PIConGPU Documentation

Release 0.4.0-dev

The PIConGPU Community

Nov 21, 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

1.1 Introduction

Section author: Axel Huebl

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

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

1.1.1 Before you Start

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

1.1.2 Ways to Install

Spack

[Spack] is a flexible package manager that can build and organize software 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.

Build from Source

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

Conda

We currently do not have an official conda install (yet). Due to pre-build binaries, performance 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. We are also working on Singularity images.

1.1.3 References

1.2 Instructions

Section author: Axel Huebl

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

See also:

You will need to understand how to use the terminal.

1.2.1 Spack

Section author: Axel Huebl

Preparation

First install spack itself via:

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

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

# build spack's dependencies via spack :)
spack bootstrap

# install a supported compiler
spack install gcc@5.4.0
spack load gcc@5.4.0
spack compiler add

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

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

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

Install

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

```
spack install picongpu %gcc@5.4.0
```

Use PIConGPU

PIConGPU can now be loaded with

```
spack load picongpu %gcc@5.4.0
```

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

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

```
spack install picongpu backend=omp2b %gcc@5.4.0
spack load picongpu backend=omp2b %gcc@5.4.0
```

See also:

You will need to understand how to use the terminal.

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

1.2.2 Docker

Section author: Axel Huebl

Preparation

First install nvidia-docker for your distribution.

Install

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

```
nvidia-docker pull ax3l/picongpu
```

Use PIConGPU

Start a pre-configured LWFA live-simulation with

```
nvidia-docker run -p 2459:2459 -t ax3l/picongpu:0.3.0 /bin/bash -lc start_lwfa
# open firefox and isaac client
```

or just open the container and run your own:

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

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

See also:

You will need to understand how to use the terminal.

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

1.2.3 From Source

Section author: Axel Huebl

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

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 section to build our dependencies.

References

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

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

See also:

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

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

1.3 Dependencies

Section author: Axel Huebl

1.3.1 Overview

1.3.2 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
- *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:*

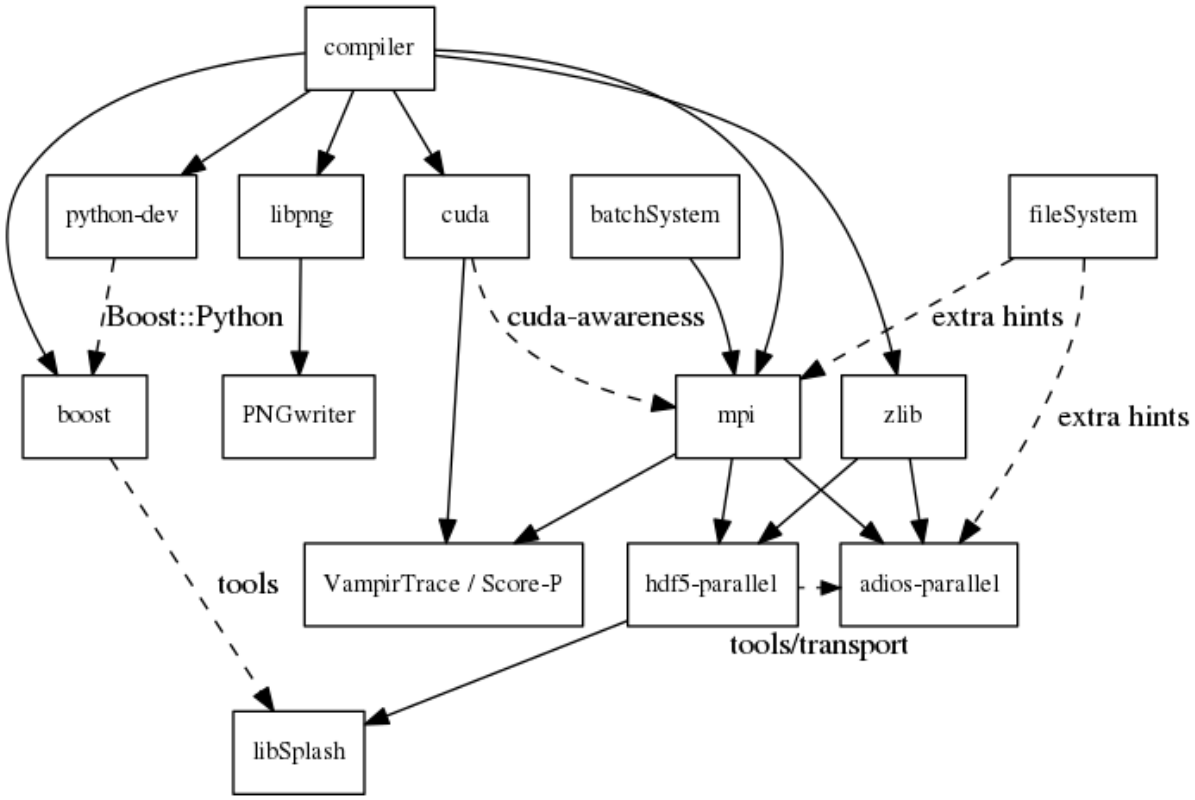


Fig. 1.1: Overview of inter-library dependencies for parallel execution of PIConGPU on a typical HPC system. Due to common binary incompatibilities between compilers, MPI and boost versions, we recommend to organize software with a version-aware package manager such as spack and to deploy a hierarchical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

  - `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`

CMake

- 3.7.0 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** 1.7+ / **MVAPICH2** 1.8+ or similar
- for running on Nvidia GPUs, perform a GPU aware MPI install *after* installing CUDA
- *Debian/Ubuntu:* `sudo apt-get install libopenmpi-dev`
- *Arch Linux:* `sudo pacman --sync openmpi`
- *Spack:*
  - *GPU support:* `spack install openmpi+cuda`
  - *CPU only:* `spack install openmpi`
- *environment:*
  - `export MPI_ROOT=<MPI_INSTALL>`
  - as long as CUDA awareness (`openmpi+cuda`) is missing: `export OMPI_MCA_mpi_leave_pinned=0`

zlib

- *Debian/Ubuntu:* `sudo apt-get install zlib1g-dev`
- *Arch Linux:* `sudo pacman --sync zlib`
- *Spack:* `spack install zlib`

boost

- 1.62.0-1.64.0 (`program_options`, `regex` , `filesystem`, `system`, `math`, `serialization` and header-only libs, optional: `fiber` with `context`, `thread`, `chrono`, `atomic`, `date_time`)
- 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-chrono-dev libboost-atomic-dev libboost-date-time-dev libboost-math-dev libboost-serialization-dev libboost-fiber-dev libboost-context-dev`

- *Arch Linux:* `sudo pacman --sync boost`
- *Spack:* `spack install boost`
- *from source:*
  - `./bootstrap.sh --with-libraries=atomic,chrono,context,date_time,fiber,filesystem,math,program_options,regex,serialization,system,thread --prefix=$HOME/lib/boost`
  - `./b2 cxxflags="-std=c++11" -j4 && ./b2 install`
- *environment:* (assumes install from source in `$HOME/lib/boost`)
  - `export BOOST_ROOT=$HOME/lib/boost`
  - `export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH`

git

- 1.7.9.5 or higher
- *Debian/Ubuntu:* `sudo apt-get install git`
- *Arch Linux:* `sudo pacman --sync git`
- *Spack:* `spack install git`

rsync

- *Debian/Ubuntu:* `sudo apt-get install rsync`
- *Arch Linux:* `sudo pacman --sync rsync`
- *Spack:* `spack install rsync`

PIConGPU Source Code

- `git clone https://github.com/ComputationalRadiationPhysics/picongpu.git $HOME/src/picongpu`
  - *optional:* update the source code with `cd $HOME/src/picongpu && git fetch && git pull`
  - *optional:* change to a different branch with `git branch` (show) and `git checkout <BranchName>` (switch)
- *environment*:
  - `export PICSRC=$PICHOME/src/picongpu`
  - `export PIC_EXAMPLES=$PICSRC/share/picongpu/examples`
  - `export PATH=$PICSRC:$PATH`
  - `export PATH=$PICSRC/src/tools/bin:$PATH`
  - `export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH`

Optional Libraries

CUDA

- 7.5+

- required if you want to run on Nvidia GPUs
- *Debian/Ubuntu:* `sudo apt-get install nvidia-cuda-toolkit`
- *Arch Linux:* `sudo pacman --sync cuda`
- *Spack:* `spack install cuda`
- at least one **CUDA** capable **GPU**
- *compute capability*: `sm_20` or higher (for CUDA 9+: `sm_30` or higher)
- full list of CUDA GPUs and their *compute capability*
- More is always better. Especially, if we are talking GPUs :-)
- *environment:*
  - `export CUDA_ROOT=<CUDA_INSTALL>`

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

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* and *zlib*)
- *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.

1.4 picongpu.profile

Section author: Axel Huebl

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

```
. $HOME/picongpu.profile
```

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

1.4.1 Hypnos (HZDR)

For these profiles to work, you need to download the *PIConGPU source code* manually.

Queue: laser (AMD Opteron 6276 CPUs)

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

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

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

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

        # Core Dependencies
        module load gcc/4.9.2
        module load cmake/3.7.2
        module load boost/1.62.0
        module load openmpi/1.8.6
        module load numactl

        # Plugins (optional)
        module load pngwriter/0.6.0
        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 getNode='qsub -I -q laser -lwalltime=00:30:00 -lnodes=1:ppn=64'

export PICSRC=/home/$(whoami)/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:bdver1"

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

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

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

Queue: k20 (Nvidia K20 GPUs)

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

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

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

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

        # Core Dependencies
        module load gcc/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/0.6.0
        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 getNode='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/$(whoami)/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"
```

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

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

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

Queue: k80 (Nvidia K80 GPUs)

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

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

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

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

        # Core Dependencies
        module load gcc/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/0.6.0
        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 getNode='qsub -I -q k80 -lwalltime=00:30:00 -lnodes=1:ppn=16'
alias getlaser='qsub -I -q laser -lwalltime=00:30:00 -lnodes=1:ppn=16'
```

```
export PICSRC=/home/$(whoami)/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37"

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

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

# "tbg" default options ########################################################
#   - PBS/Torque (qsub)
#   - "k80" queue
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="etc/picongpu/hypnos-hzdr/k80.tpl"
```

1.4.2 Hydra (HZDR)

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

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

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

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

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

        # Core Dependencies
        module load gcc/5.3.0
        module load cmake/3.7.2
        module load boost/1.62.0
        module load openmpi/1.8.6
        module load numactl

        # Plugins (optional)
        module load pngwriter/0.6.0
        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 getNode='qsub -I -q default -lwalltime=00:30:00 -lnodes=1:ppn=32'

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

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

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

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

1.4.3 Titan (ORNL)

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

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail 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>"

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

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

# 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/4.9.3

# 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/4.9.3/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 cmake3/3.9.0
module load cudatoolkit  # 7.5.18
# might fail to link with missing symbols:
#   C++11 module rebuild pending [CCS #366279]
module load boost/1.62.0
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" ~/picInputs/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 "cuda:35" \
#         -c "-DCMAKE_CXX_COMPILER=$(which scorep-CC) \
#         -DCUDA_NVCC_EXECUTABLE=$(which scorep-nvcc)" \
#         ~/picInputs/case001
#     export SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--cuda --mpp=mpi"
#     make -j
#     make install
#module load scorep

# plugins (optional) ###########################################################
module load cray-hdf5-parallel/1.8.14
module load adios/1.10.0
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=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
```

```
export PIC_BACKEND="cuda:35"

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

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

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

# "tbg" default options ################################################
export TBG_SUBMIT="qsub"
export TBG_TPLFILE="etc/picongpu/titan-ornl/batch.tpl"
```

1.4.4 Piz Daint (CSCS)

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

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

Note: For proper HDF5 detection, copy the `FindHDF5.cmake` of libSplash to PIConGPU:

```
cp $HOME/src/splash/cmake/FindHDF5.cmake $PICSRC/thirdParty/cmake-modules/
```

Note: Please find a Piz Daint quick start from October 2017 here.

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

# User Information ########################################## (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

# General Modules ##########################################################
#
# if the wrong environment is loaded we switch to the gnu environment
# note: this loads gcc/5.3.0 (6.0.4 is the version of the programming env!)
module li 2>&1 | grep "PrgEnv-cray" > /dev/null
if [ $? -eq 0 ] ; then
    module swap PrgEnv-cray PrgEnv-gnu/6.0.4
else
    module load PrgEnv-gnu/6.0.4
fi

module load CMake/3.8.1
```

```
module load cudatoolkit/8.0.61_2.4.3-6.0.4.0_3.1__gb475d12

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

# 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-0.6.0
export ADIOS_ROOT=$SCRATCH/lib/adios-1.11.1
export SPLASH_ROOT=$SCRATCH/lib/splash-1.6.0

export LD_LIBRARY_PATH=$PNGWRITER_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$ADIOS_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$SPLASH_ROOT/lib:$LD_LIBRARY_PATH

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

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

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

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

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

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

```
}
```

1.4.5 Taurus (TU Dresden)

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

Queue: gpu1 (Nvidia K20x GPUs)

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

# General modules #############################################################
#
module purge
module load oscar-modules
module load cmake/3.9.0
module load git
module load cuda/8.0.61 # gcc <= 5, intel 15-16
module load bullxmpi
module load gnuplot/4.6.1

# Compilers ###################################################################
### GCC
module load gcc/5.3.0
module load boost/1.64.0-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=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"

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

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

# "tbg" default options ########################################################
#   - SLURM (sbatch)
#   - "gpu1" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/k20x.tpl"
```

Queue: gpu2 (Nvidia K80 GPUs)

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

# General modules #############################################################
#
module purge
module load oscar-modules
module load cmake/3.9.0
module load git
module load cuda/8.0.61 # gcc <= 5, intel 15-16
module load bullxmpi
module load gnuplot/4.6.1

# Compilers ###################################################################
### GCC
module load gcc/5.3.0
module load boost/1.64.0-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=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37"

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

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

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

Queue: knl (Intel Intel Xeon Phi - Knights Landing)

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

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

# General modules #############################################################
#
module purge
module load cmake/3.9.0
module load git
module load intelmpi/2017.2.174

# Compilers ###################################################################
### ICC
module load intel/2017.2.174
# Boost needs to be build by yourself for now!
# The boost root path needs to be adjusted.
export BOOST_ROOT=$HOME/lib/boost-1.62.0
```

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

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

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

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

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

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

1.4.6 Lawrencium (LBNL)

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

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

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

        # Core Dependencies
        module load gcc
        module load cuda
        echo "WARNING: Boost version is too old! (Need: 1.62.0+)" >&2
        # module load boost/1.62.0-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=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:20"

# fix pic-create: re-enable rsync
#   ssh lrc-xfer.scs00
#   -> cp /usr/bin/rsync $HOME/bin/
export PATH=$HOME/bin:$PATH

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

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

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

1.4.7 Judge (FZJ)

(example missing)

1.4.8 Draco (MPCDF)

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

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

# User Information ############################################ (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: 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>"

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

# General Modules #############################################################
#
module purge

module load git/2.14
module load gcc/6.3
module load cmake/3.7
module load boost/gcc/1.64
module load impi/2017.3
module load hdf5-mpi/gcc/1.8.18

# Other Software ##############################################################
#
# needs to be compiled by the user
export PNGWRITER_ROOT=$HOME/lib/pngwriter-0.6.0
export SPLASH_ROOT=$HOME/lib/splash-1.6.0

export LD_LIBRARY_PATH=$PNGWRITER_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$SPLASH_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BOOST_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HDF5_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$I_MPI_ROOT/lib64:$LD_LIBRARY_PATH

export HDF5_ROOT=$HDF5_HOME

export CXX=$(which g++)
export CC=$(which gcc)

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

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

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

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

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

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

CHAPTER 2

Usage

2.1 Reference

Section author: Axel Huebl

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

2.1.1 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},
}
```

2.1.2 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.

2.2 Basics

Section author: Axel Huebl

2.2.1 Preparation

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

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

```
export SCRATCH=$HOME
```

We need a few directories to structure our workflow:

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

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

2.2.2 Step-by-Step

1. Create an Input (Parameter) Set

```
# clone the LWFA example to $HOME/picInputs/myLWFA
pic-create $PIC_EXAMPLES/LaserWakefield $HOME/picInputs/myLWFA

# switch to your input directory
cd $HOME/picInputs/myLWFA
```

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

Edit the *.param files* inside `include/picongpu/simulation_defines/param/`. Initially and when options are changed, PIConGPU *requires a re-compile*.

Now edit the *runtime (command line) arguments* in `etc/picongpu/*.cfg`. These options do *not* require a re-compile when changed (e.g. simulation size, number of GPUs, plugins, ...).

2. Compile Simulation

In our input, `.param` files are build directly into the PIConGPU binary for performance reasons. Changing or initially adding those requires a compile.

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

```
pic-build
```

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

3. Run Simulation

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

```
# example run for an interactive simulation on the same machine
tbg -s bash -c etc/picongpu/0001gpus.cfg -t etc/picongpu/bash/mpiexec.tpl $SCRATCH/
↪runs/lwfa_001
```

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

2.2.3 Further Reading

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

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

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

```
# request 16 GPUs from the PBS batch system and run on the queue k20
tbg -s qsub -c etc/picongpu/0016gpus.cfg -t etc/picongpu/hypnos-hzdr/k20.tpl
↪$SCRATCH/runs/lwfa_002
```

pic-create

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

See `pic-create --help` for more options during input 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
```

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

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

pic-build

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

In detail, it does:

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

# configure with CMake
pic-configure $OPTIONS ..

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

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

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

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

```
Build new binaries for a PIConGPU input set

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

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

You must run this command inside an input directory.

usage: pic-build [OPTIONS]

-b | --backend       - set compute backend and optionally the architecture
                       syntax: backend[:architecture]
                       supported backends: cuda, omp2b
                       (e.g.: "cuda:20;35;37;52;60" or "omp2b:native" or "omp2b")
                       default: "cuda" if not set via environment variable PIC_
↪BACKEND
                       note: architecture names are compiler dependent
-c | --cmake         - overwrite options for cmake
                       (e.g.: "-DPIC_VERBOSE=21 -DCMAKE_BUILD_TYPE=Debug")
-t <presetNumber>    - configure this preset from cmakeFlags
-h | --help          - show this help message
```

pic-configure

The tools is just a convenient wrapper for a call to CMake. It is executed from an empty build directory. You will likely not use this tool directly when using `pic-build` from above.

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

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

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

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

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

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

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

```
Configure PIConGPU with CMake

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

usage: pic-configure [OPTIONS] <inputDirectory>

-i | --install       - path were picongpu shall be installed
                       (default is <inputDirectory>)
-b | --backend       - set compute backend and optionally the architecture
                       syntax: backend[:architecture]
                       supported backends: cuda, omp2b
                       (e.g.: "cuda:20;35;37;52;60" or "omp2b:native" or "omp2b")
                       default: "cuda" if not set via environment variable PIC_
↪BACKEND
                       note: architecture names are compiler dependent
-c | --cmake         - overwrite options for cmake
                       (e.g.: "-DPIC_VERBOSE=21 -DCMAKE_BUILD_TYPE=Debug")
-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"`.

2.3 .param Files

Section author: Axel Huebl

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

2.3.1 Editing

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

```
# first switch to your input directory
cd $HOME/picInputs/myLWFA

pic-edit grid fileOutput laser
```

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

```
Edit compile-time options for a PIConGPU input set

Opens .param files in an input set with the default "EDITOR".
If a .param file is not yet part of the input set but exists in the
defaults, it will be transparently added to the input set.

You must run this command inside an input directory.

The currently selected editor is: /usr/bin/vim.basic
You can change it via the "EDITOR" environment variable.

usage: pic-edit <input>

Available <input>s:
bremsstrahlung components density dimension fieldBackground fieldSolver fileOutput␣
↪flylite grid ionizationEnergies ionizer isaac laser mallocMC memory particle␣
↪particleCalorimeter particleFilters particleMerger physicalConstants precision␣
↪pusher radiation radiationObserver seed species speciesAttributes␣
↪speciesConstants speciesDefinition speciesInitialization starter␣
↪synchrotronPhotons visColorScales visualization
```

2.3.2 Rationale

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

This comes at a small cost: when **even one of those settings is changed, you need to recompile**. Nevertheless, wasting about 5 minutes compiling on a single node is nothing compared to the time you save *at scale*!

All options that are less or non-critical for runtime performance, such as specific ranges observables in *plugins* or how many nodes shall be used, can be set in *run time configuration files (*.cfg)* and do not need a recompile when changed.

2.3.3 Files and Their Usage

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

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

2.3.4 All Files

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

PIC Core

dimension.param

The spatial dimensionality of the simulation.

Defines

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

namespace picongpu

Variables

constexpr uint32_t **simDim** = SIMDIM

grid.param

Definition of cell sizes and time step.

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

Units in reduced dimensions

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

A 2D3V simulation in a cartesian PIC simulation such 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

components.param

Select the laser profile and the field solver here.

Defines

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

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 `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`

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

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

namespace `particlePusherProbe`

Typedefs

using `picongpu::particlePusherProbe::ActualPusher = typedef void`
Also push the probe particles?

In many cases, probe particles are static throughout the simulation. This option allows to set an "actual" pusher that shall be used to also change the probe particle positions.

Examples:
- particles::pusher::Boris
- particles::pusher::[all others from above]
- void (no push)

density.param

Configure existing or define new normalized density profiles here.

During particle species creation in speciesInitialization.param, those profiles can be translated to spatial particle distributions.

namespace `picongpu`

namespace `densityProfiles`

Typedefs

using `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

speciesAttributes.param

This file defines available attributes that can be stored with each particle of a particle species.

Each attribute defined here needs to implement furthermore the traits

- Unit
- UnitDimension
- WeightingPower
- MacroWeighted in speciesAttributes.unitless for further information about these traits see therein.

namespace `picongpu`

Functions

`alias`(position)
relative (to cell origin) in-cell position of a particle

With this definition we do not define any type like float3_X, float3_64, ... This is only a name without a specialization.

`value_identifier`(uint64_t, particleId, IdProvider<*simDim*>::getNewId)
unique identifier for a particle

`picongpu::value_identifier(floatD_X, position_pic, floatD_X::create (0.))`
specialization for the relative in-cell position

`picongpu::value_identifier(float3_X, momentum, float3_X::create (0.))`
momentum at timestep t

`picongpu::value_identifier(float3_X, momentumPrev1, float3_X::create (0.))`
momentum at (previous) timestep t-1

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

`picongpu::value_identifier(int16_t, voronoiCellId, -1)`
Voronoi cell of the macro particle.

`picongpu::value_identifier(float3_X, probeE, float3_X::create (0.))`
interpolated electric field with respect to particle shape

`picongpu::value_identifier(float3_X, probeB, float3_X::create (0.))`
interpolated electric field with respect to particle shape

`picongpu::value_identifier(bool, radiationMask, false)`
masking a particle for radiation

The mask is used by the user defined filter `RadiationParticleFilter` in radiation.param to (de)select particles for the radiation calculation.

`picongpu::value_identifier(float_X, boundElectrons, float_X(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

`picongpu::value_identifier(flylite::Superconfig, superconfig, flylite::Superconfig:`
atomic superconfiguration

atomic configuration of an ion for collisional-radiative modeling, see also flylite.param

value_identifier (DataSpace<*simDim*>, totalCellIdx, DataSpace<*simDim*>)
Total cell index of a particle.

The total cell index is a N-dimensional DataSpace given by a GPU's `globalDomain.offset` + `localDomain.offset` added to the N-dimensional cell index the particle belongs to on that GPU.

alias (shape)
alias for particle shape, see also species.param

alias (particlePusher)
alias for particle pusher, see alsospecies.param

alias (ionizers)
alias for particle ionizers, see also ionizer.param

alias (ionizationEnergies)
alias for ionization energy container, see also ionizationEnergies.param

alias (synchrotronPhotons)
alias for synchrotronPhotons, see also speciesDefinition.param

alias for ion species used for bremsstrahlung

alias (bremsstrahlungPhotons)
alias for photon species used for bremsstrahlung

alias (interpolation)
alias for particle to field interpolation, see also species.param

alias (current)
alias for particle current solver, see also species.param

alias (atomicNumbers)
alias for particle flag: atomic numbers, see also ionizer.param

- only reasonable for atoms / ions / nuclei

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

- see also ionizer.param
- only reasonable for atoms / ions / nuclei

alias (populationKinetics)
alias for particle population kinetics model (e.g.

