PIConGPU Documentation

Release 0.3.0

The PIConGPU Community

Jun 16, 2017

INSTALLATION

A particle-in-cell code for GPGPUs

PIConGPU is a fully relativistic, many GPGPU, 3D3V particle-in-cell (PIC) code. The Particle-in-Cell 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 Maxwell's equations.

Generally, you want to follow those pages in-order to get started. Individual chapters are based on the information of 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.

Attention: This documentation is just getting started. Learn more about how to improve it *here* and please contribute via pull requests! :-)

Note: We also have a wiki and a general official homepage

CHAPTER 1

Installation

Installation

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 padavan, and follow the corresponding sections of the next chapters.

Ways to Install

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 can be near-ideal if architecture is choosen correctly (and/or if build directly on your hardware). You then set environment variables to find those installs.

Spack

[Spack] is a flexible package manager for HPC systems that can organize versions and dependencies for you. 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.

Conda

We currently do not have an official conda install (yet). Due to pre-build binaries, performance will be sub-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 but we already provide a dockerfile to get started. Performance might be sub-ideal if the image is not build for the specific local hardware again. Useful for small desktop or single-node runs.

Compute Environments

HPC Cluster

SysAdmin

- use *[Spack]* and auto-build modules, ideally via *[Lmod]*
- or build from source, manage binary and version incompatibilities and provide modules

User

As a user, you ideally start with a configured compiler and MPI version for your HPC system (at least). Those and further dependencies can be set up by:

- loading modules (e.g. via *[Lmod]* or *[modules]*)

or self-adding them:

- build from source
- or use *[Spack]*

Desktop

Root/Admin

Use your package manager to install drivers and core dependencies, e.g. via *apt-get install* as far as possible. Build further dependencies from source.

Alternately, use *[Spack]* for all dependencies.

User

If drivers are already installed:

- use *[Spack]*
- or use *[nvidia-docker]* (dockerfile)
- or build from source

Cloud

For single nodes, essentially the same as working via SSH on any other machine. We did not investigate deeper into multi-node cloud setups yet.

AWS

- use *[Spack]*
- or use *[nvidia-docker]* (dockerfile)
- or build from source

Google Cloud

- use *[Spack]*
- or use *[nvidia-docker]* (dockerfile)
- or build from source

References

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.

Compiling from Source

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

Compiling a project from source essentially requires three steps:

1. configure the project and find its dependencies
2. build 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.

Step-by-Step

Compling 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

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

# configurate, 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 `-DOPTION=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 still build *in-source* and use a build system called *autotools*. The syntax is still very similar:

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

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

That's all! Continue with the following chapter to build our dependencies.

References

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 systems.

Dependencies

Overview

Requirements

Mandatory

gcc

- 4.9 to 5.X (depends on your current CUDA version)
- *note:* be sure to build all libraries/dependencies with the *same* gcc version

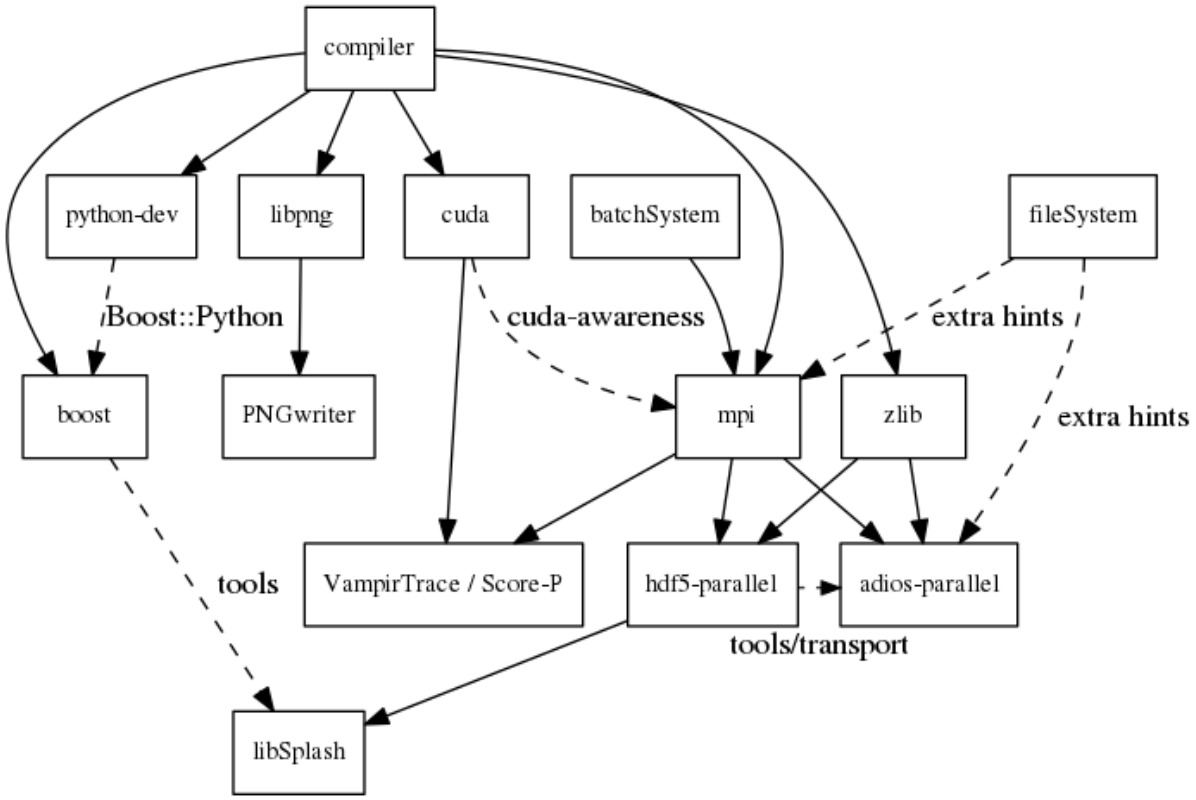

Fig. 1.1: Overview of inter-library dependencies for parallel execution of PIConGPU on a typical HPC system. Due to common binary incompatibilities between compilers, MPI and boost versions, we recommend to organize software with a version-aware package manager such as spack and to deploy a hierarchical module system such as lmod. A Lmod example setup can be found here.

- *Debian/Ubuntu:*
  - `sudo apt-get install gcc-4.9 g++-4.9 build-essential`
  - `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.9`
- *Arch Linux:*
  - `sudo pacman --sync base-devel`
  - if the installed version of **gcc** is too new, compile an older gcc
- *Spack:*
  - `spack install gcc@4.9.4`
  - make it the default in your packages.yaml or *suffix* all following `spack install` commands with a *space* and `%gcc@4.9.4`

CUDA

- 7.5+
- *Debian/Ubuntu:* `sudo apt-get install nvidia-cuda-toolkit`
- *Arch Linux:* `sudo pacman --sync cuda`
- *Spack:*
  - `curl -o cuda_7.5.18_linux.run http://developer.download.nvidia.com/compute/cuda/7.5/Prod/local_installers/cuda_7.5.18_linux.run`
  - `spack install cuda@7.5.18`
- at least one **CUDA** capable **GPU**
- *Compute capability* **sm_20** 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>`

CMake

- 3.3.0 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** 1.5.1+ / **MVAPICH2** 1.8+ or similar (GPU aware install recommended)
- *Debian/Ubuntu:* `sudo apt-get install libopenmpi-dev`
- *Arch Linux:* `sudo pacman --sync openmpi`
- *Spack:*
  - `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`

boost

- 1.57.0-1.64.0 (`program options`, `regex` , `filesystem`, `system`, `thread`, `math`, `serialization` and nearly all header-only libs)
- download from http://www.boost.org
- *Debian/Ubuntu:* `sudo apt-get install libboost-program-options-dev libboost-regex-dev libboost-filesystem-dev libboost-system-dev libboost-thread-dev libboost-math-dev libboost-serialization-dev`
- *Arch Linux:* `sudo pacman --sync boost`
- *Spack:* `spack install boost`
- *from source:*
  - `./bootstrap.sh --with-libraries=filesystem,program_options,regex,system,thread,math,serialization --prefix=$HOME/lib/boost`
  - `./b2 -j4 && ./b2 install`
- *environment:* (assumes install from source in `$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`

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=$PICHOME/src/picongpu`
  - `export PATH=$PICSRC:$PATH`
  - `export PATH=$PICSRC/src/tools/bin:$PATH`

Optional Libraries

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

pngwriter

- 0.5.6+
- *Spack:* `spack install pngwriter`
- *from source:*
  - download our modified version from github.com/pngwriter/pngwriter
  - Requires [libpng](http://www.libpng.org/)
    - *Debian/Ubuntu:* `sudo apt-get install libpng-dev`
    - *Arch Linux:* `sudo pacman --sync libpng`
  - example:
    - `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 PNGWRITER_ROOT=$HOME/lib/pngwriter`
    - `export LD_LIBRARY_PATH=$PNGWRITER_ROOT/lib:$LD_LIBRARY_PATH`

libSplash

- 1.6.0+ (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`

  - `cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/splash ~/src/splash`
  - `make install`
- *environment:* (assumes install from source in `$HOME/lib/splash`)
  - `export SPLASH_ROOT=$HOME/lib/splash`
  - `export LD_LIBRARY_PATH=$SPLASH_ROOT/lib:$LD_LIBRARY_PATH`

HDF5

- 1.8.6+
- standard shared version (no c++, enable parallel), e.g. `hdf5/1.8.5-threadsafe`
- *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 ~/build ~/lib`
  - `cd ~/src`
  - download hdf5 source code from release list of the HDF5 group, for example:
  - `wget https://www.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.14/src/hdf5-1.8.14.tar.gz`
  - `tar -xvzf hdf5-1.8.14.tar.gz`
  - `cd hdf5-1.8.14`
  - `./configure --enable-parallel --enable-shared --prefix $HOME/lib/hdf5/`
  - `make`
  - *optional:* `make test`
  - `make install`
- *environment:* (assumes install from source in `$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`, `regex`
- 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 a species initial density profiles
- compile and install exactly as *splash2txt* above

ADIOS

- 1.10.0+ (requires *MPI*, *zlib* and mxml)
- *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 ~/build ~/lib`
  - `cd ~/src`
  - `wget http://users.nccs.gov/~pnorbert/adios-1.10.0.tar.gz`
  - `tar -xvzf adios-1.10.0.tar.gz`
  - `cd adios-1.10.0`
  - `CFLAGS="-fPIC" ./configure --enable-static --enable-shared --prefix=$HOME/lib/adios --with-mpi=$MPI_ROOT --with-zlib=/usr`
  - `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

- 1.3.0+
- 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`
  - `wget -O VampirTrace-5.14.4.tar.gz "http://wwwpub.zih.tu-dresden.de/~mlieber/dcount/dcount.php?package=vampirtrace&get=VampirTrace-5.14.4.tar.gz"`
  - `tar -xvzf 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.

picongpu.profile

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:

```
. $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.

Hypnos (HZDR)

```
# Modules ######################################################################
#

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

```
        # Core Dependencies
        module load gcc/4.9.2
        module load cmake/3.7.2
        module load boost/1.62.0
        module load cuda/8.0
        module load openmpi/1.8.6.kepler.cuda80

        # Plugins (optional)
        module load pngwriter/0.5.6
        module load hdf5-parallel/1.8.15 libsplash/1.6.0

        # either use libSplash or ADIOS for file I/O
        #module load adios/1.10.0

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

#       unset MODULES_NO_OUTPUT
fi

# Environment #################################################################
#
alias getk20='qsub -I -q k20 -lwalltime=00:30:00 -lnodes=1:ppn=8'
alias getlaser='qsub -I -q laser -lwalltime=00:30:00 -lnodes=1:ppn=16'

export PICSRC=/home/`whoami`/src/picongpu
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# send me mails on 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>"

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

# Development #################################################################
#
#function make
#{
#  real_make=`which make`
#  $real_make $* 2>&1 | $HOME/grcat/usr/bin/grcat conf.gcc
#}

# "tbg" default options #######################################################
#   - PBS/Torque (qsub)
#   - "k20" queue
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="submit/hypnos-hzdr/k20_profile.tpl"
```

