex frequency shift (alpha) is polynomially downscaling from the maximum value (defined below) at the internal border of PML to 0 at the external border. The grading is given in [Taflove, Hagness], eq. (7.79), with the order denoted ‘m\_a’. Must be \(\geq 0\). Normally values are around 1.0. Unitless.

**constexpr** float\_64 ALPHA\_MAX\_SI[3] = \{0.2, , \}
Complex frequency shift in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Setting it to 0 will make PML behave as uniaxial PML. Setting it to a positive value helps to attenuate evanescent modes, but can degrade absorption of propagating modes, as described in section 7.7 and 7.11.3 in [Taflove, Hagness]. Must be \(\geq 0\). Normally values are 0 or between 0.15 and 0.3. Unit: siemens / m.

**pusher.param**
Configure particle pushers.
Those pushers can then be selected by a particle species in species.param and speciesDefinition.param

**namespace picongpu**

2.3. .param Files 143
struct particlePusherAccelerationParam
   Subclassed by picongpu::particlePusherAcceleration::UnitlessParam

Public Static Attributes

constexpr float_64 AMPLITUDEx_SI = 0.0
   Define strength of constant and homogeneous accelerating electric field in SI per dimension.
   unit: V olt / meter

constexpr float_64 AMPLITUDEy_SI = -1.e11
   The moving window propagation direction unit: Volt / meter (1e11 V/m = 1 GV/cm)

constexpr float_64 AMPLITUDEz_SI = 0.0
   unit: Volt / meter

constexpr float_64 ACCELERATION_TIME_SI = 10000.0 * picongpu::SI::DELTA_T_SI
   Acceleration duration unit: second.

namespace particlePusherAxel

Enums

enum TrajectoryInterpolationType
   Values:
   LINEAR = 1u
   NONLINEAR = 2u

Variables

constexpr TrajectoryInterpolationType TrajectoryInterpolation = LINEAR

namespace particlePusherProbe

Typedefs

using ActualPusher = void
   Also push the probe particles?

   In many cases, probe particles are static throughout the simulation. This option allows to set an
   “actual” pusher that shall be used to also change the probe particle positions.

   Examples:
   • particles::pusher::Boris
   • particles::pusher::[all others from above]
   • void (no push)

density.param

Configure existing or define new normalized density profiles here.

During particle species creation in speciesInitialization.param, those profiles can be translated to spatial particle
distributions.

namespace picongpu

   namespace densityProfiles
Typedefs

using Gaussian = GaussianImpl<GaussianParam>
using Homogenous = HomogenousImpl
using LinearExponential = LinearExponentialImpl<LinearExponentialParam>
using GaussianCloud = GaussianCloudImpl<GaussianCloudParam>
using SphereFlanks = SphereFlanksImpl<SphereFlanksParam>
using FromHDF5 = FromHDF5Impl<FromHDF5Param>
using FreeFormula = FreeFormulaImpl<FreeFormulaFunctor>

Functions

picongpu::densityProfiles::PMACC_STRUCT(GaussianParam, ( PMACC_C_VALUE (float_X, gasFactor, -1.0))( PMACC_C_VALUE ... 4.62e-5))(PMACC_C_VALUE(float_64, gasSigmaLeft_SI, 4.62e-5))(PMACC_C_VALUE(float_64, gasSigmaRight_SI, 4.62e-5)))

Profile Formula:
const float_X exponent = abs((y - gasCenter_SI) / gasSigma_SI); const float_X density = exp(gasFactor * pow(exponent, gasPower));
takes gasCenterLeft_SI for y < gasCenterLeft_SI, gasCenterRight_SI for y > gasCenterRight_SI, and exponent = 0.0 for gasCenterLeft_SI < y < gasCenterRight_SI

picongpu::densityProfiles::PMACC_STRUCT(LinearExponentialParam, ( PMACC_C_VALUE (uint32_t, vacuumCellsY, 50))( ... gasA_SI, 1.0e-3))(PMACC_C_VALUE(float_64, gasD_SI, 1.0e-3))(PMACC_C_VALUE(float_64, gasB, 0.0)))

parameter for LinearExponential profile

* Density Profile: /\   *
  * linear / \_,._ exponential
  * slope / \_,._ slope
  *
  * MAX

picongpu::densityProfiles::PMACC_STRUCT(GaussianCloudParam, ( PMACC_C_VALUE (float_X, gasFactor, -0.5))( PMACC_C_VALUE ... center_SI, 1.134e-5, 1.134e-5, 1.134e-5))(PMACC_C_VECTOR_DIM(float_64, simDim, sigma_SI, 7.0e-6, 7.0e-6, 7.0e-6)))

picongpu::densityProfiles::PMACC_STRUCT(SphereFlanksParam, ( PMACC_C_VALUE (uint32_t, vacuumCellsY, 50))( PMACC_C_VALUE ... simDim, center_SI, 8.0e-3, 8.0e-3, 8.0e-3))(PMACC_C_VALUE(float_64, exponent_SI, 1.0e3)))
The profile consists out of the composition of 3 1D profiles with the scheme: exponential increasing flank, constant sphere, exponential decreasing flank.

picongpu::densityProfiles::PMACC_STRUCT(FromHDF5Param, ( PMACC_C_STRING (filename, "gas"))(PMACC_C_STRING(datasetName, "fields/e_chargeDensity"))(PMACC_C_VALUE(uint32_t, iteration, 0))( PMACC_C_VALUE (float_X, defaultDensity, 0.0)))

struct FreeFormulaFunctor

Public Functions

HDINLINE float_X picongpu::densityProfiles::FreeFormulaFunctor::operator()(const floatD_64 & position_SI, const float3_64 & cellSize_SI)

This formula uses SI quantities only.
The profile will be multiplied by BASE_DENSITY_SI.

Return float_X density [normalized to 1.0]

Parameters
• position_SI: total offset including all slides [meter]
• cellSize_SI: cell sizes [meter]

namespace SI

Variables

cconstexpr float_64 BASE_DENSITY_SI = 1.e25
    Base density in particles per m^3 in the density profiles.

    This is often taken as reference maximum density in normalized profiles. Individual particle
    species can define a densityRatio flag relative to this value.

    unit: ELEMENTS/m^3

SpeciesAttributes.param

This file defines available attributes that can be stored with each particle of a particle species.

Each attribute defined here needs to implement furthermore the traits

• Unit
• UnitDimension
• WeightingPower
• MacroWeighted in speciesAttributes.unitless for further information about these traits see therein.

namespace picongpu

Functions

alias (position)
    relative (to cell origin) in-cell position of a particle

    With this definition we do not define any type like float3_X, float3_64, ... This is only a name without
    a specialization.

value_identifier (uint64_t, particleId, IdProvider<simDim>::getNewId)
    unique identifier for a particle

picongpu::value_identifier(floatD_X, position_pic, floatD_X::create (0.))
    specialization for the relative in-cell position

picongpu::value_identifier(float3_X, momentum, float3_X::create (0.))
    momentum at timestep t

picongpu::value_identifier(float3_X, momentumPrev1, float3_X::create (0._X))
    momentum at (previous) timestep t-1

picongpu::value_identifier(float_X, weighting, 0._X)
    weighting of the macro particle

picongpu::value_identifier(int16_t, voronoiCellId, -1)
    Voronoi cell of the macro particle.

picongpu::value_identifier(float3_X, probeE, float3_X::create (0.))
    interpolated electric field with respect to particle shape

picongpu::value_identifier(float3_X, probeB, float3_X::create (0.))
    interpolated electric field with respect to particle shape
piCongpu::value_identifier(bool, radiationMask, false)
masking a particle for radiation
The mask is used by the user defined filter RadiationParticleFilter in radiation.param to (de)select particles for the radiation calculation.

piCongpu::value_identifier(bool, transitionRadiationMask, false)
masking a particle for transition radiation
The mask is used by the user defined filter TransitionRadiationParticleFilter in transitionRadiation.param to (de)select particles for the transition radiation calculation.

piCongpu::value_identifier(float_X, boundElectrons, 0. _X)
number of electrons bound to the atom / ion
value type is float_X to avoid casts during the runtime
- float_X instead of integer types are reasonable because effective charge numbers are possible
- required for ion species if ionization is enabled
- setting it requires atomicNumbers to also be set

piCongpu::value_identifier(flylite::Superconfig, superconfig, flylite::Superconfig::create (0.))
atomic superconfiguration
atomic configuration of an ion for collisional-radiative modeling, see also flylite.param

value_identifier (DataSpace<simDim>, totalCellIdx, DataSpace<simDim>)
Total cell index of a particle.
The total cell index is a N-dimensional DataSpace given by a GPU’s globalDomain.offset + localDomain.offset added to the N-dimensional cell index the particle belongs to on that GPU.

alias (shape)
alias for particle shape, see also species.param

alias (particlePusher)
alias for particle pusher, see also species.param

alias (ionizers)
alias for particle ionizers, see also ionizer.param

alias (ionizationEnergies)
alias for ionization energy container, see also ionizationEnergies.param

alias (synchrotronPhotons)
alias for synchrotronPhotons, see also speciesDefinition.param
alias for ion species used for bremsstrahlung

alias (bremssstrahlungPhotons)
alias for photon species used for bremsstrahlung

alias (interpolation)
alias for particle to field interpolation, see also species.param

alias (current)
alias for particle current solver, see also species.param

alias (atomicNumbers)
alias for particle flag: atomic numbers, see also ionizer.param
- only reasonable for atoms / ions / nuclei
- is required when boundElectrons is set

alias (effectiveNuclearCharge)
alias for particle flag: effective nuclear charge,
• see also ionizer.param
• only reasonable for atoms / ions / nuclei

**alias** (populationKinetics)
alias for particle population kinetics model (e.g. FLYlite)
see also flylite.param

**alias** (massRatio)
alias for particle mass ratio
mass ratio between base particle, see also speciesConstants.param SI::BASE_MASS_SI and a user defined species
default value: 1.0 if unset

**alias** (chargeRatio)
alias for particle charge ratio
charge ratio between base particle, see also speciesConstants.param SI::BASE_CHARGE_SI and a user defined species
default value: 1.0 if unset

**alias** (densityRatio)
alias for particle density ratio
density ratio between default density, see also density.param SI::BASE_DENSITY_SI and a user defined species
default value: 1.0 if unset

**alias** (exchangeMemCfg)
alias to reserved bytes for each communication direction
This is an optional flag and overwrites the default species configuration in memory.param.
A memory config must be of the following form:

```c
struct ExampleExchangeMemCfg
{
    static constexpr uint32_t BYTES_EXCHANGE_X = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Y = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Z = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_CORNER = 16 * 1024;
    static constexpr uint32_t BYTES_EDGES = 16 * 1024;
};
```

**alias** (boundaryCondition)
alias to specify the boundary condition for particles
The default behavior if this alias is not given to a species is that the particles which leave the global simulation box where deleted. This also notifies all plugins that can handle leaving particles.

Note: alias boundaryCondition will be ignored if the runtime parameter --periodic is set.
The following species attributes are defined by PMacc and always stored with a particle:

```
namespace pmacc

Functions

pmacc::value_identifier(lcellId_t, localCellIdx, 0)
cell of a particle inside a supercell
```
Value is a linear cell index inside the supercell

\texttt{pmacc::value\_identifier(uint8\_t, multiMask, 0)}

state of a particle

Particle might be valid or invalid in a particle frame. Valid particles can further be marked as candidates to leave a supercell. Possible multiMask values are:

- 0 (zero): no particle (invalid)
- 1: particle (valid)
- 2 to 27: (valid) particle that is about to leave its supercell but is still stored in the current particle frame. Directions to leave the supercell are defined as follows. An ExchangeType = value - 1 (e.g. 27 - 1 = 26) means particle leaves supercell in the direction of FRONT(value=18) && TOP(value=6) && LEFT(value=2) which defines a diagonal movement over a supercell corner (18+6+2=26).

\texttt{speciesConstants.param}

Constants and thresholds for particle species.

Defines the reference mass and reference charge to express species with (default: electrons with negative charge).

\texttt{namespace picongpu}

\texttt{Variables}

\texttt{constexpr float\_X picongpu::GAMMA\_THRESH = 1.005\_X}

Threshold between relativistic and non-relativistic regime.

Threshold used for calculations that want to separate between high-precision formulas for relativistic and non-relativistic use-cases, e.g. energy-binning algorithms.

\texttt{constexpr float\_X picongpu::GAMMA\_INV\_SQUARE\_RAD\_THRESH = 0.18\_X}

Threshold in radiation plugin between relativistic and non-relativistic regime.

This limit is used to decide between a pure 1-sqrt(1-x) calculation and a 5th order Taylor approximation of 1-sqrt(1-x) to avoid halving of significant digits due to the sqrt() evaluation at x = 1/gamma^2 near 0.0. With 0.18 the relative error between Taylor approximation and real value will be below 0.001\% = 1e-5 for x=1/gamma^2 < 0.18

\texttt{namespace SI}

\texttt{Variables}

\texttt{constexpr float\_64 BASE\_MASS\_SI = ELECTRON\_MASS\_SI}

base particle mass

reference for massRatio in speciesDefinition.param

unit: kg

\texttt{constexpr float\_64 BASE\_CHARGE\_SI = ELECTRON\_CHARGE\_SI}

base particle charge

reference for chargeRatio in speciesDefinition.param

unit: C
species.param

Forward declarations for speciesDefinition.param in case one wants to use the same particle shape, interpolation, current solver and particle pusher for all particle species.

namespace picongpu

**Typedefs**

**using UsedParticleShape** = particles::shapes::TSC

Particle Shape definitions.

- particles::shapes::CIC : 1st order
- particles::shapes::TSC : 2nd order
- particles::shapes::PCS : 3rd order
- particles::shapes::P4S : 4th order

example: using UsedParticleShape = particles::shapes::CIC;

**using UsedField2Particle** = FieldToParticleInterpolation<UsedParticleShape, AssignedTrilinearInterpolation>

define which interpolation method is used to interpolate fields to particles

**using UsedParticleCurrentSolver** = currentSolver::Esirkepov<UsedParticleShape>

select current solver method

- currentSolver::Esirkepov< SHAPE > : particle shapes - CIC, TSC, PCS, P4S (1st to 4th order)
- currentSolver::VillaBune<> : particle shapes - CIC (1st order) only
- currentSolver::EmZ< SHAPE > : particle shapes - CIC, TSC, PCS, P4S (1st to 4th order)

For development purposes:

- currentSolver::currentSolver::EsirkepovNative< SHAPE > : generic version of currentSolverEsirkepov without optimization (~4x slower and needs more shared memory)

**using UsedParticlePusher** = particles::pusher::Boris

particle pusher configuration

Defining a pusher is optional for particles

- particles::pusher::Vay : better suited relativistic boris pusher
- particles::pusher::Boris : standard boris pusher
- particles::pusher::ReducedLandauLifshitz : 4th order RungeKutta pusher with classical radiation reaction

For diagnostics & modeling:

- particles::pusher::Acceleration : Accelerate particles by applying a constant electric field
- particles::pusher::Free : free propagation, ignore fields (= free stream model)
- particles::pusher::Photon : propagate with c in direction of normalized mom.
- particles::pusher::Probe : Probe particles that interpolate E & B For development purposes:

- particles::pusher::Axel : a pusher developed at HZDR during 2011 (testing)
speciesDefinition.param

Define particle species.

This file collects all previous declarations of base (reference) quantities and configured solvers for species and defines particle species. This includes “attributes” (values to store with each species) and “flags” (values & aliases for solvers to perform with the species for each timestep and ratios to base quantities). With those information, a Particles class is defined for each species and then collected in the list VectorAllSpecies.

namespace picongpu

Typedefs

using DefaultParticleAttributes = MakeSeq_t<position<position_pic>, momentum, weighting>
describe attributes of a particle

using ParticleFlagsPhotons = MakeSeq_t<particlePusher<particles::pusher::Photon>, shape<UsedParticleShape>
using PIC_Photons = Particles<PMACC_CSTRING("ph"), ParticleFlagsPhotons, DefaultParticleAttributes>

using ParticleFlagsElectrons = MakeSeq_t<particlePusher<UsedParticlePusher>, shape<UsedParticleShape>,
using PIC_Electrons = Particles<PMACC_CSTRING("e"), ParticleFlagsElectrons, DefaultParticleAttributes>

using ParticleFlagsIons = MakeSeq_t<particlePusher<UsedParticlePusher>, shape<UsedParticleShape>,
using PIC_Ions = Particles<PMACC_CSTRING("i"), ParticleFlagsIons, DefaultParticleAttributes>
using VectorAllSpecies = MakeSeq_t<PIC_Electrons, PIC_Ions>

All known particle species of the simulation.

List all defined particle species from above in this list to make them available to the PIC algorithm.

Functions

picongpu::value_identifier(float_X, MassRatioPhotons, 0. 0)
picongpu::value_identifier(float_X, ChargeRatioPhotons, 0. 0)
picongpu::value_identifier(float_X, MassRatioElectrons, 1. 0)
picongpu::value_identifier(float_X, ChargeRatioElectrons, 1. 0)
picongpu::value_identifier(float_X, MassRatioIons, 1836. 152672)
picongpu::value_identifier(float_X, ChargeRatioIons, -1. 0)
picongpu::value_identifier(float_X, DensityRatioIons, 1. 0)

particle.param

Configurations for particle manipulators.

Set up and declare functors that can be used in speciesInitialization.param for particle species initialization and manipulation, such as temperature distributions, drifts, pre-ionization and in-cell position.

namespace picongpu

namespace particles
Variables

```cpp
constexpr float_X MIN_WEIGHTING = 10.0
```
A particle with a weighting below MIN_WEIGHTING will not be created / will be deleted

```cpp
constexpr uint32_t TYPICAL_PARTICLES_PER_CELL = 2u
```
Number of maximum particles per cell during density profile evaluation.

Determines the weighting of a macro particle and with it, the number of particles “sampling”
dynamics in phase space.

namespace manipulators

Typedefs

```cpp
using AssignXDrift = unary::Drift<DriftParam, nvidia::functors::Assign>
definition of manipulator that assigns a drift in X
```

```cpp
using AddTemperature = unary::Temperature<TemperatureParam>
```

```cpp
using DoubleWeighting = generic::Free<DoubleWeightingFunctor>
definition of a free particle manipulator: double weighting
```

```cpp
using RandomEnabledRadiation = generic::FreeRng<RandomEnabledRadiationFunctor, pmacc::random::distributions::Uniform<float_X>>
```

```cpp
using RandomPosition = unary::RandomPosition
```
changes the in-cell position of each particle of a species

Functions

```cpp
picongpu::particles::manipulators::CONST_VECTOR(float_X, 3, DriftParam_direction, 1.0, 0.0, 0.0)
```
Parameter for DriftParam.

```cpp
struct DoubleWeightingFunctor
Unary particle manipulator: double each weighting.
```

Public Functions

```cpp
template<typename T_Particle>INLINE void picongpu::particles::manipulators::
```

```cpp
struct DriftParam
Parameter for a particle drift assignment.
```

Public Members

```cpp
const DriftParam_direction_t direction
```

Public Static Attributes

```cpp
constexpr float_64 gamma = 1.0
```

```cpp
struct RandomEnabledRadiationFunctor
```
Public Functions

\[
\text{template<typename T_Rng, typename T_Particle> DINLINE void picongpu::particles::manipulators::RandomEnabledRadiationFunctor::operator() (T_Rng & rng, T_Particle & particle)}
\]

Public Static Attributes

\[
\text{constexpr float}_64 \text{temperature} = 0.0
\]

namespace startPosition

Typedefs

\[
\text{using Random} = \text{RandomImpl<RandomParameter>}
\]

\[
\text{using Quiet} = \text{QuietImpl<QuietParam>}
\]

\[
\text{using OnePosition} = \text{OnePositionImpl<OnePositionParameter>}
\]

Functions

\[
\text{picongpu::particles::startPosition::CONSTVECTOR(float}_X, 3, \text{InCellOffset}, 0.0, 0.0, 0.0)
\]

\[
\text{sit directly in lower corner of the cell}
\]

\[
\text{struct OnePositionParameter}
\]

Public Members

\[
\text{const InCellOffset_t inCellOffset}
\]

Public Static Attributes

\[
\text{constexpr uint32_t numParticlesPerCell} = \text{TYPICAL_PARTICLES_PER_CELL}
\]

\[
\text{Count of particles per cell at initial state.}
\]

\[
\text{unit: none}
\]

\[
\text{struct QuietParam}
\]

Public Types

\[
\text{using numParticlesPerDimension} = \text{mCT::shrinkTo\langle mCT::Int<1, TYPICAL_PARTICLES_PER_CELL}\rangle
\]

\[
\text{Count of particles per cell per direction at initial state.}
\]

\[
\text{unit: none}
\]

\[
\text{struct RandomParameter}
\]
Public Static Attributes

```cpp
constexpr uint32_t numParticlesPerCell = TYPICAL_PARTICLES_PER_CELL
Count of particles per cell at initial state.
unit: none
```

More details on the order of initialization of particles inside a particle species can be found here. List of all pre-defined particle manipulators.

unit.param

In this file we define typical scales for normalization of physical quantities aka “units”.
Usually, a user would not change this file but might use the defined constants in other input files.

namespace picongpu

Variables

```cpp
constexpr float_64 UNIT_TIME = SI::DELTA_T_SI
Unit of time.

constexpr float_64 UNIT_LENGTH = UNIT_TIME * UNIT_SPEED
Unit of length.

constexpr float_64 UNIT_MASS = SI::BASE_MASS_SI * double(particles::TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE)
Unit of mass.

constexpr float_64 UNIT_CHARGE = -1.0 * SI::BASE_CHARGE_SI * double(particles::TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE)
Unit of charge.

constexpr float_64 UNIT_ENERGY = (UNIT_MASS * UNIT_LENGTH * UNIT_LENGTH / (UNIT_TIME * UNIT_TIME))
Unit of energy.

constexpr float_64 UNIT_EFIELD = 1.0 / (UNIT_TIME * UNIT_TIME / UNIT_MASS / UNIT_LENGTH * UNIT_CHARGE)
Unit of EField: V/m.

constexpr float_64 UNIT_BFIELD = (UNIT_MASS / (UNIT_TIME * UNIT_CHARGE))
```

namespace particles

Variables

```cpp
constexpr float_X TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE = float_64(SI::BASE_DENSITY_SI * SI::CELL_WIDTH_SI * SI::CELL_HEIGHT_SI * SI::CELL_DEPTH_SI) / float_64(particles::TYPICAL_PARTICLES_PER_CELL)
Number of particles per makro particle (= macro particle weighting) unit: none.
```

particleFilters.param

A common task in both modeling and in situ processing (output) is the selection of particles of a particle species by attributes.
Users can define such selections as particle filters in this file.
Particle filters are simple mappings assigning each particle of a species either true or false (ignore / filter out).
All active filters need to be listed in AllParticleFilters. They are then combined with VectorAllSpecies at compile-time, e.g. for plugins.
namespace particles

namespace filter

Typedefs

using AllParticleFilters = MakeSeq_t>All>
Plugins: collection of all available particle filters.
Create a list of all filters here that you want to use in plugins.
Note: filter All is defined in picongpu/particles/filter/filter.def

List of all pre-defined particle filters.

speciesInitialization.param

Initialize particles inside particle species.
This is the final step in setting up particles (defined in speciesDefinition.param) via density profiles (defined in density.param). One can then further derive particles from one species to another and manipulate attributes with “manipulators” and “filters” (defined in particle.param and particleFilters.param).

namespace picongpu

namespace particles

Typedefs

using InitPipeline = bmpl::vector<>InitPipeline defines in which order species are initialized.
the functors are called in order (from first to last functor)

List of all initialization methods for particle species.

Particles

Particles are defined in modular steps. First, species need to be generally defined in speciesDefinition.param. Second, species are initialized with particles in speciesInitialization.param.
The following operations can be applied in the picongpu::particles::InitPipeline of the latter:

Initialization

CreateDensity

template<typename T_DensityFunctor, typename T_PositionFunctor, typename T_SpeciesType = bmpl::_1>
struct CreateDensity
    Create particle distribution from a normalized density profile.
    Create particles inside a species. The created particles are macroscopically distributed according to a given normalized density profile (T_DensityFunctor). Their microscopic position inside individual cells is determined by the T_PositionFunctor.
    Note FillAllGaps is automatically called after creation.
Template Parameters

- **T_DensityFunctor**: unary lambda functor with profile description, see density.param, example: picongpu::particles::densityProfiles::Homogenous
- **T_PositionFunctor**: unary lambda functor with position description, see particle.param, examples: picongpu::particles::StartPosition::Quiet, picongpu::particles::StartPosition::Random
- **T_SpeciesType**: type or name as boost::mpl::string of the used species, see speciesDefinition.param

**Derive**

```cpp
template<typename T_SrcSpeciesType, typename T_DestSpeciesType = bmpl::_1, typename T_Filter = filter::All>
struct Derive : public picongpu::particles::ManipulateDerive<manipulators::generic::None, T_SrcSpeciesType, T_DestSpeciesType>
```

Generate particles in T_DestSpeciesType by deriving (copying) all particles and their matching attributes (except particleId) from T_SrcSpeciesType.

**Note** FillAllGaps is called on on T_DestSpeciesType after the derivation is finished.

Template Parameters

- **T_SrcSpeciesType**: type or name as boost::mpl::string of the source species
- **T_DestSpeciesType**: type or name as boost::mpl::string of the destination species
- **T_Filter**: picongpu::particles::filter, particle filter type to select source particles to derive

**Manipulate**

```cpp
template<typename T_Manipulator, typename T_Species = bmpl::_1, typename T_Filter = filter::All>
struct Manipulate : public pmacc::particles::algorithm::CallForEach<pmacc::particles::meta::FindByNameOrType<VectorAllSpecies>, detail::MakeUnaryFilteredFunctor<T_Manipulator, T_Species, T_Filter>>
```

Run a user defined manipulation for each particle of a species.

Allows to manipulate attributes of existing particles in a species with arbitrary unary functors ("manipulators").

**Warning** Does NOT call FillAllGaps after manipulation! If the manipulation deactivates particles or creates "gaps" in any other way, FillAllGaps needs to be called for the T_Species manually in the next step!

**See** picongpu::particles::manipulators

Template Parameters

- **T_Manipulator**: unary lambda functor accepting one particle species

Template Parameters

- **T_Species**: type or name as boost::mpl::string of the used species
- **T_Filter**: picongpu::particles::filter, particle filter type to select particles in T_Species to manipulate

**ManipulateDerive**

```cpp
template<typename T_Manipulator, typename T_SrcSpeciesType, typename T_DestSpeciesType = bmpl::_1, typename T_SrcFilter = filter::All>
```
**struct ManipulateDerive**

Generate particles in a species by deriving and manipulating from another species’ particles.

Create particles in `T_DestSpeciesType` by deriving (copying) all particles and their matching attributes (except `particleId`) from `T_SrcSpeciesType`. During the derivation, the particle attributes in can be manipulated with `T_ManipulateFunctor`.

**Note** `FillAllGaps` is called on on `T_DestSpeciesType` after the derivation is finished. If the derivation also manipulates the `T_SrcSpeciesType`, e.g. in order to deactivate some particles for a move, `FillAllGaps` needs to be called for the `T_SrcSpeciesType` manually in the next step!

See `picongpu::particles::manipulators`

**Template Parameters**

- `T_Manipulator`: a pseudo-binary functor accepting two particle species: destination and source,

**Template Parameters**

- `T_SrcSpeciesType`: type or name as `boost::mpl::string` of the source species
- `T_DestSpeciesType`: type or name as `boost::mpl::string` of the destination species
- `T_SrcFilter`: `picongpu::particles::filter`, particle filter type to select particles in `T_SrcSpeciesType` to derive into `T_DestSpeciesType`

**FillAllGaps**

```cpp
template<typename T_SpeciesType = bmpl::_1>
struct FillAllGaps
```

Generate a valid, contiguous list of particle frames.

Some operations, such as deactivating or adding particles to a particle species can generate “gaps” in our internal particle storage, a list of frames.

This operation copies all particles from the end of the frame list to “gaps” in the beginning of the frame list. After execution, the requirement that all particle frames must be filled contiguously with valid particles and that all frames but the last are full is fulfilled.

**Template Parameters**

- `T_SpeciesType`: type or name as `boost::mpl::string` of the particle species to fill gaps in memory

**Manipulation Functors**

Some of the particle operations above can take the following functors as arguments to manipulate attributes of particle species. A particle filter (see following section) is used to only manipulated selected particles of a species with a functor.

**Free**

```cpp
template<typename T_Functor>
struct Free : protected picongpu::particles::functor::User<T_Functor>
```

call simple free user defined manipulators

example for `particle.param`: set in cell position to zero
struct FunctorInCellPositionZero
{
    template<typename T_Particle>
    HDINLINE void operator()(T_Particle & particle)
    {
        particle[position_] = floatD_X::create(0.0);
    }
    static constexpr char const * name = "inCellPositionZero";
};

using InCellPositionZero = generic::Free<FunctorInCellPositionZero>;

Template Parameters

• T_Functor: user defined manipulators optional: can implement one host side constructor
  T_Functor() or T_Functor(uint32_t currentTimeStep)

FreeRng

template<typename T_Functor, typename T_Distribution>
struct FreeRng : protected picongpu::particles::functor::User<T_Functor>,
private picongpu::particles::functor::misc::Rng<T_Distribution> {
    call simple free user defined functor and provide a random number generator
    example for particle.param: add

#include <pmacc/nvidia/rng/distributions/Uniform_float.hpp>

struct FunctorRandomX
{
    template<typename T_Rng, typename T_Particle>
    HDINLINE void operator()(T_Rng & rng, T_Particle & particle)
    {
        particle[position_].x() = rng();
    }
    static constexpr char const * name = "randomXPos";
};

using RandomXPos = generic::FreeRng<FunctorRandomX,
pmacc::random::distributions::Uniform< float_X > >;

Template Parameters

• T_Functor: user defined unary functor
  • T_Distribution: pmacc::random::distributions, random number distribution

and to InitPipeline in speciesInitialization.param:

Manipulate<manipulators::RandomXPos, SPECIES_NAME>

FreeTotalCellOffset

template<typename T_Functor>
struct FreeTotalCellOffset : protected picongpu::particles::functor::User<T_Functor>, private picongpu::particles::functor::misc::TotalCellOffset
{
    call simple free user defined manipulators and provide the cell information

    The functor passes the cell offset of the particle relative to the total domain origin into the functor.

    example for particle.param: set a user-defined species attribute y0 (type: uint32_t) to the current total y-cell index

    struct FunctorSaveYcell
    {
        template< typename T_Particle >
        HDINLINE void operator()(DataSpace< simDim > const & particleOffsetToTotalOrigin, T_Particle & particle)
        {
            particle[ y0_ ] = particleOffsetToTotalOrigin.y();
        }
        static constexpr char const * name = "saveYcell";
    };

    using SaveYcell = unary::FreeTotalCellOffset<FunctorSaveYcell>;
}

Template Parameters
- T_Functor: user defined unary functor

CopyAttribute

using picongpu::particles::manipulators::unary::CopyAttribute = generic::Free<acc::CopyAttribute<T_DestAttribute, T_SrcAttribute>>
copy a particle source attribute to a destination attribute

This is an unary functor and operates on one particle.

Template Parameters
- T_DestAttribute: type of the destination attribute e.g. momentumPrev1
- T_SrcAttribute: type of the source attribute e.g. momentum

Drift

using picongpu::particles::manipulators::unary::Drift = generic::Free<acc::Drift<T_ParamClass, T_ValueFunctor>>
change particle’s momentum based on speed
allow to manipulate a speed to a particle

Template Parameters
- T_ParamClass: param::DriftCfg, configuration parameter
- T_ValueFunctor: pmacc::nvidia::functors::*\, binary functor type to manipulate the momentum attribute

RandomPosition

using picongpu::particles::manipulators::unary::RandomPosition = generic::FreeRng<acc::RandomPosition>
Change the in cell position.

2.3. .param Files
This functor changes the in-cell position of a particle. The new in-cell position is uniformly distributed position between [0.0;1.0).

example: add

```cpp
particles::Manipulate<RandomPosition,SPECIES_NAME>
```

to `InitPipeline in speciesInitialization.param`

**Temperature**

using `picongpu::particles::manipulators::unary::Temperature = generic::FreeRng<acc::Temperature<T_ParamClass, T_ValueFunctor>,pmacc::random::distributions::Normal<float_X>>`

change particle’s momentum based on a temperature

allow to change the temperature (randomly normal distributed) of a particle.

**Template Parameters**

- `T_ParamClass`: `param::TemperatureCfg`, configuration parameter
- `T_ValueFunctor`: `pmacc::nvidia::functors::*`, binary functor type to manipulate the momentum attribute

**Assign**

using `picongpu::particles::manipulators::binary::Assign = generic::Free<acc::Assign>`

assign attributes of one particle to another

Can be used as binary and higher order operator but only the first two particles are used for the assign operation.

Assign all matching attributes of a source particle to the destination particle. Attributes that only exist in the destination species are initialized with the default value. Attributes that only exists in the source particle will be ignored.

**DensityWeighting**

using `picongpu::particles::manipulators::binary::DensityWeighting = generic::Free<acc::DensityWeighting>`

Re-scale the weighting of a cloned species by `densityRatio`.

When deriving species from each other, the new species “inherits” the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species’ macro particles to satisfy the input `densityRatio` of it.

note: needs the `densityRatio` flag on both species, used by the GetDensityRatio trait.

**ProtonTimesWeighting**

using `picongpu::particles::manipulators::binary::ProtonTimesWeighting = generic::Free<acc::ProtonTimesWeighting>`

Re-scale the weighting of a cloned species by `numberOfProtons`.

When deriving species from each other, the new species “inherits” the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species’ macro particles to be a multiplied by the number of protons of the initial species.

As an example, this is useful when initializing a quasi-neutral, pre-ionized plasma of ions and electrons. Electrons can be created from ions via deriving and increasing their weight to avoid simulating multiple macro electrons per macro ion (with Z>1).

note: needs the `atomicNumbers` flag on the initial species, used by the GetAtomicNumbers trait.
Manipulation Filters

Most of the particle functors shall operate on all valid particles, where `filter::All` is the default assumption. One can limit the domain or subset of particles with filters such as the ones below (or define new ones).

**All**

```cpp
struct All
```

**RelativeGlobalDomainPosition**

```cpp
template<typename T_Params>
struct RelativeGlobalDomainPosition
    filter particle dependent on the global position
    Check if a particle is within a relative area in one direction of the global domain.

    Template Parameters
    • T_Params: picongpu::particles::filter::param::RelativeGlobalDomainPosition, parameter to configure the functor
```

**Free**

```cpp
template<typename T_Functor>
struct Free : protected picongpu::particles::functor::User<T_Functor>
    call simple free user defined filter
```

**Example**

```cpp
struct FunctorEachParticleAboveMiddleOfTheCell
{
    template< typename T_Particle >
    HDINLINE bool operator()( T_Particle const & particle )
    {
        bool result = false;
        if( particle[ position_ ].y() >= float_X( 0.5 ) )
            result = true;
        return result;
    }

    static constexpr char const * name = "eachParticleAboveMiddleOfTheCell";
};
```

Using `EachParticleAboveMiddleOfTheCell = generic::Free<FunctorEachParticleAboveMiddleOfTheCell>;`

**Template Parameters**

• T_Functor: user defined filter optional: can implement one host side constructor `T_Functor()` or `T_Functor(uint32_t currentTimeStep)`

**FreeRng**

```cpp
template<typename T_Functor, typename T_Distribution>
```
**FreeRng**

```cpp
struct FreeRng : protected picongpu::particles::functor::User<T_Functor>, private picongpu::particles::functor::misc::Rng<T_Distribution>
{
  call simple free user defined functor and provide a random number generator

  example for particleFilters.param: get every second particle (random sample of 50%)

  struct FunctorEachSecondParticle
  {
    template< typename T_Rng, typename T_Particle >>
    HDINLINE bool operator()( T_Rng & rng, 
        T_Particle const & particle )
    {
      bool result = false;
      if( rng() >= float_X( 0.5 ) )
        result = true;
      return result;
    }
    static constexpr char const * name = "eachSecondParticle";
  };

  using EachSecondParticle = generic::FreeRng<
    FunctorEachSecondParticle,
    pmacc::random::distributions::Uniform< float_X > >;

  Template Parameters
  • T_Functor: user defined unary functor
  • T_Distribution: pmacc::random::distributions, random number distribution
}
```

**FreeTotalCellOffset**

```cpp
template<typename T_Functor>
struct FreeTotalCellOffset : protected picongpu::particles::functor::User<T_Functor>, private picongpu::particles::functor::misc::TotalCellOffset
{
  call simple free user defined functor and provide the cell information

  example for particleFilters.param: each particle with a cell offset of 5 in X direction

  struct FunctorEachParticleInXCell5
  {
    template< typename T_Particle >>
    HDINLINE bool operator()( DataSpace< simDim > const & particleOffsetToTotalOrigin, 
        T_Particle const & particle )
    {
      bool result = false;
      if( particleOffsetToTotalOrigin.x() == 5 )
        result = true;
      return result;
    }
    static constexpr char const * name = "eachParticleInXCell5";
  };

  using EachParticleInXCell5 = generic::FreeTotalCellOffset<
    FunctorEachParticleInXCell5
    >;
}
```
Template Parameters

- T_Functor: user defined unary functor

Memory

memory.param

Define low-level memory settings for compute devices.

Settings for memory layout for supercells and particle frame-lists, data exchanges in multi-device domain-decomposition and reserved fields for temporarily derived quantities are defined here.

namespace picongpu

typedefs

using SuperCellSize = typename mCT::shrinkTo<mCT::Int<8, 8, 4>, simDim>::type
    size of a superCell
    volume of a superCell must be <= 1024

using MappingDesc = MappingDescription<simDim, SuperCellSize>
    define mapper which is used for kernel call mappings

using GuardSize = typename mCT::shrinkTo<mCT::Int<1, 1, 1>, simDim>::type
    define the size of the core, border and guard area

PIConGPU uses spatial domain-decomposition for parallelization over multiple devices with non-shared memory architecture. The global spatial domain is organized per device in three sections: the GUARD area contains copies of neighboring devices (also known as “halo”/“ghost”). The BORDER area is the outermost layer of cells of a device, equally to what neighboring devices see as GUARD area. The CORE area is the innermost area of a device. In union with the BORDER area it defines the “active” spatial domain on a device.

GuardSize is defined in units of SuperCellSize per dimension.

Variables

constexpr size_t reservedGpuMemorySize = 350 * 1024 * 1024
constexpr uint32_t fieldTmpNumSlots = 1
    number of scalar fields that are reserved as temporary fields
constexpr bool fieldTmpSupportGatherCommunication = true
    can FieldTmp gather neighbor information

If true it is possible to call the method asyncCommunicationGather() to copy data from the border of neighboring GPU into the local guard. This is also known as building up a “ghost” or “halo” region in domain decomposition and only necessary for specific algorithms that extend the basic PIC cycle, e.g. with dependence on derived density or energy fields.

struct DefaultExchangeMemCfg
    bytes reserved for species exchange buffer

This is the default configuration for species exchanges buffer sizes. The default exchange buffer sizes can be changed per species by adding the alias exchangeMemCfg with similar members like in DefaultExchangeMemCfg to its flag list.
**Public Static Attributes**

```cpp
constexpr uint32_t BYTES.Exchange.X = 1 * 1024 * 1024
constexpr uint32_t BYTES.Exchange.Y = 3 * 1024 * 1024
constexpr uint32_t BYTES.Exchange.Z = 1 * 1024 * 1024
constexpr uint32_t BYTES.Edges = 32 * 1024
constexpr uint32_t BYTES.Corner = 8 * 1024
```

**precision.param**

Define the precision of typically used floating point types in the simulation.

PIConGPU normalizes input automatically, allowing to use single-precision by default for the core algorithms. Note that implementations of various algorithms (usually plugins or non-core components) might still decide to hard-code a different (mixed) precision for some critical operations.

**mallocMC.param**

Fine-tuning of the particle heap for GPUs: When running on GPUs, we use a high-performance parallel “new” allocator (mallocMC) which can be parametrized here.

**namespace picongpu**

**Typedefs**

```cpp
using DeviceHeap = mallocMC::Allocator<mallocMC::CreationPolicies::Scatter<DeviceHeapConfig>, mallocMC::DistributionPolicies::Noop, mallocMC::OOMPolicies::ReturnNull, mallocMC::ReservePoolPolicies::SimpleCudaMalloc, mallocMC::AlignmentPolicies::Shrink<>>
```

Define a new allocator.

This is an allocator resembling the behaviour of the ScatterAlloc algorithm.