FLYlite)

see also flylite.param

alias (massRatio)
alias for particle mass ratio

mass ratio between base particle, see also speciesConstants.param `SI::BASE_MASS_SI` and a user defined species

default value: 1.0 if unset

alias (chargeRatio)
alias for particle charge ratio

charge ratio between base particle, see also speciesConstants.param `SI::BASE_CHARGE_SI` and a user defined species

default value: 1.0 if unset

alias (densityRatio)
alias for particle density ratio

density ratio between default density, see also density.param `SI::BASE_DENSITY_SI` and a user defined species

default value: 1.0 if unset

alias (exchangeMemCfg)
alias to reserved bytes for each communication direction

This is an optional flag and overwrites the default species configuration in memory.param.

A memory config must be of the following form:

```
struct ExampleExchangeMemCfg
{
    static constexpr uint32_t BYTES_EXCHANGE_X = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Y = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Z = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_CORNER = 16 * 1024;
    static constexpr uint32_t BYTES_EDGES = 16 * 1024;
};
```

The following species attributes are defined by pmacc and always stored with a particle:

namespace pmacc

Functions

pmacc::value_identifier(lcellId_t, localCellIdx, 0)
cell of a particle inside a supercell

Value is a linear cell index inside the supercell

pmacc::value_identifier(uint8_t, multiMask, 0)
state of a particle

Particle might be valid or invalid in a particle frame. Valid particles can further be marked as candidates to leave a supercell. Possible multiMask values are:

- 0 (zero): no particle (invalid)
- 1: particle (valid)
- 2 to 27: (valid) particle that is about to leave its supercell but is still stored in the current particle frame. Directions to leave the supercell are defined as follows. An ExchangeType = value - 1 (e.g. 27 - 1 = 26) means particle leaves supercell in the direction of FRONT(value=18) && TOP(value=6) && LEFT(value=2) which defines a diagonal movement over a supercell corner (18+6+2=26).

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

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 diagnostics & modeling: ———————————————

- particles::pusher::Free : free propagation, ignore fields (= free stream model)
- particles::pusher::Photon : propagate with c in direction of normalized mom.
- particles::pusher::Probe : Probe particles that interpolate E & B For development purposes: ———————————————
- particles::pusher::Axel : a pusher developed at HZDR during 2011 (testing)

speciesDefinition.param

Define particle species.

This file collects all previous declarations of base (reference) quantities and configured solvers for species and defines particle species. This includes "attributes" (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)
```

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`** = 2u
Number of maximum particles per cell during density profile evaluation.

Determines the weighting of a macro particle and with it, the number of particles "sampling" dynamics in phase space.

namespace `manipulators`

Typedefs

using `picongpu::particles::manipulators::AssignXDrift = typedef unary::Drift< D`
definition of manipulator that assigns a drift in X

using `picongpu::particles::manipulators::AddTemperature = typedef unary::Tempe`

using `picongpu::particles::manipulators::DoubleWeighting = typedef generic::Fre`
definition of a free particle manipulator: double weighting

using `picongpu::particles::manipulators::RandomEnabledRadiation = typedef gener`

using `picongpu::particles::manipulators::RandomPosition = typedef unary::Rando`
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 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< QuietParar
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

speciesInitialization.param

Available species functors in include/picongpu/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

using `picongpu::SuperCellSize = typedef typename mCT::shrinkTo< mCT::Int< 8, 8, 4 >,`
size of a superCell

volume of a superCell must be <= 1024

using `picongpu::MappingDesc = typedef MappingDescription<simDim, SuperCellSize >`
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 `fieldTmpNumSlots` = 1
number of scalar fields that are reserved as temporary fields

constexpr bool `fieldTmpSupportGatherCommunication` = true
can *FieldTmp* gather neighbor information

If `true` it is possible to call the method `asyncCommunicationGather()` to copy data from the border of neighboring GPU into the local guard. This is also known as building up a "ghost" or "halo" region in domain decomposition and only necessary for specific algorithms that extend the basic PIC cycle, e.g. with dependence on derived density or energy fields.

struct `DefaultExchangeMemCfg`
bytes reserved for species exchange buffer

This is the default configuration for species exchanges buffer sizes. The default exchange buffer sizes can be changed per species by adding the alias exchangeMemCfg with similar members like in DefaultExchangeMemCfg to its flag list.

Public Static Attributes

constexpr uint32_t `BYTES_EXCHANGE_X` = 1 * 1024 * 1024

constexpr uint32_t `BYTES_EXCHANGE_Y` = 3 * 1024 * 1024

constexpr uint32_t `BYTES_EXCHANGE_Z` = 1 * 1024 * 1024

constexpr uint32_t `BYTES_EDGES` = 32 * 1024

constexpr uint32_t `BYTES_CORNER` = 8 * 1024

precision.param

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 kinetic electron energy in MeV for the lookup table.

For electrons below this value Bremsstrahlung is not taken into account.

constexpr float_64 **MAX_ENERGY_MeV** = 200.0
Maximal kinetic electron energy in MeV for the lookup table.

Electrons above this value cause a out-of-bounds access at the lookup table. Bounds checking is enabled for "CRITICAL" log level.

constexpr float_64 **MIN_THETA** = 0.01
Minimal polar deflection angle due to screening.

See Jackson 13.5 for a rule of thumb to this value.

constexpr uint32_t **NUM_SAMPLES_KAPPA** = 32
number of lookup table divisions for the kappa axis.

Kappa is the energy loss normalized to the initial kinetic energy. The axis is scaled linearly.

constexpr uint32_t **NUM_SAMPLES_EKIN** = 32
number of lookup table divisions for the initial kinetic energy axis.

The axis is scaled logarithmically.

constexpr float_64 **MIN_KAPPA** = 1.0e-10
Kappa is the energy loss normalized to the initial kinetic energy.

This minimal value is needed by the numerics to avoid a division by zero.

namespace photon
params related to the creation and the emission angle of the photon

Variables

constexpr float_64 **SOFT_PHOTONS_CUTOFF_keV** = 5000.0
Low-energy threshold in keV of the incident electron for the creation of photons.

Below this value photon emission is neglected.

constexpr uint32_t **NUM_SAMPLES_DELTA** = 256
number of lookup table divisions for the delta axis.

Delta is the angular emission probability (normalized to one) integrated from zero to theta, where theta is the angle between the photon momentum and the final electron momentum.

The axis is scaled linearly.

constexpr uint32_t **NUM_SAMPLES_GAMMA** = 64
number of lookup table divisions for the gamma axis.

Gamma is the relativistic factor of the incident electron.

The axis is scaled logarithmically.

constexpr float_64 **MAX_DELTA** = 0.95
Maximal value of delta for the lookup table.

Delta is the angular emission probability (normalized to one) integrated from zero to theta, where theta is the angle between the photon momentum and the final electron momentum.

A value close to one is reasonable. Though exactly one was actually correct, because it would map to theta = pi (maximum polar angle), the sampling then would be bad in the ultrarelativistic case. In this regime the emission primarily takes place at small thetas. So a maximum delta close to one maps to a reasonable maximum theta.

constexpr float_64 **MIN_GAMMA** = 1.0
minimal gamma for the lookup table.

constexpr float_64 **MAX_GAMMA** = 250
maximal gamma for the lookup table.

Bounds checking is enabled for "CRITICAL" log level.

constexpr float_64 **SINGLE_EMISSION_PROB_LIMIT** = 0.4
if the emission probability per timestep is higher than this value and the log level is set to "CRITICAL" a warning will be raised.

constexpr float_64 **WEIGHTING_RATIO** = 10

synchrotronPhotons.param

Defines

ENABLE_SYNCHROTRON_PHOTONS
enable synchrotron photon emission

namespace picongpu

namespace particles

namespace synchrotronPhotons

Variables

constexpr bool **enableQEDTerm** = false
enable (disable) QED (classical) photon emission spectrum

constexpr float_64 **SYNC_FUNCS_CUTOFF** = 5.0
Above this value (to the power of three, see comments on mapping) the synchrotron functions are nearly zero.

constexpr float_64 **SYNC_FUNCS_BESSEL_INTEGRAL_STEPWIDTH** = 1.0e-3
stepwidth for the numerical integration of the bessel function for the first synchrotron function

constexpr uint32_t **SYNC_FUNCS_NUM_SAMPLES** = 8192
Number of sampling points of the lookup table.

constexpr float_64 **SOFT_PHOTONS_CUTOFF_RATIO** = 1.0
Photons of oscillation periods greater than a timestep are not created since the grid already accounts for them.

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

constexpr float_64 **SINGLE_EMISSION_PROB_LIMIT** = 0.4
if the emission probability per timestep is higher than this value and the log level is set to "CRITICAL" a warning will be raised.

ionizer.param

This file contains the proton and neutron numbers of commonly used elements of the periodic table.

The elements here should have a matching list of ionization energies in Furthermore there are parameters for specific ionization models to be found here. That includes lists of screened nuclear charges as seen by bound electrons for the aforementioned elements as well as fitting parameters of the Thomas-Fermi ionization model.

See ionizationEnergies.param. Moreover this file contains a description of how to configure an ionization model for a species. Currently each species can only be assigned exactly one ionization model.

namespace picongpu

namespace ionization
Ionization Model Configuration.

- None : no particle is ionized
- BSI : simple barrier suppression ionization
- BSIEffectiveZ : BSI taking electron shielding into account via an effective atomic number Z_eff
- ADKLinPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> linearly polarized lasers
- ADKCircPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> circularly polarized lasers
- Keldysh : Keldysh ionization model
- ThomasFermi : statistical impact ionization based on Thomas-Fermi atomic model Attention: requires 2 *FieldTmp* slots Research and development:

  See memory.param
- BSIStarkShifted : BSI for hydrogen-like atoms and ions considering the Stark upshift of ionization potentials

Usage: Add flags to the list of particle flags that has the following structure

```
  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

constexpr float_X **CUTOFF_MAX_ENERGY_KEV** = 50.0
cutoff energy for electron "temperature" calculation

In laser produced plasmas we can have different, well-separable groups of electrons. For the Thomas-Fermi ionization model we only want the thermalized "bulk" electrons. Including the high-energy "prompt" electrons is physically questionable since they do not have a large cross section for collisional ionization.

unit: keV

constexpr float_X **CUTOFF_MAX_ENERGY** = CUTOFF_MAX_ENERGY_KEV * UNITCONV_keV_to_Jou
cutoff energy for electron "temperature" calculation in SI units

constexpr float_X **CUTOFF_LOW_DENSITY** = 1.7422e27
lower density cutoff

The Thomas-Fermi model yields unphysical artifacts for low ion densities. Low ion densities imply lower collision frequency and thus less collisional ionization. The Thomas-Fermi model yields an increasing charge state for decreasing densities and electron temperatures of 10eV and above. This cutoff will be used to set the lower application threshold for charge state calculation.

Note This cutoff value should be set in accordance to FLYCHK calculations, for instance! It is not a universal value and requires some preliminary approximations!

ionizationEnergies.param

This file contains the ionization energies of commonly used elements of the periodic table.

Each atomic species in PIConGPU can represent exactly one element. The ionization energies of that element are stored in a vector which contains the *name* and *proton number* as well as a list of *energy values*. The number of ionization levels must be equal to the proton number of the element.

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 include/pmacc/math/ConstVector.hpp for finding ionization energies, http://physics.nist.gov/PhysRefData/ASD/ionEnergy.html

namespace AU

Functions

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

flylite.param

This is the configuration file for the atomic particle population kinetics model FLYlite.

Its main purpose is non-LTE collisional-radiative modeling for transient plasmas at high densities and/or interaction with (X-Ray) photon fields.

In simpler words, one can also use this module to simulate collisional ionization processes without the assumption of a local thermal equilibrium (LTE), contrary to popular collisional ionization models such as the Thomas-Fermi ionization model.