Titan (ORNL)

```
export proj=<yourProject>

# send me mails on 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>"
```

```
# basic environment #################################################
source /opt/modules/3.2.6.7/init/bash
module load craype-accel-nvidia35
module swap PrgEnv-pgi PrgEnv-gnu
# module swap gcc gcc/4.8.2 # default

# Compile for CLE nodes
#   (CMake likes to unwrap the Cray wrappers)
export CC=`which cc`
export CXX=`which CC`
export FC=`which ftn`
#export LD="/sw/xk6/altd/bin/ld"

# symbol bug work around (should not be required)
#MY_CRAY_LIBS=/opt/gcc/4.8.2/snos/lib64
#export LD_PRELOAD=$MY_CRAY_LIBS/libstdc++.so.6:$LD_PRELOAD
#export LD_PRELOAD=$MY_CRAY_LIBS/libgomp.so.1:$LD_PRELOAD
#export LD_PRELOAD=$MY_CRAY_LIBS/libgfortran.so.3:$LD_PRELOAD

# required tools and libs
module load git
module load cmake/3.5.2
module load cudatoolkit
module load boost/1.57.0
export BOOST_ROOT=$BOOST_DIR
export MPI_ROOT=$MPICH_DIR

# vampirtrace (optional) ############################################
#   pic-configure with -c "-DVAMPIR_ENABLE=ON"
#   e.g.:
#     pic-configure -c "-DVAMPIR_ENABLE=ON" ~/paramSets/case001
#module load vampirtrace/5.14.4
#export VT_ROOT=$VAMPIRTRACE_DIR

# scorep (optional) #################################################
#   pic-configure with -c "-DCMAKE_CXX_COMPILER=`which scorep-CC` \
#                          -DCUDA_NVCC_EXECUTABLE=`which scorep-nvcc`"
#   e.g.:
#     SCOREP_WRAPPER=OFF pic-configure -a 35 \
#         -c "-DCMAKE_CXX_COMPILER=`which scorep-CC` \
#         -DCUDA_NVCC_EXECUTABLE=`which scorep-nvcc`" \
#         ~/paramSets/case001
#     export SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--cuda --mpp=mpi"
#     make -j
#     make install
module load scorep/2.0

# plugins (optional) ################################################
module load cray-hdf5-parallel/1.8.14
#module load adios/1.10.0 dataspaces/1.4.0
export HDF5_ROOT=$HDF5_DIR
#export ADIOS_ROOT=$ADIOS_DIR
#export DATASPACES_ROOT=$DATASPACES_DIR

# download libSplash and compile it yourself from
#   https://github.com/ComputationalRadiationPhysics/libSplash/
export SPLASH_ROOT=$PROJWORK/$proj/lib/splash
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

# download libpng.h and compile yourself with
```

```
#   http://www.libpng.org/pub/png/libpng.html
#   tar -xvf libpng-1.6.9.tar.gz
#   ./configure --host=x86 --prefix=$PROJWORK/$proj/lib/libpng
# afterwards install pngwriter yourself:
#   https://github.com/ax3l/pngwriter#install
export LIBPNG_ROOT=$PROJWORK/$proj/lib/libpng
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$LIBPNG_ROOT/lib
export PNGWRITER_ROOT=$PROJWORK/$proj/lib/pngwriter
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PNGWRITER_ROOT/lib

# helper variables and tools ########################################
export PICSRC=$HOME/src/picongpu
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

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

export PYTHONPATH=$PYTHONPATH:$SPLASH_ROOT/bin

alias getInteractive="qsub -I -A $proj -q debug -l nodes=1,walltime=30:00"

# "tbg" default options #############################################################
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="submit/titan-ornl/batch_profile.tpl"
```

Piz Daint (CSCS)

```
# this file is loaded from all PIConGPU template files `*.tpl` therefore please
# copy this file to `$SCRATCH/picongpu.profile`
#

# General modules ###################################################################
#
# if the wrong environment is loaded we switch to the gnu environment
module li 2>&1 | grep "PrgEnv-cray" > /dev/null
if [ $? -eq 0 ] ; then
    module swap PrgEnv-cray PrgEnv-gnu/6.0.3
else
    module load PrgEnv-gnu/6.0.3
fi

module load CMake/3.6.2
module load cudatoolkit/8.0.54_2.2.8_ga620558-2.1

# Libraries #########################################################################
module load cray-mpich/7.5.0
module load cray-hdf5-parallel/1.10.0

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

# Environment #######################################################################
#
# needs to be compiled by the user
export BOOST_ROOT=$SCRATCH/lib/boost-1.62.0
export PNGWRITER_ROOT=$SCRATCH/lib/pngwriter
export ADIOS_ROOT=$SCRATCH/lib/adios-1.11.1

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PNGWRITER_ROOT/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_ROOT/lib/
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ADIOS_ROOT/lib/

export MPI_ROOT=$MPICH_DIR
export HDF5_ROOT=$HDF5_DIR

# 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

# Compile for cluster nodes
#   (CMake likes to unwrap the Cray wrappers)
export CC=`which cc`
export CXX=`which CC`

export PICSRC=$HOME/src/picongpu
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

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

# send me a mail on BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="FAIL"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

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

# helper tools #################################################################

# allocate an interactive shell for one hour
# `getInteractive 2` # allocates to interactive nodes (default: 1)
getInteractive() {
    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
}
```

Taurus (TU Dresden)

```
module purge

# General modules ##############################################################
#
module load oscar-modules
module load cmake/3.3.1 git
module load cuda/8.0.44 # gcc <= 5, intel 15-16
module load bullxmpi
module load gnuplot/4.6.1

# Compilers ####################################################################
```

```
### GCC
module load gcc/5.3.0 boost/1.60.0-gnu5.3
### ICC
#module load intel/2015.3.187 boost/1.59.0-intel2015.3.187
### PGI
#export BOOST_ROOT=$HOME/lib/boost_1_57_pgi_14_9
#export BOOST_INC=$BOOST_ROOT/include
#export BOOST_LIB=$BOOST_ROOT/lib
# must be set in `which <pgiDir>/bin/localrc`:
#   set NOSWITCHERROR=YES;
#module load pgi/14.9 boost/<noneBuildYet>

# Other Software ##############################################################
#
module load hdf5/1.8.18-gcc-5.3.0-xmpi
module load zlib/1.2.8

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

export PNGWRITER_ROOT=$HOME/lib/pngwriter
export SPLASH_ROOT=$HOME/lib/splash

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib/pngwriter/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib/splash/lib/

export PICSRC=$HOME/src/picongpu
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

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

# send me a mail on BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="ALL"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="submit/taurus-tud/k80_profile.tpl"
```

Lawrencium (LBNL)

```
if [ -f /etc/profile.d/modules.sh ]
then
        . /etc/profile.d/modules.sh
        module purge

        # Core Dependencies
        module load gcc/4.4.7
        module load cuda/5.5
        # not yet available, build boost as in `INSTALL.md`
        #module load boost/1.57.0-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
        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=$HOME/src/picongpu
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

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

# send me a mail on BEGIN, END, FAIL, REQUEUE, ALL,
# TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="ALL"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# "tbg" default options ########################################################
#   - SLURM (sbatch)
#   - fermi queue (also available: 2 K20 via k20_profile.tpl)
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="submit/lawrencium-lbnl/fermi_profile.tpl"
```

Judge (FZJ)

(example missing)

CHAPTER 2

Usage

Reference

PIConGPU is a year-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

Citation

BibTeX code:

```
@inproceedings{PIConGPU2013,
 author = {Bussmann, M. and Burau, H. and Cowan, T. E. and Debus, A. and Huebl, A.␣
↪and Juckeland, G. and Kluge, T. and Nagel, W. E. and Pausch, R. and Schmitt, F.␣
↪and Schramm, U. and Schuchart, J. and Widera, R.},
 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},
}
```

Acknowledgements

In many cases you receive support and code base maintainance 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.

See also:

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

Basics

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`.

As in our *compiling from source* section, we need a few directories to structure our workflow:

```
# source code
mkdir $HOME/src
# temporary build directory
mkdir $HOME/build

# PIConGPU input files
mkdir $HOME/paramSets
# PIConGPU simulation output
mkdir $SCRATCH/runs
```

Step-by-Step

TL;DR

("*too long, didn't read* and know how *compiling works*")

```
pic-create ~/paramSets/originalSet ~/paramSets/myLWFA

cd ~/build
pic-configure $HOME/paramSets/myLWFA
make -j install

cd ~/paramSets/myLWFA
tbg -s qsub -c submit/0016gpus.cfg -t submit/hypnos-hzdr/k20_profile.tpl $SCRATCH/
↪runs/lwfa_001
```

1. Create an Input (Parameter) Set

```
# clone the LWFA example to $HOME/paramSets/myLWFA
pic-create $PICSRC/examples/LaserWakefield/ $HOME/paramSets/myLWFA
```

Now edit `$HOME/paramSets/case001/include/simulation_defines/param/*` to change the *physical configuration of this parameter set*.

Now edit `$HOME/paramSets/case001/submit/*.cfg` to adjust *runtime parameters (simulation size, number of GPUs, plugins, ...)*.

Hint: you can further create parameter sets from parameter sets.

2. Compile Simulation

New `.param` files in inputs or changes of parameters in excisting files require a re-compile of PIConGPU. Our script `pic-configure` is a wrapper for CMake to quickly specify which parameter set and source version of PIConGPU shall be used.

```
# go to an empty build directory
cd $HOME/build
# clean it if necessary
rm -rf ../build/*

# configure case001
pic-configure $HOME/paramSets/myLWFA

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

We always configure *one* parameter set for *one* compilation. If you adjust `.param` input files just now, you can just go back to `$HOME/build` and run `make -j install` again without further need to clean the directory or configuration.

3. Run Simulation

```
# go to param set with up-to-date PIConGPU binaries
cd $HOME/paramSets/myLWFA

# example run for the HPC System "hypnos" using a PBS batch system
tbg -s qsub -c submit/0016gpus.cfg -t submit/hypnos-hzdr/k20_profile.tpl $SCRATCH/
↪runs/lwfa_001
```

This will create the directory `$SCRATCH/runs/lwfa_001` were all simulation output will be written to. `tbg` will further create a subfolder `picongpu/` in the directory of the run with the same structure as `myLWFA` to archive your input files.

Further Reading

Individual input files, their syntax and usage are explained in the following sections.

See `pic-create --help` for more options during parameter set creation:

```
pic-create create a new parameter set for simulation input
merge default picongpu parameters and a given example's input

usage: pic-create [OPTION] [src_dir] dest_dir
If no src_dir is set picongpu a default case is cloned

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

Dependencies: rsync
```

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

```
configure create a cmake call for picongpu
and get fast access to selected picongpu cmake options

usage: configure [OPTION] <parameter_DIRECTORY>
If no path_to_CMakeLists.txt is set the directory of this binary is used as source␣
↪directory.

-i | --install       - path were picongpu should be installed (default is
↪<parameter_DIRECTORY>)
-a | --arch          - set cuda architecture (semicolon separated list, e.g.: "20;
↪35;37;52;60")
-c | --cmake         - overwrite options for cmake (e.g.: -c "-DPIC_VERBOSE=1")
-t <presetNumber>    - configure this preset from cmakeFlags
-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`.

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

The `picongpu/` directory of a run can also be reused to clone parameters via `pic-create` by using this run as origin directory or to create a new binary with `configure`: e.g. `pic-configure -i $HOME/paramSets/myLWFA2 $SCRATCH/runs/lwfa_001`.

See `tbg --help` *for more information* about the `tbg` tool.