**struct DeviceHeapConfig**

configure the CreationPolicy “Scatter”

**Public Types**

```cpp
using pagesize = boost::mpl::int_<2 * 1024 * 1024>
    2MiB page can hold around 256 particle frames
using accessblocks = boost::mpl::int_<4>
    accessblocks, regionsize and wastefactor are not conclusively investigated and might be performance sensitive for multiple particle species with heavily varying attributes (frame sizes)
using regionsize = boost::mpl::int_<8>
using wastefactor = boost::mpl::int_<2>
using resetfreedpages = boost::mpl::bool_<true>
```

resetfreedpages is used to minimize memory fragmentation with varying frame sizes

**PIC Extensions**

**fieldBackground.param**

Load external background fields.
namespace picongpu

class FieldBackgroundB

Public Functions

PMACC_ALIGN (m_unitField, const float3_64)

HDINLINE FieldBackgroundB (const float3_64 unitField)

HDINLINE float3_X picongpu::FieldBackgroundB::operator()(const DataSpace < simDim > & cellIdx, const uint32_t currentStep) const

Specify your background field $B(r,t)$ here.

Parameters
• cellIdx: The total cell id counted from the start at t=0
• currentStep: The current time step

Public Static Attributes

constexpr bool InfluenceParticlePusher = false

class FieldBackgroundE

Public Functions

PMACC_ALIGN (m_unitField, const float3_64)

HDINLINE FieldBackgroundE (const float3_64 unitField)

HDINLINE float3_X picongpu::FieldBackgroundE::operator()(const DataSpace < simDim > & cellIdx, const uint32_t currentStep) const

Specify your background field $E(r,t)$ here.

Parameters
• cellIdx: The total cell id counted from the start at t = 0
• currentStep: The current time step

Public Static Attributes

constexpr bool InfluenceParticlePusher = false

class FieldBackgroundJ

Public Functions

PMACC_ALIGN (m_unitField, const float3_64)

HDINLINE FieldBackgroundJ (const float3_64 unitField)

HDINLINE float3_X picongpu::FieldBackgroundJ::operator()(const DataSpace < simDim > & cellIdx, const uint32_t currentStep) const

Specify your background field $J(r,t)$ here.

Parameters
• cellIdx: The total cell id counted from the start at t=0
• currentStep: The current time step

Public Static Attributes

constexpr bool activated = false
bremsstrahlung.param

namespace picongpu

namespace particles

namespace bremsstrahlung

namespace electron

params related to the energy loss and deflection of the incident electron

Variables

constexpr float_64 MIN_ENERGY_MeV = 0.5
Minimal kinetic electron energy in MeV for the lookup table.
For electrons below this value Bremsstrahlung is not taken into account.

constexpr float_64 MAX_ENERGY_MeV = 200.0
Maximal kinetic electron energy in MeV for the lookup table.
Electrons above this value cause a out-of-bounds access at the lookup table. Bounds checking is enabled for “CRITICAL” log level.

constexpr float_64 MIN_THETA = 0.01
Minimal polar deflection angle due to screening.
See Jackson 13.5 for a rule of thumb to this value.

constexpr uint32_t NUM_SAMPLES_KAPPA = 32
number of lookup table divisions for the kappa axis.
Kappa is the energy loss normalized to the initial kinetic energy. The axis is scaled linearly.

constexpr uint32_t NUM_SAMPLES_EKIN = 32
number of lookup table divisions for the initial kinetic energy axis.
The axis is scaled logarithmically.

constexpr float_64 MIN_KAPPA = 1.0e-10
Kappa is the energy loss normalized to the initial kinetic energy.
This minimal value is needed by the numerics to avoid a division by zero.

namespace photon

params related to the creation and the emission angle of the photon

Variables

constexpr float_64 SOFT_PHOTONS_CUTOFF_keV = 5000.0
Low-energy threshold in keV of the incident electron for the creation of photons.
Below this value photon emission is neglected.

constexpr uint32_t NUM_SAMPLES_DELTA = 256
number of lookup table divisions for the delta axis.
Delta is the angular emission probability (normalized to one) integrated from zero to theta, where theta is the angle between the photon momentum and the final electron momentum.
The axis is scaled linearly.
constexpr uint32_t NUM_SAMPLES_GAMMA = 64
number of lookup table divisions for the gamma axis.
Gamma is the relativistic factor of the incident electron.
The axis is scaled logarithmically.

constexpr float_64 MAX_DELTA = 0.95
Maximal value of delta for the lookup table.
Delta is the angular emission probability (normalized to one) integrated from zero to theta,
where theta is the angle between the photon momentum and the final electron momentum.
A value close to one is reasonable. Though exactly one was actually correct, because it
would map to theta = pi (maximum polar angle), the sampling then would be bad in the
ultrarelativistic case. In this regime the emission primarily takes place at small thetas. So
a maximum delta close to one maps to a reasonable maximum theta.

constexpr float_64 MIN_GAMMA = 1.0
minimal gamma for the lookup table.

constexpr float_64 MAX_GAMMA = 250
maximal gamma for the lookup table.
Bounds checking is enabled for “CRITICAL” log level.

constexpr float_64 SINGLE_EMISSION_PROB_LIMIT = 0.4
if the emission probability per timestep is higher than this value and the log level is set to
“CRITICAL” a warning will be raised.

constexpr float_64 WEIGHTING_RATIO = 10

synchrotronPhotons.param

Defines

ENABLE_SYNCHROTRON_PHOTONS
  enable synchrotron photon emission

namespace picongpu

namespace particles

namespace synchrotronPhotons

Variables

constexpr bool enableQEDTerm = false
  enable (disable) QED (classical) photon emission spectrum

constexpr float_64 SYNC_FUNCS_CUTOFF = 5.0
  Above this value (to the power of three, see comments on mapping) the synchrotron functions
  are nearly zero.

constexpr float_64 SYNC_FUNCS_BESSEL_INTEGRAL_STEPWIDTH = 1.0e-3
  stepwidth for the numerical integration of the bessel function for the first synchrotron function

constexpr uint32_t SYNC_FUNCS_NUM_SAMPLES = 8192
  Number of sampling points of the lookup table.

constexpr float_64 SOFT_PHOTONS_CUTOFF_RATIO = 1.0
  Photons of oscillation periods greater than a timestep are not created since the grid already
  accounts for them.
This cutoff ratio is defined as: photon-oscillation-period / timestep

```cpp
constexpr float_64 SINGLE_EMISSION_PROB_LIMIT = 0.4
```

if the emission probability per timestep is higher than this value and the log level is set to “CRITICAL” a warning will be raised.

**ionizer.param**

This file contains the proton and neutron numbers of commonly used elements of the periodic table.

The elements here should have a matching list of ionization energies in Furthermore there are parameters for specific ionization models to be found here. That includes lists of screened nuclear charges as seen by bound electrons for the aforementioned elements as well as fitting parameters of the Thomas-Fermi ionization model.

See ionizationEnergies.param. Moreover this file contains a description of how to configure an ionization model for a species. Currently each species can only be assigned exactly one ionization model.

**namespace picongpu**

**namespace ionization**

Ionization Model Configuration.

- None: no particle is ionized
- BSI: simple barrier suppression ionization
- BSIEffectiveZ: BSI taking electron shielding into account via an effective atomic number Z_eff
- ADKLinPol: Ammosov-Delone-Krainov tunneling ionization (H-like) -> linearly polarized lasers
- ADKCircPol: Ammosov-Delone-Krainov tunneling ionization (H-like) -> circularly polarized lasers
- Keldysh: Keldysh ionization model
- ThomasFermi: statistical impact ionization based on Thomas-Fermi atomic model Attention: requires 2 FieldTmp slots Research and development:
  - See memory.param
- BSIStarkShifted: BSI for hydrogen-like atoms and ions considering the Stark upshift of ionization potentials

Usage: Add flags to the list of particle flags that has the following structure

```cpp
ionizers< MakeSeq_t< particles::ionization::IonizationModel<
  Species2BCreated > > >, atomicNumbers< ionization::atomicNumbers::Element_t >,
effectiveNuclearCharge< ionization::effectiveNuclearCharge::Element_t >,
ionizationEnergies< ionization::energies::AU::Element_t >
```

**namespace atomicNumbers**

Specify (chemical) element

Proton and neutron numbers define the chemical element that the ion species is based on. This value can be non-integer for physical models taking charge shielding effects into account. It is wrapped into a struct because of C++ restricting floats from being template arguments.

See http://en.wikipedia.org/wiki/Effective_nuclear_charge
Do not forget to set the correct mass and charge via massRatio<> and chargeRatio<>!

**struct Aluminium_t**

AI-27 ~100% NA.
Public Static Attributes

```cpp
constexpr float X numberOfProtons = 13.0
constexpr float X numberOfNeutrons = 14.0
```

```cpp
struct Carbon_t
    C-12 98.9% NA.
```

Public Static Attributes

```cpp
constexpr float X numberOfProtons = 6.0
constexpr float X numberOfNeutrons = 6.0
```

```cpp
struct Copper_t
    Cu-63 69.15% NA.
```

Public Static Attributes

```cpp
constexpr float X numberOfProtons = 29.0
constexpr float X numberOfNeutrons = 34.0
```

```cpp
struct Deuterium_t
    H-2 0.02% NA.
```

Public Static Attributes

```cpp
constexpr float X numberOfProtons = 1.0
constexpr float X numberOfNeutrons = 1.0
```

```cpp
struct Gold_t
    Au-197 ~100% NA.
```

Public Static Attributes

```cpp
constexpr float X numberOfProtons = 79.0
constexpr float X numberOfNeutrons = 118.0
```

```cpp
struct Helium_t
    He-4 ~100% NA.
```

Public Static Attributes

```cpp
constexpr float X numberOfProtons = 2.0
constexpr float X numberOfNeutrons = 2.0
```

```cpp
struct Hydrogen_t
    H-1 99.98% NA.
```
Public Static Attributes

```cpp
constexpr float_X numberOfProtons = 1.0
constexpr float_X numberOfNeutrons = 0.0
```

```cpp
struct Nitrogen_t
N-14 99.6% NA.
```

Public Static Attributes

```cpp
constexpr float_X numberOfProtons = 7.0
constexpr float_X numberOfNeutrons = 7.0
```

```cpp
struct Oxygen_t
O-16 99.76% NA.
```

Public Static Attributes

```cpp
constexpr float_X numberOfProtons = 8.0
constexpr float_X numberOfNeutrons = 8.0
```

```cpp
struct Silicon_t
Si-28 ~92.23% NA.
```

Public Static Attributes

```cpp
constexpr float_X numberOfProtons = 14.0
constexpr float_X numberOfNeutrons = 14.0
```

```cpp
namespace effectiveNuclearCharge

Effective Nuclear Charge.

Due to the shielding effect of inner electron shells in an atom / ion which makes the core charge seem smaller to valence electrons new, effective, atomic core charge numbers can be defined to make the crude barrier suppression ionization (BSI) model less inaccurate.


See https://en.wikipedia.org/wiki/Effective_nuclear_charge or refer directly to the calculations by Slater or Clementi and Raimondi

IMPORTANT NOTE: You have to insert the values in REVERSE order since the lowest shell corresponds to the last ionization process!
```

```cpp
functions

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 1, Hydrogen, 1.)
picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 1, Deuterium, 1.)
picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 2, Helium, 1. 688, 1. 688)
picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 6, Carbon, 3. 136, 3. 136, 3. 217, 3. 217, 5. 673, 5. 673)
picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 7, Nitrogen, 3. 834, 3. 834, 3. 834, 3. 874, 3. 874, 6. 665, 6. 665)
picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 8, Oxygen, 4. 453, 4. 453, 4. 453, 4. 453, 4. 492, 4. 492, 7. 658, 7. 658)
```
namespace particles

namespace ionization

namespace thomasFermi

### Variables

**constexpr float**

- `TFAlpha` = 14.3139
  - Fitting parameters to average ionization degree \( Z^* = 4/3 \pi R_0^3 \cdot n(R_0) \) as an extension towards arbitrary atoms and temperatures.

- `TFBeta` = 0.6624
- `TFA1` = 3.323e-3
- `TFA2` = 9.718e-1
- `TFA3` = 9.26148e-5
- `TFA4` = 3.10165
- `TFB0` = -1.7630
- `TFB1` = 1.43175
- `TFB2` = 0.31546
- `TFC1` = -0.366667
- `TFC2` = 0.983333

- `CUTOFF_MAX_ENERGY_KEV` = 50.0
  - Cutoff energy for electron “temperature” calculation
  - In laser produced plasmas we can have different, well-separable groups of electrons. For the Thomas-Fermi ionization model we only want the thermalized “bulk” electrons. Including the high-energy “prompt” electrons is physically questionable since they do not have a large cross section for collisional ionization.
  - unit: keV

- `CUTOFF_MAX_ENERGY` = `CUTOFF_MAX_ENERGY_KEV` * `UNITCONV_keV_to_Joule`
  - Cutoff energy for electron “temperature” calculation in SI units

- `CUTOFF_LOW_DENSITY` = 1.7422e27
  - Lower ion density cutoff
  - The Thomas-Fermi model yields unphysical artifacts for low ion densities. Low ion densities imply lower collision frequency and thus less collisional ionization. The Thomas-Fermi model yields an increasing charge state for decreasing densities and electron temperatures of 10eV and above. This cutoff will be used to set the lower application threshold for charge state calculation.
  - unit: 1 / m^3
Note: This cutoff value should be set in accordance to FLYCHK calculations, for instance! It is not a universal value and requires some preliminary approximations! For instance: 1.7422e27 as a hydrogen ion number density equal to the corresponding critical electron number density for an 800nm laser. The choice of the default is motivated by the following: In laser-driven plasmas all dynamics in density regions below the critical electron density will be laser-dominated. Once ions of that density are ionized once the laser will not penetrate fully anymore and the as electrons are heated the dynamics will be collision-dominated.

```cpp
constexpr float_X CUTOFF_LOW_TEMPERATURE_EV = 1.0
```
lower electron temperature cutoff

The Thomas-Fermi model predicts initial ionization for many materials of solid density even when the electron temperature is 0.

**ionizationEnergies.param**

This file contains the ionization energies of commonly used elements of the periodic table. Each atomic species in PIConGPU can represent exactly one element. The ionization energies of that element are stored in a vector which contains the name and proton number as well as a list of energy values. The number of ionization levels must be equal to the proton number of the element.

```cpp
namespace picongpu

namespace ionization

Ionization Model Configuration.

- None: no particle is ionized
- BSI: simple barrier suppression ionization
- BSIEffectiveZ: BSI taking electron shielding into account via an effective atomic number Z_eff
- ADKLinPol: Ammosov-Delone-Krainov tunneling ionization (H-like) -> linearly polarized lasers
- ADKCircPol: Ammosov-Delone-Krainov tunneling ionization (H-like) -> circularly polarized lasers
- Keldysh: Keldysh ionization model
- ThomasFermi: statistical impact ionization based on Thomas-Fermi atomic model Attention: requires 2 FieldTmp slots Research and development:
  See memory.param
- BSIStarkShifted: BSI for hydrogen-like atoms and ions considering the Stark upshift of ionization potentials

Usage: Add flags to the list of particle flags that has the following structure

```cpp
ionizers< MakeSeq_t< particles::ionization::IonizationModel<
  ×Species2BCreated > > >,
  atomicNumbers< ionization::atomicNumbers::Element_t >,
  effectiveNuclearCharge< ionization::effectiveNuclearCharge::Element_t >,
  ionizationEnergies< ionization::energies::AU::Element_t >
```

```cpp
namespace energies

Ionization potentials.

Please follow these rules for defining ionization energies of atomic species, unless your chosen ionization model requires a different unit system than AU:::
• input of values in either atomic units or converting eV or Joule to them -> use either UNIT-CONV_eV_to_AU or SI::ATOMIC_UNIT_ENERGY for that purpose
• use float_X as the preferred data type

    example: ionization energy for ground state hydrogen: 13.6 eV 1 Joule = 1 kg \* m^2 / s^2 1 eV = 1.602e-19 J

    1 AU (energy) = 27.2 eV = 1 Hartree = 4.36e-18 J = 2 Rydberg = 2 x Hydrogen ground state binding energy

Atomic units are useful for ionization models because they simplify the formulae greatly and provide intuitively understandable relations to a well-known system, i.e. the Hydrogen atom.


See include/pmacc/math/ConstVector.hpp for finding ionization energies, http://physics.nist.gov/PhysRefData/ASD/ionEnergy.html

namespace AU

Functions

    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 1, Hydrogen, 13.59843 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 1, Deuterium, 13.60213 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 2, Helium, 24.58739 * UNITCONV_eV_to_AU, 54.41776 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 6, Carbon, 11.2603 * UNITCONV_eV_to_AU, 24.3845 * UNITCONV_eV_to_AU, 47.88778 * UNITCONV_eV_to_AU, 64.49351 * UNITCONV_eV_to_AU, 392.0905 * UNITCONV_eV_to_AU, 489.993177 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 7, Nitrogen, 14.53413 * UNITCONV_eV_to_AU, 29.60125 * UNITCONV_eV_to_AU, 77.4735 * UNITCONV_eV_to_AU, 97.89013 * UNITCONV_eV_to_AU, 552.06731 * UNITCONV_eV_to_AU, 667.04609 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 8, Oxygen, 13.61805 * UNITCONV_eV_to_AU, 35.12112 * UNITCONV_eV_to_AU, 113.8989 * UNITCONV_eV_to_AU, 138.1189 * UNITCONV_eV_to_AU, 739.3268 * UNITCONV_eV_to_AU, 871.4098 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 13, Aluminium, 5.98577 * UNITCONV_eV_to_AU, 18.8285 * UNITCONV_eV_to_AU, 398.656 * UNITCONV_eV_to_AU, 442.006 * UNITCONV_eV_to_AU, 2085.97 * UNITCONV_eV_to_AU, 2304.14 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 14, Silicon, 8.151683 * UNITCONV_eV_to_AU, 16.345845 * UNITCONV_eV_to_AU, 476.18 * UNITCONV_eV_to_AU, 523.415 * UNITCONV_eV_to_AU, 2437.65804 * UNITCONV_eV_to_AU, 2673.1774 * UNITCONV_eV_to_AU)
    picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 29, Copper, 7.72638 * UNITCONV_eV_to_AU, 20.2924 * UNITCONV_eV_to_AU, 2479.12 * UNITCONV_eV_to_AU, 2586.95 * UNITCONV_eV_to_AU, 11062.4 * UNITCONV_eV_to_AU, 11567.6 * UNITCONV_eV_to_AU)

flylite.param

This is the configuration file for the atomic particle population kinetics model FLYlite.

Its main purpose is non-LTE collisional-radiative modeling for transient plasmas at high densities and/or interaction with (X-Ray) photon fields.

In simpler words, one can also use this module to simulate collisional ionization processes without the assumption of a local thermal equilibrium (LTE), contrary to popular collisional ionization models such as the Thomas-Fermi ionization model.

This file configures the number of modeled populations for ions, spatial and spectral binning of non-LTE density and energy histograms.

namespace picongpu

    namespace flylite

2.3. .param Files 173
Typedefs

using Superconfig = types::Superconfig<float_64, populations>

using spatialAverageBox = SuperCellSize

You better not change this line, the wooooorld depends on it!

No seriously, per-supercell is the quickest way to average particle quantities such as density, energy histogram, etc. and I won’t implement another size until needed.

Variables

constexpr uint8_t populations = 3u

Number of populations (numpop)

This number defines how many configurations make up a superconfiguration

Range: [0, 255]

constexpr uint8_t ionizationStates = 29u

Ionization states of the atom (iz)

Range: [0, 255]

constexpr uint16_t energies = 512u

Number of energy bins

Energy steps used for local energy histograms

Note: no overflow- or underflow-bins are used, particles with energies outside the range (see below) are ignored

constexpr float_X electronMinEnergy = 0.0

Energy range for electron and photon histograms

Electron and photon histograms f(e) f(ph) are currently calculated in a linearly binned histogram while particles with energies outside the ranges below are ignored

Unit: eV

constexpr float_X electronMaxEnergy = 100.e3

constexpr float_X photonMinEnergy = 0.0

constexpr float_X photonMaxEnergy = 100.e3

Plugins

fileOutput.param

namespace picongpu

Typedefs

using ChargeDensity_Seq = deriveField::CreateEligible_t<VectorAllSpecies, deriveField::derivedAttributes::ChargeDensity>

FieldTmp output (calculated at runtime) ******************************.

Those operations derive scalar field quantities from particle species at runtime. Each value is mapped per cell. Some operations are identical up to a constant, so avoid writing those twice to save storage.

You can choose any of these particle to grid projections:

- Density: particle position + shape on the grid
- BoundElectronDensity: density of bound electrons note: only makes sense for partially ionized ions
• ChargeDensity: density * charge note: for species that do not change their charge state, this is the same as the density times a constant for the charge
• Energy: sum of kinetic particle energy per cell with respect to shape
• EnergyDensity: average kinetic particle energy per cell times the particle density note: this is the same as the sum of kinetic particle energy divided by a constant for the cell volume
• MomentumComponent: ratio between a selected momentum component and the absolute momentum with respect to shape
• LarmorPower: radiated Larmor power (species must contain the attribute momentumPrev1)

for debugging:
• MidCurrentDensityComponent: density * charge * velocity_component
• Counter: counts point like particles per cell
• MacroCounter: counts point like macro particles per cell

using EnergyDensity_Seq = deriveField::CreateEligible_t<VectorAllSpecies, deriveField::derivedAttributes::EnergyDensity>
using MomentumComponent_Seq = deriveField::CreateEligible_t<VectorAllSpecies, deriveField::derivedAttributes::MomentumComponent<0>>
using FieldTmpSolvers = MakeSeq_t<ChargeDensity_Seq, EnergyDensity_Seq, MomentumComponent_Seq>
FieldTmpSolvers groups all solvers that create data for FieldTmp ******.

FieldTmpSolvers is used in
See FieldTmp to calculate the exchange size

using NativeFileOutputFields = MakeSeq_t<FieldE, FieldB>
FileOutputFields: Groups all Fields that shall be dumped.
Possible native fields: FieldE, FieldB, FieldJ

using FileOutputFields = MakeSeq_t<NativeFileOutputFields, FieldTmpSolvers>
using FileOutputParticles = VectorAllSpecies
FileOutputParticles: Groups all Species that shall be dumped **********.
hint: to disable particle output set to using FileOutputParticles = MakeSeq_t< >;

isaac.param

Definition which native fields and density fields of particles will be visualizable with ISAAC.
ISAAC is an in-situ visualization library with which the PIC simulation can be observed while it is running avoiding the time consuming writing and reading of simulation data for the classical post processing of data.
ISAAC can directly visualize natives fields like the E or B field, but density fields of particles need to be calculated from PICoGPU on the fly which slightly increases the runtime and the memory consumption. Every particle density field will reduce the amount of memory left for PICoGPUs particles and fields.
To get best performance, ISAAC defines an exponential amount of different visualization kernels for every combination of (at runtime) activated fields. So furthermore a lot of fields will increase the compilation time.

namespace picongpu

namespace isaacP

Typedefs

using Particle_Seq = VectorAllSpecies
Intermediate list of native particle species of PICoGPU which shall be visualized.
using Native_Seq = MakeSeq_t<FieldE, FieldB, FieldJ>
Intermediate list of native fields of PIConGPU which shall be visualized.

using Density_Seq = deriveField::CreateEligible_t<Particle_Seq, deriveField::derivedAttributes::Density>
Intermediate list of particle species, from which density fields shall be created at runtime to visualize them.

using Fields_Seq = MakeSeq_t<Native_Seq, Density_Seq>
Compile time sequence of all fields which shall be visualized. Basically the join of Native_Seq and Density_Seq.

particleCalorimeter.param

namespace picongpu
namespace particleCalorimeter

Functions

HDINLINE float2_X picongpu::particleCalorimeter::mapYawPitchToNormedRange(const float_X yaw, const float_X pitch, const float_X maxYaw, const float_X maxPitch)
Map yaw and pitch into [0,1] respectively. These ranges correspond to the normalized histogram range of the calorimeter (0: first bin, 1: last bin). Out-of-range values are mapped to the first or the last bin.
Useful for fine tuning the spatial calorimeter resolution.

Return Two values within [-1,1]
Parameters

• yaw: -maxYaw...maxYaw
• pitch: -maxPitch...maxPitch
• maxYaw: maximum value of angle yaw
• maxPitch: maximum value of angle pitch

particleMerger.param

namespace picongpu
namespace plugins
namespace particleMerging

Variables

constexpr size_t MAX_VORONOI CELLS = 128
maximum number of active Voronoi cells per supercell.
If the number of active Voronoi cells reaches this limit merging events are dropped.

radiation.param

Definition of frequency space, number of observers, filters, form factors and window functions of the radiation plugin.
All values set here determine what the radiation plugin will compute. The observation direction is defined in a separate file `radiationObserver.param`. On the command line the plugin still needs to be called for each species the radiation should be computed for.

**Defines**

```
PIC_VERBOSE_RADIATION
    radiation verbose level: 0=nothing, 1=physics, 2=simulation_state, 4=memory, 8=critical
```

```cpp
namespace picongpu

namespace plugins

namespace radiation

```

**Typedefs**

```
using RadiationParticleFilter = picongpu::particles::manipulators::generic::Free<
    GammaFilterFunctor>;
```

filter to (de)select particles for the radiation calculation

to activate the filter:
- goto file `speciesDefinition.param`
- add the attribute `radiationMask` to the particle species

```cpp
struct GammaFilterFunctor
    select particles for radiation example of a filter for the relativistic Lorentz factor gamma
```

**Public Functions**

```
template<
    typename T_Particle>
HDINLINE void picongpu::plugins::radiation::GammaFilterFunctor::operator()(
    T_Particle & particle)
```

**Public Static Attributes**

```
constexpr float_X radiationGamma = 5.0
    Gamma value above which the radiation is calculated.
```

```cpp
namespace frequencies_from_list

Variables

```
constexpr const char *listLocation = "/path/to/frequency_list"
    path to text file with frequencies
constexpr unsigned int N_omega = 2048
    number of frequency values to compute if frequencies are given in a file [unitless]
```

```cpp
namespace linear_frequencies

Variables

```
constexpr unsigned int N_omega = 2048
    number of frequency values to compute in the linear frequency [unitless]
```

```cpp
namespace SI

```
Variables

```cpp
constexpr float_64 omega_min = 0.0
minimum frequency of the linear frequency scale in units of [1/s]

constexpr float_64 omega_max = 1.0e16
maximum frequency of the linear frequency scale in units of [1/s]
```

```cpp
namespace log_frequencies

Variables

```cpp
constexpr unsigned int N_omega = 2048
number of frequency values to compute in the logarithmic frequency [unitless]
```

```cpp
namespace SI

Variables

```cpp
constexpr float_64 omega_min = 1.0e14
minimum frequency of the logarithmic frequency scale in units of [1/s]

constexpr float_64 omega_max = 1.0e17
maximum frequency of the logarithmic frequency scale in units of [1/s]
```

```cpp
namespace parameters

Variables

```cpp
constexpr unsigned int N_observer = 256
number of observation directions
```

```cpp
namespace radFormFactor_CIC_3D
correct treatment of coherent and incoherent radiation from macro particles

Choose different form factors in order to consider different particle shapes for radiation
• radFormFactor_CIC_3D ... CIC charge distribution
• radFormFactor_TSC_3D ... TSC charge distribution
• radFormFactor_PCS_3D ... PCS charge distribution
• radFormFactor_CIC_1Dy ... only CIC charge distribution in y
• radFormFactor_Gauss_spherical ... symmetric Gauss charge distribution
• radFormFactor_Gauss_cell ... Gauss charge distribution according to cell size
• radFormFactor_incoherent ... only incoherent radiation
• radFormFactor_coherent ... only coherent radiation
```

```cpp
namespace radiationNyquist

selected mode of frequency scaling:

options:
• linear_frequencies
• log_frequencies
• frequencies_from_list
```

```cpp
Variables

```cpp
constexpr float_32 NyquistFactor = 0.5
Nyquist factor: fraction of the local Nyquist frequency above which the spectra is set to zero should be in (0, 1).
```
namespace radWindowFunctionTriangle
add a window function weighting to the radiation in order to avoid ringing effects from sharp boundaries default: no window function via radWindowFunctionNone

Choose different window function in order to get better ringing reduction radWindowFunctionTriangle radWindowFunctionHamming radWindowFunctionTriplette radWindowFunctionGauss radWindowFunctionNone

radiationObserver.param

This file defines a function describing the observation directions.
It takes an integer index from [0, picongpu::parameters::N_observer) and maps it to a 3D unit vector in $\mathbb{R}^3$ (norm=1) space that describes the observation direction in the PIConGPU cartesian coordinate system.

namespace picongpu

namespace plugins

namespace radiation

namespace radiation_observer

Functions

HDINLINE vector_64 picongpu::plugins::radiation::radiation_observer::observation_direction(const int observation_id_extern)

Compute observation angles. This function is used in the Radiation plug-in kernel to compute the observation directions given as a unit vector pointing towards a 'virtual' detector.

This default setup is an example of a 2D detector array. It computes observation directions for 2D virtual detector field with its center pointing toward the +y direction (for theta=0, phi=0) with observation angles ranging from theta = [angle_theta_start : angle_theta_end] phi = [angle_phi_start : angle_phi_end] Every observation_id_extern index moves the phi angle from its start value toward its end value until the observation_id_extern reaches N_split. After that the theta angle moves further from its start value towards its end value while phi is reset to its start value.

The unit vector pointing towards the observing virtual detector can be described using theta and phi by:  
x_value = sin(theta) * cos(phi)  
y_value = cos(theta)  
z_value = sin(theta) * sin(phi)  
These are the standard spherical coordinates.

The example setup describes an detector array of 16x16 detectors ranging from -pi/8= -22.5 degrees to +pi/8= +22.5 degrees for both angles with the center pointing toward the y-axis (laser propagation direction).

Return unit vector pointing in observation direction type: vector_64

Parameters

- observation_id_extern: int index that identifies each block on the GPU to compute the observation direction

png.param

Defines

EM_FIELD_SCALE_CHANNEL1

2.3. .param Files 179
namespace picongpu

Variables

constexpr float_64 scale_image = 1.0
constexpr bool scale_to_cellsize = true
constexpr bool white_box_per_GPU = false

namespace visPreview

Functions

DINLINE float_X picongpu::visPreview::preChannel1(const float3_X & field_B, const float3_X & field_E, const float3_X & field_J)
DINLINE float_X picongpu::visPreview::preChannel2(const float3_X & field_B, const float3_X & field_E, const float3_X & field_J)
DINLINE float_X picongpu::visPreview::preChannel3(const float3_X & field_B, const float3_X & field_E, const float3_X & field_J)

Variables

constexpr float_X picongpu::visPreview::preParticleDens_opacity = 0.25_X
constexpr float_X picongpu::visPreview::preChannel1_opacity = 1.0_X
constexpr float_X picongpu::visPreview::preChannel2_opacity = 1.0_X
constexpr float_X picongpu::visPreview::preChannel3_opacity = 1.0_X

namespace picongpu

namespace colorScales

namespace blue

Functions

HDINLINE void picongpu::colorScales::blue::addRGB(float3_X & img, const float_X value, const float_X opacity)

namespace gray

Functions

HDINLINE void picongpu::colorScales::gray::addRGB(float3_X & img, const float_X value, const float_X opacity)

namespace grayInv

Functions

HDINLINE void picongpu::colorScales::grayInv::addRGB(float3_X & img, const float_X value, const float_X opacity)

namespace green

Functions

HDINLINE void picongpu::colorScales::green::addRGB(float3_X & img, const float_X value, const float_X opacity)
Functions

```
HDINLINE void picongpu::colorScales::green::addRGB(float3_X & img, const float_X value, const float_X opacity)
```

namespace none

Functions

```
HDINLINE void picongpu::colorScales::none::addRGB(const float3_X &, const float_X, const float_X)
```

namespace red

Functions

```
HDINLINE void picongpu::colorScales::red::addRGB(float3_X & img, const float_X value, const float_X opacity)
```

transitionRadiation.param

Definition of frequency space, number of observers, filters and form factors of the transition radiation plugin.

All values set here determine what the radiation plugin will compute. On the command line the plugin still needs to be called for each species the transition radiation should be computed for.

Defines

`PIC_VERBOSE_RADIATION`
Uses the same verbose level schemes as the radiation plugin.

```
radiation verbose level: 0=nothing, 1=physics, 2=simulation_state, 4=memory, 8=critical
```

namespace picongpu

namespace plugins

namespace radiation

```namespace radFormFactor_CIC_3D
  correct treatment of coherent and incoherent radiation from macro particles

  Choose different form factors in order to consider different particle shapes for radiation
  • radFormFactor_CIC_3D ... CIC charge distribution
  • radFormFactor_TSC_3D ... TSC charge distribution
  • radFormFactor_PCS_3D ... PCS charge distribution
  • radFormFactor_CIC_1Dy ... only CIC charge distribution in y
  • radFormFactor_Gauss_spherical ... symmetric Gauss charge distribution
  • radFormFactor_Gauss_cell ... Gauss charge distribution according to cell size
  • radFormFactor_incoherent ... only incoherent radiation
  • radFormFactor_coherent ... only coherent radiation
```

namespace transitionRadiation

Typedefs

```
using GammaFilter = picongpu::particles::manipulators::generic::Free<GammaFilterFunctor>
```

filter to (de)select particles for the radiation calculation

to activate the filter:
Functions

HDINLINE float3_X picongpu::plugins::transitionRadiation::observationDirection(const int observation_id_extern)
Compute observation angles.

This function is used in the transition radiation plugin kernel to compute the observation directions given as a unit vector pointing towards a ‘virtual’ detector.

This default setup is an example of a 2D detector array. It computes observation directions for 2D virtual detector field with its center pointing toward the +y direction (for theta=0, phi=0) with observation angles ranging from theta = [angle_theta_start : angle_theta_end] phi = [angle_phi_start : angle_phi_end] Every observation_idExtern index moves the phi angle from its start value toward its end value until the observation_idExtern reaches N_split. After that the theta angle moves further from its start value towards its end value while phi is reset to its start value.

The unit vector pointing towards the observing virtual detector can be described using theta and phi by: x_value = sin(theta) * cos(phi) y_value = cos(theta) z_value = sin(theta) * sin(phi) These are the standard spherical coordinates.

The example setup describes a detector array of 128x128 detectors ranging from 0 to pi for the azimuth angle theta and from 0 to 2 pi for the polar angle phi.

If the calculation is only supposed to be done for a single azimuth or polar angle, it will use the respective minimal angle.

Return unit vector pointing in observation direction type: float3_X

Parameters
  * observation_id_extern: int index that identifies each block on the GPU to compute the observation direction

struct GammaFilterFunctor
element of a filter for the relativistic Lorentz factor gamma

Public Functions

template<typename T_Particle> HDINLINE void picongpu::plugins::transitionRadiation::GammaFilterFunctor::operator()(T_Particle & particle)

Public Static Attributes

constexpr float_X filterGamma = 5.0
Gamma value above which the radiation is calculated.

namespace linearFrequencies
units for linear frequencies distribution for transition radiation plugin

Variables

constexpr unsigned int nOmega = 512
number of frequency values to compute in the linear frequency [unitless]

namespace SI
Variables

```cpp
constexpr float_64 omegaMin = 0.0
    minimum frequency of the linear frequency scale in units of [1/s]
constexpr float_64 omegaMax = 1.06e16
    maximum frequency of the linear frequency scale in units of [1/s]
```

namespace listFrequencies
units for frequencies from list for transition radiation calculation

Variables

```cpp
constexpr char listLocation[] = "/path/to/frequency_list"
    path to text file with frequencies
constexpr unsigned int nOmega = 512
    number of frequency values to compute if frequencies are given in a file [unitless]
```

namespace logFrequencies
units for logarithmic frequencies distribution for transition radiation plugin

Variables

```cpp
constexpr unsigned int nOmega = 256
    number of frequency values to compute in the logarithmic frequency [unitless]
```

namespace SI

Variables

```cpp
constexpr float_64 omegaMin = 1.0e13
    minimum frequency of the logarithmic frequency scale in units of [1/s]
constexpr float_64 omegaMax = 1.0e17
    maximum frequency of the logarithmic frequency scale in units of [1/s]
```

namespace parameters
selected mode of frequency scaling:

unit for foil position

options:
- linearFrequencies
- logFrequencies
- listFrequencies correct treatment of coherent radiation from macro particles
These formfactors are the same as in the radiation plugin! Choose different form factors in order to consider different particle shapes for radiation

- ::picongpu::plugins::radiation::radFormFactor_CIC_3D ... CIC charge distribution
- ::picongpu::plugins::radiation::radFormFactor_TSC_3D ... TSC charge distribution
- ::picongpu::plugins::radiation::radFormFactor_PCS_3D ... PCS charge distribution
- ::picongpu::plugins::radiation::radFormFactor_CIC_1Dy ... only CIC charge distribution in y
- ::picongpu::plugins::radiation::radFormFactor_Gauss_spherical ... symmetric Gauss charge distribution
- ::picongpu::plugins::radiation::radFormFactor_Gauss_cell ... Gauss charge distribution according to cell size
- ::picongpu::plugins::radiation::radFormFactor_incoherent ... only incoherent radiation
- ::picongpu::plugins::radiation::radFormFactor_coherent ... only coherent radiation
Variables

```cpp
cconstexpr unsigned int nPhi = 128
   Number of observation directions.
   If nPhi or nTheta is equal to 1, the transition radiation will be calculated for phiMin or thetaMin respectively.

cconstexpr unsigned int nTheta = 128

cconstexpr unsigned int nObserver = nPhi * nTheta

cconstexpr float_64 thetaMin = 0.0

cconstexpr float_64 thetaMax = picongpu::PI

cconstexpr float_64 phiMin = 0.0

cconstexpr float_64 phiMax = 2 * picongpu::PI
```

```cpp
namespace SI
{

Variables

```cpp
cconstexpr float_64 foilPosition = 0.0
```

Misc

**starter.param**

**random.param**

Configure the pseudorandom number generator (PRNG).

Allows to select method and global seeds in order to vary the initial state of the parallel PRNG.

```cpp
namespace picongpu
{

namespace random
{

Typedefs

```cpp
using Generator = pmacc::random::methods::XorMin<>  
   Random number generation methods.
   It is not allowed to change the method and restart an already existing checkpoint.

   • pmacc::random::methods::XorMin
   • pmacc::random::methods::MRG32k3aMin
   • pmacc::random::methods::AlpakaRand

using SeedGenerator = seed::Value<42>
   random number start seed
   Generator to create a seed for the random number generator. Depending of the generator the seed is reproducible or or changed with each program execution.

   • seed::Value< 42 >
   • seed::FromTime
```
namespace picongpu

Variables

constexpr float_64 PI = 3.141592653589793238462643383279502884197169399

constexpr float_64 UNIT_SPEED = SI::SPEED_OF_LIGHT_SI
    Unit of speed.

constexpr float_64 SPEED_OF_LIGHT = float_X(SI::SPEED_OF_LIGHT_SI / UNIT_SPEED)

constexpr float_64 UNIT_CONV_keV_to_Joule = 1.60217646e-16

constexpr float_64 UNIT_CONV_Joule_to_keV = (1.0 / UNIT_CONV_keV_to_Joule)

constexpr float_64 UNIT_CONV_AU_to_eV = 27.21139

constexpr float_64 UNIT_CONV_eV_to_AU = (1.0 / UNIT_CONV_AU_to_eV)

namespace SI

Variables

constexpr float_64 SPEED_OF_LIGHT_SI = 2.99792458e8
    unit: m / s

constexpr float_64 MUE0_SI = PI * 4.e-7
    unit: N / A^2

constexpr float_64 EPS0_SI = 1.0 / MUE0_SI / SPEED_OF_LIGHT_SI / SPEED_OF_LIGHT_SI
    unit: C / (V m)

constexpr float_64 Z0_SI = MUE0_SI * SPEED_OF_LIGHT_SI
    impedance of free space unit: ohm

constexpr float_64 HBAR_SI = 1.054571800e-34
    reduced Planck constant unit: J * s

constexpr float_64 ELECTRON_MASS_SI = 9.109382e-31
    unit: kg

constexpr float_64 ELECTRON_CHARGE_SI = -1.602176e-19
    unit: C

constexpr float_64 ATOMIC_UNIT_ENERGY = 4.36e-18

constexpr float_64 ATOMIC_UNIT_EFIELD = 5.14e11

constexpr float_64 ATOMIC_UNIT_TIME = 2.4189e-17

constexpr float_64 N_AVOGADRO = 6.02214076e23
    Avogadro number unit: mol^-1.

Y. Azuma et al. Improved measurement results for the Avogadro constant using a 28-Si-enriched crystal, Metrologie 52, 2015, 360-375 doi:10.1088/0026-1394/52/2/360
2.3.5 Python Generator (Third party)

PoGit is a utility to generate a set of .param files and a .cfg file using a Pythonic API. PoGit is a third-party development and supports only a subset of PIConGPU compile- and run-time settings. However, the resulting output can serve as a basis to be edited as normal .param files.