This file configures the number of modeled populations for ions, spatial and spectral binning of non-LTE density and energy histograms.

namespace `picongpu`

namespace `flylite`

Typedefs

using `picongpu::flylite::Superconfig = typedef types::Superconfig< float_64, popul`

using `picongpu::flylite::spatialAverageBox = typedef SuperCellSize`
you better not change this line, the wooooorld depends on it!

no seriously, per-supercell is the quickest way to average particle quantities such as density, energy histogram, etc. and I won't implement another size until needed

Variables

constexpr uint8_t **`populations`** = 3u
number of populations (numpop)

this number defines how many configurations make up a superconfiguration

range: [0, 255]

constexpr uint8_t **`ionizationStates`** = 29u
ionization states of the atom (iz)

range: [0, 255]

constexpr uint16_t **`energies`** = 512u
number of energy bins

energy steps used for local energy histograms
Note : no overflow- or underflow-bins are used, particles with energies outside the range (see below) are ignored

constexpr float_X **`electronMinEnergy`** = 0.0
energy range for electron and photon histograms

electron and photon histograms f(e) f(ph) are currently calculated in a linearly binned histogram while particles with energies outside the ranges below are ignored

unit: eV

constexpr float_X **`electronMaxEnergy`** = 100.e3

constexpr float_X **`photonMinEnergy`** = 0.0

constexpr float_X **`photonMaxEnergy`** = 100.e3

Plugins

fileOutput.param

namespace `picongpu`

Typedefs

using picongpu::ChargeDensity_Seq = typedef deriveField::CreateEligible_t< VectorAllS
FieldTmp output (calculated at runtime) ******************************.

Those operations derive scalar field quantities from particle species at runtime. Each value is mapped per cell. Some operations are identical up to a constant, so avoid writing those twice to save storage.

you can choose any of these particle to grid projections:

- Density: particle position + shape on the grid
- ChargeDensity: density * charge note: for species that do not change their charge state, this is the same as the density times a constant for the charge
- Energy: sum of kinetic particle energy per cell with respect to shape
- EnergyDensity: average kinetic particle energy per cell times the particle density note: this is the same as the sum of kinetic particle energy divided by a constant for the cell volume
- MomentumComponent: ratio between a selected momentum component and the absolute momentum with respect to shape
- LarmorPower: radiated Larmor power (species must contain the attribute `momentumPrev1`)

for debugging:

- MidCurrentDensityComponent: density * charge * velocity_component
- Counter: counts point like particles per cell
- MacroCounter: counts point like macro particles per cell

using picongpu::EnergyDensity_Seq = typedef deriveField::CreateEligible_t< VectorAllS

using picongpu::MomentumComponent_Seq = typedef deriveField::CreateEligible_t< Vector

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 using FileOutputParticles = bmpl::vector0< >;

isaac.param

Definition which native fields and density fields of particles will be visualizable with ISAAC.

ISAAC is an in-situ visualization library with which the PIC simulation can be observed while it is running avoiding the time consuming writing and reading of simulation data for the classical post processing of data.

ISAAC can directly visualize natives fields like the E or B field, but density fields of particles need to be calculated from PIConGPU on the fly which slightly increases the runtime and the memory consumption. Every particle density field will reduce the amount of memory left for PIConGPUs particles and fields.

To get best performance, ISAAC defines an exponential amount of different visualization kernels for every combination of (at runtime) activated fields. So furthermore a lot of fields will increase the compilation time.

namespace picongpu

namespace isaacP

Typedefs

using picongpu::isaacP::Native_Seq = typedef MakeSeq_t< FieldE, FieldB, FieldJ >
Intermediate list of native fields of PIConGPU which shall be visualized.

using picongpu::isaacP::Density_Seq = typedef deriveField::CreateEligible_t< Vecto
Intermediate list of particle species, from which density fields shall be created at runtime to visualize them.

using picongpu::isaacP::Fields_Seq = typedef MakeSeq_t< Native_Seq, Density_Seq >
Compile time sequence of all fields which shall be visualized.

Basically the join of Native_Seq and Density_Seq.

particleCalorimeter.param

namespace picongpu

namespace particleCalorimeter

Functions

HDINLINE float2_X picongpu::particleCalorimeter::mapYawPitchToNormedRange(const
Map yaw and pitch into [0,1] respectively.

These ranges correspond to the normalized histogram range of the calorimeter (0: first bin, 1: last bin). Out-of-range values are mapped to the first or the last bin.

Useful for fine tuning the spatial calorimeter resolution.

Return Two values within [-1,1]
Parameters
- `yaw`: -maxYaw...maxYaw
- `pitch`: -maxPitch...maxPitch
- `maxYaw`: maximum value of angle yaw
- `maxPitch`: maximum value of angle pitch

particleMerger.param

namespace picongpu

namespace plugins

namespace particleMerging

Variables

constexpr size_t **MAX_VORONOI_CELLS** = 128
maximum number of active Voronoi cells per supercell.

If the number of active Voronoi cells reaches this limit merging events are dropped.

radiation.param

Definition of frequency space, number of observers, filters, form factors and window functions of the radiation plugin.

All values set here determine what the radiation plugin will compute. The observation direction is defined in a seperate file `radiationObserver.param`. On the comand line the plugin still needs to be called for each species the radiation should be computed for.

Defines

PIC_VERBOSE_RADIATION
radiation verbose level: 0=nothing, 1=physics, 2=simulation_state, 4=memory, 8=critical

namespace picongpu

namespace parameters

Variables

constexpr unsigned int **N_observer** = 256
number of observation directions

namespace rad_frequencies_from_list

Variables

constexpr char **listLocation**[] = "/path/to/frequency_list"
path to text file with frequencies

constexpr unsigned int **N_omega** = 2048
number of frequency values to compute if frequencies are given in a file [unitless]

namespace rad_linear_frequencies

Variables

constexpr unsigned int **N_omega** = 2048
number of frequency values to compute in the linear frequency [unitless]

namespace SI

Variables

constexpr float_64 **omega_min** = 0.0
mimimum frequency of the linear frequency scale in units of [1/s]

constexpr float_64 **omega_max** = 1.06e16
maximum frequency of the linear frequency scale in units of [1/s]

namespace rad_log_frequencies

Variables

constexpr unsigned int **N_omega** = 2048
number of frequency values to compute in the logarithmic frequency [unitless]

namespace SI

Variables

constexpr float_64 **omega_min** = 1.0e14
mimimum frequency of the logarithmic frequency scale in units of [1/s]

constexpr float_64 **omega_max** = 1.0e17
maximum frequency of the logarithmic frequency scale in units of [1/s]

namespace radFormFactor_CIC_3D
correct treatment of coherent and incoherent radiation from macro particles

Choose different form factors in order to consider different particle shapes for radiation

- radFormFactor_CIC_3D ... CIC charge distribution
- radFormFactor_TSC_3D ... TSC charge distribution
- radFormFactor_PCS_3D ... PCS charge distribution
- radFormFactor_CIC_1Dy ... only CIC charge distribution in y
- radFormFactor_Gauss_spherical ... symmetric Gauss charge distribution
- radFormFactor_Gauss_cell ... Gauss charge distribution according to cell size
- radFormFactor_incoherent ... only incoherent radiation
- radFormFactor_coherent ... only coherent radiation

namespace radiation

Typedefs

using picongpu::radiation::RadiationParticleFilter = typedef picongpu::particles::
filter to (de)select particles for the radiation calculation

to activate the filter:
- goto file `speciesDefinition.param`
- add the attribute `radiationMask` to the particle species

struct GammaFilterFunctor
select particles for radiation example of a filter for the relativistic Lorentz factor gamma

Public Functions

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

Public Static Attributes

constexpr float_X **radiationGamma** = 5.0
Gamma value above which the radiation is calculated.

namespace `radiationNyquist`
selected mode of frequency scaling:

options:

- rad_linear_frequencies
- rad_log_frequencies
- rad_frequencies_from_list

Variables

constexpr float_32 **`NyquistFactor`** = 0.5
Nyquist factor: fraction of the local Nyquist frequency above which the spectra is set to zero should be in (0, 1).

namespace `radWindowFunctionTriangle`
add a window function weighting to the radiation in order to avoid ringing effects from sharpe boundaries default: no window function via `radWindowFunctionNone`

Choose different window function in order to get better ringing reduction radWindowFunctionTriangle radWindowFunctionHamming radWindowFunctionTriplett radWindowFunctionGauss radWindowFunctionNone

radiationObserver.param

This file defines a function describing the observation directions.

It takes an integer index from [0, picongpu::parameters::N_observer) and maps it to a 3D unit vector in R^3 (norm=1) space that describes the observation direction in the PIConGPU cartesian coordinate system.

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

This default setup is an example of a 2D detector array. It computes observation directions for 2D virtual detector field with its center pointing toward the +y direction (for theta=0, phi=0) with observation angles ranging from theta = [angle_theta_start : angle_theta_end] phi = [angle_phi_start : angle_phi_end] Every observation_id_extern index moves the phi angle from its start value toward its end value until the observation_id_extern reaches N_split. After that the theta angle moves further from its start value towards its end value while phi is reset to its start value.

The unit vector pointing towards the observing virtual detector can be described using theta and phi by: x_value = sin(theta) * cos(phi) y_value = cos(theta) z_value = sin(theta) * sin(phi) These are the standard spherical coordinates.

The example setup describes an detector array of 16x16 detectors ranging from -pi/8= -22.5 degrees to +pi/8= +22.5 degrees for both angles with the center pointing toward the y-axis (laser propagation direction).

Return unit vector pointing in observation direction type: vector_64
Parameters

- `observation_id_extern`: int index that identifies each block on the GPU to compute the observation direction

visualization.param

Defines

EM_FIELD_SCALE_CHANNEL1

EM_FIELD_SCALE_CHANNEL2

EM_FIELD_SCALE_CHANNEL3

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

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

2.4 Particles

2.4.1 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, **typename** T_Filter = *filter::All*>
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
- `T_Filter`: picongpu::particles::filter, particle filter type to select particles

Inherits from *picongpu::particles::ManipulateDeriveSpecies< manipulators::generic::None, T_SrcSpeciesType, T_DestSpeciesType, T_Filter >*

Manipulate

template <**typename** T_Functor, **typename** T_SpeciesType = bmpl::_1, **typename** T_Filter = *filter::All*>
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
  - `T_Filter`: picongpu::particles::filter, particle filter type to select particles

ManipulateDeriveSpecies

template <typename *T_Functor*, typename *T_SrcSpeciesType*, typename *T_DestSpeciesType* = bmpl::_1, typename *T_Filter* = *filte*
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 include/picongpu/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
- `T_Filter`: picongpu::particles::filter, particle filter type to select particles

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

2.4.2 Manipulation Functors

Some of the particle operations above can take the following functors as arguments to manipulate attributes of particle species. A particle filter (see following section) is used to only manipulated selected particles of a species with a functor.

Free

template <typename T_Functor>
struct `picongpu::particles::manipulators::generic::`**`Free`**
call simple free user defined functor

Template Parameters

- `T_Functor`: user defined functor **optional**: can implement **one** host side constructor `T_Functor()` or `T_Functor(uint32_t currentTimeStep)`

Inherits from T_Functor

FreeRng

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

example: add

```
#include <pmacc/nvidia/rng/distributions/Uniform_float.hpp>

struct RandomXFunctor
{
    template< typename T_Rng, typename T_Particle >
    HDINLINE void operator()( T_Rng& rng, T_Particle& particle )
    {
        particle[ position_ ].x() = rng();
    }
};

using RandomXPos = FreeRng<
```

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

to `InitPipeline` in `speciesInitialization.param`

Template Parameters

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

Inherits from T_Functor, picongpu::particles::manipulators::generic::detail::Rng< T_Distribution, T_Seed, T_SpeciesType >

CopyAttribute

using picongpu::particles::manipulators::unary::CopyAttribute = typedef generic::Free< ac

copy a particle source attribute to a destination attribute

This is an unary functor and operates on one particle.