.param Files

PIC Core

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 as 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.

namespace `picongpu`

Variables

constexpr uint32_t **`ABSORBER_CELLS`**[3][2] = { {32, 32}, {32, 32}, {32, 32} }
Defines the size of the absorbing zone (in cells)

unit: none

constexpr float_X **`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

constexpr float_64 **movePoint** = 0.9
When to start moving the co-moving window.

Slide point model: A virtual photon starts at t=0 at the lower end of the global simulation box in y-direction of the simulation. When it reaches movePoint % of the global simulation box, the co-moving window starts to move with the speed of light.

Note global simulation area: there is one additional "hidden" row of gpus at the y-front, when you use the co-moving window. 1.0 would correspond to: start moving exactly when the above described "virtual photon" from the lower end of the box' Y-axis reaches the beginning of this "hidden" row of GPUs.

namespace SI

Variables

constexpr float_64 **DELTA_T_SI** = 0.8e-16
Duration of one timestep unit: seconds.

constexpr float_64 **CELL_WIDTH_SI** = 0.1772e-6
equals X unit: meter

constexpr float_64 **CELL_HEIGHT_SI** = 0.4430e-7
equals Y - the laser & moving window propagation direction unit: meter

constexpr float_64 **CELL_DEPTH_SI** = CELL_WIDTH_SI
equals Z unit: meter

dimension.param

The spatial dimensionality of the simulation.

Defines

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

namespace picongpu

Variables

constexpr uint32_t **simDim** = SIMDIM

components.param

Select the laser profile and the field solver here.

Defines

ENABLE_CURRENT 1
enable (1) or disable (0) current calculation (deprecated)

namespace picongpu

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

namespace `laserProfile`
Laser Profile Selection:

- laserNone : no laser init
- laserGaussianBeam : Gaussian beam (focusing)
- laserPulseFrontTilt : Gaussian beam with a tilted pulse envelope in 'x' direction
- laserWavepacket : wavepacket (Gaussian in time and space, not focusing)
- laserPlaneWave : a plane wave (Gaussian in time)
- laserPolynom : a polynomial laser envelope

Adjust the settings of the selected profile in laser.param

namespace `fieldSolver`
Field Solver Selection:

- fieldSolverYee : standard Yee solver
- fieldSolverLehe: Num. Cherenkov free field solver in a chosen direction
- fieldSolverDirSplitting: Sentoku's Directional Splitting Method
- fieldSolverNone: disable the vacuum update of E and B

For development purposes:

- fieldSolverYeeNative : generic version of fieldSolverYee (need more shared memory per GPU and is slow)

Adjust the settings of the selected field solver in fieldSolver.param

fieldSolver.param

Configure the selected field solver method.

You can set/modify Maxwell solver specific options in the section of each "FieldSolver".

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< simDim >:
  - default for staggered grids/Yee-scheme
  - updates E
- Binomial< simDim >: 2nd order Binomial filter
  - smooths the current before assignment in staggered grid
  - updates E & breaks local charge conservation slightly
- NoneDS< simDim >:
  - experimental assignment for all-centered/directional splitting
  - updates E & B at the same time

namespace picongpu

namespace fieldSolverDirSplitting

Typedefs

using picongpu::fieldSolverDirSplitting::CurrentInterpolation = typedef currentInt

namespace fieldSolverLehe

Lehe Solver The solver proposed by R.

Lehe et al in Phys. Rev. ST Accel. Beams 16, 021301 (2013)

Typedefs

using picongpu::fieldSolverLehe::CherenkovFreeDir = typedef CherenkovFreeDirection

Distinguish the direction where numerical Cherenkov Radiation by moving particles shall be suppressed.

using picongpu::fieldSolverLehe::CurrentInterpolation = typedef currentInterpolati

namespace fieldSolverNone

Typedefs

using picongpu::fieldSolverNone::CurrentInterpolation = typedef currentInterpolati

namespace fieldSolverYee

Typedefs

using picongpu::fieldSolverYee::CurrentInterpolation = typedef currentInterpolatio

namespace fieldSolverYeeNative

Typedefs

using picongpu::fieldSolverYeeNative::CurrentInterpolation = typedef currentInterp

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 picongpu::densityProfiles::Gaussian = typedef GaussianImpl< GaussianParam >

using picongpu::densityProfiles::Homogenous = typedef HomogenousImpl

using picongpu::densityProfiles::LinearExponential = typedef LinearExponentialImpl

using picongpu::densityProfiles::GaussianCloud = typedef GaussianCloudImpl< Gaussi

using picongpu::densityProfiles::SphereFlanks = typedef SphereFlanksImpl<SphereFla

using picongpu::densityProfiles::FromHDF5 = typedef FromHDF5Impl< FromHDF5Param >

using picongpu::densityProfiles::FreeFormula = typedef FreeFormulaImpl< FreeFormul

Functions

picongpu::densityProfiles::PMACC_STRUCT(GaussianParam, (PMACC_C_VALUE (float_X,

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 = float_X(0.0)` for `gasCenterLeft_SI < y < gasCenterRight_SI`

picongpu::densityProfiles::PMACC_STRUCT(LinearExponentialParam, (PMACC_C_VALUE

parameter for `LinearExponential` profile

```
* Density Profile: /\
*                  /  -,_
*   linear        /      -,_    exponential
*   slope        /  |       -,_ slope
*                  MAX
*
```

picongpu::densityProfiles::PMACC_STRUCT(GaussianCloudParam, (PMACC_C_VALUE (flo

picongpu::densityProfiles::PMACC_STRUCT(SphereFlanksParam, (PMACC_C_VALUE (uint

The profile consists out of the composition of 3 1D profiles with the scheme: exponential increasing flank, constant sphere, exponential decreasing flank.

```
*            ___
*  1D:  _,./   \./_   rho(r)
*
*  2D:  ..,x,..   density: . low
*       .,xxx,.            , middle
*       ..,x,..            x high (constant)
*
```

picongpu::densityProfiles::PMACC_STRUCT(FromHDF5Param, (PMACC_C_STRING (filenam

struct FreeFormulaFunctor

Public Functions

HDINLINE float_X picongpu::densityProfiles::FreeFormulaFunctor::operator()(co

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

constexpr 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

pusher.param

Configure particle pushers.

Those pushers can then be selected by a particle species in species.param and speciesDefinition.param

namespace picongpu

namespace particlePusherAxel

Enums

enum TrajectoryInterpolationType
Values:

LINEAR = 1u

NONLINEAR = 2u

Variables

constexpr *TrajectoryInterpolationType* **TrajectoryInterpolation** = LINEAR

laser.param

Configure laser profiles.

namespace picongpu

namespace laser

Variables

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

namespace laserGaussianBeam

Enums

enum `PolarisationType`
Available polarisation types.

Values:

`LINEAR_X` = 1u

`LINEAR_Z` = 2u

`CIRCULAR` = 4u

Functions

`picongpu::laserGaussianBeam::PMACC_CONST_VECTOR(float_X, MODENUMBER+ 1, LAGUERRE`

Variables

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

constexpr uint32_t **`MODENUMBER`** = 0
Use only the 0th Laguerremode for a standard Gaussian.

constexpr *PolarisationType* **`Polarisation`** = CIRCULAR
Polarization selection.

namespace `SI`

Variables

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::E
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

namespace laserNone
No Laser initialization.

namespace SI

Variables

constexpr float_64 **WAVE_LENGTH_SI** = 0.0
unit: meter

constexpr float_64 **AMPLITUDE_SI** = 0.0
unit: Volt /meter

constexpr float_64 **PULSE_LENGTH_SI** = 0.0

namespace laserPlaneWave
plane wave (use periodic boundaries!)

no transverse spacial envelope based on the electric potential Phi = Phi_0 * exp(0.5 * (x-x_0)^2 / sigma^2) * cos(k*(x - x_0) - phi) by applying -grad Phi = -d/dx Phi = E(x) we get: E = -Phi_0 * exp(0.5 * (x-x_0)^2 / sigma^2) * [k*sin(k*(x - x_0) - phi) + x/sigma^2 * cos(k*(x - x_0) - phi)]

This approach ensures that int_{-infinity}^{+infinity} E(x) = 0 for any phase if we have no transverse profile as we have with this plane wave train

Since PIConGPU requires a temporally defined electric field, we use: t = x/c and (x-x_0)/sigma = (t-t_0)/tau and k*(x-x_0) = omega*(t-t_0) with omega/k = c and tau * c = sigma and get: E = -Phi_0*omega/c * exp(0.5 * (t-t_0)^2 / tau^2) * [sin(omega*(t - t_0) - phi) + t/(omega*tau^2) * cos(omega*(t - t_0) - phi)] and define: E_0 = -Phi_0*omega/c integrationCorrectionFactor = t/(omega*tau^2)

Please consider: 1) The above formulae does only apply to a Gaussian envelope. If the plateau length is not zero, the integral over the volume will only vanish if the plateau length is a multiple of the wavelength. 2) Since we define our envelope by a sigma of the laser intensity, tau = PULSE_LENGTH * sqrt(2)

Enums

enum PolarisationType
Available polarisation types.

Values:

LINEAR_X = 1u

LINEAR_Z = 2u

CIRCULAR = 4u

Variables

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.

namespace SI

Variables

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::E
UNITCONV.

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 constexprant 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)

namespace laserPolynom
not focusing polynomial laser pulse

no phase shifts, just spacial envelope

Variables

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

namespace SI

Variables

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::E
UNITCONV.

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** = 4.0e-15
Pulse length: PULSE_LENGTH_SI = total length of polynamial laser pulse Rise time = 0.5 * PULSE_LENGTH_SI Fall time = 0.5 * PULSE_LENGTH_SI in order to compare to a gaussian pulse: rise time = sqrt{2} * T_{FWHM} unit: seconds.

constexpr float_64 **W0x_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 float_64 **W0z_SI** = W0x_SI

namespace laserPulseFrontTilt

Enums

enum PolarisationType
Available polarisation types.

Values:

LINEAR_X = 1u

LINEAR_Z = 2u

CIRCULAR = 4u

Variables

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

constexpr *PolarisationType* **Polarisation** = LINEAR_X
Polarization selection.

namespace SI

Variables

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::E
UNITCONV.

constexpr float_64 **AMPLITUDE_SI** = 1.738e13
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 **TILT_X_SI** = 0
the tilt angle between laser propagation in y-direction and laser axis in x-direction (0 degree == no tilt) unit: degree

namespace laserWavepacket
not focusing wavepaket with spacial gaussian envelope

no phase shifts, just spacial envelope including correction to laser formular derived from vector potential, so the integration along propagation direction gives 0 this is important for few-cycle laser pulses

Enums

enum PolarisationType
Available polarisation types.

Values:

LINEAR_X = 1u

LINEAR_Z = 2u

CIRCULAR = 4u

Variables

constexpr float_64 **RAMP_INIT** = 20.0
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 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 SI

Variables

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::E
UNITCONV.

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** = 7.0 * WAVE_LENGTH_SI / ::picongpu::SI::SPEED_C
The profile of the test Lasers 0 and 2 can be stretched by a constexprant 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, WO_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

particle.param

Configurations for particle manipulators.

Set up and declare functors that can be used in speciesInitalization.param for particle species initialization and manipulation, such as temperature distributions, drifts, pre-ionization and in-cell position.

namespace picongpu

namespace particles

Variables

constexpr float_X **MIN_WEIGHTING** = 10.0
a particle with a weighting below MIN_WEIGHTING will not be created / will be deleted

unit: none

constexpr uint32_t **TYPICAL_PARTICLES_PER_CELL** = 2
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

using picongpu::particles::manipulators::AssignXDrift = typedef DriftImpl< Dri
definition of manipulator that assigns a dirft in X

using picongpu::particles::manipulators::AddTemperature = typedef TemperatureIn

using picongpu::particles::manipulators::AssignXDriftToLowerHalfXPosition = typ
definition of a relative position selection that assigns a drift in X

using `picongpu::particles::manipulators::DoubleWeighting = typedef FreeImpl< Do`
definition of a free particle manipulator: double weighting

typedef *FreeRngImpl*<*RandomEnabledRadiationFunctor*, nvidia::rng::distributions::Uniform_float> `RandomEna`

using `picongpu::particles::manipulators::RandomPosition = typedef RandomPositio`
changes the in-cell position of each particle of a species

Functions

`picongpu::particles::manipulators::CONST_VECTOR(float_X, 3, DriftParam_direct`
Parameter for DriftParam.

struct `DoubleWeightingFunctor`
Unary particle manipulator: double each weighting.