2.4 Plugins

<table>
<thead>
<tr>
<th>Plugin name</th>
<th>short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADIOS</td>
<td>stores simulation data as openPMD flavoured ADIOS files [Huebl2017]</td>
</tr>
<tr>
<td>energy histogram</td>
<td>energy histograms for electrons and ions</td>
</tr>
<tr>
<td>charge conservation</td>
<td>maximum difference between electron charge density and div E</td>
</tr>
<tr>
<td>checkpoint</td>
<td>stores the primary data of the simulation for restarts.</td>
</tr>
<tr>
<td>count particles</td>
<td>count total number of macro particles</td>
</tr>
<tr>
<td>count per supercell</td>
<td>count macro particles per supercell</td>
</tr>
<tr>
<td>energy fields</td>
<td>electromagnetic field energy per time step</td>
</tr>
<tr>
<td>energy particles</td>
<td>kinetic and total energies summed over all electrons and/or ions</td>
</tr>
<tr>
<td>HDF5</td>
<td>stores simulation data as openPMD flavoured HDF5 files [Huebl2017]</td>
</tr>
<tr>
<td>ISAAC</td>
<td>interactive 3D live visualization [Matthes2016]</td>
</tr>
<tr>
<td>intensity</td>
<td>maximum and integrated electric field along the y-direction</td>
</tr>
<tr>
<td>particle calorimeter</td>
<td>spatially resolved, particle energy detector in infinite distance</td>
</tr>
<tr>
<td>particle merger</td>
<td>macro particle merging</td>
</tr>
<tr>
<td>phase space</td>
<td>calculate 2D phase space [Huebl2014]</td>
</tr>
<tr>
<td>PNG</td>
<td>pictures of 2D slices</td>
</tr>
<tr>
<td>positions particles</td>
<td>save trajectory, momentum, … of a single particle</td>
</tr>
<tr>
<td>radiation</td>
<td>compute emitted electromagnetic spectra [Pausch2012] [Pausch2014] [Pausch2018]</td>
</tr>
<tr>
<td>resource log</td>
<td>monitor used hardware resources &amp; memory</td>
</tr>
<tr>
<td>slice emittance</td>
<td>compute emittance and slice emittance of particles</td>
</tr>
<tr>
<td>slice field printer</td>
<td>print out a slice of the electric and/or magnetic and/or current field</td>
</tr>
<tr>
<td>sum currents</td>
<td>compute the total current summed over all cells</td>
</tr>
<tr>
<td>transitionRadiation</td>
<td>compute emitted electromagnetic spectra</td>
</tr>
</tbody>
</table>

2.4.1 ADIOS

Stores simulation data such as fields and particles as ADIOS files or ADIOS staging methods [Huebl2017].

External Dependencies

The plugin is available as soon as the ADIOS library is compiled in.

.param file

The corresponding .param file is fileOutput.param.

One can e.g. disable the output of particles by setting:

---
2 Either ADIOS or HDF5 is required for simulation restarts. If both are available, writing checkpoints with ADIOS is automatically preferred by the simulation.

3 Multi-Plugin: Can be configured to run multiple times with varying parameters.

5 Only runs on the CUDA backend (GPU).

4 Requires HDF5 for output.

1 On restart, plugins with that footnote overwrite their output of previous runs. Manually save the created files of these plugins before restarting in the same directory.

5 Deprecated

4 Can remember particles that left the box at a certain time step.

---
/* output all species */
using FileOutputParticles = VectorAllSpecies;
/* disable */
using FileOutputParticles = MakeSeq_t<>;

**.cfg file**

You can use --adios.period and --adios.file to specify the output period and path and name of the created fileset. For example, --adios.period 128 --adios.file simData --adios.
source 'species_all' will write only the particle species data to files of the form simData_0.bp, simData_128.bp in the default simulation output directory every 128 steps. Note that this plugin will only be available if ADIOS is found during compile configuration.

<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--adios.period</td>
<td>Period after which simulation data should be stored on disk.</td>
</tr>
<tr>
<td>--adios.file</td>
<td>Relative or absolute fileset prefix for simulation data. If relative, files are stored under simOutput.</td>
</tr>
<tr>
<td>--adios.compression</td>
<td>Set data transform compression method. See adios_config -m for which compression methods are available. This flag also influences compression for checkpoints.</td>
</tr>
<tr>
<td>--adios.aggregators</td>
<td>Set number of I/O aggregator nodes for ADIOS MPI_AGGREGATE transport method.</td>
</tr>
<tr>
<td>--adios.ost</td>
<td>Set number of I/O OSTs for ADIOS MPI_AGGREGATE transport method.</td>
</tr>
<tr>
<td>--adios.transport-param</td>
<td>Further options for transports, see ADIOS manual chapter 6.1.5. Lustre example: random_offset=1;stripe_count=4 (FS chooses OST; user chooses striping factor).</td>
</tr>
<tr>
<td>--adios.disable-meta</td>
<td>Disable on-the-fly creation of the adios journal file. Allowed values: 0 means write a journal file, 1 skips its generation.</td>
</tr>
<tr>
<td>--adios.source</td>
<td>Select data sources to dump. Default is species_all,fields_all, which dumps all fields and particle species.</td>
</tr>
</tbody>
</table>

**Note:** This plugin is a multi plugin. Command line parameter can be used multiple times to create e.g. dumps with different dumping period. In the case where a optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--adios.period 128 --adios.file simData1 --adios.source 'species_all'
--adios.period 1000 --adios.file simData2 --adios.source 'fields_all' --adios.
   disable-meta 1
```

creates two plugins:

1. dump all species data each 128th time step, **do not create** the adios journal meta file.
2. dump all field data each 1000th time step but **create** the adios journal meta file.

**Compression**

ADIOS supports various on-the-fly compression methods. Typical options:

```
# single-threaded, slow zlib
--adios.compression zlib
```

(continues on next page)
See the ADIOS manual, chapter 8.2 for full details.

See adios_config --m for available compression methods and recompile ADIOS with further dependencies if needed. Typically, ADIOS adds compressors during the configure step with options such as --with-zlib=<ZLIB_DIR> and --with-blosc=<BLOSC_DIR>.

Meta Files

Disabling on-the-fly meta (journal) file creation can improve output performance for large scale runs. After your simulation finished, make sure to run bpmeta <theoretical-meta-fileName> on created ADIOS output.

You also need to create the meta file if you skipped on-the-fly creation in checkpointing and want to restart from such a checkpoint (with ADIOS as IO backend).

Example:

```
ls simOutput/
# bp checkpoints [...]
ls simOutput/{bp,checkpoints}
# simOutput/bp:
# simData_0.bp.dir simData_100.bp.dir [...]
# simOutput/checkpoints:
# checkpoint_0.bp.dir checkpoint_2000.bp.dir

cd simOutput/bp
bpmeta simData_0.bp
bpmeta simData_100.bp
# [...]
cd ../checkpoints
bpmeta checkpoint_0.bp
bpmeta checkpoint_2000.bp

ls simOutput/{bp,checkpoints}
# simOutput/bp:
# simData_0.bp simData_0.bp.dir
# simData_100.bp simData_100.bp.dir [...]
# simOutput/checkpoints:
# checkpoint_0.bp checkpoint_0.bp.dir
# checkpoint_2000.bp checkpoint_2000.bp.dir
```

Memory Complexity

**Accelerator**

no extra allocations.

**Host**

as soon as ADIOS is compiled in, one extra mallocMC heap for the particle buffer is permanently reserved. During I/O, particle attributes are allocated one after another.
Additional Tools

See our openPMD chapter.

References

2.4.2 Charge Conservation

First the charge density of all species with respect to their shape function is computed. Then this charge density is compared to the charge density computed from the divergence of the electric field $\nabla \vec{E}$. The maximum deviation value multiplied by the cell’s volume is printed.

**Attention:** This plugin assumes a Yee-like divergence $E$ stencil! Do not use it together with other field solvers like directional splitting (for the Lehe solver it is still correct).

.cfg file

PIConGPU command line argument (for .cfg files):

```bash
--chargeConservation.period <periodOfSteps>
```

Memory Complexity

**Accelerator**

no extra allocations (needs at least one FieldTmp slot).

**Host**

negligible.

Output and Analysis Tools

A new file named `chargeConservation.dat` is generated:

```
# timestep max-charge-deviation unit[As]
0 7.59718e-06 5.23234e-17
100 8.99187e-05 5.23234e-17
200 0.000113926 5.23234e-17
300 0.00014836 5.23234e-17
400 0.000154502 5.23234e-17
500 0.000164952 5.23234e-17
```

The charge is normalized to `UNIT_CHARGE` (third column) which is the typical charge of one macro-particle. There is a up 5% difference to a native hdf5 post-processing based implementation of the charge conversation check due to a different order of subtraction. And the zero-th time step (only numerical differences) might differ more then 5% relative due to the close to zero result.
2.4.3 Checkpoint

Stores the primary data of the simulation for restarts. Primary data includes:

- electro-magnetic fields
- particle attributes
- state of random number generators and particle ID generator
- ...

**Note:** Some plugins have their own internal state. They will be notified on checkpoints to store their state themselves.

**What is the format of the created files?**

We write our fields and particles in an open markup called *openPMD*. For further details, see the according sections in *HDF5* and *ADIOS*.

**External Dependencies**

The plugin is available as soon as the *libSplash (HDF5)* or *ADIOS libraries* are compiled in.

**.cfg file**

You can use `--checkpoint.period` to specify the output period of the created checkpoints. Note that this plugin will only be available if libSplash (HDF5) or ADIOS is found during compile configuration.
<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--checkpoint.period &lt;N&gt;</td>
<td>Create checkpoints every N steps.</td>
</tr>
<tr>
<td>--checkpoint.backend &lt;IO-backend&gt;</td>
<td>IO-backend used to create the checkpoint.</td>
</tr>
<tr>
<td>--checkpoint.directory &lt;string&gt;</td>
<td>Directory inside simOutput for writing checkpoints. Default is checkpoints.</td>
</tr>
<tr>
<td>--checkpoint.file &lt;string&gt;</td>
<td>Relative or absolute fileset prefix for writing checkpoints. If relative, checkpoint files are stored under simOutput/&lt;checkpoint-directory&gt;. Default depends on the selected IO-backend.</td>
</tr>
<tr>
<td>--checkpoint.restart</td>
<td>Restart a simulation from the latest checkpoint.</td>
</tr>
<tr>
<td>--checkpoint.restart.step &lt;N&gt;</td>
<td>Select a specific restart checkpoint.</td>
</tr>
<tr>
<td>--checkpoint.restart.backend &lt;IO-backend&gt;</td>
<td>IO-backend used to load an existent checkpoint.</td>
</tr>
<tr>
<td>--checkpoint.restart.directory &lt;string&gt;</td>
<td>Directory inside simOutput containing checkpoints for a restart. Default is checkpoints.</td>
</tr>
<tr>
<td>--checkpoint.restart.file &lt;string&gt;</td>
<td>Relative or absolute fileset prefix for reading checkpoints. If relative, checkpoint files are searched under simOutput/&lt;checkpoint-directory&gt;. Default depends on the selected IO-backend&quot;.</td>
</tr>
<tr>
<td>--checkpoint.restart.chunkSize &lt;N&gt;</td>
<td>Number of particles processed in one kernel call during restart to prevent frame count blowup.</td>
</tr>
<tr>
<td>--checkpoint.restart.loop &lt;N&gt;</td>
<td>Number of times to restart the simulation after simulation has finished. This mode is intended for visualization and not all plugins support it.</td>
</tr>
<tr>
<td>--checkpoint.&lt;IO-backend&gt;*</td>
<td>Additional options to control the IO-backend</td>
</tr>
</tbody>
</table>

Depending on the available external dependencies (see above), the options for the <IO-backend> are:

- hdf5
- adios (keep in mind the note on meta-files for restarts)

**Interacting Manually with Checkpoint Data**

**Note:** Interacting with the raw data of checkpoints for manual manipulation is considered an advanced feature for experienced users.

Contrary to regular output, checkpoints contain additional data which might be confusing on the first glance. For example, some comments might be missing, all data from our concept of slides for moving window simulations will be visible, additional data for internal states of helper classes is stored as well and index tables such as openPMD particle patches are essential for parallel restarts.

### 2.4.4 Count Particles

This plugin counts the total number of *macro particles associated with a species* and writes them to a file for specified time steps. It is used mainly for debugging purposes. Only in case of constant particle density, where
each macro particle describes the same number of real particles (weighting), conclusions on the plasma density can be drawn.

.cfg file

The CountParticles plugin is always complied for all species. By specifying the periodicity of the output using the command line argument

```
--e_macroParticlesCount.period
```

(here for an electron species called e) with picongpu, the plugin is enabled. Setting

```
--e_macroParticlesCount.period 100
```

adds the number of all electron macro particles to the file ElectronsCount.dat for every 100th time step of the simulation.

Memory Complexity

Accelerator

no extra allocations.

Host

negligible.

Output

In the output file e_macroParticlesCount.dat there are three columns. The first is the integer number of the time step. The second is the number of macro particles as integer - useful for exact counts. And the third is the number of macro particles in scientific floating point notation - provides better human readability.

Known Issues

Currently, the file e_macroParticlesCount.dat is overwritten when restarting the simulation. Therefore, all previously stored counts are lost.

2.4.5 Count per Supercell

This plugin counts the total number of macro particles of a species for each super cell and stores the result in an hdf5 file. Only in case of constant particle weighting, where each macro particle describes the same number of real particles, conclusions on the plasma density can be drawn.

External Dependencies

The plugin is available as soon as the libSplash and HDF5 libraries are compiled in.

.cfg files

By specifying the periodicity of the output using the command line argument

```
--e_macroParticlesPerSuperCell.period
```

(here for an electron species e) with picongpu the plugin is enabled. Setting

```
--e_macroParticlesPerSuperCell.period 100
```

adds the number of all electron macro particles to the file e_macroParticlesCount.dat for every 100th time step of the simulation.
Accelerator

an extra permanent allocation of size_t for each local supercell.

Host

negligible.

Output

The output is stored as hdf5 file in a separate directory.

2.4.6 Energy Fields

This plugin computes the total energy contained in the electric and magnetic field of the entire volume simulated. The energy is computed for user specified time steps.

.cfg file

By setting the PIConGPU command line flag --fields_energy.period to a non-zero value the plugin computes the total field energy. The default value is 0, meaning that the total field energy is not stored. By setting e.g. --fields_energy.period 100 the total field energy is computed for time steps 0, 100, 200, ...

Memory Complexity

Accelerator

negligible.

Host

negligible.

Output

The data is stored in fields_energy.dat. There are two columns. The first gives the time step. The second is the total field energy in Joule. The first row is a comment describing the columns:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.5e+18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.5e+18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>2.5e+18</td>
<td>2.45e-22</td>
<td>2.26e-08</td>
<td>2.24e-08</td>
<td>2.5e+18</td>
<td>2.29e-08</td>
<td>2.30e-08</td>
</tr>
</tbody>
</table>

Attention: The output of this plugin computes a sum over all cells in a very naive implementation. This can lead to significant errors due to the finite precision in floating-point numbers. Do not expect the output to be precise to more than a few percent. Do not expect the output to be deterministic due to the statistical nature of the implemented reduce operation.

Please see this issue for a longer discussion and possible future implementations.
Example Visualization

Python example snippet:

```python
import numpy as np
import matplotlib.pyplot as plt

simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]

# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot(np.array([e_sum_ene[:,0],
                   label="sum")
plt.legend()
plt.show()
```

2.4.7 Energy Histogram

This plugin computes the energy histogram (spectrum) of a selected particle species and stores it to plain text files. The acceptance of particles for counting in the energy histogram can be adjusted, e.g. to model the limited acceptance of a realistic spectrometer.

.param file

The `particleFilters.param` file allows to define accepted particles for the energy histogram. A typical filter could select particles within a specified opening angle in forward direction.

.cfg files

There are several command line parameters that can be used to set up this plugin. Replace the prefix `e` for electrons with any other species you have defined, we keep using `e` in the examples below for simplicity. Currently, the plugin can be set once for each species.
### PIConGPU Command Line Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--e_energyHistogram.period</td>
<td>The output periodicity of the electron histogram. A value of 100 would mean an output at simulation time step 0, 100, 200, .... If set to a non-zero value, the energy histogram of all electrons is computed. By default, the value is 0 and no histogram for the electrons is computed.</td>
</tr>
<tr>
<td>--e_energyHistogram.filter</td>
<td>Use filtered particles. Available filters are set up in particleFilters.param.</td>
</tr>
<tr>
<td>--e_energyHistogram.binCount</td>
<td>Specifies the number of bins used for the electron histogram. Default is 1024.</td>
</tr>
<tr>
<td>--e_energyHistogram.minEnergy</td>
<td>Set the minimum energy for the electron histogram in keV. Default is 0, meaning 0 keV.</td>
</tr>
<tr>
<td>--e_energyHistogram.maxEnergy</td>
<td>Set the maximum energy for the electron histogram in keV. There is no default value. This has to be set by the user if --e_energyHistogram.period 1 is set.</td>
</tr>
</tbody>
</table>

**Note:** This plugin is a multi plugin. Command line parameter can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. For example,

```
--e_energyHistogram.period 128 --e_energyHistogram.filter all --e_energyHistogram.
--maxEnergy 10
--e_energyHistogram.period 100 --e_energyHistogram.filter all --e_energyHistogram.
--maxEnergy 20 --e_energyHistogram.binCount 512
```

creates two plugins:

1. create an electron histogram with **512 bins** each 128th time step.
2. create an electron histogram with **1024 bins** (this is the default) each 100th time step.

### Memory Complexity

**Accelerator**

an extra array with the number of bins.

**Host**

negligible.

### Output

The histograms are stored in ASCII files in the simOutput/ directory.

The file for the electron histogram is named e_energyHistogram.dat and for all other species <species>_energyHistogram.dat likewise. The first line of these files does not contain histogram data and is commented-out using #. It describes the energy binning that needed to interpret the following data. It can be seen as the head of the following data table. The first column is an integer value describing the simulation time step. The second column counts the number of real particles below the minimum energy value used for the histogram. The following columns give the real electron count of the particles in the specific bin described by the first line/header. The second last column gives the number of real particles that have a higher energy than the maximum energy used for the histogram. The last column gives the total number of particles. In total there are 4 columns more than the number of bins specified with command line arguments. Each row describes another simulation time step.

---

2.4. Plugins 195
Analysis Tools

Data Reader

You can quickly load and interact with the data in Python with:

```python
from picongpu.plugins.data import EnergyHistogramData

eh_data = EnergyHistogramData('/home/axel/runs/lwfa_001')

# show available iterations
eh_data.get_iterations(species='e')

# show available simulation times
eh_data.get_times(species='e')

# load data for a given iteration
counts, bins_keV = eh_data.get('e', species_filter='all', iteration=2000)

# load data for a given time
counts, bins_keV = eh_data.get('e', species_filter='all', time=1.3900e-14)

# get data for multiple iterations
d, bins, iteration, dt = eh_data.get(species='e', iteration=[200, 400, 8000])
```

Matplotlib Visualizer

You can quickly plot the data in Python with:

```python
from picongpu.plugins.plot_mpl import EnergyHistogramMPL
import matplotlib.pyplot as plt

# create a figure and axes
fig, ax = plt.subplots(1, 1)

# create the visualizer
eh_vis = EnergyHistogramMPL('path/to/run_dir', ax)

eh_vis.visualize(iteration=200, species='e')
plt.show()

# specifying simulation time is also possible (granted there is a matching
# iteration for that time)

eh_vis.visualize(time=2.6410e-13, species='e')
plt.show()

# plotting histogram data for multiple simulations simultaneously also works:

eh_vis = EnergyHistogramMPL([
    ('sim1', 'path/to/sim1'),
    ('sim2', 'path/to/sim2'),
    ('sim3', 'path/to/sim3')], ax)

eh_vis.visualize(species='e', iteration=10000)
plt.show()
```

The visualizer can also be used from the command line (for a single simulation only) by writing
python energy_histogram_visualizer.py

with the following command line options

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>Path to the run directory of a simulation.</td>
</tr>
<tr>
<td>-i</td>
<td>An iteration number</td>
</tr>
<tr>
<td>-s (optional, defaults to ‘e’)</td>
<td>Particle species abbreviation (e.g. ‘e’ for electrons)</td>
</tr>
<tr>
<td>-f (optional, defaults to ‘all’)</td>
<td>Species filter string</td>
</tr>
</tbody>
</table>

Alternatively, PIConGPU comes with a command line analysis tool for the energy histograms. It is based on gnuplot and requires that gnuplot is available via command line. The tool can be found in src/tools/bin/ and is called BinEnergyPlot.sh. It accesses the gnuplot script BinEnergyPlot.gnuplot in src/tools/share/gnuplot/. BinEnergyPlot.sh requires exactly three command line arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Path and filename to e_energyHistogram.dat file.</td>
</tr>
<tr>
<td>2nd</td>
<td>Simulation time step (needs to exist)</td>
</tr>
<tr>
<td>3rd</td>
<td>Label for particle count used in the graph that this tool produces</td>
</tr>
</tbody>
</table>

**Jupyter Widget**

If you want more interactive visualization, then start a jupyter notebook and make sure that ipywidgets and ipympl are installed.

After starting the notebook server write the following

```python
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
plt.ioff()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import EnergyHistogramWidget

w = EnergyHistogramWidget(run_dir_options=[
    (“scan1/sim4”, scan1_sim4),
    (“scan1/sim5”, scan1_sim5)])
display(w)
```

and then interact with the displayed widgets.

**2.4.8 Energy Particles**

This plugin computes the kinetic and total energy summed over all particles of a species for time steps specified.

**.cfg file**

Only the time steps at which the total kinetic energy of all particles should be specified needs to be set via command line argument.
PIConGPU command line option | Description
--- | ---
--e_energy.<br>period 100 | Sets the time step period at which the energy of all electrons in the simulation should be simulated. If set to e.g. 100, the energy is computed for time steps 0, 100, 200, .... The default value is 0, meaning that the plugin does not compute the particle energy.

--<species>_energy.<br>period 42 | Same as above, for any other species available.

--<species>_energy.<br>filter | Use filtered particles. All available filters will be shown with picongpu --help.

Memory Complexity

Accelerator

negligible.

Host

negligible.

Output

The plugin creates files prefixed with the species’ name and the filter name as postfix, e.g. e_energy_<filterName>.dat for the electron energies and p_energy_<filterName>.dat for proton energies. The file contains a header describing the columns.

```
#step Ekin_Joule E_Joule
0.0 0.0 0.0
```

Following the header, each line is the output of one time step. The time step is given as first value. The second value is the kinetic energy of all particles at that time step. And the last value is the total energy (kinetic + rest energy) of all particles at that time step.

**Attention:** The output of this plugin computes a sum over all particles in a very naive implementation. This can lead to significant errors due to the finite precision in floating-point numbers. Do not expect the output to be precise to more than a few percent. Do not expect the output to be deterministic due to the statistical nature of the implemented reduce operation.

Please see this issue for a longer discussion and possible future implementations.

Example Visualization

Python snippet:

```python
import numpy as np

simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
```

(continues on next page)
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]
# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot( 
    e_sum_ene[:,0],
    label="sum"
)
plt.legend()

2.4.9 HDF5

Stores simulation data such as fields and particles along with domain information, conversion units etc. as HDF5 files [Huebl2017]. It uses libSplash for writing HDF5 data. It is used for post-simulation analysis and for restarts of the simulation after a crash or an intended stop.

What is the format of the created HDF5 files?

We write our fields and particles in an open markup called openPMD. You can investigate your files via a large collection of tools and frameworks or use the native HDF5 bindings of your favorite programming language.

Resources for a quick-start:

• online tutorial
• example files
• written standard of the openPMD standard
• list of projects supporting openPMD files

External Dependencies

The plugin is available as soon as the libSplash and HDF5 libraries are compiled in.

.param file

The corresponding .param file is fileOutput.param.

One can e.g. disable the output of particles by setting:

```
/* output all species */
using FileOutputParticles = VectorAllSpecies;
/* disable */
using FileOutputParticles = MakeSeq_t< >;
```
You can use `--hdf5.period` and `--hdf5.file` to specify the output period and path and name of the created fileset. For example, `--hdf5.period 128 --hdf5.file simData --hdf5.source 'species_all'` will write only the particle species data to files of the form `simData_0.h5`, `simData_128.h5` in the default simulation output directory every 128 steps. Note that this plugin will only be available if libSplash and HDF5 is found during compile configuration.

<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--hdf5.period</code></td>
<td>Period after which simulation data should be stored on disk.</td>
</tr>
<tr>
<td><code>--hdf5.file</code></td>
<td>Relative or absolute fileset prefix for simulation data. If relative, files are stored under <code>simOutput/</code>.</td>
</tr>
<tr>
<td><code>--hdf5.source</code></td>
<td>Select data sources to dump. Default is <code>species_all,fields_all</code>, which dumps all fields and particle species.</td>
</tr>
</tbody>
</table>

**Note:** This plugin is a multi plugin. Command line parameter can be used multiple times to create e.g. dumps with different dumping period. In the case where a optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```bash
--hdf5.period 128 --hdf5.file simData1
--hdf5.period 1000 --hdf5.file simData2 --hdf5.source 'species_all'
```

creates two plugins:

1. dump **all species data** each 128th time step.
2. dump **all fields and species data** (this is the default) data each 1000th time step.

### Memory Complexity

**Accelerator**

no extra allocations.

**Host**

During I/O, each complete particle species is allocated one after an other.

### Additional Tools

See our openPMD chapter.

### References

#### 2.4.10 Intensity

The maximum amplitude of the electric field for each cell in y-cell-position in \( \text{V/m} \) and the integrated amplitude of the electric field (integrated over the entire x- and z-extent of the simulated volume and given for each y-cell-position).
Attention: There might be an error in the units of the integrated output.

Note: A renaming of this plugin would be very useful in order to understand its purpose more intuitively.

.cfg file

By setting the PIConGPU command line flag `--intensity.period` to a non-zero value the plugin computes the maximum electric field and the integrated electric field for each cell-wide slice in y-direction. The default value is 0, meaning that nothing is computed. By setting e.g. `--intensity.period 100` the electric field analysis is computed for time steps 0, 100, 200, ....

Memory Complexity

Accelerator

negligible.

Host

negligible.

Output

The output of the maximum electric field for each y-slice is stored in `Intensity_max.dat`. The output of the integrated electric field for each y-slice is stored in `Intensity_integrated.dat`.

Both files have two header rows describing the data. .. code:

```
#step_position_in_laser_propagation_direction
#step_amplitude_data[*]
```

The following odd rows give the time step and then describe the y-position of the slice at which the maximum electric field or integrated electric field is computed. The even rows give the time step again and then the data (maximum electric field or integrated electric field) at the positions given in the previews row.

Know Issues

Currently, the output file is overwritten after restart. Additionally, this plugin does not work with non-regular domains, see here. This will be fixed in a future version.

There might be an error in the units of the integrated output.

For a full list, see #327.

2.4.11 ISAAC

This is a plugin for the in-situ library ISAAC [Matthes2016] for a live rendering and steering of PIConGPU simulations.
External Dependencies

The plugin is available as soon as the ISAAC library is compiled in.

.cfg file

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--isaac.period N</td>
<td>Sets up, that every $N$ th timestep an image will be rendered. This parameter can be changed later with the controlling client.</td>
</tr>
<tr>
<td>--isaac.name NAME</td>
<td>Sets the NAME of the simulation, which is shown at the client.</td>
</tr>
<tr>
<td>--isaac.url URL</td>
<td>URL of the required and running isaac server. Host names and IPs are supported.</td>
</tr>
<tr>
<td>--isaac.port PORT</td>
<td>PORT of the isaac server. The default value is 2458 (for the in-situ plugins), but may be needed to be changed for tunneling reasons or if more than one server shall run on the very same hardware.</td>
</tr>
<tr>
<td>--isaac.width WIDTH</td>
<td>Setups the WIDTH and HEIGHT of the created image(s).</td>
</tr>
<tr>
<td>--isaac.height HEIGHT</td>
<td>Default is 1024x768.</td>
</tr>
<tr>
<td>--isaac.direct_pause</td>
<td>If activated ISAAC will pause directly after the simulation started. Useful for presentations or if you don’t want to miss the beginning of the simulation.</td>
</tr>
<tr>
<td>--isaac.quality QUALITY</td>
<td>Sets the QUALITY of the images, which are compressed right after creation. Values between 1 and 100 are possible. The default is 90, but 70 does also still produce decent results.</td>
</tr>
</tbody>
</table>

The most important settings for ISAAC are --isaac.period, --isaac.name and --isaac.url. A possible addition for your submission tbg file could be --isaac.period 1 --isaac.name !TBG_jobName --isaac.url YOUR_SERVER, where the tbg variables !TBG_jobName is used as name and YOUR_SERVER needs to be set up by yourself.

.param file

The ISAAC Plugin has an isaac.param, which specifies which fields and particles are rendered. This can be edited (in your local paramSet), but at runtime also an arbitrary amount of fields (in ISAAC called sources) can be deactivated. At default every field and every known species are rendered.

Running and steering a simulation

First of all you need to build and run the isaac server somewhere. On HPC systems, simply start the server on the login or head node since it can be reached by all compute nodes (on which the PIConGPU clients will be running).

Functor Chains

One of the most important features of ISAAC are the Functor Chains. As most sources (including fields and species) may not be suited for a direct rendering or even full negative (like the electron density field), the functor chains enable you to change the domain of your field source-wise. A date will be read from the field, the functor chain applied and then only the x-component used for the classification and later rendering of the scene. Multiply functors can be applied successive with the Pipe symbol |. The possible functors are at default:
• **mul** for a multiplication with a constant value. For vector fields you can choose different value per component, e.g. `mul(1,2,0)`, which will multiply the x-component with 1, the y-component with 2 and the z-component with 0. If less parameters are given than components exists, the last parameter will be used for all components without an own parameter.

• **add** for adding a constant value, which works the same as `mul(...)`. 

• **sum** for summarizing all available components. Unlike `mul(...)` and `add(...)` this decreases the dimension of the data to 1, which is a scalar field. You can exploit this functor to use a different component than the x-component for the classification, e.g. with `mul(0,1,0) | sum`. This will first multiply the x- and z-component with 0, but keep the y-component and then merge this to the x-component.

• **length** for calculating the length of a vector field. Like `sum` this functor reduces the dimension to a scalar field, too. However `mul(0,1,0) | sum` and `mul(0,1,0) | length` do not do the same. As `length` does not know, that the x- and z-component are 0 an expensive square root operation is performed, which is slower than just adding the components up.

• **idem** does nothing, it just returns the input data. This is the default functor chain.

Beside the functor chains the client allows to setup the weights per source (values greater than 6 are more useful for PIConGPU than the default weights of 1), the classification via transfer functions, clipping, camera steering and to switch the render mode to iso surface rendering. Furthermore interpolation can be activated. However this is quite slow and most of the time not needed for non-iso-surface rendering.

**Memory Complexity**

**Accelerator**

locally, a framebuffer with full resolution and 4 byte per pixel is allocated. For each `FieldTmp` derived field and `FieldJ` a copy is allocated, depending on the input in the `isaac.param` file.

**Host**

negligible.
Example renderings
References

2.4.12 Particle Calorimeter

A binned calorimeter of the amount of kinetic energy per solid angle and energy-per-particle. The solid angle bin is solely determined by the particle’s momentum vector and not by its position, so we are emulating a calorimeter at infinite distance.

The calorimeter takes into account all existing particles as well as optionally all particles which have already left the global simulation volume.

External Dependencies

The plugin is available as soon as the *libSplash* and *HDF5* libraries are compiled in.

*.param file*

The spatial calorimeter resolution can be customized and in *speciesDefinition.param*. Therein, a species can be also be marked for detecting particles leaving the simulation box.

*.cfg file*

All options are denoted exemplarily for the photon (*ph*) particle species here.

<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ph_calorimeter.period</td>
<td>The output periodicity of the plugin. A value of 100 would mean an output at simulation time step 0, 100, 200, ...</td>
</tr>
<tr>
<td>--ph_calorimeter.file</td>
<td>Output file suffix. Put unique name if same species + filter is used multiple times.</td>
</tr>
<tr>
<td>--ph_calorimeter.filter</td>
<td>Use filtered particles. All available filters will be shown with picongpu --help</td>
</tr>
<tr>
<td>--ph_calorimeter.numBinsYaw</td>
<td>Specifies the number of bins used for the yaw axis of the calorimeter. Defaults to 64.</td>
</tr>
<tr>
<td>--ph_calorimeter.numBinsPitch</td>
<td>Specifies the number of bins used for the pitch axis of the calorimeter. Defaults to 64.</td>
</tr>
<tr>
<td>--ph_calorimeter.numBinsEnergy</td>
<td>Specifies the number of bins used for the energy axis of the calorimeter. Defaults to 1, i.e. there is no energy binning.</td>
</tr>
<tr>
<td>--ph_calorimeter.minEnergy</td>
<td>Minimum detectable energy in keV. Ignored if numBinsEnergy is 1. Defaults to 0.</td>
</tr>
<tr>
<td>--ph_calorimeter.maxEnergy</td>
<td>Maximum detectable energy in keV. Ignored if numBinsEnergy is 1. Defaults to 1000.</td>
</tr>
<tr>
<td>--ph_calorimeter.logScale</td>
<td>En-/Disable logarithmic energy binning. Allowed values: 0 for disable, 1 enable.</td>
</tr>
<tr>
<td>--ph_calorimeter.openingYaw</td>
<td>opening angle yaw of the calorimeter in degrees. Defaults to the maximum value: 360.</td>
</tr>
<tr>
<td>--ph_calorimeter.posYaw</td>
<td>yaw coordinate of the calorimeter position in degrees. Defaults to the +y direction: 0.</td>
</tr>
<tr>
<td>--ph_calorimeter.posPitch</td>
<td>pitch coordinate of the calorimeter position in degrees. Defaults to the +y direction: 0.</td>
</tr>
</tbody>
</table>
Coordinate System

Yaw and pitch are Euler angles defining a point on a sphere’s surface, where\((0,0)\) points to the \(+y\) direction here. In the vicinity of \((0,0)\), yaw points to \(+x\) and pitch to \(+z\).

**Orientation detail:** Since the calorimeters’ three-dimensional orientation is given by just two parameters\((\text{posYaw} \text{ and posPitch})\) there is one degree of freedom left which has to be fixed. Here, this is achieved by eliminating the Euler angle roll. However, when\(\text{posPitch}\) is exactly \(+90\) or \(-90\) degrees, the choice of roll is ambiguous, depending on the yaw angle one approaches the singularity. Here we assume an approach from \(\text{yaw} = 0\).

**Tuning the spatial resolution**

By default, the spatial bin size is chosen by dividing the opening angle by the number of bins for yaw and pitch respectively. The bin size can be tuned by customizing the mapping function in \texttt{particleCalorimeter.param}.

**Memory Complexity**

**Accelerator**

Each energy bin times each coordinate bin allocates two counter\((\text{float}_X)\) permanently and on each accelerator for active and outgoing particles.

**Host**

As on accelerator.

**Output**

The calorimeters are stored in h5f-files in the \texttt{simOutput/<species>_calorimeter/<filter>} directory. The file names are \texttt{<species>_calorimeter_<file>_<sfilter>_<timestep>:_0_0_0.h5}.

The dataset within the h5f-file is located at \texttt{/data/<timestep>/calorimeter}. Depending on whether energy binning is enabled the dataset is two or three dimensional. The dataset has the following attributes:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unitSI</td>
<td>scaling factor for energy in calorimeter bins</td>
</tr>
<tr>
<td>maxYaw[deg]</td>
<td>half of the opening angle yaw.</td>
</tr>
<tr>
<td>maxPitch[deg]</td>
<td>half of the opening angle pitch.</td>
</tr>
<tr>
<td>posYaw[deg]</td>
<td>yaw coordinate of the calorimeter.</td>
</tr>
<tr>
<td>posPitch[deg]</td>
<td>pitch coordinate of the calorimeter. If energy binning is enabled:</td>
</tr>
<tr>
<td>minEnergy[keV]</td>
<td>minimal detectable energy.</td>
</tr>
<tr>
<td>maxEnergy[keV]</td>
<td>maximal detectable energy.</td>
</tr>
<tr>
<td>logScale</td>
<td>boolean indicating logarithmic scale.</td>
</tr>
</tbody>
</table>

The output in each bin is given in Joule. Divide by energy value of the bin for a unitless count per bin.