Template Parameters

- `T_DestAttribute`: type of the destination attribute e.g. `momentumPrev1`
- `T_SrcAttribute`: type of the source attribute e.g. `momentum`

Drift

using picongpu::particles::manipulators::unary::Drift = typedef generic::Free< acc::Drift

change particle's momentum based on speed

allow to manipulate a speed to a particle

Template Parameters

- `T_ParamClass`: param::DriftCfg, configuration parameter
- `T_ValueFunctor`: pmacc::nvidia::functors::*, binary functor type to manipulate the momentum attribute

RandomPosition

using picongpu::particles::manipulators::unary::RandomPosition = typedef generic::FreeRng

Change the in cell position.

This functor changes the in-cell position of a particle. The new in-cell position is uniformly distributed position between [0.0;1.0).

example: add

```
particles::Manipulate<RandomPosition,SPECIES_NAME>
```

to `InitPipeline` in `speciesInitialization.param`

Temperature

using picongpu::particles::manipulators::unary::Temperature = typedef generic::FreeRng< ac
change particle's momentum based on a temperature

allow to change the temperature (randomly normal distributed) of a particle.

Template Parameters

- `T_ParamClass`: param::TemperatureCfg, configuration parameter
- `T_ValueFunctor`: pmacc::nvidia::functors::*, binary functor type to manipulate the momentum attribute

Assign

using picongpu::particles::manipulators::binary::Assign = typedef generic::Free< acc::Assi
assign attributes of one particle to another

Can be used as binary and higher order operator but only the first two particles are used for the assign operation.

Assign all matching attributes of a source particle to the destination particle. Attributes that only exist in the destination species are initialized with the default value. Attributes that only exists in the source particle will be ignored.

DensityWeighting

using picongpu::particles::manipulators::binary::DensityWeighting = typedef generic::Free<
Re-scale the weighting of a cloned species by densityRatio.

When deriving species from each other, the new species "inherits" the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species' macro particles to satisfy the input densityRatio of it.

note: needs the densityRatio flag on both species, used by the GetDensityRatio trait.

ProtonTimesWeighting

using picongpu::particles::manipulators::binary::ProtonTimesWeighting = typedef generic::I
Re-scale the weighting of a cloned species by numberOfProtons.

When deriving species from each other, the new species "inherits" the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species' macro particles to be a multiplied by the number of protons of the initial species.

As an example, this is useful when initializing a quasi-neutral, pre-ionized plasma of ions and electrons. Electrons can be created from ions via deriving and increasing their weight to avoid simulating multiple macro electrons per macro ion (with Z>1).

note: needs the atomicNumbers flag on the initial species, used by the GetAtomicNumbers trait.

2.4.3 Manipulation Filters

Most of the particle functors shall operate on all valid particles, where filter::All is the default assumption. One can limit the domain or subset of particles with filters such as the ones below (or define new ones).

All

struct `picongpu::particles::filter::`**`All`**

RelativeGlobalDomainPosition

template <typename T_Params**>**
struct `picongpu::particles::filter::`**`RelativeGlobalDomainPosition`**

filter particle dependent on the global position

Check if a particle is within a relative area in one direction of the global domain.

Template Parameters

- `T_Params`: picongpu::particles::filter::param::RelativeGlobalDomainPosition, parameter to configure the functor

Define a New Particle Filter

Note: Not yet implemented.

2.5 Plugins

Plugin name	short description
ADIOS[2]	stores simulation data as ADIOS files
energy histogram	energy histograms for electrons and ions
charge conservation	maximum difference between electron charge density and div E
checkpoint[2]	stores the primary data of the simulation for restarts.
count particles	count total number of macro particles
count per supercell[3]	count macro particles *per supercell*
energy fields	electromagnetic field energy per time step
energy particles	kinetic and total energies summed over all electrons and/or ions
HDF5[2]	stores simulation data as libSplash-flavoured HDF5 files
ISAAC	interactive 3D live visualization
intensity[15]	maximum and integrated electric field along the y-direction
particle calorimeter[34]	spatially resolved, particle energy detector in infinite distance
particle merger	macro particle merging
phase space[3]	calculate 2D phase space
PNG	pictures of 2D slices
positions particles[15]	save trajectory, momentum, ... of a single particle
radiation[3]	compute emitted electromagnetic spectra
resource log	monitor used hardware resources & memory
slice field printer[5]	print out a slice of the electric and/or magnetic and/or current field
sum currents	compute the total current summed over all cells

[2] Either *ADIOS* or *HDF5* is required for simulation restarts. If both are available, writing checkpoints with ADIOS is automatically preferred by the simulation.

[3] Requires *HDF5* for output.

[1] Restart: Plugins with that flag overwrite their output of previous runs. *Save* the created files of these plugins before restarting in the same directory.

[5] Deprecated

[4] Can remember particles that left the box at a certain time step.

2.5.1 ADIOS

Stores simulation data such as fields and particles as ADIOS files or ADIOS staging methods.

External Dependencies

The plugin is available as soon as the *ADIOS library* is compiled in.

.param file

The corresponding `.param` file is *fileOutput.param*.

One can e.g. disable the output of particles by setting:

```
/* output all species */
using FileOutputParticles = VectorAllSpecies;
/* disable */
using FileOutputParticles = bmpl::vector0< >;
```

.cfg file

You can use `--adios.period` and `--adios.file` to specify the output period and path and name of the created fileset. For example, `--adios.period 128 --adios.file simData` will write the simulation data to files of the form `simData_0.bp`, `simData_128.bp` in the default simulation output directory every 128 steps. Note that this plugin will only be available if ADIOS is found during compile configuration.

PIConGPU command line option	description
`--adios.period`	Period after which simulation data should be stored on disk. Default is `0`, which means that no data is stored.
`--adios.file`	Relative or absolute fileset prefix for simulation data. If relative, files are stored under `simOutput`. Default is `simDataAdios`.
`--adios.compression`	Set data transform compression method. See `adios_config -m` for which compression methods are available. This flag also influences compression for checkpoints.
`--adios.aggregators`	Set number of I/O aggregator nodes for ADIOS `MPI_AGGREGATE` transport method.
`--adios.ost`	Set number of I/O OSTs for ADIOS `MPI_AGGREGATE` transport method.

Additional Tools

See our *openPMD* chapter.

2.5.2 Charge Conservation

First the charge density of all species with respect to their shape function is computed. Then this charge density is compared to the charge density computed from the divergence of the electric field $\nabla \vec{E}$. The maximum deviation value multiplied by the cell's volume is printed.

Attention: This plugin assumes a Yee-like divergence E stencil! Do not use it together with other field solvers like *directional splitting* (for the *Lehe* solver it is still correct).

.cfg file

PIConGPU command line argument (for `.cfg` files):

```
--chargeConservation.period <periodOfSteps>
```

Output and Analysis Tools

A new file named `chargeConservation.dat` is generated:

```
#timestep max-charge-deviation unit[As]
0 7.59718e-06 5.23234e-17
100 8.99187e-05 5.23234e-17
200 0.000113926 5.23234e-17
300 0.00014836 5.23234e-17
400 0.000154502 5.23234e-17
500 0.000164952 5.23234e-17
```

The charge is normalized to `UNIT_CHARGE` (third column) which is the typical charge of *one* macro-particle.

There is a up 5% difference to a native hdf5 post-processing based implementation of the charge conversation check due to a different order of subtraction. And the zero-th time step (only numerical differences) might differ more then 5% relative due to the close to zero result.

2.5.3 Checkpoint

Stores the primary data of the simulation for restarts. Primary data includes:

- electro-magnetic fields
- particle attributes
- state of random number generators and particle ID generator
- ...

Note: Some plugins have their own internal state. They will be notified on checkpoints to store their state themselves.

What is the format of the created files?

We write our fields and particles in an open markup called *openPMD*.

For further details, see the according sections in *HDF5* and *ADIOS*.

External Dependencies

The plugin is available as soon as the *libSplash (HDF5) or ADIOS libraries* are compiled in.

.cfg file

You can use `--checkpoint.period` to specify the output period of the created checkpoints. Note that this plugin will only be available if libSplash (HDF5) or ADIOS is found during compile configuration.

PIConGPU command line option	Description
`--checkpoint.period <N>`	Create checkpoints every N steps.
`--checkpoint.backend <IO-backend>`	IO-backend used to create the checkpoint.
`--checkpoint.file <string>`	Relative or absolute fileset prefix for writing checkpoints. If relative, checkpoint files are stored under `simOutput/<checkpoint-directory>`. Default depends on the selected IO-backend.
`--checkpoint.restart`	Restart a simulation from the latest checkpoint.
`--checkpoint.restart.step <N>`	Select a specific restart checkpoint.
`--checkpoint.restart.backend <IO-backend>`	IO-backend used to load a existent checkpoint.
`--checkpoint.restart.file <string>`	Relative or absolute fileset prefix for reading checkpoints. If relative, checkpoint files are searched under `simOutput/<checkpoint-directory>`. Default depends on the selected IO-backend‘‘.
`--checkpoint.restart.chunkSize <N>`	Number of particles processed in one kernel call during restart to prevent frame count blowup.
`--checkpoint.<IO-backend>.*`	Additional options to control the IO-backend

Depending on the available external dependencies (see above), the options for the `<IO-backend>` are:

- `hdf5`
- `adios`

Interacting Manually with Checkpoint Data

Note: Interacting with the *raw data of checkpoints* for manual manipulation is considered an advanced feature for experienced users.

Contrary to regular output, checkpoints contain additional data which might be confusing on the first glance. For example, some comments might be missing, all data from our concept of slides for moving window simulations will be visible, additional data for internal states of helper classes is stored as well and index tables such as openPMD particle patches are essential for parallel restarts.

2.5.4 Count Particles

This plugin counts the total number of *macro particles associated with a species* and writes them to a file for specified time steps. It is used mainly for debugging purposes. Only in case of constant particle density, where each macro particle describes the same number of real particles (weighting), conclusions on the plasma density can be drawn.

.cfg file

The *CountParticles* plugin is always complied for all species. By specifying the perodicity of the output using the comand line argument `--e_macroParticlesCount.period` (here for an electron species called `e`) with

picongpu, the plugin is enabled. Setting `--e_macroParticlesCount.period 100` adds the number of all electron like macro particles to the file *ElectronsCount.dat* for every 100th time step of the simulation.

Output

In the output file `e_macroParticlesCount.dat`, there are three columns. The first is the integer number of the time step. The second is the number of macro particles as integer - useful for exact counts. And the third is the number of macro particles in scintific floating point notation - provides better human readability.

Known Issues

Currently, the file `e_macroParticlesCount.dat` is overwritten when restarting the simulation. Therefore, all previously stored counts are lost.

2.5.5 Count per Supercell

This plugin counts the total number of *macro particles of a species* for each super cell and sstores the result in an hdf5 file. Only in case of constant particle density, where each macro particle describes the same number of real particles (weighting), conclusions on the plasma density can be drawn.

External Dependencies

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

.cfg files

By specifying the perodicity of the output using the comand line argument `--e_macroParticlesPerSuperCell.period` (here for an electron species *e*) with picongpu, the plugin is enabled. Setting `--e_macroParticlesPerSuperCell.period 100` adds the number of all electron like macro particles to the file `e_macroParticlesCount.dat` for every 100th time step of the simulation.

Output

The output is stores as hdf5 file in a separate directory.

2.5.6 Energy Fields

This plugin computes the total energy contained in the electric and magnetic field of the entire volume simulated. The energy is computed for user specified time steps.

.cfg file

By setting the PIConGPU command line flag `--fields_energy.period` to a non-zero value the plugin computes the total field energy. The default value is `0`, meaning that the total field energy is not stored. By setting e.g. `--fields_energy.period 100` the total field energy is computed for time steps *0, 100, 200, ...*.

Output

The data is stored in `fields_energy.dat`. There are two columns. The first gives the time step. The second is the total field energy in **Joule**. The first row is a comment describing the columns:

```
#step Joule
0     0.0e0
100   1.2.e-3
```

Example Visualization

Python example snippet:

```
import numpy as np

simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]
# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot(
    e_sum_ene[:,0],
    e_sum_ene[:,1] + p_sum_ene[:,1] + C_sum_ene[:,1] + N_sum_ene[:,1] + fields_sum_
↪ene[:,1],
    label="sum"
)
plt.legend()
```

2.5.7 Energy Histogram

This plugin computes the energy histogram particle species simulated with PIConGPU. The energy binning can be set up using command line parameters. A *virtual detector* can be located in the y direction of the simulation to count only particles that would reach such a detector in an experimental setup. If the detector setup is not specified, all particles of a species are considered in the histogram.