Public Functions

template <**typename** T_Particle>
`DINLINE void picongpu::particles::manipulators::DoubleWeightingFunctor::op`

struct `DriftParam`
Parameter for a particle drift assignment.

Public Members

const DriftParam_direction_t `direction`

Public Static Attributes

constexpr float_64 `gamma` = 1.0

struct `IfRelativeGlobalPositionParam`
Parameters for an assignment in a relative position selection.

Public Static Attributes

constexpr float_X `lowerBound` = 0.0

constexpr float_X `upperBound` = 0.5

constexpr uint32_t `dimension` = 0

struct `RandomEnabledRadiationFunctor`

Public Functions

template <**typename** T_Rng, **typename** T_Particle>
`DINLINE void picongpu::particles::manipulators::RandomEnabledRadiationFunc`

struct `TemperatureParam`
Parameter for a temperature assignment.

Public Static Attributes

constexpr float_64 **temperature** = 0.0

namespace startPosition

Typedefs

using picongpu::particles::startPosition::Random = typedef RandomImpl< RandomPa
definition of random particle start

using picongpu::particles::startPosition::Quiet = typedef QuietImpl< QuietParam
definition of quiet particle start

using picongpu::particles::startPosition::OnePosition = typedef OnePositionImpl
definition of one specific position for particle start

Functions

picongpu::particles::startPosition::CONST_VECTOR(float_X, 3, InCellOffset, 0.
sit directly in lower corner of the cell

struct OnePositionParameter

Public Members

const InCellOffset_t **inCellOffset**

Public Static Attributes

constexpr uint32_t **numParticlesPerCell** = TYPICAL_PARTICLES_PER_CELL
Count of particles per cell at initial state.

unit: none

struct QuietParam

Public Types

using numParticlesPerDimension = mCT::shrinkTo<mCT::Int<1, *TYPICAL_PARTICLES_PER_CELL*
Count of particles per cell per direction at initial state.

unit: none

struct RandomParameter

Public Static Attributes

constexpr uint32_t **numParticlesPerCell** = TYPICAL_PARTICLES_PER_CELL
Count of particles per cell at initial state.

unit: none

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 `picongpu::UsedParticleShape = typedef 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 CICShape = particles::shapes::CIC;

using `picongpu::UsedField2Particle = typedef FieldToParticleInterpolation< UsedPartic`
define which interpolation method is used to interpolate fields to particles

using `picongpu::UsedParticleCurrentSolver = typedef currentSolver::Esirkepov< UsedPar`
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)
- currentSolver::ZigZag< SHAPE > : particle shapes - CIC, TSC, PCS, P4S (1st to 4th order)

using `picongpu::UsedParticlePusher = typedef particles::pusher::Boris`
particle pusher configuration

Define 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 development purposes:

- particles::pusher::Axel : a pusher developed at HZDR during 2011 (testing)
- particles::pusher::Free : free propagation, ignore fields (= free stream model)
- particles::pusher::Photon : propagate with c in direction of normalized mom.

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 not define any type like float3,double3,...

This is only a name without a specialization

`value_identifier`(uint64_t, particleId, IdProvider<*simDim*>::getNewId)

`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.))`
momentum at (previous) timestep t-1

`picongpu::value_identifier(float_X, weighting, 0. 0)`
weighting of the macro particle

`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(float_X, boundElectrons, float_X(0.0))`
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

`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 species.param

`alias`(particlePusher)
alias for particle pusher

See species.param

alias (ionizers)
alias for particle ionizers

See ionizer.param

alias (ionizationEnergies)
alias for ionization energy container

See ionizationEnergies.param

alias (synchrotronPhotons)
alias for synchrotronPhotons alias for ion species used for bremsstrahlung

See speciesDefinition.param

alias (bremsstrahlungPhotons)
alias for photon species used for bremsstrahlung

alias (interpolation)
alias for particle to field interpolation

See species.param

alias (current)
alias for particle current solver

See species.param

alias (atomicNumbers)
alias for particle flag: atomic numbers

See ionizer.param

- only reasonable for atoms / ions / nuclei

alias (effectiveNuclearCharge)
alias for particle flag: effective nuclear charge

See ionizer.param

- only reasonable for atoms / ions / nuclei

alias (massRatio)
alias for particle mass ratio

mass ratio between base particle default value: 1.0 if unset

See `speciesConstants.param` SI::BASE_MASS_SI and a user defined species

alias (chargeRatio)
alias for particle charge ratio

charge ratio between base particle default value: 1.0 if unset

See `speciesConstants.param` SI::BASE_CHARGE_SI and a user defined species

alias (densityRatio)
alias for particle density ratio

density ratio between default density default value: 1.0 if unset

See `density.param` SI::BASE_DENSITY_SI and a user defined species

speciesConstants.param

Constants and thresholds for particle species.

Defines the reference mass and reference charge to express species with (default: electrons with negative charge).

namespace picongpu

Variables

constexpr float_X **GAMMA_THRESH** = float_X(1.005)
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.

constexpr float_X **GAMMA_INV_SQUARE_RAD_THRESH** = float_X(0.18)
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

namespace SI

Variables

constexpr float_64 **BASE_MASS_SI** = ELECTRON_MASS_SI
base particle mass

reference for massRatio in speciesDefinition.param

unit: kg

constexpr float_64 **BASE_CHARGE_SI** = ELECTRON_CHARGE_SI
base particle charge

reference for chargeRatio in speciesDefinition.param

unit: C

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" (lvalues to store with each species) and "flags" (rvalues & 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 picongpu::DefaultParticleAttributes = typedef MakeSeq_t< position< position_pic
describe attributes of a particle

using picongpu::ParticleFlagsPhotons = typedef bmpl::vector< particlePusher< particle

using picongpu::PIC_Photons = typedef Particles< bmpl::string< 'p', 'h' >, ParticleFl

using picongpu::ParticleFlagsElectrons = typedef bmpl::vector< particlePusher< UsedPa

using picongpu::PIC_Electrons = typedef Particles< bmpl::string< 'e' >, ParticleFlags

using picongpu::ParticleFlagsIons = typedef bmpl::vector< particlePusher< UsedParticl

using picongpu::PIC_Ions = typedef Particles< bmpl::string< 'i' >, ParticleFlagsIons,

using picongpu::VectorAllSpecies = typedef 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)

speciesInitialization.param

Available species functors in src/picongpu/include/particles/InitFunctors.hpp.

- CreateDensity<T_DensityFunctor, T_PositionFunctor, T_SpeciesType> Create particle distribution based on a density profile and an in-cell positioning. Fills a particle species (`fillAllGaps()` is called).

  See density.param

  Template Parameters

  - `T_DensityFunctor`: unary lambda functor with density description,

namespace picongpu

namespace particles

Typedefs

using picongpu::particles::InitPipeline = typedef mpl::vector<>
InitPipeline defines in which order species are initialized.

the functors are called in order (from first to last functor)

Memory

memory.param

namespace picongpu

Typedefs

typedef mCT::shrinkTo<mCT::Int<8, 8, 4>, *simDim*>::type **SuperCellSize**
size of a superCell

volume of a superCell must be <= 1024

typedef MappingDescription<*simDim*, *SuperCellSize*> **MappingDesc**
define mapper which is used for kernel call mappings

Variables

constexpr size_t **reservedGpuMemorySize** = 350 *1024*1024

constexpr uint32_t **GUARD_SIZE** = 1

constexpr uint32_t **BYTES_EXCHANGE_X** = 4 * 256 * 1024
how many bytes for buffer is reserved to communication in one direction

constexpr uint32_t **BYTES_EXCHANGE_Y** = 6 * 512 * 1024

constexpr uint32_t **BYTES_EXCHANGE_Z** = 4 * 256 * 1024

constexpr uint32_t **BYTES_CORNER** = 8 * 1024

constexpr uint32_t **BYTES_EDGES** = 32 * 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.

precision.param

namespace picongpu

mallocMC.param

namespace picongpu

Typedefs

using picongpu::DeviceHeap = typedef mallocMC::Allocator< mallocMC::CreationPolicies:

struct DeviceHeapConfig

Public Types

using pagesize = boost::mpl::int_<2 * 1024 * 1024>

using accessblocks = boost::mpl::int_<4>

using regionsize = boost::mpl::int_<8>

using wastefactor = boost::mpl::int_<2>

using resetfreedpages = boost::mpl::bool_<true>

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 < simD
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 < simD
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 < simD
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 kinectic 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 0
enable synchtrotron 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 oszillation periods greater than a timestep are not created since the grid already accounts for them.

This cutoff ratio is defined as: photon-oszillation-period / timestep

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

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

```
  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
Al-27 ~100% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 13.0

constexpr float_X **numberOfNeutrons** = 14.0

struct Carbon_t
C-12 98.9% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 6.0

constexpr float_X **numberOfNeutrons** = 6.0

struct Copper_t
Cu-63 69.15% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 29.0

constexpr float_X **numberOfNeutrons** = 34.0

struct Deuterium_t
H-2 0.02% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 1.0

constexpr float_X **numberOfNeutrons** = 1.0

struct Gold_t
Au-197 ~100% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 79.0

constexpr float_X **numberOfNeutrons** = 118.0

struct Helium_t
He-4 ~100% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 2.0

constexpr float_X **numberOfNeutrons** = 2.0

struct Hydrogen_t
H-1 99.98% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 1.0

constexpr float_X **numberOfNeutrons** = 0.0

struct Nitrogen_t
N-14 99.6% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 7.0

constexpr float_X **numberOfNeutrons** = 7.0

struct Oxygen_t
O-16 99.76% NA.

Public Static Attributes

constexpr float_X **numberOfProtons** = 8.0

constexpr float_X **numberOfNeutrons** = 8.0

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.

References: Clementi, E.; Raimondi, D. L. (1963) "Atomic Screening Constants from SCF Functions" J. Chem. Phys. 38 (11): 2686–2689. doi:10.1063/1.1733573 Clementi, E.; Raimondi, D. L.; Reinhardt, W. P. (1967) "Atomic Screening Constants from SCF Functions. II. Atoms with 37 to 86 Electrons" Journal of Chemical Physics. 47: 1300–1307. doi:10.1063/1.1712084
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!

Functions

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 1,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 1,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 2,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 6,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 7,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 8,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 13,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 29,

picongpu::ionization::effectiveNuclearCharge::PMACC_CONST_VECTOR(float_X, 79,

namespace particles

namespace ionization

namespace thomasFermi

Variables

constexpr float_X **TFAlpha** = 14.3139
Fitting parameters to average ionization degree Z* = 4/3*pi*R_0^3 * n(R_0) as an extension towards arbitrary atoms and temperatures.

See table IV of http://www.sciencedirect.com/science/article/pii/S0065219908601451 doi:10.1016/S0065-2199(08)60145-1

constexpr float_X **TFBeta** = 0.6624

constexpr float_X **TFA1** = 3.323e-3

constexpr float_X **TFA2** = 9.718e-1

constexpr float_X **TFA3** = 9.26148e-5

constexpr float_X **TFA4** = 3.10165

constexpr float_X **TFB0** = -1.7630

constexpr float_X **TFB1** = 1.43175

constexpr float_X **TFB2** = 0.31546

constexpr float_X **TFC1** = -0.366667

constexpr float_X **TFC2** = 0.983333

ionizationEnergies.param

This file contains the ionization energies and conversion to corresponding electric field strengths for different species.

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