**Note:** This plugin is a multi plugin. Command line parameters can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--ph_calorimeter.period 128 --ph_calorimeter.file calo1 --ph_calorimeter.filter all
--ph_calorimeter.period 1000 --ph_calorimeter.file calo2 --ph_calorimeter.filter all
--ph_calorimeter.logScale 1 --ph_calorimeter.minEnergy 1
```

creates two plugins:

1. calorimeter for species ph each 128th time step **with** logarithmic energy binning.
2. calorimeter for species ph each 1000th time step **without** (this is the default) logarithmic energy binning.

**Attention:** When using the plugin multiple times for the same combination of species and filter, you **must** provide a unique file suffix. Otherwise output files will overwrite each other, since only species, filter and file suffix are encoded in it.

An example use case would be two (or more) calorimeters for the same species and filter but with differing position in space or different binning, range, linear and log scaling, etc.

**Analysis Tools**

The first bin of the energy axis of the calorimeter contains all particle energy less than the minimal detectable energy whereas the last bin contains all particle energy greater than the maximal detectable energy. The inner bins map to the actual energy range of the calorimeter.

Sample script for plotting the spatial distribution and the energy distribution:

```python
f = h5.File("<path-to-hdf5-file>")
calorimeter = np.array(f["/data/<timestep>/calorimeter"])  
# spatial energy distribution
# sum up the energy spectrum
plt.imshow(np.sum(calorimeter, axis=0))
plt.show()

# energy spectrum
# sum up all solid angles
plt.plot(np.sum(calorimeter, axis=(1,2)))
plt.show()
```
2.4.13 Particle Merger

Merges macro particles that are close in phase space to reduce computational load.

.param file

In `particleMerging.param` is currently one compile-time parameter:

<table>
<thead>
<tr>
<th>Compile-Time Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_VORONOI.Cells</td>
<td>Maximum number of active Voronoi cells per supercell. If the number of active Voronoi cells reaches this limit merging events are dropped.</td>
</tr>
</tbody>
</table>

.cfg file

<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--&lt;species&gt;_merger.period</td>
<td>The output periodicity of the plug-in. A value of 100 would mean an output at simulation time step 0, 100, 200, ....</td>
</tr>
<tr>
<td>--&lt;species&gt;_merger.minParticlesToMerge</td>
<td>minimal number of macroparticles needed to merge the macroparticle collection into a single macroparticle.</td>
</tr>
<tr>
<td>--&lt;species&gt;_merger.posSpreadThreshold</td>
<td>Below this threshold of spread in position macroparticles can be merged [unit: cell edge length].</td>
</tr>
<tr>
<td>--&lt;species&gt;_merger.absMomSpreadThreshold</td>
<td>Below this absolute threshold of spread in momentum macroparticles can be merged [unit: mₑ·c]. Disabled for -1 (default).</td>
</tr>
<tr>
<td>--&lt;species&gt;_merger.relMomSpreadThreshold</td>
<td>Below this relative (to mean momentum) threshold of spread in momentum macroparticles can be merged [unit: none]. Disabled for -1 (default).</td>
</tr>
<tr>
<td>--&lt;species&gt;_merger.minMeanEnergy</td>
<td>minimal mean kinetic energy needed to merge the macroparticle collection into a single macroparticle [unit: keV].</td>
</tr>
</tbody>
</table>

Notes

- absMomSpreadThreshold and relMomSpreadThreshold are mutually exclusive
- absMomSpreadThreshold is always given in [electron mass * speed of light]!

Memory Complexity

Accelerator

no extra allocations, but requires an extra particle attribute per species, voronoiCellId.

Host

no extra allocations.

Known Limitations

- this plugin is only available with the CUDA backend
- this plugin might take a significant amount of time due to not being fully parallelized.
Reference

The particle merger implements a macro particle merging algorithm based on:

There is a slight deviation from the paper in determining the next subdivision. The implementation always tries to subdivide a Voronoi cell by positions first; momentums are only checked in case the spreads in the positions satisfy the threshold.

2.4.14 Phase Space

This plugin creates a 2D phase space image for a user-given spatial and momentum coordinate.

External Dependencies

The plugin is available as soon as the libSplash and HDF5 libraries are compiled in.

.cfg file

Example for y-pz phase space for the electron species (.cfg file macro):

```plaintext
# Calculate a 2D phase space
# - momentum range in m_e c
TGB_ePSypz="--e_phaseSpace.period 10 --e_phaseSpace.filter all --e_phaseSpace.
˓space y --e_phaseSpace.momentum pz --e_phaseSpace.min -1.0 --e_phaseSpace.max 1.0
˓"
```

The distinct options are (assuming a species e for electrons):

<table>
<thead>
<tr>
<th>Option</th>
<th>Usage Unit</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>--e_phaseSpace.period &lt;N&gt;</td>
<td>calculate each N steps</td>
<td>none</td>
</tr>
<tr>
<td>--e_phaseSpace.filter</td>
<td>Use filtered particles. Available filters are set up in particleFilters.param.</td>
<td>none</td>
</tr>
<tr>
<td>--e_phaseSpace.space &lt;x/y/z&gt;</td>
<td>spatial coordinate of the 2D phase space</td>
<td>none</td>
</tr>
<tr>
<td>--e_phaseSpace.momentum &lt;px/py/pz&gt;</td>
<td>momentum coordinate of the 2D phase space</td>
<td>none</td>
</tr>
<tr>
<td>--e_phaseSpace.min &lt;ValL&gt;</td>
<td>minimum of the momentum range</td>
<td>m_{species}</td>
</tr>
<tr>
<td>--e_phaseSpace.max &lt;ValR&gt;</td>
<td>maximum of the momentum range</td>
<td>m_{species}</td>
</tr>
</tbody>
</table>

Memory Complexity

Accelerator

locally, a counter matrix of the size local-cells of space direction times 1024 (for momentum bins) is permanently allocated.

Host

negligible.
Output

The 2D histograms are stored in `.hdf5` files in the `simOutput/phaseSpace/` directory. A file is created per species, phasespace selection and time step.

Values are given as charge density per phase space bin. In order to scale to a simpler charge of particles per `dr` and `dp`, -bin multiply by the cell volume `dV`.

Analysis Tools

Data Reader

You can quickly load and interact with the data in Python with:

```python
from picongpu.plugins.data import PhaseSpaceData
import numpy as np

ps_data = PhaseSpaceData('/home/axel/runs/lwfa_001')
# show available iterations
ps_data.get_iterations(ps="xpx", species="e", species_filter='all')

# show available simulation times
ps_data.get_times(ps="xpx", species="e", species_filter='all')

# load data for a given iteration
ps, meta = ps_data.get(ps='ypy', species='e', species_filter='all', iteration=2000)

# unit conversion from SI
mu = 1.e6  # meters to microns
e_mc_r = 1. / (9.109e-31 * 2.9979e8)  # electrons: kg * m / s to beta * gamma
Q_dr_dp = np.abs(ps) * meta.dV  # C s kg^-1 m^-2
extent = meta.extent * [mu, mu, e_mc_r, e_mc_r]  # spatial: microns, momentum:

# load data for a given time
ps, ps_meta = ps_data.get(ps='xpx', species='e', species_filter='all', time=1.3900e-14)

# load data for multiple iterations
ret = ps_data.get(ps='xpx', species='e', species_filter='all', iteration=[2000, 4000])

# data and metadata for iteration 2000
# (data is in same order as the value passed to the 'iteration' parameter)
ps, meta = ret[0]
```

Note that the spatial extent of the output over time might change when running a moving window simulation.

Matplotlib Visualizer

You can quickly plot the data in Python with:

```python
from picongpu.plugins.plot_mpl import PhaseSpaceMPL
import matplotlib.pyplot as plt

# create a figure and axes
(continues on next page)
```
fig, ax = plt.subplots(1, 1)
# create the visualizer
ps_vis = PhaseSpaceMPL('path/to/run_dir', ax)
# plot
ps_vis.visualize(ps="xpx", iteration=200, species='e', species_filter='all')
plt.show()

# specifying simulation time is also possible (granted there is a matching
-<iteration for that time>)
ps_vis.visualize(ps="xpx", time=2.6410e-13, species='e', species_filter='all')
plt.show()

# plotting data for multiple simulations simultaneously also works:
ps_vis = PhaseSpaceMPL([
    ("sim1", "path/to/sim1"),
    ("sim2", "path/to/sim2"),
    ("sim3", "path/to/sim3"),
], ax)
ps_vis.visualize(ps="xpx", iteration=10000, species="e", species_filter='all')
plt.show()

The visualizer can also be used from the command line (for a single simulation only) by writing

```
python phase_space_visualizer.py
```

with the following command line options

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>Path and filename to the run directory of a simulation.</td>
</tr>
<tr>
<td>-i</td>
<td>An iteration number</td>
</tr>
<tr>
<td>-s (optional, defaults to 'e')</td>
<td>Particle species abbreviation (e.g. 'e' for electrons)</td>
</tr>
<tr>
<td>-f (optional, defaults to 'all')</td>
<td>Species filter string</td>
</tr>
<tr>
<td>-m (optional, defaults to 'ypy')</td>
<td>Momentum string to specify the phase space</td>
</tr>
</tbody>
</table>

**Jupyter Widget**

If you want more interactive visualization, then start a jupyter notebook and make sure that ipywidgets and ipympl are installed.

After starting the notebook server write the following

```python
# this is required!
#matplotlib widget
import matplotlib.pyplot as plt
plt.ion()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import PhaseSpaceWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
# which data belongs to which simulation
w = PhaseSpaceWidget(run_dir_options=[
    ("scan1/sim4", "/path/to/scan1/sim4"),
])
```
and then interact with the displayed widgets.

Please note that per default the widget allows selection only of the ypy phase space slice for particles labelled by $e$. To visualize, for instance the ypy, xpx and ypz slices for particles labelled by $e$ (as a rule background electrons) and by $b$ (here electrons of a particle bunch) the above has to be augmented by setting $w.ps.options$ and $w.species.options$. The final script snippet then reads:

```python
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
plt.ioff()
from IPython.display import display
from picongpu.plugins.jupyter_widgets import PhaseSpaceWidget

w = PhaseSpaceWidget(run_dir_options=[
    ("scan1/sim4", "/path/to/scan1/sim4"),
    ("scan1/sim5", "/path/to/scan1/sim5"))
w.ps.set_trait('options', ('ypy', 'xpx', 'ypz'))
w.species.set_trait('options', ('e', 'b'))
display(w)
```

## Out-of-Range Behavior

Particles that are not in the range of $<ValL>/ValR>$ get automatically mapped to the lowest/highest bin respectively. Take care about that when setting your range and during analysis of the results.

### Known Limitations

- only one range per selected space-momentum-pair possible right now (naming collisions)
- charge deposition uses the counter shape for now (would need one more write to neighbors to evaluate it correctly according to the shape)
- the user has to define the momentum range in advance
- the resolution is fixed to 1024 bins in momentum and the number of cells in the selected spatial dimension
- this plugin does not yet use openPMD markup.

## References

The internal algorithm is explained in pull request #347 and in [Huebl2014].

### 2.4.15 PNG

This plugin generates images in the png format for slices through the simulated volume. It allows to draw a species density together with electric, magnetic and/or current field values. The exact field values, their coloring and their normalization can be set using *.param files. It is a very rudimentary and useful tool to get a first impression on what happens in the simulation and to verify that the parameter set chosen leads to the desired physics.
**Note:** In the near future, this plugin might be replaced by the ISAAC interactive 3D visualization.

### External Dependencies

The plugin is available as soon as the PNGwriter library is compiled in.

#### .cfg file

For electrons ($e$) the following table describes the command line arguments used for the visualization.

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--e_png.period</td>
<td>This flag requires an integer value that specifies at what periodicity the png pictures should be created. E.g. setting <code>--e_png.period 100</code> generates images for the 0th, 100th, 200th, ... time step. There is no default. If flags are not set, no pngs are created.</td>
</tr>
<tr>
<td>--e_png.axis</td>
<td>Set 2D slice through 3D volume that will be drawn. Combine two of the three dimensions $x$, $y$ and <code>$z$, the define a slice. E.g. setting </code>--e_png.axis yz` draws both the $y$ and $z$ dimension and performs a slice in $x$-direction.</td>
</tr>
<tr>
<td>--e_png.slicePoint</td>
<td>Specifies at what ratio of the total depth of the remaining dimension, the slice should be performed. The value given should lie between 0.0 and 1.0.</td>
</tr>
<tr>
<td>--e_png.folder</td>
<td>Name of the folder, where all pngs for the above setup should be stored.</td>
</tr>
</tbody>
</table>

These flags use `boost::program_options`'s `multitoken()`. Therefore, several setups can be specified e.g. to draw different slices. The order of the flags is important in this case. E.g. in the following example, two different slices are visualized and stored in different directories:

```bash
picongpu [more args]
  # first
  --e_png.period 100
  --e_png.axis xy
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXY
  # second
  --e_png.period 100
  --e_png.axis xz
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXZ
```

#### .param files

The two param files `png.param` and `pngColorScales.param` are used to specify the desired output.

**Specifying the field values using `png.param`**

Depending on the used prefix in the command line flags, electron and/or ion density is drawn. Additionally to that, three field values can be visualized together with the particle density. In order to set up the visualized field values, the `png.param` needs to be changed. In this file, a variety of other parameters used for the PngModule can be specified.

The ratio of the image can be set.
In order to scale the image, `scale_to_cellsize` needs to be set to `true` and `scale_image` needs to specify the reduction ratio of the image.

**Note:** For a 2D simulation, even a 2D image can be a quite heavy output. Make sure to reduce the preview size!

It is possible to draw the borders between the GPUs used as white lines. This can be done by setting the parameter `white_box_per_GPU` in `png.param` to `true`:

```c
const bool white_box_per_GPU = true;
```

There are three field values that can be drawn: `CHANNEL1`, `CHANNEL2` and `CHANNEL3`.

Since an adequate color scaling is essential, there several option the user can choose from.

```c
#define EM_FIELD_SCALE_CHANNEL1 -1
#define EM_FIELD_SCALE_CHANNEL2 -1
#define EM_FIELD_SCALE_CHANNEL3 -1
```

In the above example, all channels are set to auto scale. Be careful, when using a normalization other than auto-scale, depending on your setup, the normalization might fail due to parameters not set by PIConGPU. Use the other normalization options only in case of the specified scenarios or if you know, how the scaling is computed.

You can also add opacity to the particle density and the three field values:

```c
float_X const preParticleDens_opacity = 0.25;
float_X const preChannel1_opacity = 1.0;
float_X const preChannel2_opacity = 1.0;
float_X const preChannel3_opacity = 1.0;
```

and add different coloring:

```c
namespace preParticleDensCol = colorScales::red;  // draw density in red */
namespace preChannel1Col = colorScales::blue;    // draw channel 1 in blue */
namespace preChannel2Col = colorScales::green;  // draw channel 2 in green */
namespace preChannel3Col = colorScales::none;   // do not draw channel 3 */
```

The colors available are defined in `pngColorScales.param` and their usage is described below. If `colorScales::none` is used, the channel is not drawn.

In order to specify what the three channels represent, three functions can be defined in `png.param`. The define the values computed for the png visualization. The data structures used are those available in PIConGPU.
*/ png preview settings for each channel */
DINLINE float_X preChannel1( float3_X const & field_B, float3_X const & field_E,  
   float3_X const & field_J )
{
   /* Channel1 */
   /* computes the absolute value squared of the electric current */
   return math::abs2(field_J);
}

DINLINE float_X preChannel2( float3_X const & field_B, float3_X const & field_E,  
   float3_X const & field_J )
{
   /* Channel2 */
   /* computes the square of the x-component of the electric field */
   return field_E.x() * field_E.x();
}

DINLINE float_X preChannel3( float3_X const & field_B, float3_X const & field_E,  
   float3_X const & field_J )
{
   /* Channel3 */
   /* computes the negative values of the y-component of the electric field */
   return -float_X(1.0) * field_E.y();
}

Only positive values are drawn. Negative values are clipped to zero. In the above example, this feature is used for preChannel3.

**Defining coloring schemes in** `pngColorScales.param`

There are several predefined color schemes available:

- none (do not draw anything)
- gray
- grayInv
- red
- green
- blue

But the user can also specify his or her own color scheme by defining a namespace with the color name that provides an `addRGB` function:

```cpp
namespace NameOfColor /* name needs to be unique */
{
   HDINLINE void addRGB( float3_X& img, /* the already existing image */
      const float_X value, /* the value to draw */
      const float_X opacity ) /* the opacity specified */
   {
      /* myChannel specifies the color in RGB values (RedGreenBlue) with */
      /* each value ranging from 0.0 to 1.0 . */
      /* In this example, the color yellow (RGB=1,1,0) is used. */
      const float3_X myChannel( 1.0, 1.0, 0.0 );

      /* here, the previously calculated image (in case, other channels have */
      /* already */
      /* contributed to the png) is changed. */
      /* First of all, the total image intensity is reduced by the opacity of */
      /* this channel, but only in the color channels specified by this color */
      "NameOfColor".
   }
}
```
* Then, the actual values are added with the correct color (myChannel) and opacity. */
    img = img
    - opacity * float3_X( myChannel.x() * img.x(),
                          myChannel.y() * img.y(),
                          myChannel.z() * img.z() )
    + myChannel * value * opacity;
}
}

For most cases, using the predefined colors should be enough.

**Memory Complexity**

**Accelerator**

locally, memory for the local 2D slice is allocated with 3 channels in float_X.

**Host**

as on accelerator. Additionally, the master rank has to allocate three channels for the full-resolution image. This is the original size before reduction via scale_image.

**Output**

The output of this plugin are pngs stored in the directories specified by --e_png.folder or --i_png.folder. There can be as many of these folders as the user wants. The pngs follow a naming convention:

\[
<\text{species}>_\text{png}_\text{yx}_0.5_002000.png
\]

First, either <species> names the particle type. Following the 2nd underscore, the drawn dimensions are given. Then the slice ratio, specified by --e_png.slicePoint or --i_png.slicePoint, is stated in the file name. The last part of the file name is a 6 digit number, specifying the simulation time step, at which the picture was created. This naming convention allows to put all pngs in one directory and still be able to identify them correctly if necessary.

**Analysis Tools**

**Data Reader**

You can quickly load and interact with the data in Python with:

```
from picongpu.plugins.data import PNGData

png_data = PNGData('path/to/run_dir')

# get the available iterations for which output exists
iters = png_data.get_iterations(species="e", axis="yx")

# get the available simulation times for which output exists
times = png_data.get_times(species="e", axis="yx")

# pngs as numpy arrays for multiple iterations (times would also work)
pngs = png_data.get(species="e", axis="yx", iteration=iters[:3])
```

(continues on next page)
for png in pngs:
    print(png.shape)

Matplotlib Visualizer

If you are only interested in visualizing the generated png files it is even easier since you don’t have to load the data manually.

```python
from picongpu.plugins.plot_mpl import PNGMPL
import matplotlib.pyplot as plt

# create a figure and axes
fig, ax = plt.subplots(1, 1)

# create the visualizer
png_vis = PNGMPL('path/to/run_dir', ax)

# plot
png_vis.visualize(iteration=200, species='e', axis='yx')
```

The visualizer can also be used from the command line by writing

```
python png_visualizer.py
```

with the following command line options

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>Path and to the run directory of a simulation.</td>
</tr>
<tr>
<td>-i</td>
<td>An iteration number</td>
</tr>
<tr>
<td>-s</td>
<td>Particle species abbreviation (e.g. 'e' for electrons)</td>
</tr>
<tr>
<td>-f (optional, defaults to 'e')</td>
<td>Species filter string</td>
</tr>
<tr>
<td>-a (optional, defaults to 'yx')</td>
<td>Axis string (e.g. 'yx' or 'xy')</td>
</tr>
<tr>
<td>-o (optional, defaults to 'None')</td>
<td>A float between 0 and 1 for slice offset along the third dimension</td>
</tr>
</tbody>
</table>

Jupyter Widget

If you want more interactive visualization, then start a jupyter notebook and make sure that ipywidgets and ipympl are installed.

After starting the notebook server write the following

```python
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
# deactivate interactive mode
plt.ioff()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import PNGWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
```
and then interact with the displayed widgets.

## 2.4.16 Positions Particles

This plugin prints out the position, momentum, mass, macro particle weighting, electric charge and relativistic gamma factor of a particle to stdout (usually inside the simOutput/output file). It only works with test simulations that have only one particle.

### .cfg file

By setting the command line flag `--<species>_position.period` to a non-zero number, the analyzer is used. In order to get the particle trajectory for each time step the period needs to be set to 1, meaning e.g. `--e_position.period 1` for electrons. If less output is needed, e.g. only every 10th time step, the period can be set to different values, e.g. `--e_position.period 10`.

### Memory Complexity

**Accelerator**

negligible.

**Host**

negligible.

### Output

The electron trajectory is written directly to the *standard output*. Therefore, it goes both to `simOutput/output` as well as to the output file specified by the machine used (usually the `stdout` file in the main directory of the simulation). The output is ASCII-text only. It has the following format:

```
[ANALYSIS] [MPI_Rank] [COUNTER] [-<species>_position] [currentTimeStep] currentTime
  <position.x position.y position.z> <momentum.x momentum.y momentum.z> mass
  <weighting charge gamma>
```
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Rank</td>
<td>MPI rank at which prints the particle position</td>
<td>none</td>
</tr>
<tr>
<td>COUNTER</td>
<td>name of the plugin</td>
<td>always</td>
</tr>
<tr>
<td>currentTimeStep</td>
<td>simulation time step = number of PIC cycles</td>
<td>none</td>
</tr>
<tr>
<td>currentTime</td>
<td>simulation time in SI units</td>
<td>seconds</td>
</tr>
<tr>
<td>position.x _position.y</td>
<td>location of the particle in space</td>
<td>meters</td>
</tr>
<tr>
<td>momentum.x _momentum.y</td>
<td>momentum of particle</td>
<td>kg m/s</td>
</tr>
<tr>
<td>mass</td>
<td>mass of macro particle</td>
<td>kg</td>
</tr>
<tr>
<td>weighting</td>
<td>number of electrons represented by the macro particle</td>
<td>none</td>
</tr>
<tr>
<td>charge</td>
<td>charge of macro particle</td>
<td>Coulomb</td>
</tr>
<tr>
<td>gamma</td>
<td>relativistic gamma factor of particle</td>
<td>none</td>
</tr>
</tbody>
</table>

# an example output line:
[ANALYSIS] [2] [COUNTER] [e_position] [878] 1.46440742e-14 (1.032e-05 4.57851589815522e-05 5.2e-06) {0 -1.337873603181226e-21 0} 9.109382e-31 1 -1.602176e-19 4.9999985694888525

In order to extract only the trajectory information from the total output stored in stdout, the following command on a bash command line could be used:

grep "e_position" stdout > trajectory.dat

The particle data is then stored in trajectory.dat.

In order to extract e.g. the position from this line the following can be used:

cat trajectory.dat | awk '{print $7}' | sed -e "s/\{/\t/g" | sed -e 's//\t/g' > position.dat

**Known Issues**

**Attention:** This plugin only works correctly if a single particle is simulated. If more than one particle is simulated, the output becomes random, because only the information of one particle is printed. This plugin might be upgraded to work with multiple particles, but better use our HDF5 or ADIOS plugin instead and assign particleIds to individual particles.

**Attention:** Currently, both simOutput/output and stdout are overwritten at restart. All data from the plugin is lost, if these file are not backuped manually.

### 2.4.17 Radiation

The spectrally resolved far field radiation of charged macro particles.

Our simulation computes the Lienard Wiechert potentials to calculate the emitted electromagnetic spectra for different observation directions using the far field approximation.

\[
\frac{d^2 I}{d\Omega d\omega} (\omega, \vec{n}) = \frac{q^2}{16\pi^2 c_0^2} \sum_{k=1}^{N} \int_{-\infty}^{+\infty} \frac{\vec{n} \times \left( \vec{n} - \vec{\beta}_k(t) \times \dot{\vec{\beta}}_k(t) \right) \times \vec{\beta}_k(t) \left( 1 - \vec{\beta}_k(t) \cdot \vec{n} \right)^2}{\left( 1 - \vec{\beta}_k(t) \cdot \vec{n} \right)^2} \cdot e^{i\omega(t - \vec{n} \cdot \vec{r}_k(t)/c)} dt^2
\]
Details on how radiation is computed with this plugin and how the plugin works can be found in [Pausch2012]. A list of tests can be found in [Pausch2014] and [Pausch2019].

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vec{r}_k(t) )</td>
<td>The position of particle ( k ) at time ( t ).</td>
</tr>
<tr>
<td>( \vec{\beta}_k(t) )</td>
<td>The normalized speed of particle ( k ) at time ( t ). (Speed divided by the speed of light)</td>
</tr>
<tr>
<td>( \dot{\vec{\beta}}_k(t) )</td>
<td>The normalized acceleration of particle ( k ) at time ( t ). (Time derivative of the normalized speed.)</td>
</tr>
<tr>
<td>( t )</td>
<td>Time</td>
</tr>
<tr>
<td>( \vec{n} )</td>
<td>Unit vector pointing in the direction where the far field radiation is observed.</td>
</tr>
<tr>
<td>( \omega )</td>
<td>The circular frequency of the radiation that is observed.</td>
</tr>
<tr>
<td>( N )</td>
<td>Number of all (macro) particles that are used for computing the radiation.</td>
</tr>
<tr>
<td>( k )</td>
<td>Running index of the particles.</td>
</tr>
</tbody>
</table>

Currently this allows to predict the emitted radiation from plasma if it can be described by classical means. Not considered are emissions from ionization, Compton scattering or any bremsstrahlung that originate from scattering on scales smaller than the PIC cell size.

**External Dependencies**

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

**.param files**

In order to setup the radiation analyzer plugin, both the *radiation.param* and the *radiationObserver.param* have to be configured and the radiating particles need to have the attribute *momentumPrev1* which can be added in *speciesDefinition.param*.

In *radiation.param*, the number of frequencies \( N_{omega} \) and observation directions \( N_{theta} \) is defined.

**Frequency range**

The frequency range is set up by choosing a specific namespace that defines the frequency setup:

```cpp
/* choose linear frequency range */
namespace radiation_frequencies = linear_frequencies;
```

Currently you can choose from the following setups for the frequency range:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_frequencies</td>
<td>linear frequency range from SI::omega_min to SI::omega_max with ( N_{omega} ) steps</td>
</tr>
<tr>
<td>log_frequencies</td>
<td>logarithmic frequency range from SI::omega_min to SI::omega_max with ( N_{omega} ) steps</td>
</tr>
<tr>
<td>frequencies_from_list</td>
<td>( n ) frequencies taken from a text file with location listLocation[]</td>
</tr>
</tbody>
</table>

All three options require variable definitions in the according namespaces as described below:

For the *linear frequency* scale all definitions need to be in the *picongpu::plugins::radiation::linear_frequencies* namespace. The number of total sample frequencies \( N_{omega} \) need to be defined as constexpr unsigned int. In the sub-namespace SI, a minimal frequency \( omega_min \) and a maximum frequency \( omega_max \) need to be defined as constexpr float_64.

For the *logarithmic frequency* scale all definitions need to be in the *picongpu::plugins::radiation::log_frequencies* namespace. Equivalently to the linear case, three variables need to be defined: The number of total sample frequencies \( N_{omega} \) need to be defined as...
constexpr unsigned int. In the sub-namespace SI, a minimal frequency \( \omega_{\text{min}} \) and a maximum frequency \( \omega_{\text{max}} \) need to be defined as constexpr float_64.

For the file-based frequency definition, all definitions need to be in the picongpu::plugins::radiation::frequencies_from_list namespace. The number of total frequencies \( N_{\omega} \) need to be defined as constexpr unsigned int and the path to the file containing the frequency values in units of \([s^{-1}]\) needs to be given as constexpr const char * listLocation = "/path/to/frequency_list". The frequency values in the file can be separated by newlines, spaces, tabs, or any other whitespace. The numbers should be given in such a way, that c++ standard std::ifstream can interpret the number e.g., as \( 2.5344e+16 \).

**Note:** Currently, the variable listLocation is required to be defined in the picongpu::plugins::radiation::frequencies_from_list namespace, even if frequencies_from_list is not used. The string does not need to point to an existing file, as long as the file-based frequency definition is not used.

**Observation directions**

The number of observation directions \( N_{\theta} \) is defined in radiation.param, but the distribution of observation directions is given in radiationObserver.param) There, the function observation_direction defines the observation directions.

This function returns the x,y and z component of a unit vector pointing in the observation direction.

```cpp
DINLINE vector_64 observation_direction( int const observation_idExtern )
{
    /* use the scalar index const int observation_idExtern to compute an observation direction (x,y,z) */
    return vector_64( x , y , z );
}
```

**Note:** The radiationObserver.param set up will be subject to further changes. These might be namespaces that describe several preconfigured layouts or a functor if C++ 11 is included in the nvcc.

**Nyquist limit**

A major limitation of discrete Fourier transform is the limited frequency resolution due to the discrete time steps of the temporal signal. (see Nyquist-Shannon sampling theorem) Due to the consideration of relativistic delays, the sampling of the emitted radiation is not equidistantly sampled. The plugin has the option to ignore any frequency contributions that lies above the frequency resolution given by the Nyquist-Shannon sampling theorem. Because performing this check costs computation time, it can be switched off. This is done via a precompiler pragma:

```cpp
// Nyquist low pass allows only amplitudes for frequencies below Nyquist frequency
// 1 = on (slower and more memory, no Fourier reflections)
// 0 = off (faster but with Fourier reflections)
#define __NYQUISTCHECK__ 0
```

Additionally, the maximally resolvable frequency compared to the Nyquist frequency can be set.

```cpp
namespace radiationNyquist
{
    /* only use frequencies below 1/2*Omega_Nyquist */
    const float NyquistFactor = 0.5;
}
```
This allows to make a save margin to the hard limit of the Nyquist frequency. By using \texttt{NyquistFactor} = 0.5 for periodic boundary conditions, particles that jump from one border to another and back can still be considered.

\textbf{Form factor}

The \textit{form factor} is a method, which considers the shape of the macro particles when computing the radiation. More details can be found in \cite{Pausch2018} and \cite{Pausch2019}.

One can select between different macro particle shapes. Currently eight shapes are implemented. A shape can be selected by choosing one of the available namespaces:

```cpp
namespace radFormFactor = radFormFactor_CIC_3D;
```

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radFormFactor_CIC_3D</td>
<td>3D Cloud-In-Cell shape</td>
</tr>
<tr>
<td>radFormFactor_TSC_3D</td>
<td>3D Triangular shaped density cloud</td>
</tr>
<tr>
<td>radFormFactor_PCS_3D</td>
<td>3D Quadratic spline density shape (Piecewise Cubic Spline assignment function)</td>
</tr>
<tr>
<td>radFormFactor_CIC_1Dy</td>
<td>Cloud-In-Cell shape in y-direction, dot like in the other directions</td>
</tr>
<tr>
<td>radFormFactor_Gauss_symmetric</td>
<td>Symmetric Gauss charge distribution</td>
</tr>
<tr>
<td>radFormFactor_Gauss_cell</td>
<td>Gauss charge distribution according to cell size</td>
</tr>
<tr>
<td>radFormFactor_incoherent</td>
<td>Forces a completely incoherent emission by scaling the macro particle charge with the square root of the weighting</td>
</tr>
<tr>
<td>radFormFactor_coherent</td>
<td>Forces a completely coherent emission by scaling the macro particle charge with the weighting</td>
</tr>
</tbody>
</table>

\textbf{Reducing the particle sample}

In order to save computation time, only a random subset of all macro particles can be used to compute the emitted radiation. In order to do that, the radiating particle species needs the attribute \texttt{radiationMask} (which is initialized as \texttt{false}) which further needs to be manipulated, to set to true for specific (random) particles.

\textbf{Note:} The reduction of the total intensity is not considered in the output. The intensity will be (in the incoherent case) will be smaller by the fraction of marked to all particles.

\textbf{Note:} The radiation mask is only added to particles, if not all particles should be considered for radiation calculation. Adding the radiation flag costs memory.

\textbf{Note:} In future updates, the radiation will only be computed using an extra particle species. Therefore, this setup will be subject to further changes.

\textbf{Gamma filter}

In order to consider the radiation only of particles with a gamma higher than a specific threshold, the radiating particle species needs the attribute \texttt{radiationMask} (which is initialized as \texttt{false}). Using a filter functor as:
using RadiationParticleFilter = picongpu::particles::manipulators::FreeImpl<
    GammaFilterFunctor
>;

(see Bunch or Kelvin Helmholtz example for details) sets the flag to true is a particle fulfills the gamma condition.

**Note:** More sophisticated filters might come in the near future. Therefore, this part of the code might be subject to changes.

### Window function filter

A window function can be added to the simulation area to reduce ringing artifacts due to sharp transition from radiating regions to non-radiating regions at the boundaries of the simulation box. This should be applied to simulation setups where the entire volume simulated is radiating (e.g. Kelvin-Helmholtz Instability).

In `radiation.param` the precompiler variable `PIC_RADWINDOWFUNCTION` defines if the window function filter should be used or not.

```cpp
// add a window function weighting to the radiation in order
// to avoid ringing effects from sharp boundaries
// 1 = on (slower but with noise/ringing reduction)
// 0 = off (faster but might contain ringing)
#define PIC_RADWINDOWFUNCTION 0
```

If set to 1, the window function filter is used.

There are several different window function available:

```cpp
/* Choose different window function in order to get better ringing reduction
 * radWindowFunctionRectangle
 * radWindowFunctionTriangle
 * radWindowFunctionHamming
 * radWindowFunctionTriplett
 * radWindowFunctionGauss
 */
namespace radWindowFunctionRectangle { }
namespace radWindowFunctionTriangle { }
namespace radWindowFunctionHamming { }
namespace radWindowFunctionTriplett { }
namespace radWindowFunctionGauss { }
namespace radWindowFunction = radWindowFunctionTriangle;
```

By setting `radWindowFunction` a specific window function is selected.

More details can be found in [Pausch2019].

### .cfg file

For a specific (charged) species `<species>` e.g. e, the radiation can be computed by the following commands.
### Command line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--&lt;species&gt;_radiation_period</code></td>
<td>Gives the number of time steps between which the radiation should be calculated. Default is 0, which means that the radiation is never calculated and therefore off. Using 1 calculates the radiation constantly. Any value &gt;=2 is currently producing nonsense.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_dump</code></td>
<td>Period after which the calculated radiation data should be dumped to the filesystem. Default is 0, therefore never. In order to store the radiation data, a value &gt;=1 should be used.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_lastRadiation</code></td>
<td>If set, the radiation spectra summed between the last and the current dump-time-step are stored. Used for a better evaluation of the temporal evolution of the emitted radiation.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_folderLastRad</code></td>
<td>Name of the folder, in which the summed spectra for the simulation time between the last dump and the current dump are stored. Default is <code>lastRad</code>.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_totalRadiation</code></td>
<td>If set, the spectra summed from simulation start till current time step are stored.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_folderTotalRad</code></td>
<td>Folder name in which the total radiation spectra, integrated from the beginning of the simulation, are stored. Default <code>totalRad</code>.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_start</code></td>
<td>Time step, at which PIConGPU starts calculating the radiation. Default is 2 in order to get enough history of the particles.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_end</code></td>
<td>Time step, at which the radiation calculation should end. Default: 0 (stops at end of simulation).</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_radPerGPU</code></td>
<td>If set, each GPU additionally stores its own spectra without summing over the entire simulation area. This allows for a localization of specific spectral features.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_folderRadPerGPU</code></td>
<td>Name of the folder, where the GPU specific spectra are stored. Default: <code>radPerGPU</code>.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_compression</code></td>
<td>If set, the hdf5 output is compressed.</td>
</tr>
</tbody>
</table>

### Memory Complexity

#### Accelerator

Each energy bin times each coordinate bin allocates one counter (`float_X`) permanently and on each accelerator.

#### Host

As on accelerator.

#### Output

Depending on the command line options used, there are different output files.

<table>
<thead>
<tr>
<th>Command line flag</th>
<th>Output description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--&lt;species&gt;_radiation_totalRadiation</code></td>
<td>Contains ASCII files that have the total spectral intensity until the timestep specified by the filename. Each row gives data for one observation direction (same order as specified in the observer.py). The values for each frequency are separated by tabs and have the same order as specified in radiation.param. The spectral intensity is stored in the units [Js].</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_lastRadiation</code></td>
<td>Has the same format as the output of <code>totalRadiation</code>. The spectral intensity is only summed over the last radiation dump period.</td>
</tr>
<tr>
<td><code>--&lt;species&gt;_radiation_radPerGPU</code></td>
<td>Same output as <code>totalRadiation</code> but only summed over each GPU. Because each GPU specifies a spatial region, the origin of radiation signatures can be distinguished.</td>
</tr>
<tr>
<td><code>radiation-HDF5</code></td>
<td>In the folder <code>radiationHDF5</code>, hdf5 files for each radiation dump and species are stored. These are complex amplitudes in units used by PIConGPU. These are for restart purposes and for more complex data analysis.</td>
</tr>
</tbody>
</table>
Text-based output

The text-based output of `lastRadiation` and `totalRadiation` contains the intensity values in SI-units [Js]. Intensity values for different frequencies are separated by spaces, while newlines separate values for different observation directions.

In order to read and plot the text-based radiation data, a python script as follows could be used:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

# frequency definition:
# as defined in the 'radiation.param' file:
N_omega = 1024
omega_min = 0.0 # [1/s]
omega_max = 5.8869e17 # [1/s]
omega = np.linspace(omega_min, omega_max, N_omega)

# observation angle definition:
# as defined in the 'radiationObserver.param' file:
# this example assumes one used the default Bunch example
# there, the theta values are normalized to the Lorentz factor
theta_min = -1.5 # [rad/gamma]
theta_max = +1.5 # [rad/gamma]
theta = np.linspace(theta_min, theta_max, N_observer)

# load radiation text-based data
rad_data = np.loadtxt('./simOutput/lastRad/e_radiation_2820.dat')

# plot radiation spectrum
plt.figure()
plt.pcolormesh(omega, theta, rad_data, norm=LogNorm())

# add and configure colorbar
cb = plt.colorbar()
for i in cb.ax.get_yticklabels():
    i.set_fontsize(14)

# configure x-axis
plt.xlabel(r'$\omega \, \text{[1/s]}$', fontsize=18)
plt.xticks(fontsize=14)

# configure y-axis
plt.ylabel(r'$\theta / \gamma$', fontsize=18)
plt.yticks(fontsize=14)

# make plot look nice
plt.tight_layout()
plt.show()
```

HDF5 output

The hdf5 based data contains the following data structure in `/data/{iteration}/DetectorMesh/` according to the openPMD standard:

Amplitude (Group):
### Dataset (Group):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_Re</td>
<td>real part, x-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
<tr>
<td>x_Im</td>
<td>imaginary part, x-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
<tr>
<td>y_Re</td>
<td>real part, y-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
<tr>
<td>y_Im</td>
<td>imaginary part, y-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
<tr>
<td>z_Re</td>
<td>real part, z-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
<tr>
<td>z_Im</td>
<td>imaginary part, z-component of the complex amplitude</td>
<td>(N_observer, N_omega, 1)</td>
</tr>
</tbody>
</table>

**Note:** Please be aware, that despite the fact, that the SI-unit of each amplitude entry is $\sqrt{\text{Js}}$, the stored `unitSI` attribute returns $[\text{Js}]$. This inconsistency will be fixed in the future. Until this inconsistency is resolved, please multiply the datasets with the square root of the `unitSI` attribute to convert the amplitudes to SI units.

### DetectorDirection (Group):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x-component of the observation direction $\vec{n}$</td>
<td>(N_observer, 1, 1)</td>
</tr>
<tr>
<td>y</td>
<td>y-component of the observation direction $\vec{n}$</td>
<td>(N_observer, 1, 1)</td>
</tr>
<tr>
<td>z</td>
<td>z-component of the observation direction $\vec{n}$</td>
<td>(N_observer, 1, 1)</td>
</tr>
</tbody>
</table>

### DetectorFrequency (Group):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>frequency $\omega$ of virtual detector bin</td>
<td>(1, N_omega, 1)</td>
</tr>
</tbody>
</table>

Please be aware that all datasets in the hdf5 output are given in the PIConGPU-intrinsic unit system. In order to convert, for example, the frequencies $\omega$ to SI-units one has to multiply with the dataset-attribute `unitSI`.

```python
import h5py
f = h5py.File("e_radAmplitudes_2800_0_0_0.h5", "r")
omega_handler = f['/data/2800/DetectorMesh/DetectorFrequency/omega']
omega = omega_handler[0, :, 0] * omega_handler.attrs['unitSI']
f.close()
```

In order to extract the radiation data from the HDF5 datasets, PIConGPU provides a python module to read the data and obtain the result in SI-units. An example python script is given below:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from picongpu.plugins.data import RadiationData

# access HDF5 radiation file
radData = RadiationData("./simOutput/radiationHDF5/e_radAmplitudes_2820_0_0_0.h5")

# get frequencies
omega = radData.get_omega()

# get all observation vectors and convert to angle
vec_n = radData.get_vector_n()
gamma = 5.0
theta_norm = np.arctan(vec_n[:, 0]/vec_n[:, 1]) * gamma

# get spectrum over observation angle
spectrum = radData.get_Spectra()
```

(continues on next page)
There are various methods besides `get_Spectra()` that are provided by the python module. If a method exists for `_x` (or `_X`) it also exists for `_y` and `_z` (_Y and _Z) accordingly.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.get_omega()</td>
<td>get frequency ( \omega ) of virtual detector bin in units of ([1/\text{s}])</td>
</tr>
<tr>
<td>.get_vector_n()</td>
<td>get observation direction ( \hat{n} )</td>
</tr>
<tr>
<td>.get_Spectra()</td>
<td>get spectrum ( \frac{d^2 I}{d \omega d\Omega} ) in units of ([\text{Js}])</td>
</tr>
<tr>
<td>.get_Polarization_X()</td>
<td>get spectrum but only for polarization in x-direction</td>
</tr>
<tr>
<td>.get_Amplitude_x()</td>
<td>get x-component of complex amplitude (unit: ([\sqrt{\text{Js}}]))</td>
</tr>
<tr>
<td>.get_timestep()</td>
<td>the iteration (timestep) at which the data was produced (unit: PIC-cycles)</td>
</tr>
</tbody>
</table>

**Note:** Modules for visualizing radiation data and a widget interface to explore the data interactively will be developed in the future.

**Analyzing tools**

In `picongp/src/tools/bin`, there are tools to analyze the radiation data after the simulation.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotRadiation</td>
<td>Reads ASCII radiation data and plots spectra over angles as color plots. This is a python script that has its own help. Run <code>plotRadiation --help</code> for more information.</td>
</tr>
<tr>
<td>radiationSyntheticDetector</td>
<td>Reads ASCII radiation data and statistically analysis the spectra for a user specified region of observation angles and frequencies. This is a python script that has its own help. Run <code>radiationSyntheticDetector --help</code> for more information.</td>
</tr>
<tr>
<td>smooth.py</td>
<td>Python module needed by <code>plotRadiation</code>.</td>
</tr>
</tbody>
</table>

**Known Issues**

The plugin supports multiple radiation species but spectra (frequencies and observation directions) are the same for all species.
2.4.18 Resource Log

Writes resource information such as rank, position, current simulation step, particle count, and cell count as json or xml formatted string to output streams (file, stdout, stderr).

.cfg file

Run the plugin for each nth time step: --resourceLog.period n

The following table will describes the settings for the plugin:

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--resourceLog.properties</td>
<td>Selects properties to write [rank, position, currentStep, particleCount, cellCount]</td>
</tr>
<tr>
<td>--resourceLog.format</td>
<td>Selects output format [json, jsonpp, xml, xmlpp]</td>
</tr>
<tr>
<td>--resourceLog.stream</td>
<td>Selects output stream [file, stdout, stderr]</td>
</tr>
<tr>
<td>--resourceLog.prefix</td>
<td>Selects the prefix for the file stream name</td>
</tr>
</tbody>
</table>

Memory Complexity

Accelerator

no extra allocation.

Host

negligible.

Output / Example

Using the options

```
--resourceLog.period 1  
--resourceLog.stream stdout  
--resourceLog.properties rank position currentStep particleCount cellCount  
--resourceLog.format jsonpp
```

will write resource objects to stdout such as:

```
[1,1]<stdout>:  "resourceLog": {  
[1,1]<stdout>:   "rank": "1",  
[1,1]<stdout>:   "position": {  
[1,1]<stdout>:     "x": "0",  
[1,1]<stdout>:     "y": "1",  
[1,1]<stdout>:     "z": "0"  
[1,1]<stdout>:   },  
[1,1]<stdout>:   "currentStep": "357",  
[1,1]<stdout>:   "cellCount": "1048576",  
[1,1]<stdout>:   "particleCount": "2180978"  
[1,1]<stdout>: }  
[1,1]<stdout>:}
```

For each format there exists always a non pretty print version to simplify further processing:
2.4.19 Slice Emittance

The plugin computes the total emittance and the slice emittance (for ten combined cells in the longitudinal direction).

Currently, it outputs only the emittance of the transverse momentum space x-px.

More details on the implementation and tests can be found in the master’s thesis [Rudat2019].

External Dependencies

None

.param file

None for now. In the future, adding more compile-time configurations might become necessary (e.g., striding of data output).

.cfg file

All options are denoted for the electron (e) particle species here.

<table>
<thead>
<tr>
<th>PIConGPU command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--e_emittance.period arg</td>
<td>compute slice emittance [for each n-th step], enable plugin by setting a non-zero value A value of 100 would mean an output at simulation time step 0, 100, 200, ....</td>
</tr>
<tr>
<td>--e_emittance.filter arg</td>
<td>Use filtered particles. All available filters will be shown with picongpu --help</td>
</tr>
</tbody>
</table>

Memory Complexity

Accelerator

Each \( x^2, p_x^2 \) and \( x \times p_x \) summation value as well as the number of real electrons \( g\text{Count\_e} \) needs to be stored as float_64 for each \( y \)-cell.

Host

as on accelerator (needed for MPI data transfer)

Output

Note: This plugin is a multi plugin. Command line parameters can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--e_emittance.period 1000 --e_emittance.filter all
--e_emittance.period 100 --e_emittance.filter highEnergy
```
creates two plugins:

1. slice emittance for species e each 1000th time step for all particles.
2. slice emittance for species e each 100th time step only for particles with high energy (defined by filter).