.cfg files

There are several command line parameters that can be used to set up this plugin. Replace the prefix `e` for electrons with any other species you have defined, we keep using `e` in the examples below for simplicity. Currently, the plugin can be set *once for each species*.

PIConGPU command line option	description
`--e_energyHistogram.period`	The ouput periodicity of the **electron** histogram. A value of `100` would mean aoutput at simulation time step *0, 100, 200, ...* If set to a non-zero value, the energy histogram of all **electrons** is computed. By default, the value is `0` and no histogram for the electrons is computed.
`--e_energyHistogram.binCount`	Specifies the number of bins used for the **electron** histogram. Default is `1024`.
`--e_energyHistogram.minEnergy`	Set the minimum energy for the **electron** histogram in *keV*. Default is `0`, meaning *0 keV*.
`--e_energyHistogram.maxEnergy`	Set the maximum energy for the **electron** histogram in *keV*. There is **no default value**. This has to be set by the user if `--e_energyHistogram.period 1` is set.
`--e_energyHistogram.distanceToDetector`	Distance in *meter* of a **electron** detector located far away in y direction with slit opening in x and z direction. If set to *non-zero* value, only particles that would reach the detector are considered in the histogram. Default `0`, meaning all particles are considered in the **electron** histogram and no detector is assumed.
`--e_energyHistogram.slitDetectorX`	Width of the **electron** detector in *meter*. If not set, all particles are counted.
`--e_energyHistogram.slitDetectorZ`	Hight of the **electron** detector in *meter*. If not set, all particles are counted.

Output

The histograms are stored in ASCII files in the `simOutput/` directory.

The file for the electron histogram is named `e_energyHistogram.dat` and for all other species `<species>_energyHistogram.dat` likewise. The first line of these files does not contain histogram data and is commented-out using `#`. It describes the energy binning that needed to interpret the following data. It can be seen as the head of the following data table. The first column is an integer value describing the simulation time step. The second column counts the number of real particles below the minimum energy value used for the histogram. The following columns give the real electron count of the particles in the specific bin described by the first line/header. The second last column gives the number of real particles that have a higher energy than the maximum energy used for the histogram. The last column gives the total number of particles. In total there are 4 columns more than the number of bins specified with command line arguments. Each row describes another simulation time step.

Analysis Tools

You can quickly plot the data in Python with:

```
from picongpu.plugins.energy_histogram import EnergyHistogram
import matplotlib.pyplot as plt

energy_histogram = EnergyHistogram('/home/axel/runs/lwfa_001')
energies, bins = energy_histogram.get('e', iteration=2000)

plt.plot( bins, energies )
plt.show()
```

Alternatively, PIConGPU comes with a command line analysis tool for the energy histograms. It is based on *gnuplot* and requires that gnuplot is available via command line. The tool can be found in `src/tools/bin/` and is called `BinEnergyPlot.sh`. It accesses the gnuplot script `BinEnergyPlot.gnuplot` in `src/tools/share/gnuplot/`. `BinEnergyPlot.sh` requires exactly three command line arguments:

Argument	Value
1st	Path and filename to *e_energyHistogram.dat* file.
2nd	Simulation time step (needs to exist)
3rd	Label for particle count used in the graph that this tool produces.

2.5.8 Energy Particles

This plugin computes the **kinetic and total energy summed over all particles** of a species for time steps specified.

.cfg file

Only the time steps at which the total kinetic energy of all particles should be specified needs to be set via command line argument.

PIConGPU command line option	Description
`--e_energy.period 100`	Sets the time step period at which the energy of all **electrons** in the simulation should be simulated. If set to e.g. `100`, the energy is computed for time steps *0, 100, 200, ...* The default value is `0`, meaning that the plugin does not compute the particle energy.
`--<species>_energy.period 42`	Same as above, for any other species available.
`--<species>_energy.filter`	Use filtered particles. All available filters will be shown with `picongpu --help`

Output

The plugin creates files prefixed with the species' name and the filter name as postfix, e.g. *e_energy_<filterName>.dat* for the electron energies and *p_energy_<filterName>.dat* for proton energies. The file contains a header describing the columns.

```
#step Ekin_Joule E_Joule
0.0   0.0        0.0
```

Following the header, each line is the output of one time step. The time step is given as first value. The second value is the kinetic energy of all particles at that time step. And the last value is the total energy (kinetic + rest energy) of all particles at that time step.

Example Visualization

Python snippet:

```
import numpy as np

simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]
# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
```

```
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot(
    e_sum_ene[:,0],
    e_sum_ene[:,1] + p_sum_ene[:,1] + C_sum_ene[:,1] + N_sum_ene[:,1] + fields_sum_
↪ene[:,1],
    label="sum"
)
plt.legend()
```

2.5.9 HDF5

Stores simulation data such as fields and particles along with domain information, conversion units etc. as HDF5 files. It uses libSplash for writing HDF5 data. It is used for post-simulation analysis and for **restarts** of the simulation after a crash or an intended stop.

What is the format of the created HDF5 files?

We write our fields and particles in an open markup called **openPMD**. You can investigate your files via a large collection of tools and frameworks or your use the native HDF5 bindings of your favourite programming language.

Resources for a quick-start:

- online tutorial
- example files
- written standard of the openPMD standard
- list of projects supporting openPMD files

External Dependencies

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

.param file

The corresponding `.param` file is *fileOutput.param*.

One can e.g. disable the output of particles by setting:

```
/* output all species */
using FileOutputParticles = VectorAllSpecies;
/* disable */
using FileOutputParticles = bmpl::vector0< >;
```

.cfg file

You can use `--hdf5.period` and `--hdf5.file` to specify the output period and path and name of the created fileset. For example, `--hdf5.period 128 --hdf5.file simData` will write the simulation data to files of the form `simData_0.h5`, `simData_128.h5` in the default simulation output directory every 128 steps. Note that this plugin will only be available if libSplash and HDF5 is found during compile configuration.

PIConGPU command line option	Description
`--hdf5.period`	Period after which simulation data should be stored on disk. Default is `0`, which means that no data is stored.
`--hdf5.file`	Relative or absolute fileset prefix for simulation data. If relative, files are stored under `simOutput/`. Default is `h5`.

Additional Tools

See our *openPMD* chapter.

2.5.10 Intensity

The maximum amplitude of the electric field for each cell in y-cell-position in **V/m** and the integrated amplitude of the electric field (integrated over the entirer x- and z-extent of the simulated volume and given for each y-cell-position).

Attention: There might be an error in the units of the integrated output.

Note: A renaming of this plugin would be very useful in order to understand its purpose more intuitively.

.cfg file

By setting the PIConGPU command line flag `--intensity.period` to a non-zero value the plugin computes the maximum electric field and the integrated electric field for each cell-wide slice in y-direction. The default value is `0`, meaning that nothing is computed. By setting e.g. `--intensity.period 100` the electric field analysis is computed for time steps *0, 100, 200, ...*

Output

The output of the maximum electric field for each y-slice is stored in `Intensity_max.dat`. The output of the integrated electric field for each y-slice is stored in `Intensity_integrated.dat`.

Both files have two header rows describing the data. .. code:

```
#step position_in_laser_propagation_direction
#step amplitude_data[*]
```

The following odd rows give the time step and then describe the y-position of the slice at which the maximum electric field or integrated electric field is computed. The even rows give the time step again and then the data (maximum electric field or integrated electric field) at the positions given in the previews row.

Know Issues

Currently, the output file is overwritten after restart. Additionally, this plugin does not work with non-regular domains, see here . This will be fixed in a future version.

There might be an error in the units of the integrated output.

For a full list, see #327 .

2.5.11 ISAAC

This is a plugin for the in-situ library ISAAC for a live rendering and steering of PIConGPU simulations.

External Dependencies

The plugin is available as soon as the *ISAAC library* is compiled in.

.cfg file

Command line option	Description
`--isaac.period N`	Sets up, that every *N* th timestep an image will be rendered. This parameter can be changed later with the controlling client.
`--isaac.name NAME`	Sets the *NAME* of the simulation, which is shown at the client.
`--isaac.url URL`	*URL* of the required and running isaac server. Host names and IPs are supported.
`--isaac.port PORT`	*PORT* of the isaac server. The default value is `2458` (for the in-situ plugins), but may be needed to be changed for tunneling reasons or if more than one server shall run on the very same hardware.
`--isaac.width WIDTH`	Setups the *WIDTH* and *HEIGHT* of the created image(s).
`--isaac.height HEIGHT`	Default is `1024x768`.
`--isaac.direct_pause`	If activated ISAAC will pause directly after the simulation started. Useful for presentations or if you don't want to miss the beginning of the simulation.
`--isaac.quality QUALITY`	Sets the *QUALITY* of the images, which are compressed right after creation. Values between `1` and `100` are possible. The default is `90`, but `70` does also still produce decent results.

The most important settings for ISAAC are `--isaac.period`, `--isaac.name` and `--isaac.url`. A possible addition for your submission `tbg` file could be `--isaac.period 1 --isaac.name !TBG_jobName --isaac.url YOUR_SERVER`, where the tbg variables `!TBG_jobName` is used as name and `YOUR_SERVER` needs to be set up by yourself.

.param file

The ISAAC Plugin has an *isaac.param*, which specifies which fields and particles are rendered. This can be edited (in your local paramSet), but at runtime also an arbitrary amount of fields (in ISAAC called *sources*) can be deactivated. At default every field and every known species are rendered.

Running and steering a simulation

First of all you need to build and run the isaac server somewhere. On HPC systems, simply start the server on the login or head node since it can be reached by all compute nodes (on which the PIConGPU clients will be running).

Functor Chains

One of the most important features of ISAAC are the **Functor Chains**. As most sources (including fields and species) may not be suited for a direct rendering or even full negative (like the electron density field), the functor chains enable you to change the domain of your field source-wise. A date will be read from the field, the functor

chain applied and then **only the x-component** used for the classification and later rendering of the scene. Multiply functors can be applied successive with the Pipe symbol `|`. The possible functors are at default:

- **mul** for a multiplication with a constant value. For vector fields you can choose different value per component, e.g. `mul(1,2,0)`, which will multiply the x-component with 1, the y-component with 2 and the z-component with 0. If less parameters are given than components exists, the last parameter will be used for all components without an own parameter.
- **add** for adding a constant value, which works the same as `mul(...)`.
- **sum** for summarizing all available components. Unlike `mul(...)` and `add(...)` this decreases the dimension of the data to `1`, which is a scalar field. You can exploit this functor to use a different component than the x-component for the classification, e.g. with `mul(0,1,0) | sum`. This will first multiply the x- and z-component with 0, but keep the y-component and then merge this to the x-component.
- **length** for calculating the length of a vector field. Like *sum* this functor reduces the dimension to a scalar field, too. However `mul(0,1,0) | sum` and `mul(0,1,0) | length` do not do the same. As `length` does not know, that the x- and z-component are 0 an expensive square root operation is performed, which is slower than just adding the components up.
- **idem** does nothing, it just returns the input data. This is the default functor chain.

Beside the functor chains the client allows to setup the weights per source (values greater than 6 are more useful for PIConGPU than the default weights of 1), the classification via transfer functions, clipping, camera steering and to switch the render mode to iso surface rendering. Furthermore interpolation can be activated. However this is quite slow and most of the time not needed for non-iso-surface rendering.

Example renderings

2.5.12 Particle Calorimeter

A binned calorimeter of the amount of kinetic energy per solid angle and energy-per-particle.

The solid angle bin is solely determined by the particle's momentum vector and not by its position, so we are emulating a calorimeter at infinite distance.

The calorimeter takes into account all existing particles as well as optionally all particles which have already left the global simulation volume.

External Dependencies

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

.param file

The spatial calorimeter resolution can be customized and in *speciesDefinition.param*. Therein, a species can be also be marked for detecting particles leaving the simulation box.

.cfg file

All options are denoted for the photon (`ph`) particle species here.