```
  ionizers< MakeSeq_t< particles::ionization::IonizationModel<␣
↪Species2BCreated > > >,
  atomicNumbers< ionization::atomicNumbers::Element_t >,
  effectiveNuclearCharge< ionization::effectiveNuclearCharge::Element_t >,
  ionizationEnergies< ionization::energies::AU::Element_t >
```

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 UNITCONV_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.

for PMACC_CONST_VECTOR usage, Reference: Kramida, A., Ralchenko, Yu., Reader, J., and NIST ASD Team (2014) NIST Atomic Spectra Database (ver. 5.2), [Online] Available: http://physics.nist.gov/asd [2017, February 8] National Institute of Standards and Technology, Gaithersburg, MD

See libPMacc/include/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, Hydroge

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 1, Deuteri

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 2, Helium,

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 6, Carbon,

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 7, Nitroge

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 8, Oxygen,

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 13, Alumin

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 29, Copper

picongpu::ionization::energies::AU::PMACC_CONST_VECTOR(float_X, 79, Gold,

Plugins

fileOutput.param

namespace picongpu

Typedefs

using picongpu::ChargeDensity_Seq = typedef bmpl::transform< VectorAllSpecies, Create

using picongpu::EnergyDensity_Seq = typedef bmpl::transform< VectorAllSpecies, Create

using picongpu::MomentumComponent_Seq = typedef bmpl::transform< VectorAllSpecies, Cr

using picongpu::FieldTmpSolvers = typedef MakeSeq_t< ChargeDensity_Seq, EnergyDensity

FieldTmpSolvers groups all solvers that create data for *FieldTmp* ******.

FieldTmpSolvers is used in

See *FieldTmp* to calculate the exchange size

using picongpu::NativeFileOutputFields = typedef MakeSeq_t< FieldE, FieldB >
FileOutputFields: Groups all Fields that shall be dumped.

Possible native fields: *FieldE*, *FieldB*, *FieldJ*

using picongpu::FileOutputFields = typedef MakeSeq_t< NativeFileOutputFields, FieldTm

using picongpu::FileOutputParticles = typedef VectorAllSpecies
FileOutputParticles: Groups all Species that shall be dumped **********.

hint: to disable particle output set to typedef bmpl::vector0< > FileOutputParticles;

isaac.param

namespace picongpu

namespace isaacP

Typedefs

using picongpu::isaacP::Native_Seq = typedef MakeSeq_t< FieldE, FieldB, FieldJ >

using picongpu::isaacP::Density_Seq = typedef bmpl::transform< VectorAllSpecies, C

using picongpu::isaacP::Fields_Seq = typedef MakeSeq_t< Native_Seq, Density_Seq >

particleCalorimeter.param

namespace picongpu

namespace particleCalorimeter

Functions

HDINLINE float2_X picongpu::particleCalorimeter::mapYawPitchToNormedRange(const
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

radiation.param

Defines

PIC_VERBOSE_RADIATION 3

namespace picongpu

namespace parameters

Variables

constexpr unsigned int **N_observer** = 256

namespace rad_frequencies_from_list

Variables

constexpr unsigned int **N_omega** = 2048

namespace rad_linear_frequencies

Variables

constexpr unsigned int **N_omega** = 2048

namespace SI

Variables

constexpr float_64 **omega_min** = 0.0

constexpr float_64 **omega_max** = 1.06e16

namespace rad_log_frequencies

Variables

constexpr unsigned int **N_omega** = 2048

namespace SI

Variables

constexpr float_64 **omega_min** = 1.0e14

constexpr float_64 **omega_max** = 1.0e17

namespace radiation

Typedefs

using picongpu::radiation::RadiationParticleFilter = typedef picongpu::particles::

struct GammaFilterFunctor
select particles for radiation

Public Functions

template <typename T_Particle>
DINLINE void picongpu::radiation::GammaFilterFunctor::operator()(T_Particle &

Public Static Attributes

constexpr float_X **radiationGamma** = 5.0

namespace radiationNyquist

Variables

constexpr float_32 **NyquistFactor** = 0.5

radiationObserver.param

namespace picongpu

namespace radiation_observer

Functions

HDINLINE vector_64 picongpu::radiation_observer::observation_direction(const int

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

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

visualization.param

Defines

EM_FIELD_SCALE_CHANNEL1 -1

EM_FIELD_SCALE_CHANNEL2 -1

EM_FIELD_SCALE_CHANNEL3 -1

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, co

DINLINE float_X picongpu::visPreview::preChannel2(const float3_X & field_B, co

DINLINE float_X picongpu::visPreview::preChannel3(const float3_X & field_B, co

Variables

constexpr float_X **preParticleDens_opacity** = 0.25

constexpr float_X **preChannel1_opacity** = 1.0

constexpr float_X **preChannel2_opacity** = 1.0

constexpr float_X **preChannel3_opacity** = 1.0

visColorScales.param

namespace picongpu

namespace colorScales

namespace blue

Functions

HDINLINE void picongpu::colorScales::blue::addRGB(float3_X & img, const floa

namespace gray

Functions

HDINLINE void picongpu::colorScales::gray::addRGB(float3_X & img, const floa

namespace grayInv

Functions

HDINLINE void picongpu::colorScales::grayInv::addRGB(float3_X & img, const f

namespace green

Functions

HDINLINE void picongpu::colorScales::green::addRGB(float3_X & img, const flo

namespace none

Functions

HDINLINE void picongpu::colorScales::none::addRGB(const float3_X &, const f

namespace red

Functions

HDINLINE void picongpu::colorScales::red::addRGB(float3_X & img, const float

Misc

starter.param

namespace `picongpu`

seed.param

namespace `picongpu`

Enums

enum `Seeds`

Values:

`TEMPERATURE_SEED` = 255845

`POSITION_SEED` = 854666252

`IONIZATION_SEED` = 431630977

`FREERNG_SEED` = 99991

struct `GlobalSeed`

global seed

global seed to derive GPU local seeds from

- vary it to shuffle pseudo random generators for exactly same simulation
- note: even when kept constant, highly parallel simulations do not ensure 100% deterministic simulations on the floating point level

Public Functions

uint32_t `operator()` ()

physicalConstants.param

namespace `picongpu`

Variables

constexpr float_64 `PI` = 3.14159265358979323846264338327950288419716939937

constexpr float_64 `UNITCONV_keV_to_Joule` = 1.60217646e-16

constexpr float_64 `UNITCONV_Joule_to_keV` = (1.0 / UNITCONV_keV_to_Joule)

constexpr float_64 `UNITCONV_AU_to_eV` = 27.21139

constexpr float_64 `UNITCONV_eV_to_AU` = (1.0 / UNITCONV_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 **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

Particles

Initialization

The following operations can be applied in the picongpu::particles::InitPipeline inside speciesInitialization.param:

CreateDensity

template <**typename** T_DensityFunctor, **typename** T_PositionFunctor, **typename** T_SpeciesType = bmpl::_1>
struct `picongpu::particles::CreateDensity`
create density based on a normalized profile and a position profile

constructor with current time step of density and position profile is called after the density profile is created `fillAllGaps()` is called

Template Parameters

- `T_DensityFunctor`: unary lambda functor with profile description
- `T_PositionFunctor`: unary lambda functor with position description
- `T_SpeciesType`: type of the used species

DeriveSpecies

template <**typename** T_SrcSpeciesType, **typename** T_DestSpeciesType = bmpl::_1>
struct `picongpu::particles::DeriveSpecies`
derive species out of a another species

after the species is derived `fillAllGaps()` on T_DestSpeciesType is called copy all attributes from the source species except `particleId` to the destination species

Template Parameters

- `T_SrcSpeciesType`: source species
- `T_DestSpeciesType`: destination species

Inherits from *picongpu::particles::ManipulateDeriveSpecies< manipulators::NoneImpl, T_SrcSpeciesType, T_DestSpeciesType >*

Manipulate

template <**typename** T_Functor, **typename** T_SpeciesType = bmpl::_1>
struct `picongpu::particles::`**`Manipulate`**
run a user defined functor for every particle

•constructor with current time step is called for the functor on the host side

•**Warning** `fillAllGaps()` is not called

Template Parameters

- `T_Functor`: unary lambda functor
- `T_SpeciesType`: type of the used species

ManipulateDeriveSpecies

template <typename *T_ManipulateFunctor*, typename *T_SrcSpeciesType*, typename *T_DestSpeciesType* = bmpl::_1>
struct `picongpu::particles::`**`ManipulateDeriveSpecies`**
derive species out of a another species

after the species is derived `fillAllGaps()` on T_DestSpeciesType is called copy all attributes from the source species except `particleId` to the destination species

See src/picongpu/include/particles/manipulators

Template Parameters

- `T_ManipulateFunctor`: a pseudo-binary functor accepting two particle species: destination and source,

Template Parameters

- `T_SrcSpeciesType`: source species
- `T_DestSpeciesType`: destination species

FillAllGaps

template <**typename** T_SpeciesType = bmpl::_1>
struct `picongpu::particles::`**`FillAllGaps`**
call method fill all gaps of a species

Template Parameters

- `T_SpeciesType`: type of the species

Manipulation

Some of the particle operations above can further take the following functors as arguments to manipulate attributes of particle species:

AssignImpl

struct picongpu::particles::manipulators::**AssignImpl**

CopyAttribute

using picongpu::particles::manipulators::CopyAttribute = typedef FreeImpl< detail::CopyAtt
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

DensityWeighting

struct picongpu::particles::manipulators::**DensityWeighting**

DriftImpl

template <typename T_ParamClass, **typename** T_ValueFunctor, **typename** T_SpeciesType>
struct picongpu::particles::manipulators::**DriftImpl**
Inherits from T_ValueFunctor

FreeImpl

template <typename T_Functor>
struct picongpu::particles::manipulators::**FreeImpl**
generic manipulator to create user defined manipulators

Template Parameters

- T_Functor: user defined functor
  - must implement `void operator()(ParticleType)` **or** `void operator()(ParticleType1, ParticleType2)`
  - **optional**: can implement **one** host side constructor `T_Functor()` or `T_Functor(uint32_t currentTimeStep)`

Inherits from T_Functor

FreeRngImpl

template <typename T_Functor, **typename** T_Distribution, **typename** T_SpeciesType>
struct picongpu::particles::manipulators::**FreeRngImpl**
call simple free user defined functor and provide a random number generator

example: add

```
#include "nvidia/rng/distributions/Uniform_float.hpp"

struct RandomXFunctor
{
    template< typename T_Rng, typename T_Particle >
    DINLINE void operator()( T_Rng& rng, T_Particle& particle )
    {
```

```
        particle[ position_ ].x() = rng();
    }
};

typedef FreeRngImpl<
   RandomXFunctor,
   nvidia::rng::distributions::Uniform_float
> RandomXPos;
particles::Manipulate< RandomXPos, SPECIES_NAME >
```

to `InitPipeline` in `speciesInitialization.param`

Template Parameters

- `T_Functor`: user defined unary functor
- `T_Distribution`: random number distribution
- `T_SpeciesType`: type of the species that shall be manipulated

Inherits from T_Functor

IfRelativeGlobalPositionImpl

template <typename T_ParamClass, **typename** T_Functor, **typename** T_SpeciesType>
struct `picongpu::particles::manipulators::IfRelativeGlobalPositionImpl`
Inherits from T_Functor

ProtonTimesWeighting

struct `picongpu::particles::manipulators::ProtonTimesWeighting`

RandomPositionImpl

template <typename T_SpeciesType>
struct `picongpu::particles::manipulators::RandomPositionImpl`
Change the in cell position.

This functor changes the in-cell position of each particle of a species

example: add

```
typedef particles::manipulators::RandomPositionImpl<> RandomPosition;
particles::Manipulate<RandomPosition,SPECIES_NAME>
```

to `InitPipeline` in `speciesInitialization.param`

Template Parameters

- `T_SpeciesType`: type of the species that shall be manipulated

SetAttributeImpl

template <typename T_ParamClass, **typename** T_ValueFunctor, **typename** T_SpeciesType>
struct `picongpu::particles::manipulators::SetAttributeImpl`
Inherits from T_ValueFunctor

TemperatureImpl

template <typename T_ParamClass, **typename** T_ValueFunctor, **typename** T_SpeciesType>
struct `picongpu::particles::manipulators::TemperatureImpl`
Inherits from T_ValueFunctor

Plugins

TBG

todo: explain idea and use case

- what is a batch system
- cfg files
- tpl files
- behaviour (existing dirs, submission, environment)

Usage

```
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"] [-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, ...)
                                 Default: [submitsystem] via export TBG_SUBMIT
-t | --tpl      [file]         - Template to create a batch file from.
                                 tbg will use stdin, if no file is specified.
                                 Default: [templateFile] via export TBG_TPLFILE
-o                             - Overwrite any template variable:
                                 spaces within the right side of assign are not␣
↪allowed
                                 e.g. -o "VARNAME1=10 VARNAME2=5"
                                 Overwriting is done after cfg file was executed
-h | --help                    - Shows help (this output).