**Analysis Tools**

The output is a text file with the first line as a comment describing the content. The first column is the time step. The second column is the total emittance (of all particles defined by the filter). Each following column is the emittance if the slice at ten cells around the position given in the comment line.

```python
data = np.loadtxt("<path-to-emittance-file>")
timesteps = data[:, 0]
total_emittance = data[:, 1]
slice_emittance = data[:, 2:]

# time evolution of total emittance
plt.plot(timesteps, total_emittance)
plt.xlabel("time step")
plt.ylabel("emittance")
plt.show()

# plot slice emittance over time and longitudinal (y) position
plt.imshow(slice_emittance)
plt.xlabel("y position [arb.u.]")
plt.ylabel("time [arb.u.]")
cb = plt.colorbar()
cb.set_label("emittance")
plt.show()
```

**References**

2.4.20 Slice Field Printer

Outputs a 2D slice of the electric, magnetic and/or current field in SI units. The slice position and the field can be specified by the user.

**.cfg file**

The plugin works on electric, magnetic, and current fields. For the electric field, the prefix `--E_slice` for all command line arguments is used. For the magnetic field, the prefix `--B_slice` is used. For the current field, the prefix `--J_slice` is used.

The following table will describe the setup for the electric field. The same applied to the magnetic field. Only the prefix has to be adjusted.
<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--E_slice.period</td>
<td>The periodicity of the slice print out. If set to a non-zero value, e.g. to <code>--E_slice.period 100</code>, the slices are generated for every 100th simulation time step.</td>
</tr>
<tr>
<td>--E_slice.fileName</td>
<td>Name of the output file. Setting <code>--E_slice.fileName myName</code> will result in output files like <code>myName_100.dat</code>.</td>
</tr>
<tr>
<td>--E_slice.plane</td>
<td>Defines the plane that the slice will be parallel to. The plane is defined by its orthogonal axis. By using 0 for the x-axis, 1 for the y-axis and 2 for the z-axis, all standard planes can be selected. E.g. choosing the x-y-plane is done by setting the orthogonal axis to the z-axis by giving the command line argument <code>--E_slice.plane 2</code>.</td>
</tr>
<tr>
<td>--E_slice.slicePoint</td>
<td>Defines the position of the slice on the orthogonal axis. E.g. when the x-y-plane was selected, the slice position in z-direction has to be set. This is done using a value between 0.0 and 1.0. E.g. by setting <code>--E_slice.slicePoint 0.5</code>, the slice is centered.</td>
</tr>
</tbody>
</table>

This plugin supports using multiple slices. By setting the command line arguments multiple times, multiple slices are printed to file. As an example, the following command line will create two slices:

```bash
picongpu # [...] --E_slice.period 100 --E_slice.fileName slice1 --E_slice.plane 2 --E_slice.slicePoint 0.5
--E_slice.period 50 --E_slice.fileName slice2 --E_slice.plane 0 --E_slice.slicePoint 0.25
```

The first slice is a cut along the x-y axis. It is printed every 100th step. It cuts through the middle of the z-axis and the data is stored in files like `slice1_100.dat`. The second slice is a cut along the y-z axis. It is printed every 50th step. It cuts through the first quarter of the x-axis and the data is stored in files like `slice2_100.dat`.

### 2D fields

In the case of 2D fields, the plugin outputs a 1D slice. Be aware that `--E_slice.plane` still refers to the orthogonal axis, i.e. `--E_slice.plane 1` outputs a line along the x-axis and `--E_slice.plane 0` along the y-axis.

### Memory Complexity

**Accelerator**

the local slice is permanently allocated in the type of the field (float3_X).

**Host**

as on accelerator.

**Output**

The output is stored in an ASCII file for every time step selected by `.period` (see How to set it up?). The 2D slice is stored as lines and rows of the ASCII file. Spaces separate rows and newlines separate lines. Each entry is of the format `{1.1e-1,2.2e-2,3.3e.3}` giving each value of the vector field separately e.g. `{E_x,E_y,E_z}`.

In order to read this data format, there is a python module in `lib/python/picongpu/plugins/sliceFieldReader.py`. The function `readFieldSlices` needs a data file (file or filename) with data from the plugin and returns the data as numpy-array of size `(N_y, N_x, 3)`.
Known Issues

See issue #348.
Should be solved with pull request #548.

2.4.21 Sum Currents

This plugin computes the total current integrated/added over the entire volume simulated.

`.cfg` file

The plugin can be activated by setting a non-zero value with the command line flag `--sumcurr.period`. The value set with `--sumcurr.period` is the periodicity, at which the total current is computed. E.g. `--sumcurr.period 100` computes and prints the total current for time step 0, 100, 200, ....

Memory Complexity

Accelerator

negligible.

Host

negligible.

Output

The result is printed to `standard output`. Therefore, it goes both to `.simOutput/output` as well as to the output file specified by the machine used (usually the `stdout` file in the main directory of the simulation). The output is ASCII-text only. It has the following format:

```
[ANALYSIS] _rank [COUNTER] [SumCurrents] _currentTimeStep _current.x _current.y _current.z Abs:_absCurrent
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>_rank</td>
<td>MPI rank at which prints the particle position</td>
<td>none</td>
</tr>
<tr>
<td>_currentTimeStep</td>
<td>simulation time step = number of PIC cycles</td>
<td>none</td>
</tr>
<tr>
<td>_current.x</td>
<td>electric current</td>
<td>Ampere per second</td>
</tr>
<tr>
<td>_current.y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_current.z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_absCurrent</td>
<td>magnitude of current</td>
<td>Ampere per second</td>
</tr>
</tbody>
</table>

In order to extract only the total current information from the output stored in `stdout`, the following command on a bash command line could be used:

```
grep SumCurrents stdout > totalCurrent.dat
```

The plugin data is then stored in `totalCurrent.dat`. 
Known Issues

Currently, both output and stdout are overwritten at restart. All data from the plugin is lost, if these file are not backuped manually.

2.4.22 Transition Radiation

The spectrally resolved far field radiation created by electrons passing through a metal foil.

Our simulation computes the transition radiation to calculate the emitted electromagnetic spectra for different observation angles.

\[
\frac{d^2 W}{d\omega d\Omega} = \frac{e^2 N_e}{(4\pi\epsilon_0)\pi^2 c} \left\{ \left[ \int d^3 \mathbf{p} g(\mathbf{p}^2 + \mathbf{E}_\parallel^2 + \mathbf{E}_\perp^2) \right] + (N_e - 1) \left[ |\int d^3 \mathbf{p} g \mathbf{E}_\parallel F|^2 + |\int d^3 \mathbf{p} g \mathbf{E}_\perp F|^2 \right] \right\}
\]

\[
\mathbf{E}_\parallel = \frac{u \cos \psi (u \sin \psi \cos \phi - (1 + u^2)^{1/2} \sin \theta)}{N(\theta, u, \psi, \phi)}
\]

\[
\mathbf{E}_\perp = \frac{u^2 \cos \psi \sin \psi \sin \phi \cos \theta}{N(\theta, u, \psi, \phi)}
\]

\[
N(\theta, u, \psi, \phi) = \left[ (1 + u^2)^{1/2} - u \sin \psi \cos \phi \sin \theta \right]^2 - u^2 \cos^2 \psi \cos^2 \theta
\]

\[
F = \frac{1}{g(p)} \int d^2 \mathbf{p} e^{-i\mathbf{k}_\perp \cdot \mathbf{v}_\perp} \int d\omega e^{-i\gamma(\omega - \mathbf{k}_\parallel \cdot \mathbf{v}_\parallel) / \gamma} h(r, p)
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_e)</td>
<td>Amount of real electrons</td>
</tr>
<tr>
<td>(\psi)</td>
<td>Azimuth angle of momentum vector from electrons to y-axis of simulation</td>
</tr>
<tr>
<td>(\theta)</td>
<td>Azimuth angle of observation vector</td>
</tr>
<tr>
<td>(\phi)</td>
<td>Polar angle between momentum vector from electrons and observation vector</td>
</tr>
<tr>
<td>(\omega)</td>
<td>The circular frequency of the radiation that is observed.</td>
</tr>
<tr>
<td>(h(r, p))</td>
<td>Normalized phasespace distribution of electrons</td>
</tr>
<tr>
<td>(g(p))</td>
<td>Normalized momentum distribution of electrons</td>
</tr>
<tr>
<td>(g(p))</td>
<td>Normalized momentum distribution of electrons</td>
</tr>
<tr>
<td>(k)</td>
<td>Wavevector of electrons</td>
</tr>
<tr>
<td>(\mathbf{v})</td>
<td>Velocity vector of electrons</td>
</tr>
<tr>
<td>(u)</td>
<td>Normalized momentum of electrons (\beta\gamma)</td>
</tr>
<tr>
<td>(\mathbf{E})</td>
<td>Normalized energy of electrons</td>
</tr>
<tr>
<td>(N)</td>
<td>Denominator of normalized energies</td>
</tr>
<tr>
<td>(F)</td>
<td>Normalized formfactor of electrons, contains phase informations</td>
</tr>
</tbody>
</table>

This plugin allows to predict the emitted virtual transition radiation, which would be caused by the electrons in the simulation box passing through a virtual metal foil which is set at a specific location. The transition radiation can only be calculated for electrons at the moment.

External Dependencies

There are no external dependencies.

.param files

In order to setup the transition radiation plugin, the transitionRadiation.param has to be configured and the radiating particles need to have the attributes weighting, momentum, location, and
transitionRadiationMask (which can be added in `speciesDefinition.param`) as well as the flags `massRatio` and `chargeRatio`.

In `transitionRadiation.param`, the number of frequencies `N_omega` and observation directions `N_theta` and `N_phi` are defined.

**Frequency range**

The frequency range is set up by choosing a specific namespace that defines the frequency setup:

```cpp
/* choose linear frequency range */
namespace radiation_frequencies = linear_frequencies;
```

Currently you can choose from the following setups for the frequency range:

<table>
<thead>
<tr>
<th>namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_frequencies</td>
<td>linear frequency range from <code>SI::omega_min</code> to <code>SI::omega_max</code> with <code>N_omega</code> steps</td>
</tr>
<tr>
<td>log_frequencies</td>
<td>logarithmic frequency range from <code>SI::omega_min</code> to <code>SI::omega_max</code> with <code>N_omega</code> steps</td>
</tr>
<tr>
<td>frequencies_from_list</td>
<td><code>N_omega</code> frequencies taken from a text file with location <code>listLocation[]</code></td>
</tr>
</tbody>
</table>

All three options require variable definitions in the according namespaces as described below:

For the **linear frequency** scale all definitions need to be in the `picongpu::plugins::transitionRadiation::linear_frequencies` namespace. The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **logarithmic frequency** scale all definitions need to be in the `picongpu::plugins::transitionRadiation::log_frequencies` namespace. Equivalently to the linear case, three variables need to be defined: The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **file-based frequency** definition, all definitions need to be in the `picongpu::plugins::transitionRadiation::frequencies_from_list` namespace. The number of total frequencies `N_omega` need to be defined as `constexpr unsigned int` and the path to the file containing the frequency values in units of `[s^{-1}]` needs to be given as `constexpr const char * listLocation = "/path/to/frequency_list";`. The frequency values in the file can be separated by newlines, spaces, tabs, or any other whitespace. The numbers should be given in such a way, that C++ standard `std::ifstream` can interpret the number e.g., as `2.5344e+16`.

**Note:** Currently, the variable `listLocation` is required to be defined in the `picongpu::plugins::transitionRadiation::frequencies_from_list` namespace, even if `frequencies_from_list` is not used. The string does not need to point to an existing file, as long as the file-based frequency definition is not used.

**Observation directions**

The number of observation directions `N_theta` and the distribution of observation directions is defined in `transitionRadiation.param`. There, the function `observation_direction` defines the observation directions.

This function returns the x, y and z component of a **unit vector** pointing in the observation direction.
DINLINE vector_64
observation_direction( int const observation_id_extern )
{
    /* use the scalar index const int observation_id_extern to compute an
       * observation direction (x,y,y) */
    return vector_64( x, y, z );
}

Note: The transitionRadiation.param set up will be subject to further changes, since the
radiationObserver.param it is based on is subject to further changes. These might be namespaces that
describe several preconfigured layouts or a functor if C++ 11 is included in the nvcc.

Foil Position

If one wants to virtually propagate the electron bunch to a foil in a further distance to get a rough estimate of the
effect of the divergence on the electron bunch, one can include a foil position. A foil position which is unequal to
zero, adds the electrons momentum vectors onto the electron until they reach the given y-coordinate. To contain
the longitudinal information of the bunch, the simulation window is actually virtually moved to the foil position
and not each single electron.

namespace SI
{
    // y position of the foil to calculate transition radiation at
    // leave at 0 for no virtual particle propagation
    constexpr float_64 foilPosition = 0.0;
}

Note: This is an experimental feature, which was not verified yet.

Macro-particle form factor

The macro-particle form factor is a method, which considers the shape of the macro particles when computing
the radiation.

One can select between different macro particle shapes. Currently eight shapes are implemented. A shape can be
selected by choosing one of the available namespaces:

/* choosing the 3D CIC-like macro particle shape */
namespace radFormFactor = radFormFactor_CIC_3D;

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radFormFactor_CIC_3D</td>
<td>3D Cloud-In-Cell shape</td>
</tr>
<tr>
<td>radFormFactor_TSC_3D</td>
<td>3D Triangular shaped density cloud</td>
</tr>
<tr>
<td>radFormFactor_PCS_3D</td>
<td>3D Quadratic spline density shape (Piecewise Cubic Spline assignment function)</td>
</tr>
<tr>
<td>radFormFactor_CIC_1Dy</td>
<td>Cloud-In-Cell shape in y-direction, dot like in the other directions</td>
</tr>
<tr>
<td>radFormFactor_Gauss_symmetric</td>
<td>symmetric Gauss charge distribution</td>
</tr>
<tr>
<td>radFormFactor_Gauss_cell</td>
<td>Gaussian charge distribution according to cell size</td>
</tr>
</tbody>
</table>
| radFormFactor_incoherent| forces a completely incoherent emission by scaling the macro particle charge
with the square root of the weighting |
| radFormFactor_coherent  | forces a completely coherent emission by scaling the macro particle charge
with the weighting |
**Note:** One should not confuse this macro-particle form factor with the form factor \( F \), which was previously mentioned. This form factor is equal to the macro-particle shape, while \( F \) contains the phase information of the whole electron bunch. Both are necessary for a physically correct transition radiation calculation.

**Gamma filter**

In order to consider the radiation only of particles with a gamma higher than a specific threshold. In order to do that, the radiating particle species needs the flag `transitionRadiationMask` (which is initialized as `false`) which further needs to be manipulated, to set to true for specific (random) particles.

Using a filter functor as:

```cpp
using GammaFilter = picongpu::particles::manipulators::generic::Free<
    GammaFilterFunctor
>;
```

(see TransitionRadiation example for details) sets the flag to true if a particle fulfills the gamma condition.

**Note:** More sophisticated filters might come in the near future. Therefore, this part of the code might be subject to changes.

**.cfg file**

For a specific (charged) species `<species>` e.g. e, the radiation can be computed by the following commands.

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--&lt;species&gt;_transRad.period</code></td>
<td>Gives the number of time steps between which the radiation should be calculated.</td>
</tr>
</tbody>
</table>

**Memory Complexity**

**Accelerator**

two counters (`float_X`) and two counters (`complex_X`) are allocated permanently

**Host**

as on accelerator.

**Output**

Contains `ASCII` files in `simOutput/transRad` that have the total spectral intensity until the timestep specified by the filename. Each row gives data for one observation direction (same order as specified in the `observer.py`). The values for each frequency are separated by `tabs` and have the same order as specified in `transitionRadiation.param`. The spectral intensity is stored in the units `[J s]`. 
Analysing tools

The `transition_radiation_visualizer.py` in `lib/python/picongpu/plugins/plot_mpl` can be used to analyze the radiation data after the simulation. See `transition_radiation_visualizer.py --help` for more information. It only works, if the input frequency are on a divided logarithmically!

Known Issues

The output is currently only physically correct for electron passing through a metal foil.

References

- **Synthetic characterization of ultrashort electron bunches using transition radiation** Carstens, F.-O., Bachelor thesis on the transition radiation plugin, https://doi.org/10.5281/zenodo.3469663

2.4.23 Period Syntax

Most plugins allow to define a period on how often a plugin shall be executed (notified). Its simple syntax is: `<period>` with a simple number.

Additionally, the following syntax allows to define intervals for periods:

```
<start>[:<end>][:<period>]
```

- `<start>`: begin of the interval; default: 0
- `<end>`: end of the interval, including the upper bound; default: end of the simulation
- `<period>`: notify period within the interval; default: 1

Multiple intervals can be combined via a comma separated list.

Examples

- 42 every 42th time step
- :: equal to just writing 1, every time step from start (0) to the end of the simulation
- 11:11 only once at time step 11
- 10:100:2 every second time step between steps 10 and 100 (included)
- 42,30:50:10: at steps 30 40 42 50 84 126 168 ...
- 5,10: at steps 0 5 10 15 20 25 ... (only executed once per step in overlapping intervals)
2.4.24 Python Postprocessing

In order to further work with the data produced by a plugin during a simulation run, PIConGPU provides python tools that can be used for reading data and visualization. They can be found under \texttt{lib/python/picongpu/plugins}.

It is our goal to provide at least three modules for each plugin to make postprocessing as convenient as possible:

1. a data reader (inside the \texttt{data} subdirectory)
2. a matplotlib visualizer (inside the \texttt{plot_mpl} subdirectory)
3. a jupyter widget visualizer (inside the \texttt{jupyter_widgets} subdirectory) for usage in jupyter-notebooks

Further information on how to use these tools can be found at each plugin page.

If you would like to help in developing those classes for a plugin of your choice, please read \textit{python postprocessing}.

References

2.5 TBG

\textit{Section author: Axel Huebl}

\textit{Module author: René Widera}

Our tool \textit{template batch generator} (tbg) abstracts program runtime options from technical details of supercomputers. On a desktop PC, one can just execute a command interactively and instantaneously. Contrarily on a supercomputer, resources need to be shared between different users efficiently via \textit{job scheduling}. Scheduling on today’s supercomputers is usually done via \textit{batch systems} that define various queues of resources.

An unfortunate aspect about batch systems from a user’s perspective is, that their usage varies a lot. And naturally, different systems have different resources in queues that need to be described.

PIConGPU runtime options are described in \textit{configuration files} (.cfg). We abstract the description of queues, resource acquisition and job submission via \textit{template files} (.tpl). For example, a .cfg file defines how many \textit{devices} shall be used for computation, but a .tpl file calculates how many \textit{physical nodes} will be requested. Also, .tpl files takes care of how to spawn a process when scheduled, e.g. with mpiexec and which flags for networking details need to be passed. After combining the \textit{machine independent} (portable) .cfg file from user input with the \textit{machine dependent} .tpl file, tbg can submit the requested job to the batch system.

Last but not least, one usually wants to store the input of a simulation with its output. tbg conveniently automates this task before submission. The .tpl and the .cfg files that were used to start the simulation can be found in \texttt{<tbg destination dir>/tbg/} and can be used together with the .param files from \texttt{<tbg destination dir>/input/.../param/} to recreate the simulation setup.

In summary, PIConGPU runtime options in .cfg files are portable to any machine. When accessing a machine for the first time, one needs to write template .tpl files, abstractly describing how to run PIConGPU on the specific queue(s) of the batch system. We ship such template files already for a set of supercomputers, interactive execution and many common batch systems. See $\texttt{PICSRC/etc/picongpu/}$ and our list of systems with \textit{.profile files} for details.

2.5.1 Usage

\begin{verbatim}
TBG (template batch generator)
create a new folder for a batch job and copy in all important files

usage: tbg -c [cfgFile] [-s [submitsystem]] [-t [templateFile]]
        [-o "VARNAME1=10 VARNAME2=5"] [-f] [-h]
        [projectPath] destinationPath

-c | --cfg   [file]  - Configuration file to set up batch file.
            Default: [cfgFile] via export TBG_CFGFILE
-s | --submit [command] - Submit command (qsub, "qsub -h", sbatch, ...)

(continues on next page)
\end{verbatim}
2.5.2 .cfg File Macros

Feel free to copy & paste sections of the files below into your .cfg, e.g. to configure complex plugins:

```plaintext
# Copyright 2014-2020 Felix Schmitt, Axel Huebl, Richard Pausch, Heiko Burau
#
# This file is part of PIConGPU.
#
# PIConGPU is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# PIConGPU is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with PIConGPU.
# If not, see <http://www.gnu.org/licenses/>.

# This file describes sections and variables for PIConGPU's
# TBG batch file generator.
# These variables basically wrap PIConGPU command line flags.
# To see all flags available for your PIConGPU binary, run
# picongpu --help. The available flags depend on your configuration flags.
#
# Flags that target a specific species e.g. electrons (--e_png) or ions
# (--i_png) must only be used if the respective species is activated (configure
# flags).
```
## Section: Required Variables

Variables in this section are necessary for PIConGPU to work properly and should not be removed. However, you are free to adjust them to your needs, e.g. setting the number of GPUs in each dimension.

### # Batch system walltime

TBG_wallTime="1:00:00"

### # Number of devices in each dimension (x,y,z) to use for the simulation

TBG_devices_x=1
TBG_devices_y=2
TBG_devices_z=1

### # Size of the simulation grid in cells as "X Y Z"

Note: the number of cells needs to be an exact multiple of a supercell and has to be at least 3 supercells per device. The size of a supercell (in cells) is defined in `memory.param`

TBG_gridSize="128 256 128"

### # Number of simulation steps/iterations as "N"

TBG_steps="100"

### # disable grid size auto adjustment

TBG_disableGridAutoAdjustment="--autoAdjustGrid off"

## Section: Optional Variables

You are free to add and remove variables here as you like. The only exception is TBG_plugins which is used to forward your variables to the TBG program. This variable can be modified but should not be removed!

### # Variables which are created by TBG (should be self-descriptive)

TBG_jobName
TBG_jobNameShort
TBG_cfgPath
TBG_cfgFile
TBG_projectPath
TBG_dstPath

### # version information on startup

TBG_version="--versionOnce"

### # Regex to describe the static distribution of the cells for each device

default: equal distribution over all devices

example for -d 2 4 1 -g 128 192 12

TBG_gridDist="--gridDist '64(2)' '64,32(2),64'"

### # Specifies whether the grid is periodic (1) or not (0) in each dimension (X,Y,Z).

(continues on next page)
# Default: no periodic dimensions
TBG_periodic="--periodic 1 0 1"

# Enables moving window (sliding) in your simulation
TBG_movingWindow="-m"

# Defines when to start sliding the window.
# The window starts sliding at the time required to pass the distance of
# windowMovePoint * (global window size in y) when moving with the speed of light
TBG_windowMovePoint="--windowMovePoint 0.9"

# stop the moving window after given simulation step
TBG_stopWindow="--stopWindow 1337"

### Placeholder for multi data plugins:
#### placeholders must be substituted with the real data name
####
#### <species> = species name e.g. e (electrons), i (ions)
#### <field> = field names e.g. FieldE, FieldB, FieldJ

The following flags are available for the radiation plugin.
For a full description, see the plugins section in the online wiki.
#--<species>_radiation.period Radiation is calculated every .period steps.
    --Currently 0 or 1
#--<species>_radiation.dump Period, after which the calculated radiation data
    --should be dumped to the file system
#--<species>_radiation.lastRadiation If flag is set, the spectra summed
    --between the last and the current dump-time-step are stored
#--<species>_radiation.folderLastRad Folder in which the summed spectra are
    --stored
#--<species>_radiation.totalRadiation If flag is set, store spectra summed
    --from simulation start till current time step
#--<species>_radiation.folderTotalRad Folder in which total radiation spectra
    --are stored
#--<species>_radiation.start Time step to start calculating the radition
#--<species>_radiation.end Time step to stop calculating the radiation
#--<species>_radiation.radPerGPU If flag is set, each GPU stores its own
    --spectra without summing the entire simulation area
#--<species>_radiation.folderRadPerGPU Folder where the GPU specific spectras
    --are stored
#--e_<species>_radiation.compression If flag is set, the hdf5 output will be
    --compressed.
TBG_radiation="--<species>_radiation.period 1 --<species>_radiation.dump 2 --
    --<species>_radiation.totalRadiation
    --<species>_radiation.lastRadiation --<species>_radiation.start
    --<species>_radiation.end 3000"

# The following flags are available for the transition radiation plugin.
# For a full description, see the plugins section in the online documentation.
#--<species>_transRad.period Gives the number of time steps between which the
    --radiation should be calculated.
TBG_transRad="--<species>_transRad.period 1000"

# Create 2D images in PNG format every .period steps.
# The slice plane is defined using .axis [yx,yz] and .slicePoint (offset from origin)
# as a float within [0.0,1.0].
# The output folder can be set with .folder.
# Can be used more than once to print different images, e.g. for YZ and YX planes.

```
TBG_<species>_pngYZ="--<species>_png.period 10 --<species>_png.axis yz --<species>_png.slicePoint 0.5 --<species>_png.folder pngElectronsYZ"
TBG_<species>_pngYX="--<species>_png.period 10 --<species>_png.axis yx --<species>_png.slicePoint 0.5 --<species>_png.folder pngElectronsYX"
```

# Enable macro particle merging

```
TBG_<species>_merger="--<species>_merger.period 100 --<species>_merger.minParticlesToMerge 8 --<species>_merger.posSpreadThreshold 0.2 --<species>_merger.absMomSpreadThreshold 0.01"
```

# Notification period of position plugin (single-particle debugging)

```
TBG_<species>_pos_dbg="--<species>_position.period 1"
```

# Create a particle-energy histogram [in keV] per species for every .period steps

```
TBG_<species>_histogram="--<species>_energyHistogram.period 500 --<species>_energyHistogram.binCount 1024 \
--<species>_energyHistogram.minEnergy 0 --<species>_energyHistogram.maxEnergy 500000 \
--<species>_energyHistogram.filter all"
```

# Calculate a 2D phase space
# - requires parallel libSplash for HDF5 output
# - momentum range in \(m_{<species>} c\)

```
TBG_<species>_PSxpx="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter all --<species>_phaseSpace.space x --<species>_phaseSpace.momentum px --<species>_phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSxpz="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter all --<species>_phaseSpace.space x --<species>_phaseSpace.momentum pz --<species>_phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSypx="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter all --<species>_phaseSpace.space y --<species>_phaseSpace.momentum px --<species>_phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSypz="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter all --<species>_phaseSpace.space y --<species>_phaseSpace.momentum pz --<species>_phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
```

# Write out slices of field data for every .period step

```
TBG_EField_slice="/E_slice.period 100 --E_slice.fileName sliceE --E_slice.plane 2 --E_slice.slicePoint 0.5"
TBG_BField_slice="/B_slice.period 100 --B_slice.fileName sliceB --B_slice.plane 2 --B_slice.slicePoint 0.5"
TBG_JField_slice="/J_slice.period 100 --J_slice.fileName sliceJ --J_slice.plane 2 --J_slice.slicePoint 0.5"
```

# Sum up total energy every .period steps for
# - species (--<species>_energy)
# - fields (<!--fields_energy)

```
TBG_sumEnergy="/fields_energy.period 10 --<species>_energy.period 10 --<species>_energy.filter all"
```

# Count the number of macro particles per species for every .period steps

(continues on next page)
TBG_macroCount="--<species>_macroParticlesCount.period 100"

# Count makro particles of a species per super cell
TBG_countPerSuper="--<species>_macroParticlesPerSuperCell.period 100 --<species>_macroParticlesPerSuperCell.period 100"

# Dump simulation data (fields and particles) to HDF5 files using libSplash.
# Data selected in .source is dumped every .period steps to the fileset .file.
TBG_hdf5="--hdf5.period 100 --hdf5.file simData --hdf5.source 'species_all,fields_all'

# Dump simulation data (fields and particles) to ADIOS files.
# Data is dumped every .period steps to the fileset .file.
TBG_adios="--adios.period 100 --adios.file simData --adios.source 'species_all,fields_all'

# see 'adios_config -m', e.g., for on-the-fly zlib compression
# (compile ADIOS with --with-zlib=<ZLIB_ROOT>)
# --adios.compression zlib
# or
# --adios.compression blosc:threshold=2048,shuffle=bit,lvl=1,threads=6,
# compressor=zstd
# for parallel large-scale parallel file-systems:
# --adios.aggregators <N * 3> --adios.ost <N>
# avoid writing meta file on massively parallel runs
# --adios.disable-meta <B>
# B = 0 is equal to false, B = 1 is true
# specify further options for the transports, see ADIOS manual
# chapter 6.1.5, e.g., 'random_offset=1;stripe_count=4'
# --adios.transport-params "semicolon_separated_list"
# select data sources for the dump
# --adios.source "comma_separated_list_of_data_sources"

# Create a checkpoint that is restartable every --checkpoint.period steps
# http://git.io/PToFYg
TBG_checkpoint="--checkpoint.period 1000"

# Select the backend for the checkpoint, available are hdf5 and adios
# --checkpoint.backend adios
# hdf5
# Available backend options are exactly as in --adios.* and --hdf5.* and can be set
# via:
# --checkpoint.<IO-backend>.* <value>
# e.g.:
# --checkpoint.adios.compression zlib
# --checkpoint.adios.disable-meta 1
# Note: if you disable ADIOS meta files in checkpoints, make sure to run
# 'bpmeta' on your checkpoints before restarting from them!

# Restart the simulation from checkpoint created using TBG_checkpoint
TBG_restart="--checkpoint.restart"

# Select the backend for the restart (must fit the created checkpoint)
# --checkpoint.restart.backend adios
# hdf5
# By default, the last checkpoint is restarted if not specified via
# --checkpoint.restart.step 1000
# To restart in a new run directory point to the old run where to start from
# --checkpoint.restart.directory /path/to/simOutput/checkpoints

# Presentation mode: loop a simulation via restarts
# does either start from 0 again or from the checkpoint specified with
# --checkpoint.restart.step as soon as the simulation reached the last time step;
# in the example below, the simulation is run 5000 times before it shuts down
TBG_restartLoop="--checkpoint.restart.loop 5000"

# Live in situ visualization using ISAAC
# Initial period in which a image shall be rendered
# --isaac.period PERIOD
# Name of the simulation run as seen for the connected clients
# --isaac.name NAME
# URL of the server
# --isaac.url URL
# Number from 1 to 100 decribing the quality of the transceived jpeg image.
# Smaller values are faster sent, but of lower quality
# --isaac.quality QUALITY
# Resolution of the rendered image. Default is 1024x768
# --isaac.width WIDTH
# --isaac.height HEIGHT
# Pausing directly after the start of the simulation
# --isaac.directPause
# By default the ISAAC Plugin tries to reconnect if the sever is not available
# at start or the servers crashes. This can be deactivated with this option
# --isaac.reconnect false
TBG_isaac="--isaac.period 1 --isaac.name !TBG_jobName --isaac.url <server_url>"
TBG_isaac_quality="--isaac.quality 90"
TBG_isaac_resolution="--isaac.width 1024 --isaac.height 768"
TBG_isaac_pause="--isaac.directPause"
TBG_isaac_reconnect="--isaac.reconnect false"

# Print the maximum charge deviation between particles and div E to textfile
# -->'chargeConservation.dat':
TBG_chargeConservation="--chargeConservation.period 100"

# Particle calorimeter: (virtually) propagates and collects particles to infinite_
#--distance
TBG_<species>_calorimeter="--<species>_calorimeter.period 100 --<species>_--
calorimeter.openingYaw 90 --<species>_calorimeter.openingPitch 30
--<species>_calorimeter.numBinsEnergy 32 --<species>_--
calorimeter.minEnergy 10 --<species>_calorimeter.maxEnergy 1000
--<species>_calorimeter.logScale 1 --<species>_calorimeter.
--file filePrefix --<species>_calorimeter.filter all"

# Resource log: log resource information to streams or files
# set the resources to log by --resourceLog.properties [rank, position,_
# currentStep, particleCount, cellCount]
# set the output stream by --resourceLog.stream [stdout, stderr, file]
# set the prefix of filestream --resourceLog.prefix [prefix]
# set the output format by (pp == pretty print) --resourceLog.format jsonpp [json,
# jsonpp, xml, xmlpp]
# The example below logs all resources for each time step to stdout in the pretty_
# print json format
TBG_resourceLog="--resourceLog.period 1 --resourceLog.stream stdout
--resourceLog.properties rank position currentStep particleCount,cellCount
--resourceLog.format jsonpp"

# Section: Program Parameters
# This section contains TBG internal variables, often composed from required
# variables. These should not be modified except when you know what you are doing!
2.5.3 Batch System Examples

Section author: Axel Huebl, Richard Pausch

Slurm

Slurm is a modern batch system, e.g. installed on the Taurus cluster at TU Dresden, Hemera at HZDR, Cori at NERSC, among others.

Job Submission

PIConGPU job submission on the Taurus cluster at TU Dresden:

• `tbg -s sbatch -c etc/picongpu/0008gpus.cfg -t etc/picongpu/taurus-tud/k80.tpl $SCRATCH/runs/test-001`

Job Control

• interactive job:
  
  - `salloc --time=1:00:00 --nodes=1 --ntasks-per-node=2 --cpus-per-task=8 --partition gpu-interactive`
  
  - `e.g. srun "hostname"`
  
  - GPU allocation on taurus requires an additional flag, e.g. for two GPUs `--gres=gpu:2`

• details for my jobs:
  
  - `scontrol -d show job 12345 all details for job with <job id> 12345`
  
  - `squeue -u $(whoami) -l all jobs under my user name`

• details for queues:
  
  - `squeue -p queueName -l list full queue`
  
  - `squeue -p queueName --start (show start times for pending jobs)`
  
  - `squeue -p queueName -l -t R (only show running jobs in queue)`
  
  - `sinfo -p queueName (show online/offline nodes in queue)`
  
  - `sview (alternative on taurus: module load llview and llview)`
- `scontrol show partition queueName`

  • communicate with job:
    - `scancel <job id>` abort job
    - `scancel -s <signal number> <job id>` send signal or signal name to job
    - `scontrol update timelimit=4:00:00 jobid=12345` change the walltime of a job
    - `scontrol update jobid=12345 dependency=afterany:54321` only start job after job with id 54321 has finished
    - `scontrol hold <job id>` prevent the job from starting
    - `scontrol release <job id>` release the job to be eligible for run (after it was set on hold)

**PBS**

PBS (for Portable Batch System) is a widely distributed batch system that comes in several implementations (open, professional, etc.). It is used, e.g. on Hypnos at HZDR.

**Job Submission**

PIConGPU job submission on the Hypnos cluster at HZDR:

- `tbg -s qsub -c etc/picongpu/0008gpus.cfg -t etc/picongpu/hypnos-hzdr/k20.tpl /bigdata/hplsim/<...>/test-001`

Where `<...>` is one of:

- external/$(whoami)
- internal:
  - scratch/$(whoami)
  - development/$(whoami)
  - production/<project name>

**Job Control**

- interactive job:
  - `qsub -I -q k20 -lwalltime=12:00:00 -lnodes=1:ppn=8`

- details for my jobs:
  - `qstat -f 12345 all details for job with <job id> 12345`
  - `qstat -u $(whoami)` all jobs under my user name

- details for queues:
  - `qstat -a queueName show all jobs in a queue`
  - `pbs_free -l` compact view on free and busy nodes
  - `pbsnodes` list all nodes and their detailed state (free, busy/job-exclusive, offline)

- communicate with job:
  - `qdel <job id>` abort job
  - `qsig -s <signal number> <job id>` send signal or signal name to job
  - `qalter -lwalltime=12:00:00 <job id>` change the walltime of a job
PIConGPU Documentation, Release 0.5.0

- qalter -Wdepend=afterany:54321 12345 only start job 12345 after job with id 54321 has finished
- qhold <job id> prevent the job from starting
- qrls <job id> release the job to be eligible for run (after it was set on hold)

LSF

LSF (for Load Sharing Facility) is an IBM batch system (bsub/BSUB). It is used, e.g. on Summit at ORNL.

Job Submission

PIConGPU job submission on the Summit cluster at Oak Ridge National Lab:

- tbg -s bsub -c etc/picongpu/0008gpus.cfg -t etc/picongpu/summit-ornl/gpu.tpl $PROJWORK/$proj/test-001

Job Control

- interactive job:
  - bsub -P $proj -W 2:00 -nnodes 1 -Is /bin/bash
- details for my jobs:
  - bjobs 12345 all details for job with <job id> 12345
  - bjobs [-l] all jobs under my user name
  - jobstat -u $(whoami) job eligibility
  - bjdepinfo 12345 job dependencies on other jobs
- details for queues:
  - bqueues list queues
- communicate with job:
  - bkill <job id> abort job
  - bpeek [-f] <job id> peek into stdout/stderr of a job
  - bkill -s <signal number> <job id> send signal or signal name to job
  - bckhpnt and brestart checkpoint and restart job (untested/unimplemented)
  - bmod -W 1:30 12345 change the walltime of a job (currently not allowed)
  - bstop <job id> prevent the job from starting
  - bresume <job id> release the job to be eligible for run (after it was set on hold)

References


2.6 Python

This section contains python utilities for more comfortable working with PIConGPU.
2.6.1 Memory Calculator

To aid you in the planning and setup of your simulation PIConGPU provides python tools for educated guesses on simulation parameters. They can be found under lib/python/picongpu/utils.

Calculate the memory requirement per device.