PIConGPU command line option	Description
`--ph_calorimeter.period`	The ouput periodicity of the plugin. A value of `100` would mean an output at simulation time step *0, 100, 200, ...*
`--ph_calorimeter.numBinsYaw`	Specifies the number of bins used for the yaw axis of the calorimeter. Defaults to `64`.
`--ph_calorimeter.numBinsPitch`	Specifies the number of bins used for the pitch axis of the calorimeter. Defaults to `64`.
`--ph_calorimeter.numBinsEnergy`	Specifies the number of bins used for the energy axis of the calorimeter. Defaults to `1`, i.e. there is no energy binning.
`--ph_calorimeter.minEnergy`	Minimum detectable energy in keV. Ignored if `numBinsEnergy` is `1`. Defaults to `0`.
`--ph_calorimeter.maxEnergy`	Maximum detectable energy in keV. Ignored if `numBinsEnergy` is `1`. Defaults to `1000`.
`--ph_calorimeter.logScale`	if given the energy binning is logarithmic.
`--ph_calorimeter.openingYaw`	opening angle yaw of the calorimeter in degrees. Defaults to the maximum value: `360`.
`--ph_calorimeter.openingPitch`	opening angle pitch of the calorimeter in degrees. Defaults to the maximum value: `180`.
`--ph_calorimeter.posYaw`	yaw coordinate of the calorimeter position in degrees. Defaults to the +y direction: `0`.
`--ph_calorimeter.posPitch`	pitch coordinate of the calorimeter position in degrees. Defaults to the +y direction: `0`.

Coordinate System

Yaw and pitch are Euler angles defining a point on a sphere's surface, where `(0,0)` points to the `+y` direction here. In the vicinity of `(0,0)`, yaw points to `+x` and pitch to `+z`.

Orientation detail: Since the calorimeters's three-dimensional orientation is given by just two parameters (*posYaw* and *posPitch*) there is one degree of freedom left which has to be fixed. Here, this is achieved by eliminating the Euler angle roll. However, when *posPitch* is exactly `+90` or `-90` degrees, the choice of roll is ambiguous, depending on the yaw angle one approaches the singularity. Here we assume an approach from `yaw = 0`.

Tuning the spatial resolution

By default, the spatial bin size is chosen by dividing the opening angle by the number of bins for yaw and pitch respectively. The bin size can be tuned by customizing the mapping function in `particleCalorimeter.param`.

Detection of outgoing particles

In order to detect particles which are leaving the global simulation volume, the calorimeter plugin has to be triggered whenever this happens. This is established by adding the `GuardHandlerCallPlugins` template argument to `ParticleDescription` (see *speciesDefinition.param*):

```
// example for photons
using PIC_Photons = Particles<
    ParticleDescription<
        bmpl::string<'p', 'h'>,
        SuperCellSize,
        DefaultAttributesSeq,
        ParticleFlagsPhotons,
```

```
        GuardHandlerCallPlugins
    >
> ;
```

Please make sure to have `picongpu/GuardHandlerCallPlugins.hpp` included.

Output

The calorimeters are stored in hdf5-files in the `simOutput/<species>_calorimeter` directory. The dataset within the hdf5-file is located at `/data/<timestep>/calorimeter`. Depending on whether energy binning is enabled the dataset is two or three dimensional. The dataset has the following attributes:

Attribute	Description
`unitSI`	conversion factor from calorimeter value to Joule.
`maxYaw[deg]`	half of the opening angle yaw.
`maxPitch[deg]`	half of the opening angle pitch.
`posYaw[deg]`	yaw coordinate of the calorimeter.
`posPitch[deg]`	pitch coordinate of the calorimeter. If energy binning is enabled:
`minEnergy[keV]`	minimal detectable energy.
`maxEnergy[keV]`	maximal detectable energy.
`logScale`	boolean indicating logarithmic scale.

Analysis Tools

The first bin of the energy axis of the calorimeter contains all particle energy less than the minimal detectable energy whereas the last bin contains all particle energy greater than the maximal detectable energy. The inner bins map to the actual energy range of the calorimeter.

Sample script for plotting the spatial distribution and the energy distribution:

```
f = h5.File("<path-to-hdf5-file>")
calorimeter = np.array(f["/data/<timestep>/calorimeter"])

# spatial energy distribution
# sum up the energy spectrum
plt.imshow(np.sum(calorimeter, axis=0))
plt.show()

# energy spectrum
# sum up all solid angles
plt.plot(np.sum(calorimeter, axis=(1,2)))
plt.show()
```

2.5.13 Particle Merger

Merges macro particles that are close in phase space to reduce computational load.

.param file

In *particleMerging.param* is currently one compile-time parameter:

Compile-Time Option	Description
`MAX_VORONOI_CELLS`	Maximum number of active Voronoi cells per supercell. If the number of active Voronoi cells reaches this limit merging events are dropped.

.cfg file

PIConGPU command line option	Description
`--<species>_merger.period`	The ouput periodicity of the plugin. A value of `100` would mean an output at simulation time step *0, 100, 200, ...*
`--<species>_merger.minParticlesToMerge`	minimal number of macroparticles needed to merge the macroparticle collection into a single macroparticle.
`--<species>_merger.posSpreadThreshold`	Below this threshold of spread in position macroparticles can be merged [unit: cell edge length].
`--<species>_merger.absMomSpreadThreshold`	Below this absolute threshold of spread in momentum macroparticles can be merged [unit: $m_{e^-} \cdot c$]. Disabled for `-1` (default).
`--<species>_merger.relMomSpreadThreshold`	Below this relative (to mean momentum) threshold of spread in momentum macroparticles can be merged [unit: none]. Disabled for `-1` (default).
`--<species>_merger.minMeanEnergy`	minimal mean kinetic energy needed to merge the macroparticle collection into a single macroparticle [unit: keV].

Notes

- `absMomSpreadThreshold` and `relMomSpreadThreshold` are mutually exclusive
- `absMomSpreadThreshold` is always given in [electron mass * speed of light]!

Reference

The particle merger implements a macro particle merging algorithm based on:

Luu, P. T., Tueckmantel, T., & Pukhov, A. (2016). Voronoi particle merging algorithm for PIC codes. Computer Physics Communications, 202, 165-174.

2.5.14 Phase Space

This plugin creates a 2D phase space image for a user-given spatial and momentum coordinate.

External Dependencies

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

.cfg file

Example for *y-pz* phase space for the *electron* species (`.cfg` file macro):

```
# Calculate a 2D phase space
# - momentum range in m_e c
TGB_ePSypz="--e_phaseSpace.period 10 --e_phaseSpace.space y --e_phaseSpace.
↪momentum pz --e_phaseSpace.min -1.0 --e_phaseSpace.max 1.0"
```

The distinct options are (assuming a species `e` for electrons):

Option	Usage	Unit
`--e_phaseSpace.period <N>`	calculate each N steps	*none*
`--e_phaseSpace.space <x/y/z>`	spatial coordinate of the 2D phase space	*none*
`--e_phaseSpace.momentum <px/py/pz>`	momentum coordinate of the 2D phase space	*none*
`--e_phaseSpace.min <ValL>`	minimum of the momentum range	$m_{\text{species}}c$
`--e_phaseSpace.max <ValR>`	maximum of the momentum range	$m_{\text{species}}c$

Output

The 2D histograms are stored in `.hdf5` files in the `simOutput/phaseSpace/` directory. A file is created per species, phasespace selection and time step.

Values are given as *charge density* per phase space bin. In order to scale to a simpler *charge of particles* per $\mathrm{d}r_i$ and $\mathrm{d}p_i$ -bin multiply by the cell volume `dV`.

Analysis

The easiest way is to load the data in Python:

```
from picongpu.plugins.phase_space import PhaseSpace
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import numpy as np


# load data
phase_space = PhaseSpace('/home/axel/runs/foil_001')
e_ps, e_ps_meta = phase_space.get('e', ps='ypy', iteration=1000)

# unit conversion from SI
mu = 1.e6  # meters to microns
e_mc_r = 1. / (9.1e-31 * 2.9979e8)  # electrons: kg * m / s to beta * gamma

# plotting
plt.imshow(
    np.abs(e_ps).T * e_ps_meta.dV,
    extent = e_ps_meta.extent * [mu, mu, e_mc_r, e_mc_r],
    interpolation = 'nearest',
    aspect = 'auto',
    origin='lower',
    norm = LogNorm()
)

# annotations
cbar = plt.colorbar()
cbar.set_label(r'$Q / \mathrm{d}r \mathrm{d}p$ [$\mathrm{C s kg^{-1} m^{-2}}$]')

ax = plt.gca()
ax.set_xlabel(r'${0}$'.format(e_ps_meta.r) + r' [$\mathrm{\mu m}$]')
ax.set_ylabel(r'$p_{0}$ [$\beta\gamma$]'.format(e_ps_meta.p))

plt.show()
```

Note that the spatial extent of the output over time might change when running a moving window simulation.

Out-of-Range Behavior

Particles that are *not* in the range of `<ValL>`/`<ValR>` get automatically mapped to the lowest/highest bin respectively. Take care about that when setting your range and during analysis of the results.

Known Limitations

- only one range per selected space-momentum-pair possible right now (naming collisions)
- charge deposition uses the counter shape for now (would need one more write to neighbours to get it correct to the shape)

- the user has to define the momentum range in advance
- the resolution is fixed to `1024 bins` in momentum and the number of cells in the selected spatial dimension
- this plugin does not yet use *openPMD markup*.

References

The internal algorithm is explained in pull request #347 and in *[Huebl2014]*.

2.5.15 PNG

This plugin generates **images in the png format** for slices through the simulated volume. It allows to draw a **species density** together with electric, magnetic and/or current field values. The exact field values, their coloring and their normalization can be set using `*.param` files. It is a very rudimentary and useful tool to get a first impression on what happens in the simulation and to verify that the parameter set chosen leads to the desired physics.

Note: In the near future, this plugin might be replaced by the ISAAC interactive 3D visualization.

External Dependencies

The plugin is available as soon as the *PNGwriter library* is compiled in.

.cfg file

For **electrons** (`e`) the following table describes the command line arguments used for the visualization.

Command line option	Description
`--e_png.period`	This flag requires an integer value that specifies at what periodicity the png pictures should be created. E.g. setting `--e_png.period 100` generates images for the *0th, 100th, 200th, ...* time step. There is no default. If flags are not set, no pngs are created.
`--e_png.axis`	Set 2D slice through 3D volume that will be drawn. Combine two of the three dimensions `x`, `y``and ``z`, the define a slice. E.g. setting `--e_png.axis yz` draws both the y and z dimension and performes a slice in x-direction.
`--e_png.slicePoint`	Specifies at what ratio of the total depth of the remaining dimension, the slice should be performed. The value given should lie between `0.0` and `1.0`.
`--e_png.folder`	Name of the folder, where all pngs for the above setup should be stored.

These flags use `boost::program_options`'s `multitoken()`. Therefore, **several setups** can be specified e.g. to draw different slices. The order of the flags is important in this case. E.g. in the following example, two different slices are visualized and stored in different directories:

```
picongpu [more args]
  # first
  --e_png.period 100
  --e_png.axis xy
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXY
  # second
  --e_png.period 100
  --e_png.axis xz
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXZ
```

.param files

The two param files *visualization.param* and *visColorScales.param* are used to specify the desired output.

Specifying the field values using `visualization.param`

Depending on the used prefix in the command line flags, electron and/or ion density is drawn. Additionally to that, three field values can be visualized together with the particle density. In order to set up the visualized field values, the *visualization.param* needs to be changed. In this file, a variety of other parameters used for the PngModule can be specified.

The ratio of the image can be set.

```
/* scale image before write to file, only scale if value is not 1.0 */
const double scale_image = 1.0;

/* if true image is scaled if cellsize is not quadratic, else no scale */
const bool scale_to_cellsize = true;
```

In order to scale the image, `scale_to_cellsize` needs to be set to `true` and `scale_image` needs to specify the reduction ratio of the image.

Note: For a 2D simulation, even a 2D image can be a quite heavy output. Make sure to reduce the preview size!