[projectPath]                  - Project directory containing source code and
                                 binaries
                                 Default: current directory
destinationPath                - Directory for simulation output.


TBG exports the following variables, which can be used in cfg and tpl files at
any time:
 TBG_jobName                   - name of the job
 TBG_jobNameShort              - short name of the job, without blanks
 TBG_cfgPath                   - absolute path to cfg file
 TBG_cfgFile                   - full absolute path and name of cfg file
 TBG_projectPath               - absolute project path (see optional parameter
                                 projectPath)
 TBG_dstPath                   - absolute path to destination directory
```

Example with Slurm

Job Submission

PIConGPU job submission on the *Taurus* cluster at *TU Dresden*:

- *tbg -s sbatch -c submit/0008gpus.cfg -t submit/taurus-tud/k80_profile.tpl $SCRATCH/runs/test123*

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*
  - `` ` squeue -u `whoami` -l` ``
- 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 12345* abort job
  - *scancel -s Number 12345* send signal or signal name to job
  - *scontrol update timelimit=4:00:00 jobid=12345* change the walltime of the job
  - *scontrol update jobid=12345 dependency=afterany:54321* only start the job after job with id *54321* has finished
  - *scontrol hold jobid=12345* prevent the job from starting
  - *scontrol release jobid=12345* or in short *scontrol release 12345* release the job to be eligible for run (after it was set on hold)

.cfg File Macros

Feel free to copy & paste sections of the files below into your *.cfg*, e.g. to configure complex plugins:

```
# Copyright 2014-2017 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 avalable 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).
##
## If not stated otherwise, variables/flags must not be used more than once!
################################################################################

################################################################################
## Section: Required Variables
## Variables in this secton 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 GPUs in each dimension (x,y,z) to use for the simulation
TBG_gpu_x=1
TBG_gpu_y=2
TBG_gpu_z=1

# Size of the simulation grid in cells as "-g 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 GPU,
#       the size of a supercell (in cells) is defined in `memory.param`
TBG_gridSize="-g 128 256 128"

# Number of simulation steps/iterations as "-s N"
TBG_steps="-s 100"


################################################################################
## 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!
##
## Please add all variables you define in this section to TBG_plugins.
################################################################################

# Variables which are created by TBG (should be self-descriptive)
TBG_jobName
TBG_jobNameShort
TBG_cfgPath
TBG_cfgFile
TBG_projectPath
TBG_dstPath


# Regex to describe the static distribution of the cells for each GPU
# default: equal distribution over all GPUs
# 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).
# Default: no periodic dimensions
TBG_periodic="--periodic 1 0 1"


# Enables moving window (sliding) in your simulation
TBG_movingWindow="-m"

################################################################################
## 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.omegaList     If spectrum frequencies are taken from a file,
↪ this gives the path to this list
#--<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␣
↪2800 --<species>_radiation.end 3000"


# 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"


# 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"


# 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.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.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.space␣
↪y --<species>_phaseSpace.momentum px --<species>_phaseSpace.min -1.0 --<species>_
↪phaseSpace.max 1.0"
TBG_<species>_PSypy="--<species>_phaseSpace.period 10 --<species>_phaseSpace.space␣
↪y --<species>_phaseSpace.momentum py --<species>_phaseSpace.min -1.0 --<species>_
↪phaseSpace.max 1.0"
TBG_<species>_PSypz="--<species>_phaseSpace.period 10 --<species>_phaseSpace.space␣
↪y --<species>_phaseSpace.momentum pz --<species>_phaseSpace.min -1.0 --<species>_
↪phaseSpace.max 1.0"


# Sum up total energy every .period steps for
# - species   (--<species>_energy)
# - fields    (--fields_energy)
TBG_sumEnergy="--fields_energy.period 10 --<species>_energy.period 10"


# Count the number of macro particles per species for every .period steps
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 is dumped every .period steps to the fileset .file.
TBG_hdf5="--hdf5.period 100 --hdf5.file simData"

# 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"
# see 'adios_config -m', e.g., for on-the-fly zlib compression
#     (compile ADIOS with --with-zlib=<ZLIB_ROOT>)
#   --adios.compression zlib
# 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
# specify further options for the transports, see ADIOS manual
# chapter 6.1.5, e.g., 'random_offset=1;stripe_count=4'
#                      (FS chooses OST;user chooses striping factor)
#   --adios.transport-params "semicolon_separated_list"
```

```
# Create a checkpoint that is restartable every --checkpoints steps
#   http://git.io/PToFYg
TBG_checkpoints="--checkpoints 1000"

# Restart the simulation from checkpoints created using TBG_checkpoints
TBG_restart="--restart"
# By default, the last checkpoint is restarted if not specified via
#   --restart-step 1000
# To restart in a new run directory point to the old run where to start from
#   --restart-directory /path/to/simOutput/checkpoints

# Presentation mode: loop a simulation via soft restart
#   does either start from 0 again or from the checkpoint specified with
#   --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
# Note: does currently not work with `Radiation` plugin
TBG_softRestarts="--softRestarts 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"

# Connect to a live-view server (start the server in advance)
TBG_liveViewYX="--<species>_liveView.period 1 --<species>_liveView.slicePoint 0.5 -
↪-<species>_liveView.ip 10.0.2.254 \
               --<species>_liveView.port 2020 --<species>_liveView.axis yx"
TBG_liveViewYZ="--<species>_liveView.period 1 --<species>_liveView.slicePoint 0.5 -
↪-<species>_liveView.ip 10.0.2.254 \
               --<species>_liveView.port 2021 --<species>_liveView.axis yz"


# 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"
```

```
# 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!
################################################################################

# Number of compute devices in each dimension as "-d X Y Z"
TBG_devices="-d !TBG_gpu_x !TBG_gpu_y !TBG_gpu_z"


# Combines all declared variables. These are passed to PIConGPU as command line␣
↪flags.
# The program output (stdout) is stored in a file called output.stdout.
TBG_programParams="!TBG_devices     \
                   !TBG_gridSize    \
                   !TBG_steps       \
                   !TBG_plugins"

# Total number of GPUs
TBG_tasks="$(( TBG_gpu_x * TBG_gpu_y * TBG_gpu_z ))"
```

Example Setups

Bremsstrahlung: Emission of Bremsstrahlung from Laser-Foil Interaction

- author: Heiko Burau <h.burau (at) hzdr.de>
- maintainer: Heiko Burau <h.burau (at) hzdr.de>

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 kinectic 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 appearence 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

Bunch: Thomson scattering from laser electron-bunch interaction

- author: Richard Pausch <r.pausch (at) hzdr.de>, Rene Widera <r.widera (at) hzdr.de>
- maintainer: Richard Pausch <r.pausch (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

Empty: Default PIC Algorithm

- author: Axel Huebl <a.huebl (at) hzdr.de>
- maintainer: 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 *src/picongpu/include/simulation_defines/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

KelvinHelmholtz: Kelvin-Helmholtz Instability

- author: Axel Huebl <a.huebl (at) hzdr.de>, E. Paulo Alves, Thomas Grismayer
- maintainer: Axel Huebl <a.huebl (at) hzdr.de>

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. Modifiying the ion species' mass to resample positrons instead is a test we perform regularly to control numerical heating and charge conservation.

References

LaserWakefield: Laser Electron Acceleration

- author: Axel Huebl <a.huebl (at) hzdr.de>, René Widera, Heiko Burau, Richard Pausch, Marco Garten

- maintainer: Axel Huebl <a.huebl (at) hzdr.de>

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 challening 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

WarmCopper: Average Charge State Evolution of Copper Irradiated by a Laser

- author: Axel Huebl <a.huebl (at) hzdr.de>, Hyun-Kyung Chung
- maintainer: Axel Huebl <a.huebl (at) hzdr.de>

This setup initializes a homogenous, non-moving, copper block irradiated by a laser with 10^18 W/cm^3 as a benchmark for *[SCFLY]*[1] 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 <Z> 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.

References

Workflows

This section contains typical user workflows and best practices.

Setting the Number of Cells

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 GPU needs to:

1. contain an integer *multiple* of supercells
2. at least *three* supercells

Supercell sizes in terms of number of cells are set in *memory.param* and are by default `8x8x4` for 3D3V simulations on GPUs. For 2D3V simulations, `16x16` is usually a good supercell size, however the default is simply cropped to `8x8`, so make sure to change it to get more performance.

Changing the Resolution with a Fixed Target

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] In PIConGPU, we generally refer to the implemented subset of *SCFLY* (solving Non-LTE population kinetics) as *FLYlite*.

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 (transveral) positioning of targets in *density.param*
5. *recompile*

CHAPTER 3

Models

The Particle-in-Cell Algorithm

For now, please refer to the textbooks *[BirdsallLangdon]*, *[HockneyEastwood]*, our *latest paper on PIConGPU* and *[Huebl2014]* (chapters 2.3, 3.1 and 3.4).

References

Landau-Lifschitz Radiation Reaction

To do

References

Ionization

Field Ionization

Get started here https://github.com/ComputationalRadiationPhysics/picongpu/wiki/Ionization-in-PIConGPU

Collisional Ionization

Implemented LTE Model: Thomas-Fermi Ionization

In-development: NLTE Models

References

Photons

Radiation reaction and (hard) photons: why and when are they needed. Models we implemented and verified:

- *Landau-Lifschitz Model (semi-classical)*
- QED Models (Synchrotron & Bremsstrahlung)

Would be great to add your Diploma Thesis 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").

Attention: Bremsstrahlung: The individual electron direction and gamma emission are not correlated. (momentum is microscopically / per e- not conserved, only collectively.)

Attention: "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 ...).

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).

References

CHAPTER 4

Post-Processing

Python

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

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

Matplotlib

One common way to visualize plots:

- http://matplotlib.org/faq/usage_faq.html#usage
- https://gist.github.com/ax3l/fc123cb94f59d440f952

Jupyter

Access, share, modify, run and interact with your python scripts from your browser:

https://jupyter.readthedocs.io

openPMD-viewer

A library that reads and visualizes data in our HDF5 files. Provides an API to correctly convert units to SI, interpret iteration steps correctly, annotate axis and much more. Also provides an interactive GUI for fast exploration via Jupyter notebooks.

https://github.com/openPMD/openPMD-viewer/tree/master/tutorials

yt-project (dev)

Starting with yt 3.4, our HDF5 output, which uses the openPMD markup, can be read, processed and visualized with yt.

http://yt-project.org/docs/dev/

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#documentation

openPMD

Please see https://github.com/ComputationalRadiationPhysics/picongpu/wiki/Post-processing-and-Visualization for now.

ParaView

Please see https://github.com/ComputationalRadiationPhysics/picongpu/wiki/ParaView for now.

CHAPTER 5

Development

How to Participate as a Developer

Contents

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 1.7.9.5

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
- you may like to visit http://mac.github.com/

Windows:

- see here
- just kidding, it's this link
- please use UTF8 for your files and take care of line endings

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 *avoide 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.)

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 PIConGPU** 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 pushed your local branch 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.

Coding Guide Lines

Well - there are some! ;)

- Please follow our recommendations in our wiki page.
- The `uncrustify` script picongpu_uncrustify.cfg can be used for auto-formatting your code.

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=libPMacc && 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 ist `PIConGPU` (case sensitive!) and adds the GPLv3+ only.

Files in the directory `thirdParty/` are only imported from remote repositories as written in our LICENSE. 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.

Commit Rules

See our commit rules page

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!

Sphinx

In the following section we explain how to contribute to this documentation.

If you are reading the HTML version on http://picongpu.readthedocs.io and want to improve or correct existing pages, check the "Edit on GitHub" link on the right upper corner of each document.

Alternatively, go to *docs/source* in our source code and follow the directory structure of reStructuredText (`.rst`) files there. For intrusive changes, like structural changes to chapters, please open an issue to discuss them beforehand.

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:

```
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
```

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!