```
from picongpu.utils import MemoryCalculator
```

class picongpu.utils.memory_calculator.MemoryCalculator

Memory requirement calculation tool for PIConGPU

Contains calculation for fields, particles, random number generator and the calorimeter plugin. In-situ methods other than the calorimeter so far use up negligible amounts of memory on the device.

__init__ (n_x, n_y, n_z, precision_bits=32)

Class constructor

Parameters

- n_x (int) – number of cells in x direction (per device)
- n_y (int) – number of cells in y direction (per device)
- n_z (int) – number of cells in z direction (per device)
- precision_bits (int) – floating point precision for PIConGPU run

mem_req_by_calorimeter (n_energy, n_yaw, n_pitch, value_size=None)

Memory required by the particle calorimeter plugin. Each of the (n_energy x n_yaw x n_pitch) bins requires a value (32/64 bits). The whole calorimeter is represented twice on each device, once for particles in the simulation and once for the particles that leave the box.

Parameters

- n_energy (int) – number of bins on the energy axis
- n_yaw (int) – number of bins for the yaw angle
- n_pitch (int) – number of bins for the pitch angle
- value_size (int) – value size in particle calorimeter {unit: byte} (default: 4)

Returns

req_mem – required memory {unit: bytes} per device

Return type

int

mem_req_by_fields (n_x=None, n_y=None, n_z=None, field_tmp_slots=1, particle_shape_order=2, sim_dim=3, pml_n_x=0, pml_n_y=0, pml_n_z=0)

Memory reserved for fields on each device

Parameters

- n_x (int) – number of cells in x direction (per device)
- n_y (int) – number of cells in y direction (per device)
- n_z (int) – number of cells in z direction (per device)
- field_tmp_slots (int) – number of slots for temporary fields (see PIConGPU memory.param: fieldTmpNumSlots)
- particle_shape_order (int) – numerical order of the particle shape (see PIConGPU species.param: e.g. particles::shapes::PCS : 3rd order)
- sim_dim (int) – simulation dimension (available for PIConGPU: 2 and 3)
- pml_n_x (int) – number of PML cells in x direction, combined for both sides
- pml_n_y (int) – number of PML cells in y direction, combined for both sides
• `pml_n_z (int)` – number of PML cells in z direction, combined for both sides

**Returns** `req_mem` – required memory [unit: bytes] per device

**Return type** `int`

`mem_req_by_particles` (target_n_x=None, target_n_y=None, target_n_z=None, num_additional_attributes=0, particles_per_cell=2, sim_dim=3)
Memory reserved for all particles of a species on a device. We currently neglect the constant species memory.

**Parameters**

• `target_n_x (int)` – number of cells in x direction containing the target
• `target_n_y (int)` – number of cells in y direction containing the target
• `target_n_z (int)` – number of cells in z direction containing the target
• `num_additional_attributes (int)` – number of additional attributes like e.g. `boundElectrons`
• `particles_per_cell (int)` – number of particles of the species per cell
• `sim_dim (int)` – simulation dimension (available for PIConGPU: 2 and 3)

**Returns** `req_mem` – required memory [unit: bytes] per device and species

**Return type** `int`

`mem_req_by_rng` (n_x=None, n_y=None, n_z=None, generator_method='XorMin')
Memory reserved for the random number generator state on each device.

Check `random.param` for a choice of random number generators. If you find that your required RNG state is large (> 300 MB) please see `memory.param` for a possible adjustment of the reservedGpuMemorySize.

**Parameters**

• `n_x (int)` – number of cells in x direction (per device)
• `n_y (int)` – number of cells in y direction (per device)
• `n_z (int)` – number of cells in z direction (per device)
• `generator_method (str)` – random number generator method - influences the state size per cell possible options: “XorMin”, “MRG32k3aMin”, “AlpakaRand” - (GPU default: “XorMin”) - (CPU default: “AlpakaRand”)

**Returns** `req_mem` – required memory [unit: bytes] per device

**Return type** `int`

### 2.7 Example Setups

#### 2.7.1 Bremsstrahlung: Emission of Bremsstrahlung from Laser-Foil Interaction

*Section author: Heiko Burau &lt;h.burau (at) hzdr.de&gt;*

*Module author: Heiko Burau &lt;h.burau (at) hzdr.de&gt;*

This is a simulation of a flat solid density target hit head-on by a high-intensity laser pulse. At the front surface free electrons are accelerated up to ultra relativistic energies and start travelling through the bulk then. Meanwhile, due to ion interaction, the hot electrons lose a small fraction of their kinetic energy in favor of emission of Bremsstrahlung-photons. Passing over the back surface hot electrons are eventually reflected and re-enter the foil in opposite direction. Because of the ultra-relativistic energy Bremsstrahlung (BS) is continuously emitted mainly along the direction of motion of the electron. The BS-module models the electron-ion scattering as three
single processes, including electron deflection, electron deceleration and photon creation with respect to the emission angle. Details of the implementation and the numerical model can be found in [BurauDipl]. Details of the theoretical description can be found in [Jackson] and [Salvat].

This 2D test simulates a laser pulse of a_0=40, lambda=0.8µm, w0=1.5µm in head-on collision with a fully pre-ionized gold foil of 2µm thickness.

**Checks**

- check appearance of photons moving along (forward) and against (backward) the incident laser pulse direction.
- check photon energy spectrum in both directions for the forward moving photons having a higher energy.

**References**

2.7.2 Bunch: Thomson scattering from laser electron-bunch interaction

*Section author: Richard Pausch <r.pausch (at) hzdr.de>*

*Module author: Richard Pausch <r.pausch (at) hzdr.de>, Rene Widera <r.widera (at) hzdr.de>*

This is a simulation of an electron bunch that collides head-on with a laser pulse. Depending on the number of electrons in the bunch, their momentum and their distribution and depending on the laser wavelength and intensity, the emitted radiation differs. A general description of this simulation can be found in [PauschDipl]. A detailed analysis of this bunch simulation can be found in [Pausch13]. A theoretical study of the emitted radiation in head-on laser electron collisions can be found in [Esarey93].

This test simulates an electron bunch with a relativistic gamma factor of gamma=5.0 and with a laser with a_0=1.0. The resulting radiation should scale with the number of real electrons (incoherent radiation).

**References**

2.7.3 Empty: Default PIC Algorithm

*Section author: Axel Huebl <a.huebl (at) hzdr.de>*

This is an “empty” example, initializing a default particle-in-cell cycle with default algorithms [BirdsallLangdon] [HockneyEastwood] but without a specific test case. When run, it iterates a particle-in-cell algorithm on a vacuum without particles or electro-magnetic fields initialized, which are the default .param files in include/picongpu/param/.

This is a case to demonstrate and test these defaults are still (syntactically) working. In order to set up your own simulation, there is no need to overwrite all .param files but only the ones that are different from the defaults. As an example, just overwrite the default laser (none) and initialize a species with a density distribution.

**References**

2.7.4 FoilLCT: Ion Acceleration from a Liquid-Crystal Target

*Section author: Axel Huebl*

*Module author: Axel Huebl, T. Kluge*

The following example models a laser-ion accelerator in the [TNSA] regime. An optically over-dense target (n_{max} = 192n_c) consisting of a liquid-crystal material 8CB (4-octyl-4’-cyanobiphenyl) C_{21}H_{25}N is used.

Irradiated with a high-power laser pulse with a_0 = 5 the target is assumed to be partly pre-ionized due to realistic laser contrast and pre-pulses to O^{2+}, H^+ and N^{2+} while being slightly expanded on its surfaces (modeled as
exponential density slope). The overall target is assumed to be initially quasi-neutral and the 8CB ion components are are not demixed in the surface regions. Surface contamination with, e.g. water vapor is neglected.

The laser is assumed to be in focus and approximated as a plane wave with temporally Gaussian intensity envelope of $\tau_{FWHM} = 25$ fs.

This example is used to demonstrate:

- an ion acceleration setup with
- composite, multi ion-species target material
- quasi-neutral initial conditions
- ionization models for field ionization and collisional ionization

with PIConGPU.

References

2.7.5 KelvinHelmholtz: Kelvin-Helmholtz Instability

Section author: Axel Huebl <a.huebl (at) hzdr.de>
Module author: Axel Huebl <a.huebl (at) hzdr.de>, E. Paulo Alves, Thomas Grismayer

This example simulates a shear-flow instability known as the Kelvin-Helmholtz Instability in a near-relativistic setup as studied in [Alves12], [Grismayer13], [Bussmann13]. The default setup uses a pre-ionized quasi-neutral hydrogen plasma. Modifying the ion species’ mass to resample positrons instead is a test we perform regularly to control numerical heating and charge conservation.

References

2.7.6 LaserWakefield: Laser Electron Acceleration

Section author: Axel Huebl <a.huebl (at) hzdr.de>
Module author: Axel Huebl <a.huebl (at) hzdr.de>, René Widera, Heiko Burau, Richard Pausch, Marco Garten

Setup for a laser-driven electron accelerator [TajimaDawson] in the blowout regime of an underdense plasma [Modena] [PukhovMeyerterVehn]. A short (fs) laser beam with ultra-high intensity ($a_0 >> 1$), modeled as a finite Gaussian beam is focussed in a hydrogen gas target. The target is assumed to be pre-ionized with negligible temperature. The relevant area of interaction is followed by a co-moving window, in whose time span the movement of ions is considered irrelevant which allows us to exclude those from our setup.

This is a demonstration setup to get a visible result quickly and test available methods and I/O. The plasma gradients are unphysically high, the resolution of the laser wavelength is seriously bad, the laser parameters (e.g. pulse length, focusing) are challenging to achieve technically and interaction region is too close to the boundaries of the simulation box. Nevertheless, this setup will run on a single GPU in full 3D in a few minutes, so just enjoy running it and interact with our plugins!

References

2.7.7 TransitionRadiation : Transtion Radiation

Section author: Finn-Ole Carstens <f.carstens (at) hzdr.de>

This example simulates the coherent and incoherent transition radiation created by an electron bunch in-situ. The implemented transition radiation follows the studies from [Schroeder2004] and [Downer2018]. The transition radiation is computed for an infinitely large interface perpendicular to the y-axis of the simulation.
The electron bunch in this setup is moving with a 45° angle in the x-y plane with a Lorentz-factor of $\gamma = 100$. The bunch has a Gaussian distribution with $\sigma_y = 3.0 \mu m$. The results can be interpreted with the according python script `lib/python/picongpu/plugins/plot_mpl/transition_radiation_visualizer.py`.

References

2.7.8 WarmCopper: Average Charge State Evolution of Copper Irradiated by a Laser

Section author: Axel Huebl <a.huebl (at) hzdr.de>

Module author: Axel Huebl <a.huebl (at) hzdr.de>, Hyun-Kyung Chung

This setup initializes a homogenous, non-moving, copper block irradiated by a laser with $10^{18} \text{ W/cm}^3$ as a benchmark for [SCFLY] atomic population dynamics. We follow the setup from [FLYCHK] page 10, figure 4 assuming a quasi 0D setup with homogenous density of a 1+ ionized copper target. The laser (not modeled) already generated a thermal electron density at 10, 100 or 1000 eV and a delta-distribution like “hot” electron distribution with 200 keV (directed stream). The observable of interest is $\langle Z \rangle$ over time of the copper ions. For low thermal energies, collisional excitation, de-excitation and recombinations should be sufficient to reach the LTE state after about 0.1-1 ps. For higher initial temperatures, radiative rates get more relevant and the Non-LTE steady-state solution can only be reached correctly when also adding radiative rates.

Note: FLYlite is still in development!

References

2.8 Workflows

This section contains typical user workflows and best practices.

2.8.1 Setting the Number of Cells

Section author: Axel Huebl

Together with the grid resolution in `grid.param`, the number of cells in our `.cfg` files determine the overall size of a simulation (box). The following rules need to be applied when setting the number of cells:

Each device needs to:

1. contain an integer multiple of supercells
2. at least three supercells
3. for non periodic boundary conditions, the number of absorbing boundary cells for devices at the simulation boundary (see `grid.param`) must fit into the local volume

The grid size will be automatically adjusted if the conditions above are not fulfilled. This behavior can be disabled by using the command line option `--autoAdjustGrid off`

Supercell sizes in terms of number of cells are set in `memory.param` and are by default $8 \times 8 \times 4$ for 3D3V simulations on GPUs. For 2D3V simulations, $16 \times 16$ is usually a good supercell size, however the default is simply cropped to $8 \times 8$, so make sure to change it to get more performance.

---

1 In PICconGPU, we generally refer to the implemented subset of SCFLY (solving Non-LTE population kinetics) as FLYlite.
2.8.2 Changing the Resolution with a Fixed Target

Section author: Axel Huebl

One often wants to refine an already existing resolution in order to model a setup more precisely or to be able to model a higher density.

1. change cell sizes and time step in `grid.param`
2. change number of GPUs in `.cfg` file
3. change number of number of cells and distribution over GPUs in `.cfg` file
4. adjust (transversal) positioning of targets in `density.param`
5. recompile

2.8.3 Calculating the Memory Requirement per Device

Section author: Marco Garten

The planning of simulations for realistically sized problems requires a careful estimation of memory usage and is often a trade-off between resolution of the plasma, overall box size and the available resources. The file `memory_calculator.py` contains a class for this purpose.

The following paragraph shows the use of the `MemoryCalculator` for the 4.cfg setup of the FoilLCT example.

It is an estimate for how much memory is used per device if the whole target would be fully ionized but does not move much. Of course, the real memory usage depends on the case and the dynamics inside the simulation. We calculate the memory of just one device per row of GPUs in laser propagation direction. We hereby assume that particles are distributed equally in the transverse direction like it is set up in the FoilLCT example.

We encourage to try out this script with different settings, to see how they influence the distribution of the total memory requirement between devices.

```python
from picongpu.utils import MemoryCalculator
from math import ceil

cell_size = 0.8e-6 / 384.  # 2.083e-9 m
y0 = 0.5e-6  # position of foil front surface (m)
y1 = 1.5e-6  # position of foil rear surface (m)
L = 10e-9  # pre-plasma scale length (m)
L_cutoff = 4 * L  # pre-plasma length (m)

sim_dim = 2
# number of cells in the simulation
Nx_all, Ny_all, Nz_all = 256, 1280, 1
# number of GPU rows in each direction
x_rows, y_rows, z_rows = 2, 2, 1
# number of cells per GPU
Nx, Ny, Nz = Nx_all / x_rows, Ny_all / y_rows, Nz_all / z_rows

vacuum_cells = ceil((y0 - L_cutoff) / cell_size)  # in front of the target
# target cells (between surfaces + pre-plasma)
target_cells = ceil((y1 - y0 + 2 * L_cutoff) / cell_size)
# number of cells (y direction) on each GPU row
GPU_rows = [0] * y_rows

cells_to_spread = vacuum_cells + target_cells
# spread the cells on the GPUs
for ii, _ in enumerate(GPU_rows):
    if cells_to_spread >= Ny:
        GPU_rows[ii] = Ny
```

(continues on next page)
cells_to_spread -= Ny
else:
    GPU_rows[ii] = cells_to_spread
    break
# remove vacuum cells from the front rows
extra_cells = vacuum_cells
for ii, _ in enumerate(GPU_rows):
    if extra_cells >= Ny:
        GPU_rows[ii] = 0
        extra_cells -= Ny
    else:
        GPU_rows[ii] -= extra_cells
        break

cells_to_spread -= Ny

# remove vacuum cells from the front rows
extra_cells = vacuum_cells
for ii, _ in enumerate(GPU_rows):
    if extra_cells >= Ny:
        GPU_rows[ii] = 0
        extra_cells -= Ny
    else:
        GPU_rows[ii] -= extra_cells
        break

cells_to_spread -= Ny

# remove vacuum cells from the front rows
extra_cells = vacuum_cells
for ii, _ in enumerate(GPU_rows):
    if extra_cells >= Ny:
        GPU_rows[ii] = 0
        extra_cells -= Ny
    else:
        GPU_rows[ii] -= extra_cells
        break

pmc = MemoryCalculator(Nx, Ny, Nz)

# typical number of particles per cell which is multiplied later for
# each species and its relative number of particles
N_PPC = 6

# conversion factor to megabyte
megabyte = 1.0 / (1024 * 1024)

target_x = Nx  # full transverse dimension of the GPU
target_z = Nz

def sx(n):
    return {1: "st", 2: "nd", 3: "rd"}.get(n if n < 20
    else int(str(n)[-1]), "th")

for row, target_y in enumerate(GPU_rows):
    print("{}{} row of GPUs:"
          .format(row + 1, sx(row + 1)))
    print("* Memory requirement per GPU:")
    # field memory per GPU
    field_gpu = pmc.mem_req_by_fields(Nx, Ny, Nz, field_tmp_slots=2,
                                      particle_shape_order=2, sim_dim=sim_dim)
    print(" + fields: {:.2f} MB".format(
        field_gpu * megabyte))

    # electron macroparticles per supercell
    e_PPC = N_PPC * ({
        # H,C,N pre-ionization - higher weighting electrons
        3
        # electrons created from C ionization
        + (6 - 2)
        # electrons created from N ionization
        + (7 - 2)
    }
    # particle memory per GPU - only the target area contributes here
    e_gpu = pmc.mem_req_by_particles(
        target_x, target_y, target_z,
        num_additional_attributes=0,
        particles_per_cell=e_PPC
    )
    H_gpu = pmc.mem_req_by_particles(
        target_x, target_y, target_z,
        # no bound electrons since H is pre-ionized
        num_additional_attributes=0,
        particles_per_cell=N_PPC
    )
C_gpu = pmc.mem_req_by_particles(
    target_x, target_y, target_z,  
    num_additional_attributes=1,  # number of bound electrons  
    particles_per_cell=N_PPC
)
N_gpu = pmc.mem_req_by_particles(
    target_x, target_y, target_z,  
    num_additional_attributes=1,  
    particles_per_cell=N_PPC
)
# memory for calorimeters  
cal_gpu = pmc.mem_req_by_calorimeter(
    n_energy=1024, n_yaw=360, n_pitch=1
) * 2  # electrons and protons  
# memory for random number generator states  
rng_gpu = pmc.mem_req_by_rng(Nx, Ny, Nz)

print(" + species:")
print(" - e: {:.2f} MB".format(e_gpu * megabyte))
print(" - H: {:.2f} MB".format(H_gpu * megabyte))
print(" - C: {:.2f} MB".format(C_gpu * megabyte))
print(" - N: {:.2f} MB".format(N_gpu * megabyte))
print(" + RNG states: {:.2f} MB".format(
    rng_gpu * megabyte))
print(" + particle calorimeters: {:.2f} MB".format(
    cal_gpu * megabyte))

mem_sum = field_gpu + e_gpu + H_gpu + C_gpu + N_gpu + rng_gpu + cal_gpu
print("* Sum of required GPU memory: {:.2f} MB".format(
    mem_sum * megabyte))

This will give the following output:

1st row of GPUs:
* Memory requirement per GPU:
  + fields: 4.58 MB
  + species:
    - e: 114.16 MB
    - H: 9.51 MB
    - C: 10.74 MB
    - N: 10.74 MB
  + RNG states: 1.88 MB
  + particle calorimeters: 5.62 MB
* Sum of required GPU memory: 157.23 MB
2nd row of GPUs:
* Memory requirement per GPU:
  + fields: 4.58 MB
  + species:
    - e: 27.25 MB
    - H: 2.27 MB
    - C: 2.56 MB
    - N: 2.56 MB
  + RNG states: 1.88 MB
  + particle calorimeters: 5.62 MB
* Sum of required GPU memory: 46.72 MB

If you have a machine or cluster node with NVIDIA GPUs you can find out the available memory size by typing nvidia-smi on a shell.
# 2.8.4 Setting the Laser Initialization Cut-Off

*Section author: Axel Huebl*

Laser profiles for simulation are modeled with a temporal envelope. A common model assumes a Gaussian intensity distribution over time which by definition never sets to zero, so it needs to be cut-off to a reasonable range.

In `laser.param` each profile implements the cut-off to start (and end) initializing the laser profile via a parameter `PULSE_INIT t_{\text{init}}` (sometimes also called `RAMP_INIT`). \( t_{\text{init}} \) is given in units of the `PULSE_LENGTH \tau` which is implemented laser-profile dependent (but usually as \( \sigma_I \) of the standard Gaussian of intensity \( I = E^2 \)).

For a fixed target in distance \( d \) to the lower \( y = 0 \) boundary of the simulation box, the maximum intensity arrives at time:

\[
    t_{\text{laserPeakOnTarget}} = \frac{t_{\text{init}} \cdot \tau}{2} + \frac{d}{c_0}
\]

or in terms of discrete time steps \( \Delta t \):

\[
    \text{step}_{\text{laserPeakOnTarget}} = \frac{t_{\text{laserPeakOnTarget}}}{\Delta t}.
\]

**Note:** Moving the spatial plane of initialization of the laser pulse via `initPlaneY` does not change the formula above. The implementation covers this spatial offset during initialization.

# 2.8.5 Definition of Composite Materials

*Section author: Axel Huebl*

The easiest way to define a composite material in PIConGPU is starting relative to an idealized full-ionized electron density. As an example, let's use \( \text{C}_{21}\text{H}_{25}\text{N} \) ("8CB") with a plasma density of \( n_{e,\text{max}} = 192 n_c \) contributed by the individual ions relatively as:

- Carbon: \( 21 \cdot 6/N_{\Sigma e} \)
- Hydrogen: \( 25 \cdot 1/N_{\Sigma e} \)
- Nitrogen: \( 1 \cdot 7/N_{\Sigma e} \)

and \( N_{\Sigma e} = 21\text{C} \cdot 6\text{C}^+ + 25\text{H} \cdot 1\text{H}^+ + 1\text{N} \cdot 7\text{N}^+ = 158 \).

Set the idealized electron density in `density.param` as a reference and each species’ relative `densityRatio` from the list above accordingly in `speciesDefinition.param` (see the input files in the `FoilLCT` example for details).

In order to initialize the electro-magnetic fields self-consistently, read quasi-neutral initialization.

# 2.8.6 Quasi-Neutral Initialization

*Section author: Axel Huebl*

In order to initialize the electro-magnetic fields self-consistently, one needs to fulfill Gauss’s law \( \nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \) (and \( \nabla \cdot \mathbf{B} = 0 \)). The trivial solution to this equation is to start field neutral by microscopically placing a charge-compensating amount of free electrons on the same position as according ions.

## Fully Ionized Ions

For fully ionized ions, just use `ManipulateDerive` in `speciesInitialization.param` and derive macro-electrons \( 1 : 1 \) from macro-ions but increase their weighting by \( 1 : Z \) of the ion.
using InitPipeline = bmpl::vector<
    /* density profile from density.param and *
     * start position from particle.param */
CreateDensity<
    densityProfiles::YourSelectedProfile,
    startPosition::YourStartPosition,
    Carbon
>,
/* create a macro electron for each macro carbon but increase its *
   * weighting by the ion's proton number so it represents all its *
   * electrons after an instantaneous ionization */
ManipulateDerive<
    manipulators::ProtonTimesWeighting,
    Carbon,
    Electrons
>;

If the Carbon species in this example has an attribute boundElectrons (optional, see speciesAttributes.param and speciesDefinition.param) and its value is not manipulated the default value is used (zero bound electrons, fully ionized). If the attribute boundElectrons is not added to the Carbon species the charge state is considered constant and taken from the chargeRatio< ... > particle flag.

Partly Ionized Ions

For partial pre-ionization, the FoilLCT example shows a detailed setup. First, define a functor that manipulates the number of bound electrons in particle.param, e.g. to twice pre-ionized.

```cpp
#include "picongpu/particles/traits/GetAtomicNumbers.hpp"
// ...
namespace manipulators
{
    //! ionize ions twice
    struct TwiceIonizedImpl
    {
        template<typename T_Particle>
        DINLINE void operator()(
            T_Particle& particle
        )
        {
            constexpr float_X protonNumber =
                GetAtomicNumbers< T_Particle >::type::numberOfProtons;
            particle[boundElectrons_] = protonNumber - float_X(2.);
        }
    };

    //! definition of TwiceIonizedImpl manipulator
    using TwiceIonized = generic::Free< TwiceIonizedImpl >;
} // namespace manipulators

Then again in speciesInitialization.param set your initialization routines to:

using InitPipeline = bmpl::vector<
    /* density profile from density.param and *
     * start position from particle.param */
CreateDensity<
    densityProfiles::YourSelectedProfile,
    startPosition::YourStartPosition,
    Carbon
>,
/* create a macro electron for each macro carbon but increase its *
   * weighting by the ion's proton number so it represents all its *
   * electrons after an instantaneous ionization */
ManipulateDerive<
    manipulators::ProtonTimesWeighting,
    Carbon,
    Electrons
>;

(continues on next page)
Carbon
>IRMWARE /* partially pre-ionize the carbons by manipulating the carbon's
> 'boundElectrons' attribute,
> * functor defined in particle.param: set to C2+ */
Manipulate<
  manipulators::TwiceIonized,
  Carbon
>,
> /* does also manipulate the weighting x2 while deriving the electrons
> * ("twice pre-ionized") since we set carbon as C2+ */
ManipulateDerive<
  manipulators::binary::UnboundElectronsTimesWeighting,
  Carbon,
  Electrons
>;

2.8.7 Probe Particles

Section author: Axel Huebl

Probe particles ("probes") can be used to record field quantities at selected positions over time.

As a geometric data-reduction technique, analyzing the discrete, regular field of a particle-in-cell simulation only
at selected points over time can greatly reduce the need for I/O. Such particles are often arranged at isolated points,
regularly as along lines, in planes or in any other user-defined manner.

Probe particles are usually neutral, non-interacting test particles that are statically placed in the simulation or
co-moving with along pre-defined path. Self-consistently interacting particles are usually called tracer particles.

Workflow

• speciesDefinition.param: create a species specifically for probes and add fieldE and fieldB
  attributes to it for storing interpolated fields

```cpp
using ParticleFlagsProbes = MakeSeq_t<
  particlePusher< particles::pusher::Probe >,
  shape< UsedParticleShape >,
  interpolation< UsedField2Particle >
>;

using Probes = Particles<
  PMACC_CSTRING( "probe" ),
  ParticleFlagsProbes,
  MakeSeq_t<
    position< position_pic >,
    probeB,
    probeE
  >
>;
```

and add it to VectorAllSpecies:

```cpp
using VectorAllSpecies = MakeSeq_t<
  Probes,
  // ...
>;
```

• density.param: select in which cell a probe particle shall be placed, e.g. in each 4th cell per direction:
// put probe particles every 4th cell in X, Y(, Z)

using ProbeEveryFourthCell = EveryNthCellImpl<mCT::UInt32<
  4,
  4,
  4>
>

• particle.param: initialize the individual probe particles in-cell, e.g. always in the left-lower corner and only one per selected cell

const float_X, 3,
InCellOffset,
/* each x, y, z in-cell position component
* in range [0.0, 1.0) */
0.0,
0.0,
0.0
};

struct OnePositionParameter
{
  static constexpr uint32_t numParticlesPerCell = 1u;
  const InCellOffset_t inCellOffset;
};

using OnePosition = OnePositionImpl<OnePositionParameter>;

• speciesInitialization.param: initialize particles for the probe just as with regular particles

using InitPipeline = bmpl::vector<
  // ...
  CreateDensity<
    densityProfiles::ProbeEveryFourthCell,
    startPosition::OnePosition,
    Probes
  >
>;

• fileOutput.param: make sure the tracer particles are part of FileOutputParticles

// either all via VectorAllSpecies or just select
using FileOutputParticles = MakeSeq_t<Probes>;

Known Limitations

Note: currently, only the electric field $\vec{E}$ and the magnetic field $\vec{B}$ can be recorded

Note: we currently do not support time averaging

Warning: If the probe particles are dumped in the file output, the instantaneous fields they recorded will be one time step behind the last field update (since our runOneStep pushed the particles first and then calls the field solver).
2.8.8 Tracer Particles

Section author: Axel Huebl

Tracer particles are like probe particles, but interact self-consistently with the simulation. They are usually used to visualize representative particle trajectories of a larger distribution.

Workflow

- speciesDefinition.param: create a species specifically for tracer particles
  - add the particle attribute particleId to your species’ Particles< ... > class (third argument, T_Attributes)
  - optional: add fieldE and fieldB attributes to the species to store fields as in probes
- create tracer particles by either
  - speciesInitialization.param: initializing a low percentage of your initial density inside this species or
  - speciesInitialization.param: assigning the target (electron) species of an ion’s ionization routine to the tracer species or
  - speciesInitialization.param: moving some particles of an already initialized species to the tracer species (upcoming)
- fileOutput.param: output the tracer particles

Known Limitations

- currently, only the electric field \( \vec{E} \) and the magnetic field \( \vec{B} \) can be recorded
- we currently do not support time averaging

2.8.9 Particle Filters

Section author: Axel Huebl

A common task in both modeling, initializing and in situ processing (output) is the selection of particles of a particle species by attributes. PIConGPU implements such selections as particle filters.

Particle filters are simple mappings assigning each particle of a species either true or false (ignore / filter out). These filters can be defined in particleFilters.param.

Example

Let us select particles with momentum vector within a cone with an opening angle of five degrees (pinhole):

```cpp
namespace picongpu
{
namespace particles
{
namespace filter
{
    struct FunctorParticlesForwardPinhole
    {
        static constexpr char const * name = "forwardPinhole";
        template< typename T_Particle >
        HDINLINE bool operator()( (continues on next page)```
T_Particle const & particle
{
    bool result = false;
    float3_X const mom = particle[ momentum_ ];
    float_X const absMom = math::abs( mom );
    if( absMom > float_X( 0. ) )
    {
        /* place detector in y direction, "infinite distance" to target,
         * and five degree opening angle
         */
        constexpr float_X openingAngle = 5.0 * PI / 180.;
        float_X const dotP = mom.y() / absMom;
        float_X const degForw = math::acos( dotP );
        if( math::abs( degForw ) <= openingAngle * float_X( 0.5 ) )
            result = true;
    }
    return result;
}

using ParticlesForwardPinhole = generic::Free<
    FunctorParticlesForwardPinhole
>;

and add ParticlesForwardPinhole to the AllParticleFilters list:

using AllParticleFilters = MakeSeq_t<
    All,
    ParticlesForwardPinhole
>;

Limiting Filters to Eligible Species

Besides the list of pre-defined filters with parametrization, users can also define generic, “free” implementations as shown above. All filters are added to AllParticleFilters and then combined with all available species from VectorAllSpecies (see speciesDefinition.param).

In the case of user-defined free filters we can now check if each species in VectorAllSpecies fulfills the requirements of the filter. That means: if one accesses specific attributes or flags of a species in a filter, they must exist or will lead to a compile error.

As an example, probe particles usually do not need a momentum attribute which would be used for an energy filter. So they should be ignored from compilation when combining filters with particle species.

In order to exclude all species that have no momentum attribute from the ParticlesForwardPinhole filter, specialize the C++ trait SpeciesEligibleForSolver. This trait is implemented to be checked during compile time when combining filters with species:

// ...

namespace traits
{
}
template<
    typename T_Species
>
struct SpeciesEligibleForSolver<
    T_Species,
    filter::ParticlesForwardPinhole
>
{
    using type = typename pmacc::traits::HasIdentifiers<
        typename T_Species::FrameType,
        MakeSeq_t< momentum >
    >::type;
};
} // namespace traits
} // namespace particles
} // namespace picongpu
3.1 The Particle-in-Cell Algorithm

Section author: Axel Huebl

Please also refer to the textbooks [BirdsallLangdon], [HockneyEastwood], our latest paper on PIConGPU and the works in [Huebl2014] and [Huebl2019].

3.1.1 System of Equations

\[
\begin{align*}
\nabla \cdot \mathbf{E} &= \frac{1}{\varepsilon_0} \sum_s \rho_s \\
\nabla \cdot \mathbf{B} &= 0 \\
\nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\
\nabla \times \mathbf{B} &= \mu_0 \left( \sum_s \mathbf{J}_s + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right)
\end{align*}
\]

for multiple particle species \(s\). \(\mathbf{E}(t)\) represents the electric, \(\mathbf{B}(t)\) the magnetic, \(\rho_s\) the charge density and \(\mathbf{J}_s(t)\) the current density field.

Except for normalization of constants, PIConGPU implements the governing equations in SI units.

3.1.2 Relativistic Plasma Physics

The 3D3V particle-in-cell method is used to describe many-body systems such as plasmas. It approximates the Vlasov–Maxwell–Equation

\[
\frac{\partial}{\partial t} f_s(x, v, t) + v \cdot \nabla_x f_s(x, v, t) + \frac{q_s}{m_s} \left[ \mathbf{E}(x, t) + v \times \mathbf{B}(x, t) \right] \cdot \nabla_v f_s(x, v, t) = 0 \quad (3.1)
\]

with \(f_s\) as the distribution function of a particle species \(s\), \(x, v, t\) as position, velocity and time and \(\frac{q_s}{m_s}\) the charge to mass-ratio of a species. The momentum is related to the velocity by \(p = \gamma m_s v\).

The equations of motion are given by the Lorentz force as

\[
\begin{align*}
\frac{d}{dt} V_s(t) &= \frac{q_s}{m_s} \left[ \mathbf{E}(X_s(t), t) + \mathbf{V}_s(t) \times \mathbf{B}(X_s(t), t) \right] \\
\frac{d}{dt} X_s(t) &= V_s(t)
\end{align*}
\]

Attention: TODO: write proper relativistic form
\( \mathbf{X}_s = (x_1, x_2, \ldots)_s \) and \( \mathbf{V}_s = (v_1, v_2, \ldots)_s \) are vectors of *marker* positions and velocities, respectively, which describe the ensemble of particles belonging to species \( s \).

**Note:** Particles in a particle species can have different charge states in PIConGPU. In the general case, \( \frac{q}{m_s} \) is not required to be constant per particle species.

### 3.1.3 Electro-Magnetic PIC Method

**Fields** such as \( \mathbf{E}(t), \mathbf{B}(t) \) and \( \mathbf{J}(t) \) are discretized on a regular mesh in Eulerian frame of reference (see [EulerLagrangeFrameOfReference]).

The distribution of **Particles** is described by the distribution function \( f_s(\mathbf{x}, \mathbf{v}, t) \). This distribution function is sampled by markers (commonly referred to as *macro-particles*). The temporal evolution of the distribution function is simulated by advancing the markers over time according to the Vlasov–Maxwell–Equation in Lagrangian frame (see eq. (3.1) and [EulerLagrangeFrameOfReference]).

Markers carry a spatial shape of order \( n \) and a delta-distribution in momentum space. In most cases, these shapes are implemented as B-splines and are pre-integrated to *assignment functions* \( S \) of the form:

\[
S^0(x) = \begin{cases} 
1 & \text{if } 0 \leq x_1 \\
0 & \text{else}
\end{cases}
\]

\[
S^n(x) = (S^{n-1} * S^0)(x) = \int_{x-1}^{x} S^{n-1} \xi d\xi
\]

PIConGPU implements these up to order \( n = 4 \). The three dimensional marker shape is a multiplicative union of B-splines \( S^n(x, y, z) = S^n(x)S^n(y)S^n(z) \).

### 3.1.4 References

### 3.2 Landau-Lifschitz Radiation Reaction

**Module author:** Richard Pausch, Marija Vranic

To do

### 3.2.1 References

### 3.3 Field Ionization

**Section author:** Marco Garten

**Module author:** Marco Garten


PIConGPU features an adaptable ionization framework for arbitrary and combinalbe ionization models.

**Note:** Most of the calculations and formulae in this section of the docs are done in the **Atomic Units (AU)** system.

\[
\hbar = e = m_e = 1
\]
### Table 1: Atomic Unit System

<table>
<thead>
<tr>
<th>AU</th>
<th>SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>$5.292 \cdot 10^{-11}$ m</td>
</tr>
<tr>
<td>time</td>
<td>$2.419 \cdot 10^{-17}$ s</td>
</tr>
<tr>
<td>energy</td>
<td>$4.360 \cdot 10^{-18}$ J (= 27.21 eV = 1 Rydberg)</td>
</tr>
<tr>
<td>electrical field</td>
<td>$5.142 \cdot 10^{11}$ V m</td>
</tr>
</tbody>
</table>

#### 3.3.1 Overview: Implemented Models

<table>
<thead>
<tr>
<th>Ionization regime</th>
<th>implemented model</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiphoton</td>
<td>None, yet</td>
<td></td>
</tr>
<tr>
<td>Tunneling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keldysh</td>
<td>[BauerMulser1999]</td>
</tr>
<tr>
<td></td>
<td>ADKLinPol</td>
<td>[DeloneKrainov]</td>
</tr>
<tr>
<td></td>
<td>ADKCircPol</td>
<td>[DeloneKrainov]</td>
</tr>
<tr>
<td>Barrier Suppression</td>
<td>BSI</td>
<td>[MulserBauer2010]</td>
</tr>
<tr>
<td></td>
<td>BSI\text{EffectiveZ} (R&amp;D)</td>
<td>[ClementiRaimondi1963]</td>
</tr>
<tr>
<td></td>
<td>BSI\text{StarkShifted} (R&amp;D)</td>
<td>[ClementiRaimondi1967, BauerMulser1999]</td>
</tr>
</tbody>
</table>

**Attention:** Models marked with “(R&D)” are under research and development and should be used with care.

#### 3.3.2 Usage

Input for ionization models is defined in `speciesDefinition.param`, `ionizer.param` and `ionizationEnergies.param`.

#### 3.3.3 Barrier Suppression Ionization

The so-called barrier-suppression ionization regime is reached for strong fields where the potential barrier binding an electron is completely suppressed.

#### 3.3.4 Tunneling Ionization

Tunneling ionization describes the process during which an initially bound electron quantum-mechanically tunnels through a potential barrier of finite height.

**Keldysh**

$$\Gamma_K = \frac{(6\pi)^{1/2}}{2^{5/4}} E_{ip}^{1/2} \left( \frac{F}{(2E_{ip})^{3/2}} \right)^{1/2} \exp \left( -\frac{2(2E_{ip})^{3/2}}{3F} \right)$$

The Keldysh ionization rate has been implemented according to the equation (9) in [BauerMulser1999]. See also [Keldysh] for the original work.

**Note:** Assumptions:
• low field - perturbation theory
• $\omega_{\text{laser}} \ll E_{\text{ip}}$
• $F \ll F_{\text{BSI}}$
• tunneling is instantaneous

Ammosov-Delone-Krainov (ADK)

$$\Gamma_{\text{ADK}} = \left(\frac{3n^* F}{\pi Z^3} \right)^{\frac{1}{3}} \frac{F D^2}{8 \pi Z} \exp \left( -\frac{2 Z^3}{3 n^* F} \right)$$  \hspace{1cm} (3.2)

$$D \equiv \left(\frac{4 e Z^3}{F n^*} \right)^{\frac{1}{3}} \quad n^* \equiv \frac{Z}{\sqrt{2 E_{\text{ip}}}}$$  \hspace{1cm} (3.3)

We implemented equation (7) from [DeloneKrainov] which is a simplified result assuming s-states (since we have no atomic structure implemented, yet). Leaving out the pre-factor distinguishes ADKCircPol from ADKLinPol. ADKLinPol results from replacing an instantaneous field strength $F$ by $F \cos(\omega t)$ and averaging over one laser period.

**Attention:** Be aware that $Z$ denotes the residual ion charge and not the proton number of the nucleus!

In the following comparison one can see the ADKLinPol ionization rates for the transition from Carbon II to III (meaning 1+ to 2+). For a reference the rates for Hydrogen as well as the barrier suppression field strengths $F_{\text{BSI}}$ have been plotted. They mark the transition from the tunneling to the barrier suppression regime.

![Comparison of ADK ionization rates for Carbon-II and Hydrogen](image-url)

When we account for orbital structure in shielding of the ion charge $Z$ according to [ClementiRaimondi1963] in BSIEffectiveZ the barrier suppression field strengths of Hydrogen and Carbon-II are very close to one
another. One would expect much earlier ionization of Hydrogen due to lower ionization energy. The following image shows how this can be explained by the shape of the ion potential that is assumed in this model.

![Effective atomic potentials of Carbon-II and Hydrogen in homogeneous electric field $F_{BSI}$ (C-II)](image)

$$V_{eff} = -\frac{Z}{|x|} + Fx$$

with $F = F_{BSI}(C-II)$

3.3.5 Predicting Charge State Distributions

Especially for underdense targets, it is possible to already give an estimate for how the laser pulse ionizes a specific target. Starting from an initially unionized state, calculating ionization rates for each charge state for a given electric field via a Markovian approach of transition matrices yields the charge state population for each time.

Here, we show an example of Neon gas ionized by a Gaussian laser pulse with maximum amplitude $a_0 = 10$ and pulse duration (FWHM intensity) of 30 fs. The figure shows the ionization rates and charge state population produced by the ADKLinPol model obtained from the pulse shape in the lower panel, as well as the step-like ionization produced by the BSI model.

You can test the implemented ionization models yourself with the corresponding module shipped in picongpu/lib/python.

```python
import numpy as np
import scipy.constants as sc
from picongpu.utils import FieldIonization

# instantiate class object that contains functions for
# - ionization rates
# - critical field strengths (BSI models)
# - laser intensity conversion
FI = FieldIonization()

# dictionary with atomic units
AU = FI.atomic_unit

# residual charge state AFTER ionization (continues on next page)
```
Note: ADK rates were calculated from the intensity envelope below.

ADK: Ionization Rate $\Gamma$ (AU$^{-1}$)

ADK: Relative Population (%)

BSI: Relative Population (%)

Laser Intensity $I$ (W/cm$^2$)
3.3.6 References

3.4 Collisional Ionization

3.4.1 LTE Models

*Module author: Marco Garten*

Implemented LTE Model: Thomas-Fermi Ionization according to [More1985]


The implementation of the Thomas-Fermi model takes the following input quantities.

- ion proton number $Z$
- ion species mass density $\rho$
- electron “temperature” $T$

Due to the nature of our simulated setups it is also used in non-equilibrium situations. We therefore implemented additional conditions to mediate unphysical behavior but introduce arbitrariness.
Here is an example of hydrogen (in blue) and carbon (in orange) that we would use in a compound plastic target, for instance. The typical plastic density region is marked in green. Two of the artifacts can be seen in this plot:

1. Carbon is predicted to have an initial charge state \( \langle Z \rangle > 0 \) even at \( T = 0 \text{ eV} \).
2. Carbon is predicted to have a charge state of \( \langle Z \rangle \approx 2 \) at solid plastic density and electron temperature of \( T = 10 \text{ eV} \) which increases even as the density decreases. The average electron kinetic energy at such a temperature is 6.67 eV which is less than the 24.4 eV of binding energy for that state. The increase in charge state with decreasing density would lead to very high charge states in the pre-plasmas that we model.

1. Super-thermal electron cutoff

   We calculate the temperature according to \( T_e = \frac{2}{3} \frac{E_{\text{kin},e}}{m_e} \), in units of electron volts. We thereby assume an ideal electron gas. Via the variable \texttt{CUTOFF\_MAX\_ENERGY\_KEV} in \texttt{ionizer.param} the user can exclude electrons with kinetic energy above this value from average energy calculation. That is motivated by a lower interaction cross section of particles with high relative velocities.

2. Lower ion-density cutoff

   The Thomas-Fermi model displays unphysical behaviour for low ion densities in that it predicts an increasing charge state for decreasing ion densities. This occurs already for electron temperatures of 10 eV and the effect increases as the temperature increases. For instance in pre-plasmas of solid density targets the charge state would be overestimated where
   - on average electron energies are not large enough for collisional ionization of a respective charge state
   - ion density is not large enough for potential depression
   - electron-ion interaction cross sections are small due to small ion density

   It is strongly suggested to do approximations for every setup or material first. To that end, a parameter scan with \texttt{[FLYCHK]} can help in choosing a reasonable value.

3. Lower electron-temperature cutoff

   Depending on the material the Thomas-Fermi prediction for the average charge state can be unphysically high. For some materials it predicts non-zero charge states at 0 temperature. That can be a reasonable approximation for metals and their electrons in the conduction band. Yet this cannot be generalized for all materials and therefore a cutoff should be explicitly defined.

   - define via \texttt{CUTOFF\_LOW\_TEMPERATURE\_EV} in \texttt{ionizer.param}

### 3.4.2 NLTE Models

*Module author: Axel Huebl*

in development

### 3.5 Photons

*Section author: Axel Huebl*

*Module author: Heiko Burau*

Radiation reaction and (hard) photons: why and when are they needed. Models we implemented ([Gonoskov] and [Furry]) and verified:

- Landau-Lifschitz Model (semi-classical)
- QED Models (Synchrotron & Bremsstrahlung)
Would be great to add your Diploma Thesis [Burau2016] talk with pictures and comments here. Please add notes and warnings on the models’ assumptions for an easy guiding on their usage :)

**Note:** Assumptions in Furry-picture and Volkov-States: classical em wave part and QED “pertubation”. EM fields on grid (Synchrotron) and density modulations (Bremsstrahlung) need to be locally constant compared to radiated coherence interval (“constant-crossed-field approximation”).

<table>
<thead>
<tr>
<th><strong>Attention:</strong></th>
<th>Bremsstrahlung: The individual electron direction and gamma emission are not correlated. (momentum is microscopically / per e- not conserved, only collectively.)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Attention:</strong></th>
<th>“Soft” photons from low energy electrons will get underestimated in intensity below a threshold of . . . . Their energy is still always conserved until cutoff (defined in . . . ).</th>
</tr>
</thead>
</table>

**Note:** An electron can only emit a photon with identical weighting. Otherwise, the statistical variation of their energy loss would be weighting dependent (note that the average energy loss is unaffected by that).

### 3.5.1 References
CHAPTER
FOUR

POST-PROCESSING

4.1 Python

Section author: Axel Huebl

If you are new to python, get your hands on the tutorials of the following important libraries to get started.

- https://www.python.org/about/gettingstarted/
- https://docs.python.org/3/tutorial/index.html

4.1.1 Numpy

Numpy is the universal swiss army knife for working on ND arrays in python.

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

4.1.2 Matplotlib

One common way to visualize plots:

- http://matplotlib.org/faq/usage_faq.html#usage
- https://gist.github.com/ax3l/fc123cb94f59d440f952

4.1.3 Jupyter

Access, share, modify, run and interact with your python scripts from your browser:

https://jupyter.readthedocs.io

4.1.4 openPMD-viewer

An exploratory framework that visualizes and analyzes data in our HDF5 files thanks to their openPMD markup. Automatically converts units to SI, interprets iteration steps as time series, annotates axes and provides some domain specific analysis, e.g. for LWFA. Also provides an interactive GUI for fast exploration via Jupyter notebooks.

- Project Homepage
- Tutorial
4.1.5 openPMD-api

A data library that reads (and writes) data in our openPMD files (HDF5 and ADIOS) to and from Numpy data structures. Provides an API to correctly convert units to SI, interprets iteration steps correctly, etc.

- Manual
- Examples

4.1.6 yt-project

With yt 3.4 or newer, our HDF5 output, which uses the openPMD markup, can be read, processed and visualized with yt.

- Project Homepage
- Data Loading
- Data Tutorial

4.1.7 pyDive (experimental)

pyDive provides numpy-style array and file processing on distributed memory systems (“numpy on MPI” for data sets that are much larger than your local RAM). pyDive is currently not ready to interpret openPMD directly, but can work on generated raw ADIOS and HDF5 files.

https://github.com/ComputationalRadiationPhysics/pyDive

4.2 openPMD

Section author: Axel Huebl
Module author: Axel Huebl

Our HDF5 and ADIOS use a specific internal markup to structure physical quantities called openPMD. If you hear of it for the first time you can find a quick online tutorial on it here.

As a user of PIConGPU, you will be mainly interested in our python tools and readers, that can read openPMD, e.g. into:

- read & write data: openPMD-api (manual)
- visualization and analysis, including an exploratory Jupyter notebook GUI: openPMD-viewer (tutorial)
- yt-project (tutorial)
- ParaView
- VisIt
- converter tools: openPMD-converter
- full list of projects using openPMD

If you intend to write your own post-processing routines, make sure to check out our example files, the formal, open standard on openPMD and a list of projects that already support openPMD.
4.3 ParaView

Section author: Axel Huebl
Module author: Axel Huebl

Please see https://github.com/ComputationalRadiationPhysics/picongpu/wiki/ParaView for now.
5.1 How to Participate as a Developer

5.1.1 Contents

1. Code - Version Control
   • Install git
   • git
   • git for svn users
2. GitHub Workflow
   • In a Nutshell
   • How to Fork From Us
   • Keep Track of Updates
   • Pull Requests or Being Social
   • Maintainer Notes
3. Commit Rules
4. Test Suite Examples

5.1.2 Code - Version Control

If you are familiar with git, feel free to jump to our github workflow section.

install git

Debian/Ubuntu:
   • sudo apt-get install git
   • make sure git --version is at least at version 1.7.10

Optional one of these. There are nice GUI tools available to get an overview on your repository.
   • gitk git-gui qgit gitg

Mac:
   • see here

Windows:
Configure your global git settings:

- `git config --global user.name NAME`
- `git config --global user.email EMAIL@EXAMPLE.com`
- `git config --global color.ui "auto"` (if you like colors)
- `git config --global pack.threads "0"` (improved performance for multi cores)

You may even improve your level of awesomeness by:

- `git config --global alias.pr "pull --rebase"` (see how to avoid merge commits)
- `git config --global alias.pm "pull --rebase mainline"` (to sync with the mainline by git pm dev)
- `git config --global alias.st "status -sb"` (short status version)
- `git config --global alias.l "log --oneline --graph --decorate --first-parent"` (single branch history)
- `git config --global alias.la "log --oneline --graph --decorate --all"` (full branch history)
- `git config --global rerere.enable 1` (see git rerere)

- More alias tricks:
  - `git config --get-regexp alias` (show all aliases)
  - `git config --global --unset alias.<Name>` (unset alias <Name>)

**git**

Git is a *distributed version control system*. It helps you to keep your software development work organized, because it keeps track of changes in your project. It also helps to come along in teams, crunching on the same project. Examples:

- Arrr, dare you other guys! Why did you change my precious main.cpp, too!?
- Who introduced that awesome block of code? I would like to pay for a beer as a reward.
- Everything is wrong now, why did this happen and when?
- What parts of the code changed since I went on vacation (to a conference, phd seminar, mate fridge, . . . )?

If version control is totally new to you (that’s good, because you are not spoiled) - please refer to a beginners guide first.

- `git - the simple guide`

15 minutes guide at try.github.io

Since git is *distributed*, no one really needs a server or services like github.com to use git. Actually, there are even very good reasons why one should use git even for local data, e.g. a master thesis (or your collection of ascii art dwarf hamster pictures).

Btw, **fun fact warning**: Linus Torvalds, yes the nice guy with the pinguin stuff and all that, developed git to maintain the Linux kernel. So that’s cool, by definition.

A nice overview about the humongous number of tutorials can be found at stackoverflow.com . . . but we may like to start with a git cheat sheet (is there anyone out there who knows more than 1% of all git commands available?)

- git-tower.com (print the 1st page)
• github.com - “cheat git” gem (a cheat sheet for the console)
• kernel.org Everyday GIT with 20 commands or so
• an other interactive, huge cheat sheet (nice overview about stash - workspace - index - local/remote repositories)

Please spend a minute to learn how to write useful git commit messages (caption-style, maximum characters per line, use blank lines, present tense). Read our commit rules and use keywords.

If you like, you can credit someone else for your next commit with:

• git commit --author "John Doe <johns-github-mail@example.com>"

**git for svn users**

If you already used version control systems before, you may enjoy the git for svn users crash course.

Anyway, please keep in mind to use git not like a centralized version control system (e.g. not like svn). Imagine git as your own private svn server waiting for your commits. For example Github.com is only one out of many sources for updates. (But of course, we agree to share our finished, new features there.)

### 5.1.3 GitHub Workflow

Welcome to github! We will try to explain our coordination strategy (I am out of here!) and our development workflow in this section.

**In a Nutshell**

Create a GitHub account and prepare your basic git config.

Prepare your forked copy of our repository:

• fork picongpu on GitHub
• git clone git@github.com:<YourUserName>/picongpu.git (create local copy)
• git remote add mainline git@github.com:ComputationalRadiationPhysics/picongpu.git (add our main repository for updates)
• git checkout dev (switch to our, its now your, dev branch to start from)

Start a topic/feature branch:

• git checkout -b <newFeatureName> (start a new branch from dev and check it out)
• hack hack
• git add <yourChangedFiles> (add changed and new files to index)
• git commit (commit your changes to your local repository)
• git pull --rebase mainline dev (update with our remote dev updates and avoid a merge commit)

Optional, clean up your feature branch. That can be dangerous:

• git pull (if you pushed your branch already to your public repository)
• git pull --rebase mainline dev (apply the mainline updates to your feature branch)
• git log ..mainline/dev, git log --oneline --graph --decorate --all (check for related commits and ugly merge commits)
• git rebase mainline/dev (re-apply your changes after a fresh update to the mainline/dev, see here)
• **git rebase -i mainline/dev** (squash related commits to reduce the complexity of the features history during a pull request)

**Publish** your feature and start a pull request:

• **git push -u origin <newFeatureName>** (push your local branch to your github profile)
• Go to your GitHub page and open a pull request, e.g. by clicking on compare & review
• Select ComputationalRadiationPhysics:dev instead of the default master branch
• Add additional updates (if requested to do so) by push-ing to your branch again. This will update the pull request.

**How to fork from us**

To keep our development fast and conflict free, we recomment you to fork our repository and start your work from our dev (development) branch in your private repository. Simply click the Fork button above to do so.

Afterwards, **git clone your repository** to your local machine. But that is not it! To keep track of the original dev repository, add it as another remote.

• **git remote add mainline https://github.com/ComputationalRadiationPhysics/picongpu.git**
• **git checkout dev** (go to branch dev)

Well done so far! Just start developing. Just like this? No! As always in git, start a new branch with **git checkout -b topic-<yourFeatureName>** and apply your changes there.

**Keep track of updates**

We consider it a **best practice not to modify** neither your master nor your dev branch at all. Instead you can use it to pull --ff-only new updates from the original repository. Take care to **switch to dev** by **git checkout dev** to start new feature branches from dev.

So, if you like to do so, you can even keep track of the original dev branch that way. Just start your new branch with **git branch --track <yourFeatureName> mainline/dev** instead. This allows you to immediatly pull or fetch from our dev and avoids typing (during **git pull --rebase**). Nevertheless, if you like to push to your forked (== origin) repository, you have to say e.g. **git push origin <branchName>** explicitly.

You should **add updates** from the original repository on a regular basis or at least when you finished your feature.

• commit your local changes in your feature branch: **git commit**

Now you could do a normal merge of the latest mainline/dev changes into your feature branch. That is indeed possible, but will create an ugly merge commit. Instead try to first update the point where you branched from and apply your changes again. That is called a **rebase** and is indeed less harmful as reading the sentence before:

• **git checkout <yourFeatureName>**
• **git pull --rebase mainline dev** (in case of an emergency, hit **git rebase --abort**)

Now solve your conflicts, if there are any, and you got it! Well done!

**Pull requests or being social**

How to propose that your awesome feature (we know it will be awesome!) should be included in the mainline PICongGPU version?

Due to the so called pull requests in GitHub, this quite easy (yeah, sure). We start again with a forked repository of our own. You already created a new feature branch starting from our dev branch and commited your changes. Finally, you publish you local branch via a push to your GitHub repository: **git push -u origin <yourLocalBranchName>**
Now let’s start a review. Open the GitHub homepage, go to your repository and switch to your pushed feature branch. Select the green compare & review button. Now compare the changes between your feature branch and our dev.

Everything looks good? Submit it as a pull request (link in the header). Please take the time to write an extensive description.

• What did you implement and why?
• Is there an open issue that you try to address (please link it)?
• Do not be afraid to add images!

The description of the pull request is essential and will be referred to in the change log of the next release.

Please consider to change only one aspect per pull request (do not be afraid of follow-up pull requests!). For example, submit a pull request with a bug fix, another one with new math implementations and the last one with a new awesome implementation that needs both of them. You will see, that speeds up review time a lot!

Speaking of those, a fruitful ( wuhu, we love you - don’t be scared ) discussion about your submitted change set will start at this point. If we find some things you could improve ( That looks awesome, all right! ), simply change your local feature branch and push the changes back to your GitHub repository, to update the pull request. (You can now rebase follow-up branches, too.)

One of our maintainers will pick up the pull request to coordinate the review. Other regular developers that are competent in the topic might assist.

Sharing is caring! Thank you for participating, you are great!

maintainer notes

• do not push to the main repository on a regular basis, use pull request for your features like everyone else
• never do a rebase on the mainline repositories (this causes heavy problems for everyone who pulls them)
• on the other hand try to use pull -rebase to avoid merge commits (in your local/topic branches only)
• do not vote on your own pull requests, wait for the other maintainers
• we try to follow the strategy of a-successful-git-branching-model

Last but not least, help.github.com has a very nice FAQ section.

More best practices.

5.1.4 Commit Rules

See our commit rules page

5.1.5 Test Suite Examples

You know a useful setting to validate our provided methods? Tell us about it or add it to our test sets in the examples/ folder!
5.2 PIConGPU Commit Rulez

We agree on the following simple rules to make our lives easier :)

- Stick to the style below for commit messages
- Commit compiling patches for the main branches (master and dev), you can be less strict for (unshared) topic branches

5.2.1 Commit Messages

Let’s go for an example:

Use the 1st line as a topic, stay <= 50 chars

- the blank line between the “topic” and this “body” is MANDATORY
- use several key points with - or * for additional information
- stay <= 72 characters in this “body” section
- avoid blank lines in the body

5.2.2 Compile Tests

We provide an (interactive/automated) script that compiles all examples within the examples/ directory in your branch.

This helps a lot to maintain various combinations of options in the code (like different solvers, boundary conditions, . . . ).
CompileTest

Assume

- repo=<pathToYourPIConGPUgitDirectory>
- tmpPath=<tmpFolder>

Now run the tests with

- $repo/compile -l $repo/examples/ $tmpPath

Further options are:

- -q : continue on errors
- -j <N> : run <N> tests in parallel (note: do NOT omit the number <N>)

If you ran your test with, let’s say -l -q -j 4, and you got errors like

[compileSuite] [error] In PIC_EXTENSION_PATH:PATH=../params/ThermalTest/cmakePreset_0:
CMAKE_INSTALL_PREFIX:PATH=../params/ThermalTest/cmakePreset_0 (./.build) make install

check the specific test’s output (in this case examples/ThermalTest with CMake preset #0) with:

- less -R $tmpPath/build/build_ThermalTest_cmakePreset_0/compile.log

Compile Tests - Single Example

Compile all CMake presets of a single example with:

- $repo/compile $repo/examples/ $tmpPath
Compile Tests - Cluster Example:

- Request an interactive job (to release some load from the head node) `qsub -I -q laser -lwalltime=03:00:00 -lnodes=1:ppn=64`
- Use a non-home directory, e.g. `tmpPath=/net/cns/projects/HPL/<yourTeam>/` `<yourName>/tmp_tests/`
- Compile like a boss! `<pathToYourPIConGPUgitDirectory>/compile -l -q -j 60 <pathToYourPIConGPUgitDirectory>/examples/ $tmpPath`
- Wait for the thumbs up/down :) 

5.3 Repository Structure

Section author: Axel Huebl

5.3.1 Branches

- `master`: the latest stable release, always tagged with a version
- `dev`: the development branch where all features start from and are merged to
- `release-X.Y.Z`: release candidate for version X.Y.Z with an upcoming release, receives updates for bug fixes and documentation such as change logs but usually no new features

5.3.2 Directory Structure

- `include/`
  - C++ header and source files
  - set `-I` here
  - prefixed with project name
- `lib/`
  - pre-compiled libraries
  - `python/`
    - modules, e.g. for RT interfaces, pre* & post-processing
    - set `PYTHONPATH` here
- `etc/`
  - (runtime) configuration files
  - `picongpu/`
    - `tbg` templates (as long as PIConGPU specific, later on to `share/tbg/`)
    - network configurations (e.g. infiniband)
    - `score-p` and `vampir-trace` filters
- `share/`
  - examples, documentation
  - `picongpu/`
    - `completions/`: bash completions
    - `examples/`: each with same structure as /
5.4 Coding Guide Lines

Section author: Axel Huebl

See also:
Our coding guide lines are documented in this repository.

5.4.1 Source Style

For contributions, **an ideal patch blends in the existing coding style around it** without being noticed as an addition when applied. Nevertheless, please make sure **new files** follow the styles linked above as strict as possible from the beginning.

Unfortunately, we currently do not have tools available to auto-format all aspects of our style guidelines. Since we want to focus on the content of your contribution, we try to cover as much as possible by automated tests which you always have to pass. Nevertheless, we will not enforce the still uncovered, **non-semantic aspects** of style in a **pedantic** way until we find a way to automate it fully.

(That also means that we do not encourage manual style-only changes of our existing code base, since both you and us have better things to do than adding newlines and spaces manually. Doxygen and documentation additions are always welcome!)

5.4.2 License Header

Please **add the according license header** snippet to your **new files**:

- for **PIConGPU (GPLv3+)**: `src/tools/bin/addLicense <FileName>`
- for libraries (**LGPLv3+ & GPLv3+**): `export PROJECT_NAME=PMacc && src/tools/bin/addLicense <FileName>`
- **delete other headers**: `src/tools/bin/deleteHeadComment <FileName>`
- **add license to all .hpp files within a directory (recursive)**: `export PROJECT_NAME=PIConGPU && src/tools/bin/findAndDo <PATH> "*.hpp" src/tools/bin/addLicense`
- **the default project name is PIConGPU (case sensitive!) and add the GPLv3+ only**

Files in the directory **thirdParty/** are only imported from remote repositories. If you want to improve them, submit your pull requests there and open an issue for our **maintainers** to update to a new version of the according software.

5.5 Sphinx

Section author: Axel Huebl, Marco Garten

In the following section we explain how to contribute to this documentation.
5.5.1 Build Locally

This document is build based on free open-source software, namely Sphinx, Doxygen (C++ APIs as XML) and Breathe (to include doxygen XML in Sphinx). A web-version is hosted on ReadTheDocs.

The following requirements need to be installed (once) to build our documentation successfully:

```bash
cd docs/
# doxygen is not shipped via pip, install it externally,
# from the homepage, your package manager, conda, etc.
# example:
sudo apt-get install doxygen
# python tools & style theme
pip install -r requirements.txt # --user
```

In order to not break any of your existing Python configurations, you can also create a new environment that you only use for building the documentation. Since it is possible to install doxygen with conda, the following demonstrates this.

```bash
cd docs/
# create a bare conda environment containing just all the requirements
# for building the picongpu documentation
# note: also installs doxygen inside this environment
conda env create --file picongpu-docs-env.yml
# start up the environment as suggested during its creation e.g.
conda activate picongpu-docs-env
# or
source activate picongpu-docs-env
```

With all documentation-related software successfully installed, just run the following commands to build your docs locally. Please check your documentation build is successful and renders as you expected before opening a pull request!

```bash
# skip this if you are still in docs/
 cd docs/
# parse the C++ API documentation,
# enjoy the doxygen warnings!
doxygen
# render the `.rst` files and replace their macros within
# enjoy the breathe errors on things it does not understand from doxygen :) 
makes html
# open it, e.g. with firefox :)
firefox build/html/index.html
# now again for the pdf :)
makes latexpdf
# open it, e.g. with okular
build/latex/PIConGPU.pdf
```
5.5.2 Useful Links

- A primer on writing restFUL files for sphinx
- Why You Shouldn’t Use “Markdown” for Documentation
- Markdown Limitations in Sphinx

5.6 Doxygen

Section author: Axel Huebl

An online version of our Doxygen build can be found at http://computationalradiationphysics.github.io/picongpu

We regularly update it via

```bash
git checkout gh-pages
# optional argument: branch or tag name
./update.sh
git commit -a
git push
```

This section explains what is done when this script is run to build it manually.

5.6.1 Requirements

First, install Doxygen and its dependencies for graph generation.

```bash
# install requirements (Debian/Ubuntu)
sudo apt-get install doxygen graphviz

# enable HTML output in our Doxyfile
sed -i 's/GENERATE_HTML.*=.*NO/GENERATE_HTML = YES/' docs/Doxyfile
```

5.6.2 Build

Now run the following commands to build the Doxygen HTML documentation locally.

```bash
cd docs/

# build the doxygen HTML documentation
doxygen

# open the generated HTML pages, e.g. with firefox
firefox html/index.html
```

5.7 Clang Tools

Section author: Axel Huebl

We are currently integrating support for Clang Tools [ClangTools] such as clang-tidy and clang-format. Clang Tools are fantastic for static source code analysis, e.g. to find defects, automate style formatting or modernize code.
5.7.1 Install

At least LLVM/Clang 3.9 or newer is required. On Debian/Ubuntu, install them via:

```
sudo apt-get install clang-tidy-3.9
```

5.7.2 Usage

Currently, those tools work only with CPU backends of PIConGPU. For example, enable the OpenMP backend via:

```
# in an example
mkdir .build
cd build
pic-configure -c"-DALPAKA_ACC_CPU_B_OMP2_T_SEQ_ENABLE=ON" ..
```

We try to auto-detect clang-tidy. If that fails, you can set a manual hint to an adequate version via 

```
-DCLANG_TIDY_BIN
```

in CMake:

```
pic-configure -c"-DALPAKA_ACC_CPU_B_OMP2_T_SEQ_ENABLE=ON -DCLANG_TIDY_BIN=\$(which \n-clang-tidy-3.9)" ..
```

If a proper version of clang-tidy is found, we add a new clang-tidy build target:

```
# enable verbose output to see all warnings and errors
make VERBOSE=true clang-tidy
```

5.8 Important PIConGPU Classes

This is very, very small selection of classes of interest to get you started.

5.8.1 MySimulation

```c++
class MySimulation : public pmacc::SimulationHelper<simDim> {
    Global simulation controller class.
    Initialises simulation data and defines the simulation steps for each iteration.

    Template Parameters
    - DIM: the dimension (2-3) for the simulation

    Public Functions

    MySimulation() Constructor.

    virtual void pluginRegisterHelp (po::options_description &desc)
    Register command line parameters for this plugin.
    Parameters are parsed and set prior to plugin load.

    Parameters
    - desc: boost::program_options description
```
std::string pluginGetName () const
    Return the name of this plugin for status messages.

virtual void pluginLoad ()
virtual void pluginUnload ()

void notify (uint32_t currentStep)
    Notification callback.
    For example Plugins can set their requested notification frequency at the PluginConnector

Parameters
    • currentStep: current simulation iteration step

virtual void init ()
    Initialize simulation.
    Does hardware selections/reservations, memory allocations and initializes data structures as empty.

virtual uint32_t fillSimulation ()
    Fills simulation with initial data after init()

    Return returns the first step of the simulation (can be >0 for, e.g., restarts from checkpoints)

virtual void runOneStep (uint32_t currentStep)
    Run one simulation step.

Parameters
    • currentStep: iteration number of the current step

virtual void movingWindowCheck (uint32_t currentStep)
    Check if moving window work must do.
    If no moving window is needed the implementation of this function can be empty

Parameters
    • currentStep: simulation step

virtual void resetAll (uint32_t currentStep)
    Reset the simulation to a state such as it was after init() but for a specific time step.
    Can be used to call fillSimulation() again.

void slide (uint32_t currentStep)

virtual void setInitController (IInitPlugin *initController)

MappingDesc *getMappingDescription ()

5.8.2 FieldE

class FieldE : public picongpu::fields::EMFieldBase
    Representation of the electric field.
    Stores field values on host and device and provides data synchronization between them.
    Implements interfaces defined by SimulationFieldHelper< MappingDesc > and ISimulationData.
5.8.3 FieldB

class FieldB : public picongpu::fields::EMFieldBase
   Representation of the magnetic field.
   Stores field values on host and device and provides data synchronization between them.
   Implements interfaces defined by SimulationFieldHelper< MappingDesc > and ISimulationData.

5.8.4 FieldJ

class FieldJ : public pmacc::SimulationFieldHelper<MappingDesc>, public pmacc::ISimulationData
   Representation of the current density field.
   Stores field values on host and device and provides data synchronization between them.
   Implements interfaces defined by SimulationFieldHelper< MappingDesc > and ISimulationData.

5.8.5 FieldTmp

class FieldTmp : public pmacc::SimulationFieldHelper<MappingDesc>, public pmacc::ISimulationData
   Representation of the temporary scalar field for plugins and temporary particle data mapped to grid (charge density, energy density, etc.)
   Stores field values on host and device and provides data synchronization between them.
   Implements interfaces defined by SimulationFieldHelper< MappingDesc > and ISimulationData.

5.8.6 Particles

template<typename T_Name, typename T_FLAGS, typename T_ATTRIBUTES>

class Particles : public pmacc::ParticlesBase<ParticleDescription<T_Name, SuperCellSize, T_ATTRIBUTES, T_FLAGS>, bmpl::if_>
   particle species

   Template Parameters

   • T_Name: name of the species [type boost::mpl::string]
   • T_Attributes: sequence with attributes [type boost::mpl forward sequence]
   • T_Flags: sequence with flags e.g. solver [type boost::mpl forward sequence]

   Public Types

   template<>
   using SpeciesParticleDescription = pmacc::ParticleDescription<T_Name, SuperCellSize, T_Attributes, T_Flags, typename bmpl::if_>
   template<>
   using ParticlesBaseType = ParticlesBase<SpeciesParticleDescription, picongpu::MappingDesc, DeviceHeap>
   template<>
   using FrameType = typename ParticlesBaseType::FrameType
   template<>
   using FrameTypeBorder = typename ParticlesBaseType::FrameTypeBorder
   template<>
   using ParticlesBoxType = typename ParticlesBaseType::ParticlesBoxType
Public Functions

**Particles**

```cpp
Particles(const std::shared_ptr<DeviceHeap> &heap, picongpu::MappingDesc cellDescription, SimulationDataId datasetID)
```

void `createParticleBuffer()`

void `update(uint32_t const currentStep)`

```cpp
template<typename T_DensityFunctor, typename T_PositionFunctor>
void initDensityProfile(T_DensityFunctor &densityFunctor, T_PositionFunctor &positionFunctor, const uint32_t currentStep)
```

```cpp
template<typename T_SrcName, typename T_SrcAttributes, typename T_SrcFlags, typename T_ManipulateFunctor, typename T_SrcFilterFunctor>
void deviceDeriveFrom(Particles<T_SrcName, T_SrcAttributes, T_SrcFlags> &src, T_ManipulateFunctor &manipulateFunctor, T_SrcFilterFunctor &srcFilterFunctor)
```

SimulationDataId `getUniqueId()`

Return the globally unique identifier for this simulation data.

void `synchronize()`

Synchronizes simulation data, meaning accessing (host side) data will return up-to-date values.

void `syncToDevice()`

Synchronize data from host to device.

Public Static Functions

```cpp
static pmacc::traits::StringProperty getStringProperties()
```

5.8.7 ComputeGridValuePerFrame

template<class T_ParticleShape, class T_DerivedAttribute>

```cpp
class ComputeGridValuePerFrame
```

Public Types

```cpp
template<>
using AssignmentFunction = typename T_ParticleShape::ChargeAssignment
```

```cpp
template<>
using LowerMargin = typename pmacc::math::CT::make_Int<simDim, lowerMargin>::type
```

```cpp
template<>
using UpperMargin = typename pmacc::math::CT::make_Int<simDim, upperMargin>::type
```

Public Functions

```cpp
HDINLINE ComputeGridValuePerFrame()
```

```cpp
HDINLINE float1_64 picongpu::particles::particleToGrid::ComputeGridValuePerFrame::getUnit() const
```

Return solver unit

5.8. Important PIConGPU Classes
HINLINE std::vector< float_64 > picongpu::particles::particleToGrid::ComputeGridValuePerFrame::getUnitDimension() const
return powers of the 7 base measures for this solver
characterizing the unit of the result of the solver in SI (length L, mass M, time T, electric current I,
thermodynamic temperature theta, amount of substance N, luminous intensity J)

template<typename FrameType, typename TVecSuperCell, typename BoxTmp, typename T_Acc>

Public Static Functions

HINLINE std::string picongpu::particles::particleToGrid::ComputeGridValuePerFrame::getName() const
return name of the this solver
Return name of solver

Public Static Attributes

constexpr int supp = AssignmentFunction::support
constexpr int lowerMargin = supp / 2
constexpr int upperMargin = (supp + 1) / 2

5.9 Important pmacc Classes

This is very, very small selection of classes of interest to get you started.

Note: Please help adding more Doxygen doc strings to the classes described below. As an example,
here is a listing of possible extensive docs that new developers find are missing: https://github.com/
ComputationalRadiationPhysics/picongpu/issues/776

5.9.1 Environment

template<uint32_t T_dim>
class Environment : public pmacc::detail::Environment
Global Environment singleton for PMacc.

Public Functions

pmacc::GridController<T_dim> & GridController ()
get the singleton GridController
Return instance of GridController

pmacc::SubGrid<T_dim> & SubGrid ()
get the singleton SubGrid
Return instance of SubGrid

pmacc::FileSystem<T_dim> & FileSystem ()
get the singleton FileSystem
Return instance of FileSystem
void `initDevices` (DataSpace<T_dim> `devices`, DataSpace<T_dim> `periodic`) create and initialize the environment of PMacc

Usage of MPI or device(accelerator) function calls before this method are not allowed.

**Parameters**

- `devices`: number of devices per simulation dimension
- `periodic`: periodicity each simulation dimension (0 == not periodic, 1 == periodic)

void `initGrids` (DataSpace<T_dim> `globalDomainSize`, DataSpace<T_dim> `localDomainSize`, DataSpace<T_dim> `localDomainOffset`) initialize the computing domain information of PMacc

**Parameters**

- `globalDomainSize`: size of the global simulation domain [cells]
- `localDomainSize`: size of the local simulation domain [cells]
- `localDomainOffset`: local domain offset [cells]

`Environment (const Environment&)` Environment &`operator=` (const Environment&)

**Public Static Functions**

```cpp
static Environment<T_dim> & get ()
get the singleton Environment< DIM >

Return instance of Environment<DIM>
```

### 5.9.2 DataConnector

**class DataConnector**

Singleton class which collects and shares simulation data.

All members are kept as shared pointers, which allows their factories to be destroyed after sharing ownership with our `DataConnector`.

**Public Functions**

```cpp
bool hasId (SimulationDataId `id`)
Returns if data with identifier id is shared.

Return if dataset with id is registered

Parameters

- `id`: id of the Dataset to query

void initialise (AbstractInitialiser &`initialiser`, uint32_t `currentStep`)
Initialises all Datasets using initialiser.

After initialising, the Datasets will be invalid.

Parameters

- `initialiser`: class used for initialising Datasets
```
void \texttt{share} (const std::shared_ptr<ISimulationData> &\texttt{data})

Register a new Dataset and share its ownership.

If a Dataset with the same id already exists, a runtime\_error is thrown. (Check with \texttt{DataConnector::hasId} when necessary.)

Parameters

\begin{itemize}
  \item \texttt{data}: simulation data to share ownership
\end{itemize}

void \texttt{consume} (std::unique_ptr<ISimulationData> \texttt{data})

Register a new Dataset and transfer its ownership.

If a Dataset with the same id already exists, a runtime\_error is thrown. (Check with \texttt{DataConnector::hasId} when necessary.) The only difference from \texttt{share()} is transfer of ownership.

Parameters

\begin{itemize}
  \item \texttt{data}: simulation data to transfer ownership
\end{itemize}

void \texttt{deregister} (SimulationDataId \texttt{id})

End sharing a dataset with identifier \texttt{id}.

Parameters

\begin{itemize}
  \item \texttt{id}: id of the dataset to remove
\end{itemize}

void \texttt{clean}()

Unshare all associated datasets.

template<class \texttt{TYPE}>

std::shared_ptr<\texttt{TYPE}> \texttt{get} (SimulationDataId \texttt{id}, bool \texttt{noSync} = false)

Returns shared pointer to managed data.

Reference to data in Dataset with identifier \texttt{id} and type \texttt{TYPE} is returned. If the Dataset status in invalid, it is automatically synchronized. Increments the reference counter to the dataset specified by \texttt{id}. This reference has to be released after all read/write operations before the next synchronize()/\texttt{getData()} on this data are done using \texttt{releaseData()}.

Return returns a reference to the data of type \texttt{TYPE}

Template Parameters

\begin{itemize}
  \item \texttt{TYPE}: if of the data to load
\end{itemize}

Parameters

\begin{itemize}
  \item \texttt{id}: id of the Dataset to load from
  \item \texttt{noSync}: indicates that no synchronization should be performed, regardless of dataset status
\end{itemize}

void \texttt{releaseData} (SimulationDataId)

Indicate a data set gotten temporarily via.

See \texttt{getData} is not used anymore

Parameters

\begin{itemize}
  \item \texttt{id}: id for the dataset previously acquired using \texttt{getData()}
\end{itemize}
Friends

friend pmacc::DataConnector::detail::Environment

5.9.3 DataSpace

template<unsigned T_Dim>
class DataSpace : public pmacc::math::Vector<int, T_Dim>
A T_Dim-dimensional data space.

DataSpace describes a T_Dim-dimensional data space with a specific size for each dimension. It only
describes the space and does not hold any actual data.

Template Parameters

• T_Dim: dimension (1-3) of the dataspace

Public Types

template<>
using BaseType = math::Vector<int, T_Dim>

Public Functions

HDINLINE DataSpace ()
default constructor.
Sets size of all dimensions to 0.

HDINLINE DataSpace (dim3 value)
constructor.
Sets size of all dimensions from cuda dim3.

HDINLINE DataSpace (uint3 value)
constructor.
Sets size of all dimensions from cuda uint3 (e.g. threadIdx/blockIdx)

HDINLINE DataSpace (const DataSpace<Dim>& value)

HDINLINE DataSpace (int x)
Constructor for DIM1-dimensional DataSpace.
Parameters

• x: size of first dimension

HDINLINE DataSpace (int x, int y)
Constructor for DIM2-dimensional DataSpace.
Parameters

• x: size of first dimension
  • y: size of second dimension

HDINLINE DataSpace (int x, int y, int z)
Constructor for DIM3-dimensional DataSpace.
Parameters

- \( x \): size of first dimension
- \( y \): size of second dimension
- \( z \): size of third dimension

HDINLINE `DataSpace(const BaseType & vec)`

HDINLINE `DataSpace(const math::Size_t<T_Dim> & vec)`

HDINLINE `int pmacc::DataSpace::getDim() const`

Returns number of dimensions (T_Dim) of this `DataSpace`.

Return number of dimensions

HDINLINE `bool pmacc::DataSpace::isOneDimensionGreaterThan(const DataSpace < T_Dim > & other) const`

Evaluates if one dimension is greater than the respective dimension of other.

Return true if one dimension is greater, false otherwise

Parameters

- `other`: `DataSpace` to compare with

HDINLINE `operator math::Size_t<T_Dim> () const`

HDINLINE `operator dim3 () const`

Public Static Functions

static HDINLINE `DataSpace<T_Dim> pmacc::DataSpace::create(int value = 1)`

Give `DataSpace` where all dimensions set to init value.

Return the new `DataSpace`

Parameters

- `value`: value which is set for all dimensions

Public Static Attributes

`constexpr int Dim = T_Dim`

5.9.4 Vector

Warning: doxygenclass: Cannot find class “pmacc::Vector” in doxygen xml output for project “PIConGPU” from directory: ./xml

5.9.5 SuperCell

template<class T_FrameType>

class SuperCell
Public Functions

HDINLINE SuperCell()

HDINLINE T_FrameType* pmacc::SuperCell::FirstFramePtr()
HDINLINE T_FrameType* pmacc::SuperCell::LastFramePtr()
HDINLINE T_FrameType const* pmacc::SuperCell::FirstFramePtr() const
HDINLINE T_FrameType const* pmacc::SuperCell::LastFramePtr() const
HDINLINE bool pmacc::SuperCell::mustShift() const
HDINLINE void pmacc::SuperCell::setMustShift(bool const value)
HDINLINE uint32_t pmacc::SuperCell::getSizeLastFrame() const
HDINLINE uint32_t pmacc::SuperCell::getNumParticles() const
HDINLINE void pmacc::SuperCell::setNumParticles(uint32_t const size)

PMACC_ALIGN (firstFramePtr, T_FrameType *)
PMACC_ALIGN (lastFramePtr, T_FrameType *)

5.9.6 GridBuffer

template<class TYPE, unsigned DIM, class BORDERTYPE = TYPE>
class GridBuffer : public pmacc::HostDeviceBuffer<TYPE, DIM>

GridBuffer represents a DIM-dimensional buffer which exists on the host as well as on the device.

GridBuffer combines a HostBuffer and a DeviceBuffer with equal sizes. Additionally, it allows sending data from and receiving data to these buffers. Buffers consist of core data which may be surrounded by border data.

Template Parameters
- TYPE: datatype for internal Host- and DeviceBuffer
- DIM: dimension of the buffers
- BORDERTYPE: optional type for border data in the buffers. TYPE is used by default.

Public Types

typedef Parent::DataBoxType DataBoxType

Public Functions

GridBuffer(const GridLayout<DIM> &gridLayout, bool sizeOnDevice = false)

Constructor.

Parameters
- gridLayout: layout of the buffers, including border-cells
- sizeOnDevice: if true, size information exists on device, too.

GridBuffer(const DataSpace<DIM> &dataSpace, bool sizeOnDevice = false)

Constructor.

Parameters
• **dataSpace**: `DataSpace` representing buffer size without border-cells

• **sizeOnDevice**: if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

```cpp
GridBuffer<DeviceBuffer<TYPE, DIM> &otherDeviceBuffer, const GridLayout<DIM> &gridLayout, bool sizeOnDevice = false)
```

**Constructor.**

**Parameters**

- **otherDeviceBuffer**: DeviceBuffer which should be used instead of creating own DeviceBuffer
- **gridLayout**: layout of the buffers, including border-cells
- **sizeOnDevice**: if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

```cpp
GridBuffer<HostBuffer<TYPE, DIM> &otherHostBuffer, const DataSpace<DIM> &offsetHost, DeviceBuffer<TYPE, DIM> &otherDeviceBuffer, const DataSpace<DIM> &offsetDevice, const GridLayout<DIM> &gridLayout, bool sizeOnDevice = false)
```

**virtual ~GridBuffer()**

**Destructor.**

```cpp
void addExchange(uint32_t dataPlace, const Mask &receive, DataSpace<DIM> guardingCells, uint32_t communicationTag, bool sizeOnDeviceSend, bool sizeOnDeviceReceive)
```

Add Exchange in `GridBuffer` memory space.

An Exchange is added to this `GridBuffer`. The exchange buffers use the same memory as this `GridBuffer`.

**Parameters**

- **dataPlace**: place where received data is stored [GUARD | BORDER] if dataPlace=GUARD than copy other BORDER to my GUARD if dataPlace=BORDER than copy other GUARD to my BORDER
- **receive**: a Mask which describes the directions for the exchange
- **guardingCells**: number of guarding cells in each dimension
- **communicationTag**: unique tag/id for communication
- **sizeOnDeviceSend**: if true, internal send buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)
- **sizeOnDeviceReceive**: if true, internal receive buffers must store their size additionally on the device

```cpp
void addExchange(uint32_t dataPlace, const Mask &receive, DataSpace<DIM> guardingCells, uint32_t communicationTag, bool sizeOnDevice = false)
```

Add Exchange in `GridBuffer` memory space.

An Exchange is added to this `GridBuffer`. The exchange buffers use the same memory as this `GridBuffer`.

**Parameters**
• dataPlace: place where received data is stored [GUARD | BORDER] if dataPlace=GUARD than copy other BORDER to my GUARD if dataPlace=BORDER than copy other GUARD to my BORDER
• receive: a Mask which describes the directions for the exchange
• guardingCells: number of guarding cells in each dimension
• communicationTag: unique tag/id for communication
• sizeOnDevice: if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

```cpp
void addExchangeBuffer(const Mask &receive, const DataSpace<DIM> &dataSpace, uint32_t communicationTag, bool sizeOnDeviceSend, bool sizeOnDeviceReceive)
```

Add Exchange in dedicated memory space.

An Exchange is added to this GridBuffer. The exchange buffers use their own memory instead of using the GridBuffer's memory space.

Parameters

• receive: a Mask which describes the directions for the exchange
• dataSpace: size of the newly created exchange buffer in each dimension
• communicationTag: unique tag/id for communication
• sizeOnDeviceSend: if true, internal send buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)
• sizeOnDeviceReceive: if true, internal receive buffers must store their size additionally on the device

```cpp
bool hasSendExchange(uint32_t ex) const
```

Returns whether this GridBuffer has an Exchange for sending in ex direction.

Return true if send exchanges with ex direction exist, otherwise false

Parameters

• ex: exchange direction to query
bool hasReceiveExchange(uint32_t ex) const
    Returns whether this GridBuffer has an Exchange for receiving from ex direction.

Return true if receive exchanges with ex direction exist, otherwise false

Parameters
• ex: exchange direction to query

Exchange<BORDERTYPE, DIM> & getSendExchange(uint32_t ex) const
    Returns the Exchange for sending data in ex direction.

Returns an Exchange which for sending data from this GridBuffer in the direction described by ex.

Return the Exchange for sending data

Parameters
• ex: the direction to query

Exchange<BORDERTYPE, DIM> & getReceiveExchange(uint32_t ex) const
    Returns the Exchange for receiving data from ex direction.

Returns an Exchange which for receiving data to this GridBuffer from the direction described by ex.

Return the Exchange for receiving data

Parameters
• ex: the direction to query

Mask getSendMask() const
    Returns the Mask describing send exchanges.

Return Mask for send exchanges

Mask getReceiveMask() const
    Returns the Mask describing receive exchanges.

Return Mask for receive exchanges

EventTask communication()
    Starts sync data from own device buffer to neighbor device buffer.

Asynchronously starts synchronization data from internal DeviceBuffer using added Exchange buffers. This operation runs sequential to other code but intern asynchronous

EventTask asyncCommunication(EventTask serialEvent)
    Starts sync data from own device buffer to neighbor device buffer.

Asynchronously starts synchronization data from internal DeviceBuffer using added Exchange buffers.

EventTask asyncSend(EventTask serialEvent, uint32_t sendEx)

EventTask asyncReceive(EventTask serialEvent, uint32_t recvEx)

GridLayout<DIM> getGridLayout()
    Returns the GridLayout describing this GridBuffer.

Return the layout of this buffer
## Protected Attributes

- bool `hasOneExchange`
- uint32_t `lastUsedCommunicationTag`
- GridLayout<DIM> `gridLayout`
- Mask `sendMask`
- Mask `receiveMask`

```cpp
template<> ExchangeIntern<BORDERTYPE, DIM> *sendExchanges[27]
```

```cpp
template<> ExchangeIntern<BORDERTYPE, DIM> *receiveExchanges[27]
```

```cpp
template<> EventTask receiveEvents[27]
```

```cpp
template<> EventTask sendEvents[27]
```

- uint32_t `maxExchange`

## 5.9.7 SimulationFieldHelper

template<class CellDescription>

class SimulationFieldHelper

### Public Types

```cpp
typedef CellDescription MappingDesc
```

### Public Functions

```cpp
SimulationFieldHelper (CellDescription description)
```

```cpp
virtual ~SimulationFieldHelper ()
```

```cpp
virtual void reset (uint32_t currentStep) = 0
```

Reset is as well used for init.