```
# 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 :)
make html

# open it, e.g. with firefox :)
firefox build/html/index.html

# now again for the pdf :)
make latexpdf

# open it, e.g. with okular
build/latex/PIConGPU.pdf
```

Useful Links

- A primer on writing restFUL files for sphinx
- Why You Shouldn't Use "Markdown" for Documentation
- Markdown Limitations in Sphinx

Important PIConGPU Classes

This is very, very small selection of classes of interest to get you started.

MySimulation

class `picongpu::MySimulation`

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

Inherits from *PMacc::SimulationHelper< simDim >*

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.

Return plugin name

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** ()

FieldE

class picongpu::**FieldE**
Inherits from *PMacc::SimulationFieldHelper< MappingDesc >*, PMacc::ISimulationData

FieldB

class picongpu::**FieldB**
Inherits from *PMacc::SimulationFieldHelper< MappingDesc >*, PMacc::ISimulationData

FieldJ

class picongpu::**FieldJ**
Inherits from *PMacc::SimulationFieldHelper< MappingDesc >*, PMacc::ISimulationData

FieldTmp

class picongpu::**FieldTmp**
Tmp (at the moment: scalar) field for plugins and tmp data like "gridded" particle data (charge density, energy density, ...)

Inherits from *PMacc::SimulationFieldHelper< MappingDesc >*, PMacc::ISimulationData

Particles

template <**typename** T_Name, **typename** T_Flags, **typename** T_Attributes>
class picongpu::**Particles**
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]

Inherits from *PMacc::ParticlesBase< ParticleDescription< T_Name, SuperCellSize, T_Attributes, T_Flags >, MappingDesc, DeviceHeap >*, PMacc::ISimulationData

Public Types

typedef ParticleDescription<T_Name, *SuperCellSize*, T_Attributes, T_Flags> **SpeciesParticleDescription**

typedef ParticlesBase<*SpeciesParticleDescription*, *MappingDesc*, DeviceHeap> **ParticlesBaseType**

typedef *ParticlesBaseType*::FrameType **FrameType**

typedef *ParticlesBaseType*::FrameTypeBorder **FrameTypeBorder**

typedef *ParticlesBaseType*::ParticlesBoxType **ParticlesBoxType**

Public Functions

Particles(**const** std::shared_ptr<DeviceHeap> &*heap*, *MappingDesc cellDescription*, SimulationDataId *datasetID*)

void **createParticleBuffer**()

void **init**()

void **update**(uint32_t *currentStep*)

template <**typename** T_DensityFunctor, **typename** T_PositionFunctor>
void **initDensityProfile**(T_DensityFunctor &*densityFunctor*, T_PositionFunctor &*positionFunctor*, **const** uint32_t *currentStep*)

template <**typename** T_SrcName, **typename** T_SrcAttributes, **typename** T_SrcFlags, **typename** T_ManipulateFunctor>
void **deviceDeriveFrom**(*Particles*<T_SrcName, T_SrcAttributes, T_SrcFlags> &*src*, T_ManipulateFunctor &*manipulateFunctor*)

template <**typename** T_Functor>
void **manipulateAllParticles**(uint32_t *currentStep*, T_Functor &*functor*)

SimulationDataId **getUniqueId**()
Return the globally unique identifier for this simulation data.

Return globally unique identifier

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

static PMacc::traits::StringProperty **getStringProperties**()

ComputeGridValuePerFrame

template <**class** T_ParticleShape, **class** T_DerivedAttribute>
class `picongpu::particleToGrid::`**ComputeGridValuePerFrame**

Public Types

typedef T_ParticleShape::ChargeAssignment **AssignmentFunction**

typedef PMacc::math::CT::make_Int<*simDim*, *lowerMargin*>::type **LowerMargin**

typedef PMacc::math::CT::make_Int<*simDim*, *upperMargin*>::type **UpperMargin**

Public Functions

HDINLINE **ComputeGridValuePerFrame**()

HDINLINE float1_64 picongpu::particleToGrid::ComputeGridValuePerFrame::getUnit() co
return unit for this solver

Return solver unit

HINLINE std::vector< float_64 > picongpu::particleToGrid::ComputeGridValuePerFrame:
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)

HINLINE std::string picongpu::particleToGrid::ComputeGridValuePerFrame::getName() co
return name of the this solver

Return name of solver

template <**class** FrameType, **class** TVecSuperCell, **class** BoxTmp>
DINLINE void picongpu::particleToGrid::ComputeGridValuePerFrame::operator()(FrameTyp

Public Static Attributes

constexpr int **supp** = AssignmentFunction::support

constexpr int **lowerMargin** = supp / 2

constexpr int **upperMargin** = (supp + 1) / 2

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

Environment

template <uint32_t *T_dim*>
class PMacc::**Environment**
Global *Environment* singleton for PMacc.

Inherits from PMacc::detail::Environment

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

static *Environment*<T_dim> &**get**()
get the singleton Environment< DIM >

Return instance of Environment<DIM >

DataConnector

class `PMacc::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

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
- `currentStep`: current simulation step

void **share**(**const** std::shared_ptr<ISimulationData> &*data*)
Registers a new Dataset with data and identifier id.

If a Dataset with identifier id already exists, a runtime_error is thrown. (Check with *DataConnector::hasId* when necessary.)

Parameters

- `data`: simulation data to share ownership

void **unshare**(SimulationDataId *id*)
End sharing a dataset with identifier id.

Parameters

- `id`: id of the dataset to remove

void **clean**()
Unshare all associated datasets.

template <class TYPE>
std::shared_ptr<TYPE> **get**(SimulationDataId *id*, bool *noSync* = false)
Returns shared pointer to managed data.

Reference to data in Dataset with identifier id and type TYPE is returned. If the Dataset status in invalid, it is automatically synchronized. Increments the reference counter to the dataset specified by id. This reference has to be released after all read/write operations before the next synchronize()/getData() on this data are done using *releaseData()*.

Return returns a reference to the data of type TYPE

Template Parameters

- `TYPE`: if of the data to load

Parameters

- `id`: id of the Dataset to load from
- `noSync`: indicates that no synchronization should be performed, regardless of dataset status

void **releaseData**(SimulationDataId)
Indicate a data set gotten temporarily via.

See getData is not used anymore

Parameters

- `id`: id for the dataset previously acquired using getData()

Friends

friend **PMacc::DataConnector::detail::Environment**

DataSpace

template <unsigned *DIM*>
class `PMacc::DataSpace`
A DIM-dimensional data space.

DataSpace describes a 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

- `DIM`: dimension (1-3) of the dataspace

Inherits from PMacc::math::Vector< int, DIM >

Public Types

typedef math::Vector<int, DIM> **BaseType**

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<DIM> &*vec*)

HDINLINE int PMacc::DataSpace::getDim() const
Returns number of dimensions (DIM) of this *DataSpace*.

Return number of dimensions

HINLINE bool PMacc::DataSpace::isOneDimensionGreaterThan(const DataSpace < DIM > &
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<DIM>**() **const**

HDINLINE **operator dim3**() **const**

Public Static Functions

static HDINLINE DataSpace<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 setfor all dimensions

Public Static Attributes

constexpr int **Dim** =DIM

Vector

Warning: doxygenclass: Cannot find class "PMacc::Vector" in doxygen xml output for project "PIConGPU" from directory: ../xml

SuperCell

template <**class** TYPE>
class `PMacc::SuperCell`

Public Functions

HDINLINE **SuperCell**()

HDINLINE TYPE* PMacc::SuperCell::FirstFramePtr()

HDINLINE TYPE* PMacc::SuperCell::LastFramePtr()

HDINLINE const TYPE* PMacc::SuperCell::FirstFramePtr() const

HDINLINE const TYPE* PMacc::SuperCell::LastFramePtr() const

HDINLINE bool PMacc::SuperCell::mustShift()

HDINLINE void PMacc::SuperCell::setMustShift(bool value)

HDINLINE lcellId_t PMacc::SuperCell::getSizeLastFrame()

HDINLINE void PMacc::SuperCell::setSizeLastFrame(lcellId_t size)

PMACC_ALIGN(firstFramePtr, TYPE *)

PMACC_ALIGN(lastFramePtr, TYPE *)

GridBuffer

template <class *TYPE*, unsigned *DIM*, class *BORDERTYPE* = TYPE>
class `PMacc::GridBuffer`

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.

Inherits from PMacc::HostDeviceBuffer< TYPE, DIM >

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)

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)

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.

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

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)

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 the 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

void **addExchangeBuffer**(**const** Mask &*receive*, **const** *DataSpace*<DIM> &*dataSpace*, uint32_t *communicationTag*, bool *sizeOnDevice* = false)

Add Exchange in dedicated memory space.

An Exchange is added to this *GridBuffer*. The exchange buffers use the 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
- `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)

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**

template<>
ExchangeIntern<BORDERTYPE, DIM> ***sendExchanges**[27]

template<>
ExchangeIntern<BORDERTYPE, DIM> ***receiveExchanges**[27]

template<>
EventTask **receiveEvents**[27]

template<>
EventTask **sendEvents**[27]

uint32_t **maxExchange**

SimulationFieldHelper

template <class *CellDescription*>
class `PMacc::SimulationFieldHelper`

Public Types

typedef CellDescription **MappingDesc**

Public Functions

SimulationFieldHelper(CellDescription *description*)

virtual ~SimulationFieldHelper()

virtual void **reset**(uint32_t *currentStep*) = 0
Reset is as well used for init.

virtual void **syncToDevice**() = 0
Synchronize data from host to device.

Protected Attributes

CellDescription **cellDescription**

ParticlesBase

template <typename *T_ParticleDescription*, class *T_MappingDesc*, typename *T_DeviceHeap*>
class `PMacc::ParticlesBase`
Inherits from *PMacc::SimulationFieldHelper< T_MappingDesc >*

Public Types

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>

typedef *BufferType*::FrameType **FrameType**

typedef *BufferType*::FrameTypeBorder **FrameTypeBorder**

typedef *BufferType*::ParticlesBoxType **ParticlesBoxType**

typedef ParticleDescription::HandleGuardRegion **HandleGuardRegion**

typedef ParticlesTag **SimulationDataTag**

Public Functions

void **fillAllGaps**()

void **fillBorderGaps**()

void **deleteGuardParticles**(uint32_t *exchangeType*)

template <uint32_t *T_area*>
void **deleteParticlesInArea**()

void **bashParticles**(uint32_t *exchangeType*)

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

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**

ParticleDescription

Warning: doxygenclass: Cannot find class "PMacc::ParticleDescription" in doxygen xml output for project "PIConGPU" from directory: ../xml

ParticleBox

Warning: doxygenclass: Cannot find class "PMacc::ParticleBox" in doxygen xml output for project "PIConGPU" from directory: ../xml

Frame

Warning: doxygenclass: Cannot find class "PMacc::Frame" in doxygen xml output for project "PIConGPU" from directory: ../xml

IPlugin

class `PMacc::IPlugin`

Inherits from 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`: cuurent 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)

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 descpription.

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**

PluginConnector

class `PMacc::PluginConnector`
Plugin registration and management class.

Public Functions

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

void **loadPlugins**()
Calls load on all registered, not loaded plugins.

void **unloadPlugins**()
Unloads all registered, loaded plugins.

std::list<po::options_description> **registerHelp**()
Publishes command line parameters for registered plugins.

Return list of boost program_options command line parameters

void **setNotificationPeriod**(INotify **notifiedObj*, uint32_t *period*)
Set the notification period.

Parameters

- `notifiedObj`: the object to notify, e.g. an *IPlugin* instance
- `period`: notification period

void **notifyPlugins**(uint32_t *currentStep*)
Notifies plugins that data should be dumped.

Parameters

- `currentStep`: current simulation iteration step

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

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)

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

friend **PMacc::PluginConnector::detail::Environment**

SimulationHelper

template <unsigned *DIM*>
class PMacc::**SimulationHelper**

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)

Inherits from *PMacc::IPlugin*

Public Functions

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.

uint32_t **checkpointPeriod**

std::string **checkpointDirectory**

uint32_t **numCheckpoints**

int32_t **restartStep**

std::string **restartDirectory**

bool **restartRequested**

const std::string **CHECKPOINT_MASTER_FILE**

std::string **author**

ForEach

template <typename *T_MPLSeq*, typename *T_Functor*, typename *T_Accessor* = compileTime::accessors::Identity< >>
struct `PMacc::algorithms::forEach::ForEach`

Compile-Time for each for Boost::MPL Type Lists.

Example: MPLSeq = boost::mpl::vector<int,float> Functor = any unary lambda functor Accessor = lambda operation identity

Template Parameters

- `T_MPLSeq`: A mpl sequence that can be accessed by mpl::begin, mpl::end, mpl::next
- `T_Functor`: An unary lambda functor with a HDINLINE void operator()(...) method _1 is substituted by Accessor's result using boost::mpl::apply with elements from T_MPLSeq. The maximum number of parameters for the operator() is limited by PMACC_MAX_FUNCTOR_OPERATOR_PARAMS
- `T_Accessor`: An unary lambda operation

definition: F(X) means boost::apply<F,X>

call: ForEach<MPLSeq,Functor,Accessor>()(42); unrolled code: Functor(Accessor(int))(42); Functor(Accessor(float))(42);

Public Types

typedef bmpl::transform<T_MPLSeq, *ReplacePlaceholder*<bmpl::_1>>::type **SolvedFunctors**

typedef boost::mpl::begin<*SolvedFunctors*>::type **begin**

typedef boost::mpl::end<*SolvedFunctors*>::type **end**

typedef detail::CallFunctorOfIterator<*begin*, *end*> **NextCall**

typedef detail::CallFunctorOfIterator<*end*, *end*> **Functor**

Public Functions

template <typename... *T_Types*>
PMACC_NO_NVCC_HDWARNING HDINLINE void PMacc::algorithms::forEach::ForEach::operator

template <typename... *T_Types*>
PMACC_NO_NVCC_HDWARNING HDINLINE void PMacc::algorithms::forEach::ForEach::operator

template <**typename** X>
struct ReplacePlaceholder

Inherits from boost::mpl::apply1< T_Functor, bmpl::apply1< T_Accessor, X >::type >

Kernel Start

template <typename *T_KernelFunctor*>
struct `PMacc::exec::Kernel`
wrapper for the user kernel functor

contains debug information like filename and line of the kernel call

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 <**typename** T_VectorGrid, **typename** T_VectorBlock>
HINLINE auto PMacc::exec::Kernel::operator()(T_VectorGrid const & gridExtent, T_Vect
configured kernel object

this objects contains the functor and the starting parameter

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(...) PMacc::exec::kernel(__VA_ARGS__, __FILE__, static_cast< size_t >(__LINE__))
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

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, ...) PMACC_PP_STRUCT_DEF(BOOST_PP_CAT(BOOST_PP_CAT(pmacc_,name),__COUNTER__)
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, ...) (0,(type,name,dim,__VA_ARGS__))
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) (1,(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) (2,(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, ...) (5,(type,name, type(__VA_ARGS__)))
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, ...) (5, \ (\ (PMacc::math::Vector<type,dim>), \ name, \ PMacc::math::Vector<type,dim>(__VA_ARGS__) \ ) \ )
create changeable member vector