```cpp
virtual void syncToDevice () = 0
```

Synchronize data from host to device.

```cpp
CellDescription getCellDescription () const
```

### Protected Attributes

```cpp
CellDescription cellDescription
```

## 5.9.8 ParticlesBase

template<typename T_ParticleDescription, class T_MappingDesc, typename T_DeviceHeap>

class ParticlesBase : public pmacc::SimulationFieldHelper<T_MappingDesc>

## 5.9. Important pmacc Classes
Public Types

```cpp
enum [anonymous]

Values:

Dim = MappingDesc::Dim
Exchanges = traits::NumberOfExchanges<Dim>::value
TileSize = math::CT::volume<typename MappingDesc::SuperCellSize>::type::value
```

typedef ParticlesBuffer<ParticleDescription, typename MappingDesc::SuperCellSize, T_DeviceHeap, MappingDesc::Dim> BufferType

typedef BufferType::FrameType FrameType

typedef BufferType::FrameTypeBorder FrameTypeBorder

typedef ParticleDescription::ParticlesBoxType ParticlesBoxType

typedef ParticleDescription::HandleGuardRegion HandleGuardRegion

typedef ParticleTag SimulationDataTag
```

Public Functions

```cpp
void fillAllGaps()

void fillBorderGaps()

void deleteGuardParticles(uint32_t exchangeType)

template<uint32_t T_area>
void deleteParticlesInArea()

void copyGuardToExchange(uint32_t exchangeType)

copy guard particles to intermediate exchange buffer

Warning This method resets the number of particles in the processed supercells even if there are
particles left in the supercell and does not guarantee that the last frame is contiguous filled.
Call fillAllGaps afterwards if you need a valid number of particles and a contiguously filled last
frame.

void insertParticles(uint32_t exchangeType)

ParticlesBoxType getDeviceParticlesBox()

ParticlesBoxType getHostParticlesBox(const int64_t memoryOffset)

BufferType &getParticlesBuffer()

void reset(uint32_t currentStep)

Reset is as well used for init.
```

Protected Functions

```cpp
ParticlesBase(const std::shared_ptr<T_DeviceHeap> &deviceHeap, MappingDesc description)

virtual ~ParticlesBase()

template<uint32_t AREA>
void shiftParticles()
```
template<uint32_t/Area> void fillGaps();

**Protected Attributes**

BufferType *particlesBuffer

### 5.9.9 ParticleDescription

**Warning:** doxygenclass: Cannot find class “pmacc::ParticleDescription” in doxygen xml output for project “PIConGPU” from directory: ../xml

### 5.9.10 ParticleBox

**Warning:** doxygenclass: Cannot find class “pmacc::ParticleBox” in doxygen xml output for project “PIConGPU” from directory: ../xml

### 5.9.11 Frame

**Warning:** doxygenclass: Cannot find class “pmacc::Frame” in doxygen xml output for project “PIConGPU” from directory: ../xml

### 5.9.12 IPlugin

```
class IPlugin : public pmacc::INotify
    Subclassed by picongpu::ISimulationPlugin, picongpu::ISimulationStarter, pmacc::SimulationHelper<
    DIM>, pmacc::SimulationHelper< simDim >
```

**Public Functions**

IPlugin();

virtual ~IPlugin();

virtual void load();

virtual void unload();

bool isLoaded();

virtual void checkpoint (uint32_t currentStep, const std::string checkpointDirectory) = 0

Notifies plugins that a (restartable) checkpoint should be created for this timestep.

**Parameters**

- currentStep: current simulation iteration step
- checkpointDirectory: common directory for checkpoints

virtual void restart (uint32_t restartStep, const std::string restartDirectory) = 0

Restart notification callback.
Parameters

• **restartStep**: simulation iteration step to restart from
• **restartDirectory**: common restart directory (contains checkpoints)

```cpp
virtual void pluginRegisterHelp (po::options_description &desc) = 0
    Register command line parameters for this plugin.
    Parameters are parsed and set prior to plugin load.

Parameters

• **desc**: boost::program_options description

virtual std::string pluginGetName () const = 0
    Return the name of this plugin for status messages.

Return plugin name

virtual void onParticleLeave (const std::string&, const int32_t)
    Called each timestep if particles are leaving the global simulation volume.
    This method is only called for species which are marked with the GuardHandlerCallPlugins policy in their description.
    The order in which the plugins are called is undefined, so this means read-only access to the particles.

Parameters

• **speciesName**: name of the particle species
• **direction**: the direction the particles are leaving the simulation

uint32_t getLastCheckpoint () const
    When was the plugin checkpointed last?

Return last checkpoint’s time step

void setLastCheckpoint (uint32_t currentStep)
    Remember last checkpoint call.

Parameters

• **currentStep**: current simulation iteration step

Protected Functions

virtual void pluginLoad()

virtual void pluginUnload()

Protected Attributes

bool loaded
uint32_t lastCheckpoint
### 5.9.13 PluginConnector

**class PluginConnector**

Plugin registration and management class.

**Public Functions**

```cpp
void registerPlugin (IPlugin *plugin)
```

Register a plugin for loading/unloading and notifications.

Plugins are loaded in the order they are registered and unloaded in reverse order. To trigger plugin notifications, call

See setNotificationPeriod after registration.

**Parameters**

- `plugin`: plugin to register

```cpp
void loadPlugins ()
```

Calls load on all registered, not loaded plugins.

```cpp
void unloadPlugins ()
```

Unloads all registered, loaded plugins.

```cpp
std::list<po::options_description> registerHelp ()
```

Publishes command line parameters for registered plugins.

**Return** list of boost program_options command line parameters

```cpp
void setNotificationPeriod (INotify *notifiedObj, std::string const &period)
```

Set the notification period.

**Parameters**

- `notifiedObj`: the object to notify, e.g. an `IPlugin` instance
- `period`: notification period

```cpp
void notifyPlugins (uint32_t currentStep)
```

Notifies plugins that data should be dumped.

**Parameters**

- `currentStep`: current simulation iteration step

```cpp
void checkpointPlugins (uint32_t currentStep, const std::string checkpointDirectory)
```

Notifies plugins that a restartable checkpoint should be dumped.

**Parameters**

- `currentStep`: current simulation iteration step
- `checkpointDirectory`: common directory for checkpoints

```cpp
void restartPlugins (uint32_t restartStep, const std::string restartDirectory)
```

Notifies plugins that a restart is required.

**Parameters**

- `restartStep`: simulation iteration to restart from
• restartDirectory: common restart directory (contains checkpoints)

```cpp
template<typename Plugin>
std::vector<Plugin *> getPluginsFromType()
    Get a vector of pointers of all registered plugin instances of a given type.

    Return  vector of plugin pointers

    Template Parameters
    • Plugin: type of plugin

std::list<IPlugin *> getAllPlugins() const
    Return a copied list of pointers to all registered plugins.
```

**Friends**

```cpp
friend pmacc::PluginConnector::detail::Environment
```

5.9.14 SimulationHelper

```cpp
template<unsigned DIM>
class SimulationHelper : public pmacc::IPlugin
    Abstract base class for simulations.
    Use this helper class to write your own concrete simulations by binding pure virtual methods.

    Template Parameters
    • DIM: base dimension for the simulation (2-3)

Public Types

```cpp
template<>
using SeqOfTimeSlices = std::vector<pluginSystem::TimeSlice>
```

Public Functions

```cpp
SimulationHelper()  Constructor.

virtual ~SimulationHelper()  

virtual void runOneStep (uint32_t currentStep) = 0
    Must describe one iteration (step).
    This function is called automatically.

virtual void init() = 0
    Initialize simulation.
    Does hardware selections/reservations, memory allocations and initializes data structures as empty.

virtual uint32_t fillSimulation() = 0
    Fills simulation with initial data after init()

    Return  returns the first step of the simulation (can be >0 for, e.g., restarts from checkpoints)
```
virtual void resetAll (uint32_t currentStep) = 0
Reset the simulation to a state such as it was after init() but for a specific time step.
Can be used to call fillSimulation() again.

virtual void movingWindowCheck (uint32_t currentStep) = 0
Check if moving window work must do.
If no moving window is needed the implementation of this function can be empty

Parameters
• currentStep: simulation step

virtual void dumpOneStep (uint32_t currentStep)
Notifies registered output classes.
This function is called automatically.

Parameters
• currentStep: simulation step

GridController<DIM> &getGridController ()

void dumpTimes (TimeIntervall &tSimCalculation, TimeIntervall&, double &roundAvg, uint32_t currentStep)

void startSimulation ()
Begin the simulation.

virtual void pluginRegisterHelp (po::options_description &desc)
Register command line parameters for this plugin.
Parameters are parsed and set prior to plugin load.

Parameters
• desc: boost::program_options description

std::string pluginGetName () const
Return the name of this plugin for status messages.

Return plugin name

void pluginLoad ()

void pluginUnload ()

void restart (uint32_t restartStep, const std::string restartDirectory)
Restart notification callback.

Parameters
• restartStep: simulation iteration step to restart from
• restartDirectory: common restart directory (contains checkpoints)

void checkpoint (uint32_t currentStep, const std::string checkpointDirectory)
Notifies plugins that a (restartable) checkpoint should be created for this timestep.

Parameters
• currentStep: cuurent simulation iteration step
• checkpointDirectory: common directory for checkpoints

Protected Functions

std::vector<uint32_t> readCheckpointMasterFile ()
     Reads the checkpoint master file if any and returns all found checkpoint steps.

     Return vector of found checkpoints steps in order they appear in the file

Protected Attributes

uint32_t runSteps
uint32_t softRestarts
     Presentations: loop the whole simulation softRestarts times from initial step to runSteps.
std::string checkpointPeriod
SeqOfTimeSlices seqCheckpointPeriod
std::string checkpointDirectory
uint32_t numCheckpoints
int32_t restartStep
std::string restartDirectory
bool restartRequested

const std::string CHECKPOINT_MASTER_FILE
std::string author

5.9.15 ForEach

Warning: doxygenstruct: Cannot find class “meta::ForEach” in doxygen xml output for project “PIConGPU” from directory: ../xml

5.9.16 Kernel Start

template<typename T_KernelFunctor>
struct Kernel
     wrapper for the user kernel functor
     contains debug information like filename and line of the kernel call

Public Types

template<>
using KernelType = T_KernelFunctor
Public Functions

HINLINE Kernel (T_KernelFunctor const &kernelFunctor, std::string const &file = std::string(), size_t const line = 0)

Return

Parameters

- gridExtent: grid extent configuration for the kernel
- blockExtent: block extent configuration for the kernel
- sharedMemByte: dynamic shared memory used by the kernel (in byte)

Template Parameters

- T_VectorGrid: type which defines the grid extents (type must be castable to CUDA dim3)
- T_VectorBlock: type which defines the block extents (type must be castable to CUDA dim3)

Parameters

- gridExtent: grid extent configuration for the kernel
- blockExtent: block extent configuration for the kernel
- sharedMemByte: dynamic shared memory used by the kernel (in byte)

Public Members

T_KernelFunctor const m_kernelFunctor
functor

std::string const m_file
file name from where the kernel is called

size_t const m_line
line number in the file

PMACC_KERNEL(...)
create a kernel object out of a functor instance
this macro add the current filename and line number to the kernel object

Parameters

- ...: instance of kernel functor

5.9.17 Struct Factory

Syntax to generate structs with all members inline. Allows to conveniently switch between variable and constant defined members without the need to declare or initialize them externally. See for example PIConGPU’s density.param for usage.

PMACC_STRUCT(name, ...)
generate a struct with static and dynamic members
PMACC_STRUCT(StructAlice,
   // constant member variable
   (PMACC_C_VALUE(float, varFoo, -1.0))
   // lvalue member variable
   (PMACC_VALUE(float, varFoo, -1.0))
   // constant vector member variable
   (PMACC_C_VECTOR_DIM(double, 3, vectorBarC, 1.134e-5, 1.134e-5, 1.134e-5))
   // lvalue vector member variable
   (PMACC_VECTOR_DIM(double, 3, vectorBarC, 1.134e-5, 1.134e-5, 1.134e-5))
   // constant string member variable
   (PMACC_C_STRING(someString, "anythingYouWant: even spaces!"))
   // plain C++ member
   PMACC_EXTENT( 
      using float_64 = double;
      static constexpr int varBar = 42;
   );
);

Note do not forget the surrounding parenthesize for each element of a sequence

Parameters
- name: name of the struct
- . . . : preprocessor sequence with TypeMemberPair's e.g. (PMACC_C_VALUE(int,a,2))

PMACC_C_VECTOR_DIM (type, dim, name, ...)
create static const member vector that needs no memory inside of the struct

PMACC_C_VECTOR_DIM(float_64, simDim, center_SI, 1.134e-5, 1.134e-5, 1.134e-5);
// is syntactically equivalent to
static const Vector<float_64,simDim> center_SI = Vector<float_64,simDim>(1.134e-5, 1.134e-5, 1.134e-5);

Parameters
- type: type of an element
- dim: number of vector components
- name: member variable name
- . . . : enumeration of init values (number of components must be greater or equal than dim)

PMACC_C_VALUE (type, name, value)
create static constexpr member

PMACC_C_VALUE(float_64, power_SI, 2.0);
// is syntactically equivalent to
static constexpr float_64 power_SI = float_64(2.0);

Parameters
- type: type of the member
- name: member variable name
- value: init value

PMACC_VALUE (type, name, initValue)
create changeable member
PMACC_VALUE(float_64, power_SI, 2.0);
// is the equivalent of
float_64 power_SI(2.0);

Parameters
- `type`: type of the member
- `name`: member variable name
- `value`: init value

PMACC_VECTOR(type, name, ...)
create changeable member vector

PMACC_VECTOR(float2_64, center_SI, 1.134e-5, 1.134e-5);
// is the equivalent of
float2_64 center_SI(1.134e-5, 1.134e-5);

Parameters
- `type`: type of an element
- `name`: member variable name
- `...`: enumeration of init values

PMACC_VECTOR_DIM(type, dim, name, ...)
create changeable member vector

PMACC_VECTOR_DIM(float_64, simDim, center_SI, 1.134e-5, 1.134e-5);
// is the equivalent of
Vector<float_64,3> center_SI(1.134e-5, 1.134e-5, 1.134e-5);

Parameters
- `type`: type of an element
- `dim`: number of vector components
- `name`: member variable name
- `...`: enumeration of init values (number of components must be equal to dim)

PMACC_C_STRING(name, initValue)
create static const character string

PMACC_C_STRING(filename, "fooFile.txt");
// is syntactically equivalent to
static const char* filename = (char*)"fooFile.txt";

Parameters
- `name`: member variable name
- `char_string`: character string

PMACC_EXTENT(...)
create any code extension

PMACC_EXTENT(typedef float FooFloat;)
// is the equivalent of
typedef float FooFloat;
Parameters

• ...: any code

5.9.18 Identifier

Construct unique types, e.g. to name, access and assign default values to particle species’ attributes. See for example PIConGPU’s speciesAttributes.param for usage.

value_identifier (in_type, name, in_default)

define a unique identifier with name, type and a default value

The created identifier has the following options: getValue() - return the user defined value getName() - return the name of the identifier ::type - get type of the value

Parameters

• in_type: type of the value
• name: name of identifier
• in_value: user defined value of in_type (can be a constructor of a class)

e.g. value_identifier(float,length,0.0f) typedef length::type value_type; // is float value_type x = length::getValue(); //set x to 0.f printf(“Identifier name: %s,length::getName()); //print Identifier name: length

to create a instance of this value_identifier you can use: length() or length_

alias (name)

create an alias

an alias is a unspecialized type of an identifier or a value_identifier

example: alias(aliasName); //create type varname

Parameters

• name: name of alias

to specialize an alias do: aliasName<valueIdentifierName> to create an instance of this alias you can use: aliasName(); or aliasName_

get type which is represented by the alias typedef typename traits::Resolve<name>::type resolved_type;

5.10 Python Postprocessing Tool Structure

Each plugin should implement at least the following Python classes.

1. A data reader class responsible for loading the data from the simulation directory
2. A visualizer class that outputs a matplotlib plot
3. A jupyter-widget class that exposes the parameters of the matplotlib visualizer to the user via other widgets.

The repository directory for PIConGPU Python modules for plugins is lib/python/picongpu/plugins/.

5.10.1 Data Reader

The data readers should reside in the lib/python/picongpu/plugins/data directory. There is a base class in base_reader.py defining the interface of a reader. Each reader class should derive from this class and implement the interface functions not implemented in this class.

class picongpu.plugins.data.base_reader.DataReader (run_directory) 

Base class that all data readers should inherit from.
__init__(run_directory)

Parameters run_directory(string) – path to the run directory of PIConGPU (the path before simOutput/)

_get_for_iteration(iteration, **kwargs)

Get the data for a given iteration.

Returns

• The data for the requested parameters in a plugin
• dependent format and type.

get (**kwargs)

Parameters

• 'iteration' or 'time' should be present in the kwargs. (Either) –
  • both are given, the 'time' argument is converted to (If) –
    • iteration and data for the iteration matching the time (an) –
      • returned. (is) –
    • time (float or np.array of float or None.) – If None, data for all available times is returned.
    • iteration (int or np.array of int or None.) – If None, data for all available iterations is returned.

Returns

• The data for the requested parameters in a plugin
• dependent format and type.

get_data_path (**kwargs)

Returns

Return type A string with the path to the underlying data file.

get_dt ()

Return the timestep for the chosen simulation.

get_iterations (**kwargs)

Returns

• An array with unsigned integers of iterations for which data is available.

get_times (**kwargs)

Returns

• An array of floats of simulation time steps for which data is available

To shorten the import statements for the readers, please also add an entry in the __init__.py file of the data directory.
5.10.2 Matplotlib visualizer

The visualizers should reside in the lib/python/picongpu/plugins/plot_mpl/ directory. The module names should end on _visualizer.py and the class name should only be Visualizer.

To shorten the import statements for the visualizers, please also add an entry in the __init__.py file of the plot_mpl directory with an alias that ends on “MPL”.

There is a base class for visualization found in base_visualizer.py which already handles the plotting logic. It uses (possibly multiple) instances of the data reader classes for accessing the data. Visualizing data simultaneously for more than one scan is supported by creating as many readers and plot objects as there are simulations for visualization. After getting the data, it ensures that (for performance reasons) a matplotlib artist is created only for the first plot and later only gets updated with fresh data.

```python
class picongpu.plugins.plot_mpl.base_visualizer.Visualizer(reader_cls, run_directories=None, ax=None):
    Abstract base class for matplotlib visualizers that implements the visualization logic. Classes that derive from this class need to write their own implementations for the following functions in order to work:

    _create_data_reader(self, run_directory), _create_plt_obj(self, ax) _update_plt_obj(self)

    Note: When using classes derived from this within jupyter notebooks, use %matplotlib notebook mode.

    __init__(reader_cls, run_directories=None, ax=None)
    Initialize the reader and data as member parameters.

    _check_and_fix_run_dirs(run_directories)
    Check variable type for the run_directories and change to list of tuples if necessary. This can be overridden in derived classes to e.g. restrict to single simulation visualization.

    _create_plt_obj(idx)
    Sets `self.plt_obj` to an instance of a matplotlib.artist.Artist object (or derived classes) created by using `self.ax` which can later be updated by feeding new data into it. Only called on the first call for visualization.

    _remove_plt_obj(idx)
    Removes the plt_obj at position idx from the current plot and sets it to None so that in a subsequent visualization call the plt_obj is created again.

    _update_plt_obj(idx)
    Take the `self.data` member, interpret it and feed it into the `self.plt_obj`.
```

Parameters

- **run_directories** *(list of tuples of length 2)* — or single tuple of length 2 or list of strings or string. If tuples are specified, they have to be of the following form (sim_label, sim_path) and consist of strings with ‘sim_label’ being a short string used e.g. in plot legends and ‘sim_path’ leading to the run directory of PICongGPU (the path before simOutput/). If only a string or list of strings is given, labels are generated by enumeration. If None, the user is responsible for providing run_directories later on via set_run_directories() before calling visualize().

- **reader_cls** *(class)* — handle of a PICongGPU Data reader class (not string!) which inherited from BaseReader in plugins.data.base_reader.

- **ax** *(matplotlib.axes)* —

Returns

- **a list of tuples, each of the form**

  - **(simulation_label, path_to_simulation).**

```
**adjust_plot**( **kwargs)

Executed after the plotting is done for adjusting legends etc...

**clear_cbar**( )

Clear colorbars if present. Should be implemented in derived classes that use colorbars.

**set_run_directories**( run_directories)

Prepare everything for a fresh start with new run_directories

**visualize**( **kwargs)

1. gathers the data for the selected kwargs
2. removes plot elements for sources which have no data
3. plot the data 3.a Creates the 'plt_obj' if it does not exist 3.b Updates the 'plt_obj' with the new data. 4. adjusts the plot

All new plugins should derive from this class.

When implementing a new visualizer you have to perform the following steps:

1. Let your visualizer class inherit from the **Visualizer** class in base **visualizer.py** and call the base class constructor with the correct data reader class.

2. Implement the _create_plt_obj(self, idx) function. This function needs to access the plotting data from the self.data member (this is the data structure as returned by the data readers .get(...) function, create some kind of matplotlib artist by storing it in the self.plt_obj member variable at the correct index specified by the idx variable (which corresponds to the data of the simulation at position idx that is passed in construction).

3. Implement the _update_plt_obj(self, idx) function. This is called only after a valid self.plt_obj was created. It updates the matplotlib artist with new data. Therefore it again needs to access the plotting data from the self.data member and call the data update API for the matplotlib artist (normally via .set_data(...).

5.10.3 Jupyter Widget

The widget is essentially only a wrapper around the matplotlib visualizer that allows dynamical adjustment of the parameters the visualizer accepts for plotting. This allows to adjust e.g. species, filter and other plugin-dependent options without having to write new lines of Python code.

The widgets should reside in the **lib/python/picongpu/plugins/jupyter_widgets/ directory. The module names should end on _widget.py**.

To shorten the import statements for the widgets, please also add an entry in the __init__.py file of the jupyter_widget directory.

There is a base class for visualization found in base **widget.py** which already handles most of the widget logic.

```python
class picongpu.plugins.jupyter_widgets.base_widget.BaseWidget ( plot_mpl_cls, run_dir_options=None, fig=None, output_widget=None, **kwargs)

Basic widget class that wraps a corresponding plot_mpl visualizer. It handles selection of scans, simulations and iterations. It also allows to expose the parameters of the corresponding plot_mpl visualizer via jupyter widgets to the user. Only classes derived from this base class should be used!

Note: In order to work, those objects should be used in %matplotlib widget mode and interactive plotting should be switched off (by using plt.ioff()).

__init__ (plot_mpl_cls, run_dir_options=None, fig=None, output_widget=None, **kwargs)```

5.10. Python Postprocessing Tool Structure
Parameters

- **run_dir_options** *(list)* – list of tuples with label and filepath
- **plot_mpl_cls** *(a valid plot_mpl class handle (not string!))* – Specifies the underlying plot_mpl visualizer that will be used.
- **fig** *(a matplotlib figure instance. (optional))* – If no figure is provided, the widget will create its own figure. Otherwise the widget will draw into the provided figure and will not create its own.
- **output_widget** *(None or instance of ipywidgets.Output)* – used for capturing messages from the visualizers. If None, a new output widget is created and displayed as a child. If not None, it is not displayed but only used for capturing output and the owner of the output is responsible for displaying it.
- **kwargs** *(options for plotting, passed on to matplotlib commands.)* –

```python
_create_sim_dropdown(options)
```
Provide the widget for selection of simulations. Can be overridden in derived classes if some of those widgets are not necessary. Note: Make sure that no value of the widget is selected initially since otherwise initial plotting after creation of the widget might not work (since the constructor sets the value to the first available which leads to the visualization callback being triggered.)

Returns

**Return type** a jupyter widget that allows selection of value(s)

```python
_create_widgets_for_vis_args()
```
Provide additional UI widget elements that expose the parameters of the underlying plot_mpl visualizer instance that the user should be able to modify.

Returns

- **a dict mapping keyword argument names of the PIC visualizer**
- **to the widgets that are used for adjusting those values.**
- **Those widgets should be created in this function as well.**
- **Note** *(no callback for plotting needs to be registered, this is done)*
- **automatically during construction.**

```python
_get_widget_args()
```

Returns

- **a dict mapping keyword argument names of the PIC visualizer**
- **to the values of the corresponding widget elements.**

```python
_handle_run_dir_selection_callback(change)
```
Callback function when user selects a subset of the available simulations.

```python
_init_fig_and_ax(fig, **kwargs)
```
Creates the figure and the ax as members.

```python
_make_drop_val_compatible(val)
```
Depending on the type of self.sim_drop we have to assign a single value or a tuple to the self.sim_drop.value. This function converts the ‘val’ in a way to be compatible with the expected type.

```python
_show_run_dir_options_in_dropdown()
```
Make the labels of the run_dir_options lookup table available for selection as options for the drop-down.
_update_available_sim_times()
Computes the intersection of simulation times that are present in all simulations currently selected. It automatically plots the iteration step that best matches the specified simulation time.

__update_plot_mpl_run_dir__(selected_sims)
Passes the selected simulations to the visualizer instance.

**Parameters**

- **selected_sims** *(list)* – list of simulation labels which will be translated to their path.

__visualize_callback__(change)
Callback that is executed when one of the extra ui elements changes or the iteration changes.

capture_output()
Used as decorator for capturing output of member functions.

It allows to switch between visualizations for different simulation times (iterations) and different simulations.

When implementing a new widget you have to perform the following steps:

1. Let the widget class inherit from the `BaseWidget` class in `base_widget.py` and call the base class constructor with the correct matplotlib visualizer class.

   ```python
   from .base_widget import BaseWidget
   class NewPluginWidget(BaseWidget):
   ```

2. In the constructor, call the base class constructor with the matplotlib visualizer class as `plot_mpl_cls` keyword.

   The base class will then create an instance of the visualizer class and delegate the plotting job to it.

   ```python
   # taken from lib/python/picongpu/plugins/jupyter_widgets/energy_histogram_widget.py
   from .base_widget import BaseWidget
   from picongpu.plugins.plot_mpl import EnergyHistogramMPL
   class EnergyHistogramWidget(BaseWidget):
       def __init__(self, run_dir_options, fig=None, **kwargs):
           BaseWidget.__init__(self,
                               EnergyHistogramMPL,
                               run_dir_options,
                               fig,
                               **kwargs)
   ```

3. Implement the `_create_widgets_for_vis_args(self)` function.

   This function has to define jupyter widgets as member variables of the class to allow interactive manipulation of parameters the underlying matplotlib visualizer is capable of handling. It needs to return a dictionary using the parameter names the matplotlib visualizer accepts as keys and the widget members that correspond to these parameters as values.

   ```python
   # taken from lib/python/picongpu/plugins/jupyter_widgets/energy_histogram_widget.py
   def _create_widgets_for_vis_args(self):
       # widgets for the input parameters
       self.species = widgets.Dropdown(description="Species",
                                        options=['e'],
                                        value='e')
       self.species_filter = widgets.Dropdown(description="Species_filter",
                                              options=['all'],
                                              value="all")

       return {'species': self.species,
               'species_filter': self.species_filter}
   ```
5.11 Index of Doxygen Documentation

This command is currently taking up to 2 GB of RAM, so we can’t run it on read-the-docs:

```bash
doxgenindex::
   project  PIConGPU
   path     '../xml'
   outline
   no-link
```
CHAPTER
SIX

PROGRAMMING PATTERNS

See also:
In order to follow this section, you need to understand the CUDA programming model.

6.1 Lockstep Programming Model

Section author: René Widera, Axel Huebl

The lockstep programming model structures code that is evaluated collectively and independently by workers (physical threads). Actual processing is described by one-dimensional index domains of virtual workers which can even be changed within a kernel. Mathematically, index domains are none-injective, total functions on physical workers.

An index domain is independent from data but can be mapped to a data domain, e.g. one to one or with more complex mappings.

Code which is implemented by the lockstep programming model is free of any dependencies between the number of worker and processed data elements. To simplify the implementation, each index within a domain can be seen as a virtual worker which is processing one data element (like the common workflow to programming CUDA). Each worker $i$ can be executed as $N_i$ virtual workers ($1 : N_i$).

6.1.1 pmacc helpers

```cpp
template<uint32_t T_domainSize, uint32_t T_workerSize, uint32_t T_simdSize = 1u>
struct IdxConfig
    describe a constant index domain
describe the size of the index domain and the number of workers to operate on the domain

Template Parameters
    • T_domainSize: number of indices in the domain
    • T_workerSize: number of worker working on T_domainSize
    • T_simdSize: SIMD width
```

```cpp
template<typename T_Type, typename T_IdxConfig>
struct CtxArray : public pmacc::memory::Array<T_Type, T_IdxConfig::numCollIter * T_IdxConfig::simdSize>, public T_IdxConfig
    Static sized array for a local variable.
```

The array is designed to hold context variables in lock step programming. A context variable is just a local variable of a virtual worker. Allocating and using a context array allows to propagate virtual worker states over subsequent lock steps. A context array for a set of virtual workers is owned by their (physical) worker.

The number of elements depends on the index domain size and the number of workers to process the indices.

```cpp
template<typename T_IdxConfig>
```
struct ForEachIdx : public T_IdxConfig
    
execute a functor for each index

Distribute the indices even over all worker and execute a user defined functor. There is no guarantee in which order the indices will be processed.

Template Parameters

- T_IdxConfig: index domain description

### 6.1.2 Common Patterns

#### Collective Loop

- each worker needs to pass a loop N times
- in this example, there are more dates than workers that process them

```cpp
// `frame` is a list which must be traversed collectively
while( frame.isValid() )
{
    uint32_t const workerIdx = threadIdx.x;
    using ParticleDomCfg = IdxConfig<
        frameSize,
        numWorker
    >;
    ForEachIdx< ParticleDomCfg > forEachParticle( workerIdx );
    forEachParticle(
        [&]( uint32_t const linearIdx, uint32_t const idx )
        {
            // independent work
        });
}
```

#### Non-Collective Loop

- each virtual worker increments a private variable

```cpp
uint32_t const workerIdx = threadIdx.x;
using ParticleDomCfg = IdxConfig<
    frameSize,
    numWorkers
>
;
ForEachIdx< ParticleDomCfg > forEachParticle( workerIdx );
memory::CtxArray< int, ParticleDomCfg > vWorkerIdx( 0 );
forEachParticle(
    [&]( uint32_t const linearIdx, uint32_t const idx )
    {
        vWorkerIdx[ idx ] = linearIdx;
        for( int i = 0; i < 100; i++ )
            vWorkerIdx[ idx ]++;
    });
```

#### Create a Context Variable

- ... and initialize with the index of the virtual worker
```cpp
uint32_t const workerIdx = threadIdx.x;
using ParticleDomCfg = IdxConfig<
  frameSize,
  numWorkers
>;
memory::CtxArray< int, ParticleDomCfg > vIdx{
  workerIdx,
  [&]{
    uint32_t const linearIdx, uint32_t const
    { return linearIdx;
    }
  };
  // is equal to
memory::CtxArray< int, ParticleDomCfg > vIdx;
ForEachIdx< ParticleDomCfg > forEachParticle{ workerIdx }{
  [&]{
    uint32_t const linearIdx, uint32_t const idx
    { vIdx[ idx ] = linearIdx;
    }
  };
}
```

**Using a Master Worker**

- only one virtual worker (called master) of all available numWorkers manipulates a shared data structure for all others

```cpp
// example: allocate shared memory (uninitialized)
PMACC_SMEM(
  finished,
  bool
);
uint32_t const workerIdx = threadIdx.x;
ForEachIdx<
  IdxConfig<
    1,
    numWorkers
  >
> onlyMaster{ workerIdx };
// manipulate shared memory
onlyMaster(
  [&]{
    uint32_t const, uint32_t const
    { finished = true;
    }
  };
  // important: synchronize now, in case upcoming operations (with
  // other workers) access that manipulated shared memory section
  __syncthreads();
```

**6.1. Lockstep Programming Model**


get_dt () (picongpu.plugins.data.base_reader.DataReader method), 315
get_iterations () (picongpu.plugins.data.base_reader.DataReader method), 315
get_times () (picongpu.plugins.data.base_reader.DataReader method), 315

M
mem_req_by_calorimeter () (picongpu.utils.memory_calculator.MemoryCalculator method), 249
mem_req_by_fields () (picongpu.utils.memory_calculator.MemoryCalculator method), 249
mem_req_by_particles () (picongpu.utils.memory_calculator.MemoryCalculator method), 250
mem_req_by_rng () (picongpu.utils.memory_calculator.MemoryCalculator method), 250
MemoryCalculator (class in picongpu.utils.memory_calculator), 249

P
picongpu::FieldB (C++ class), 292
picongpu::FieldE (C++ class), 291
picongpu::FieldJ (C++ class), 292
picongpu::fields::laserProfiles::ExpRampWithPrepulse (C++ class), 136
picongpu::fields::laserProfiles::GaussianBeam (C++ class), 131
picongpu::fields::laserProfiles::None (C++ class), 142
picongpu::fields::laserProfiles::PlaneWave (C++ class), 140
picongpu::fields::laserProfiles::Polynorn (C++ class), 139
picongpu::fields::laserProfiles::PulseFrontTilt (C++ class), 133
picongpu::fields::laserProfiles::Wavepacket (C++ class), 135
picongpu::FieldTmp (C++ class), 292
picongpu::MySimulation (C++ class), 290
picongpu::MySimulation::fillSimulation (C++ function), 291
picongpu::MySimulation::getMappingDescription (C++ function), 291
picongpu::MySimulation::init (C++ function), 291
picongpu::MySimulation::movingWindowCheck (C++ function), 291
picongpu::MySimulation::MySimulation (C++ function), 290
picongpu::MySimulation::notify (C++ function), 291
picongpu::MySimulation::pluginGetName (C++ function), 290
picongpu::MySimulation::pluginLoad (C++ function), 291
picongpu::MySimulation::pluginRegisterHelp (C++ function), 290
picongpu::MySimulation::pluginUnload (C++ function), 291
picongpu::MySimulation::resetAll (C++ function), 291
picongpu::MySimulation::runOneStep (C++ function), 291
picongpu::MySimulation::setInitController (C++ function), 291
picongpu::Particles (C++ class), 292
picongpu::particles::CreateDensity (C++ class), 155
picongpu::particles::Derive (C++ class), 156
picongpu::Particles::deviceDeriveFrom (C++ function), 293
picongpu::particles::FillAllGaps (C++ class), 157
picongpu::particles::filter::All (C++ class), 161
picongpu::particles::filter::filter::generic::Free (C++ class), 161
picongpu::particles::filter::filter::generic::FreeRng (C++ class), 161
picongpu::particles::filter::filter::generic::FreeTotalCe (C++ class), 162
picongpu::Particles::getGlobalDomain (C++ function), 293
picongpu::Particles::getStringProperties (C++ function), 293
picongpu::Particles::initDensityProfile (C++ function), 293
picongpu::particles::Manipulate (C++ class), 156
picongpu::particles::ManipulateDerive (C++ class), 156
picongpu::particles::manipulators::binary::Assign (C++ type), 160
picongpu::particles::manipulators::binary::DensityWeighting (C++ type), 160
picongpu::particles::manipulators::binary::ProtonTimesWeighting (C++ type), 160
picongpu::particles::manipulators::generic::Free (C++ class), 157
picongpu::particles::manipulators::generic::FreeRng (C++ class), 158
picongpu::particles::manipulators::unary::CopyAtt
pmacc::SuperCell::SuperCell  \textit{(C++ function)}, 299
PMACC_C_STRING \textit{(C macro)}, 313
PMACC_C_VALUE \textit{(C macro)}, 312
PMACC_C_VECTOR_DIM \textit{(C macro)}, 312
PMACC_EXTENT \textit{(C macro)}, 313
PMACC_KERNEL \textit{(C macro)}, 311
PMACC_STRUCT \textit{(C macro)}, 311
PMACC_VALUE \textit{(C macro)}, 312
PMACC_VECTOR \textit{(C macro)}, 313
PMACC_VECTOR_DIM \textit{(C macro)}, 313

\textbf{S}

set_run_directories() \textit{(picongpu.plugins.plot_mpl.base_visualizer.Visualizer method)}, 317

\textbf{V}

value_identifier \textit{(C macro)}, 314
visualize() \textit{(picongpu.plugins.plot_mpl.base_visualizer.Visualizer method)}, 317
Visualizer \textit{(class in picongpu.plugins.plot_mpl.base_visualizer)}, 316