```
PMACC_VECTOR_DIM(float_64, simDim, center_SI, 1.134e-5, 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) (3,(_,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(...) (4,(_,_,__VA_ARGS__))
create any code extension

```
PMACC_EXTENT(typedef float FooFloat;)
// is the equivalent of
typedef float FooFloat;
```

Parameters

- ...: any code

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) identifier(name, \ typedef in_type type; \ static HDINLINE type getValue() \ { \ return in_default; \ } \ static std::string getName() \ { \ return std::string(#name); \ } \ )
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) PMACC_alias(name,__COUNTER__)
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;

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:

doxygenindex::

project PIConGPU

path '../xml'

outline

no-link

Bibliography

[Spack] T. Gamblin and contributors. *A flexible package manager that supports multiple versions, configurations, platforms, and compilers*, SC '15 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2015), DOI:10.1145/2807591.2807623, https://github.com/LLNL/spack

[modules] J.L. Furlani, P.W. Osel. *Abstract Yourself With Modules*, Proceedings of the 10th USENIX conference on System administration (1996), http://modules.sourceforge.net

[Lmod] R. McLay and contributors. *Lmod: An Environment Module System based on Lua, Reads TCL Modules, Supports a Software Hierarchy*, https://github.com/TACC/Lmod

[nvidia-docker] Nvidia Corporation and contributors. *Build and run Docker containers leveraging NVIDIA GPUs*, https://github.com/NVIDIA/nvidia-docker

[CMake] Kitware Inc. *CMake: Cross-platform build management tool*, http://cmake.org/

[BurauDipl] H. Burau. *Entwicklung und Überprüfung eines Photonenmodells für die Abstrahlung durch hochenergetische Elektronen*, Diploma Thesis TU Dresden (2016), https://doi.org/10.5281/zenodo.192116

[Jackson] J.D. Jackson. *Electrodynamics*, Wiley-VCH Verlag GmbH & Co. KGaA (1975), http://onlinelibrary.wiley.com/doi/10.1002/9783527600441.oe014

[Salvat] F. Salvat, J. Fernández-Varea, J. Sempau, X. Llovet. *Monte carlo simulation of bremsstrahlung emission by electrons*, Radiation Physics and Chemistry (2006), https://dx.doi.org/10.1016/j.radphyschem.2005.05.008

[PauschDipl] Richard Pausch. *Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics*, Diploma Thesis TU Dresden (2012), http://www.hzdr.de/db/Cms?pOid=38997

[Pausch13] R. Pausch, A. Debus, R. Widera, K. Steiniger, A. Huebl, H. Burau, M. Bussmann, U. Schramm. *How to test and verify radiation diagnostics simulations within particle-in-cell frameworks*, Nuclear Instruments and Methods in Physics Research Section A (2013), http://dx.doi.org/10.1016/j.nima.2013.10.073

[Esarey93] E. Esarey, S. Ride, P. Sprangle. *Nonlinear Thomson scattering of intense laser pulses from beams and plasmas*, Physical Review E (1993), http://dx.doi.org/10.1103/PhysRevE.48.3003

[BirdsallLangdon] C.K. Birdsall, A.B. Langdon. *Plasma Physics via Computer Simulation*, McGraw-Hill (1985), ISBN 0-07-005371-5

[HockneyEastwood] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, CRC Press (1988), ISBN 0-85274-392-0

[Alves12] E.P. Alves, T. Grismayer, S.F. Martins, F. Fiuza, R.A. Fonseca, L.O. Silva. *Large-scale magnetic field generation via the kinetic kelvin-helmholtz instability in unmagnetized scenarios*, The Astrophysical Journal Letters (2012), https://dx.doi.org/10.1088/2041-8205/746/2/L14

[Grismayer13] T. Grismayer, E.P. Alves, R.A. Fonseca, L.O. Silva. *dc-magnetic-field generation in unmagnetized shear flows*, Physical Reveview Letters (2013), https://doi.org/10.1103/PhysRevLett.111.015005

[Bussmann13] M. Bussmann, H. Burau, T.E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W.E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, R. Widera. *Radiative Signatures of the Relativistic Kelvin-Helmholtz Instability*, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (2013), http://doi.acm.org/10.1145/2503210.2504564

[TajimaDawson] T. Tajima, J.M. Dawson. *Laser electron accelerator*, Physical Review Letters (1979), https://dx.doi.org/10.1103/PhysRevLett.43.267

[Modena] A. Modena, Z. Najmudin, A.E. Dangor, C.E. Clayton, K.A. Marsh, C. Joshi, V. Malka, C. B. Darrow, C. Danson, D. Neely, F.N. Walsh. *Electron acceleration from the breaking of relativistic plasma waves*, Nature (1995), https://dx.doi.org/10.1038/377606a0

[PukhovMeyerterVehn] A. Pukhov and J. Meyer-ter-Vehn. *Laser wake field acceleration: the highly non-linear broken-wave regime*, Applied Physics B (2002), https://dx.doi.org/10.1007/s003400200795

[FLYCHK] H.-K. Chung, M.H. Chen, W.L. Morgan, Y. Ralchenko, R.W. Lee. *FLYCHK: Generalized population kinetics and spectral model for rapid spectroscopic analysis for all elements*, High Energy Density Physics I (2005), https://dx.doi.org/10.1016/j.hedp.2005.07.001

[SCFLY] H.-K. Chung, M.H. Chen, R.W. Lee. *Extension of atomic configuration sets of the Non-LTE model in the application to the Ka diagnostics of hot dense matter*, High Energy Density Physics III (2007), https://dx.doi.org/10.1016/j.hedp.2007.02.001

[BirdsallLangdon] C.K. Birdsall, A.B. Langdon. *Plasma Physics via Computer Simulation*, McGraw-Hill (1985), ISBN 0-07-005371-5

[HockneyEastwood] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, CRC Press (1988), ISBN 0-85274-392-0

[Huebl2014] A. Huebl. *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), https://doi.org/10.5281/zenodo.15924

[Vranic2016] M. Vranic, J.L. Martins, R.A. Fonseca, L.O. Silva. *Classical radiation reaction in particle-in-cell simulations*, Computer Physics Communications 204, 114-151 (2016), https://dx.doi.org/10.1016/j.cpc.2016.04.002

[BauerMulser] D. Bauer and P. Mulser. *Exact field ionization rates in the barrier-suppression regime from numerical time-dependent Schrödinger-equation calculations*, Physical Review A 59, 569 (1999), https://dx.doi.org/10.1103/PhysRevA.59.569

[Keldysh] L.V. Keldysh. *Ionization in the field of a strong electromagnetic wave*, Soviet Physics JETP 20, 1307-1314 (1965), http://jetp.ac.ru/cgi-bin/dn/e_020_05_1307.pdf

[Gonoskov] A. Gonoskov, S. Bastrakov, E. Efimenko, A. Ilderton, M. Marklund, I. Meyerov, A. Muraviev, A. Sergeev, I. Surmin, E. Wallin. *Extended particle-in-cell schemes for physics in ultrastrong laser fields: Review and developments*, Physical Review E 92, 023305 (2015), https://dx.doi.org/10.1103/PhysRevE.92.023305

[Furry] W. Furry. *On bound states and scattering in positron theory*, Physical Review 81, 115 (1951), https://doi.org/10.1103/PhysRev.81.115

[Burau2016] H. Burau. *Entwicklung und Überprüfung eines Photonenmodells für die Abstrahlung durch hochenergetische Elektronen* (German), Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2016), https://doi.org/10.5281/zenodo.192116

Index

A

P

V