It is possible to draw the borders between the GPUs used as white lines. This can be done by setting the parameter *white_box_per_GPU* in *visualization.param* to *true*

```
const bool white_box_per_GPU = true;
```

There are three field values that can be drawn: *CHANNEL1*, *CHANNEL2* and *CHANNEL3*.

Since an adequate color scaling is essential, there several option the user can choose from.

```
// normalize EM fields to typical laser or plasma quantities
//-1: Auto: enable adaptive scaling for each output
// 1: Laser: typical fields calculated out of the laser amplitude
// 2: Drift: typical fields caused by a drifting plasma
// 3: PlWave: typical fields calculated out of the plasma freq.,
// assuming the wave moves approx. with c
// 4: Thermal: typical fields calculated out of the electron temperature
// 5: BlowOut: typical fields, assuming that a LWFA in the blowout
// regime causes a bubble with radius of approx. the laser's
// beam waist (use for bubble fields)
#define EM_FIELD_SCALE_CHANNEL1 -1
#define EM_FIELD_SCALE_CHANNEL2 -1
#define EM_FIELD_SCALE_CHANNEL3 -1
```

In the above example, all channels are set to **auto scale**. **Be careful**, when using other normalizations than auto scale, because depending on your set up, the normalization might fail due to parameters not set by PIConGPU. *Use the other normalization options only in case of the specified scenarios or if you know, how the scaling is computed. *

You can also add opacity to the particle density and the three field values:

```
// multiply highest undisturbed particle density with factor
float_X const preParticleDens_opacity = 0.25;
float_X const preChannel1_opacity = 1.0;
float_X const preChannel2_opacity = 1.0;
float_X const preChannel3_opacity = 1.0;
```

and add different coloring:

```
// specify color scales for each channel
namespace preParticleDensCol = colorScales::red;  /* draw density in red */
namespace preChannel1Col = colorScales::blue;     /* draw channel 1 in blue */
namespace preChannel2Col = colorScales::green;    /* draw channel 2 in green */
namespace preChannel3Col = colorScales::none;     /* do not draw channel 3 */
```

The colors available are defined in `visColorScales.param` and their usage is described below. If `colorScales::none` is used, the channel is not drawn.

In order to specify what the three channels represent, three functions can be defined in *visualization.param*. The define the values computed for the png visualization. The data structures used are those available in PIConGPU.

```
/* png preview settings for each channel */
DINLINE float_X preChannel1( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_J )
{
    /* Channel1
     * computes the absolute value squared of the electric current */
    return math::abs2(field_J);
}

DINLINE float_X preChannel2( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_J )
{
    /* Channel2
     * computes the square of the x-component of the electric field */
    return field_E.x() * field_E.x();
}

DINLINE float_X preChannel3( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_J )
{
    /* Channel3
     * computes the negative values of the y-component of the electric field
     * positive field_E.y() return as negative values and are NOT drawn */
    return -float_X(1.0) * field_E.y();
}
```

Only positive values are drawn. Negative values are clipped to zero. In the above example, this feature is used for *preChannel3*.

Defining coloring schemes in *visColorScales.param*

There are several predefined color schemes available:

- none (do not draw anything)
- gray
- grayInv
- red
- green
- blue

But the user can also specify his or her own color scheme by defining a namespace with the color name that provides an ‘addRGB‘function:

```
namespace NameOfColor /* name needs to be unique */
{
    HDINLINE void addRGB( float3_X& img, /* the already existing image */
                          const float_X value, /* the value to draw */
```

```
                        const float_X opacity ) /* the opacity specified */
    {
        /* myChannel specifies the color in RGB values (RedGreenBlue) with
         * each value ranging from 0.0 to 1.0 .
         * In this example, the color yellow (RGB=1,1,0) is used. */
        const float3_X myChannel( 1.0, 1.0, 0.0 );

        /* here, the previously calculated image (in case, other channels have␣
↪already
         * contributed to the png) is changed.
         * First of all, the total image intensity is reduced by the opacity of␣
↪this
         * channel, but only in the color channels specified by this color
↪"NameOfColor".
         * Then, the actual values are added with the correct color (myChannel)␣
↪and opacity. */
        img = img
              - opacity * float3_X( myChannel.x() * img.x(),
                                    myChannel.y() * img.y(),
                                    myChannel.z() * img.z() )
              + myChannel * value * opacity;
    }
}
```

For most cases, using the predefined colors should be enough.

Output

The output of this plugin are pngs stored in the directories specified by `--e_png.folder` or `--i_png.folder`. There can be as many of these folders as the user wants. The pngs follow a naming convention:

```
<species>_png_yx_0.5_002000.png
```

First, either `<species>` names the particle type. Following the 2nd underscore, the drawn dimensions are given. Then the slice ratio, specified by `--e_png.slicePoint` or `--i_png.slicePoint`, is stated in the file name. The last part of the file name is a 6 digit number, specifying the simulation time step, at which the picture was created. This naming convention allows to put all pngs in one directory and still be able to identify them correctly if necessary.

2.5.16 Positions Particles

This plugin prints out the *position, momentum, mass, macro particle weighting, electric charge and relativistic gamma factor* of a particle to `stdout` (usually inside the `simOutput/output` file). **It only works with test simulations that have only one particle.**

.cfg file

By setting the command line flag `--<species>_position.period` to a non-zero number, the analyzer is used. In order to get the particle trajectory for each time step the period needs to be set to `1`, meaning e.g. `--e_position.period 1` for electrons. If less output is needed, e.g. only every 10th time step, the period can be set to different values, e.g. `--e_position.period 10`.

Output

The electron trajectory is written directly to the *standard out*. Therefore, it goes both to `./simOutput/output` as well as to the output file specified by the machine used (usually the `stdout` file in the main directory of the simulation). The output is ASCII-text only. It has the following format:

```
[ANALYSIS] [MPI_Rank] [COUNTER] [<species>_position] [currentTimeStep] currentTime
↪{position.x position.y position.z} {momentum.x momentum.y momentum.z} mass␣
↪weighting charge gamma
```

Value	Description	Unit
`MPI_Rank`	MPI rank at which prints the particle position	*none*
`COUNTER`	name of the plugin | always `<species>_position`	
`currentTimeStep`	simulation time step = number of PIC cycles	*none*
`currentTime`	simulation time in SI units	seconds
`position.x` `_position.y` `_position.z`	location of the particle in space	meters
`momentum.x` `_momentum.y` `_momentum.z`	momentum of particle	kg m/s
`mass`	mass of macro particle	kg
`weighting`	number of electrons represented by the macro particle	*none*
`charge`	charge of macro particle	Coulomb
`gamma`	relativistic gamma factor of particle	*none*

```
# an example output line:
[ANALYSIS] [2] [COUNTER] [e_position] [878] 1.46440742e-14 {1.032e-05 4.
↪570851689815522e-05 5.2e-06} {0 -1.
337873603181226e-21 0} 9.109382e-31 1 -1.602176e-19 4.999998569488525
```

In order to extract only the trajectory information from the total output stored in *stdout*, the following command on a bash command line could be used:

```
grep "e_position" stdout > trajectory.dat
```

The particle data is then stored in `trajectory.dat`.

In order to extract e.g. the position from this line the following can be used:

```
cat trajectory.dat | awk '{print $7}' | sed -e "s/{//g" | sed -e 's/}//g' | sed -e
↪'s/,/\t/g' > position.dat
```

Known Issues

Attention: This plugin only works correctly if a single particle is simulated. If more than one particle is simulated, the output becomes random, because only the information of one particle is printed. This plugin might be upgraded to work with multiple particles, but better use our HDF5 or ADIOS plugin instead and assign ‘particleId‘s to individual particles.

Attention: Currently, both *output* and ‘stdout‘are overwritten at restart. All data from the plugin is lost, if these file are not backuped manually.

2.5.17 Radiation

The spectrally resolved far field radiation of charged macro particles.

Our simulation computes the Lienard Wiechert potentials to calculate the emitted electromagnetic spectra for

different observation directions using the far field approximation.

$$\frac{\mathrm{d}^2 I}{\mathrm{d}\,\Omega\,\mathrm{d}\,\omega}(\omega, \vec{n}) = \left| \sum_{k=1}^{N} \int_{-\infty}^{+\infty} \frac{\vec{n} \times \left[\left(\vec{n} - \vec{\beta}_k(t) \right) \times \dot{\vec{\beta}}_k(t) \right]}{\left(1 - \vec{\beta}_k(t) \cdot \vec{n} \right)^2} \cdot \mathrm{e}^{\mathrm{i}\,\omega (t - \vec{n} \cdot \vec{r}_k(t)/c)}\,\mathrm{d}\,t \right|^2$$

Variable	Meaning
$\vec{r}_k(t)$	The position of particle *k* at time *t*.
$\vec{\beta}_k(t)$	The normalized speed of particle *k* at time *t*. (Speed devided by the speed of light)
$\dot{\vec{\beta}}_k(t)$	The normalized acceleration of particle *k* at time *t*. (Time derivative of the normalized speed.)
t	Time
$\vec{n}$	Unit vector pointing in the direction where the far field radiation is observed.
ω	The circular frequency of the radiation that is observed.
N	Number of all (macro) particles that are used for computing the radiation.
k	Running index of the particles.

Currently this allows to predict the emitted radiation from plasmas if it can be described by classical means. Not considered are emissions from ionization, Compton scattering or any bremsstrahlung that originate from scattering on scales smaller than the PIC cell size.

External Dependencies

The plugin is available as soon as the *libSplash and HDF5 libraries* are compiled in.

.param files

In order to setup the radiation analyzer plugin, both the *radiation.param* and the *radiationObserver.param* have to be configured **and** the radiating particles need to have the attribute `momentumPrev1` which can be added in *speciesDefinition.param*.

In *radiationConfig.param*, the number of frequencies `N_omega` and observation directions `N_theta` is defined.

Frequency range

The frequency range is set up by choosing a specific namespace that defines the frequency setup

```
/* choose linear frequency range */
namespace radiation_frequencies = rad_linear_frequencies;
```

Currently you can choose from the following setups for the frequency range:

namespace	Description
`rad_linear_frequencies`	linear frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`rad_log_frequencies`	logarithmic frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`rad_frequencies_from_list`	`N_omega` frequencies taken from a text file with location `listLocation[]`

Observation directions

The number of observation directions *N_theta* is defined in *radiation.param*, but the distribution of observation directions is given in *radiationObserver.param.param*) There, the function `observation_direction` defines the observation directions.

This function returns the x,y and z component of a **unit vector** pointing in the observation direction.

```
DINLINE vector_64
observation_direction( int const observation_id_extern )
{
    /* use the scalar index const int observation_id_extern to compute an
     * observation direction (x,y,y) */
    return vector_64( x , y , z );
}
```

Note: The `radiationObserver.param` set up will be subject to **further changes**. These might be *namespaces* that describe several preconfigured layouts or a functor if *C++ 11* is included in the *nvcc*.

Nyquist limit

A major limitation of discrete Fourier transform is the limited frequency resolution due to the discrete time steps of the temporal signal. (see Nyquist-Shannon sampling theorem) Due to the consideration of relativistic delays, the sampling of the emitted radiation is not equidistantly sampled. The plugin has the option to ignore any frequency contributions that lies above the frequency resolution given by the Nyquist-Shannon sampling theorem. Because performing this check costs computation time, it can be switched off. This is done via a precompiler pragma:

```
// Nyquist low pass allows only amplitudes for frequencies below Nyquist frequency
// 1 = on (slower and more memory, no Fourier reflections)
// 0 = off (faster but with Fourier reflections)
#define __NYQUISTCHECK__ 0
```

Additionally, the maximally resolvable frequency compared to the Nyquist frequency can be set.

```
namespace radiationNyquist
{
    /* only use frequencies below 1/2*Omega_Nyquist */
    const float NyquistFactor = 0.5;
}
```

This allows to make a save margin to the hard limit of the Nyquist frequency. By using `NyquistFactor = 0.5` for periodic boundary conditions, particles that jump from one border to another and back can still be considered.

Form factor

The *form factor* is still an experimental method trying to consider the shape of the macro particles when computing the radiation. By default, it should be switched off by setting `__COHERENTINCOHERENTWEIGHTING__` to zero.

```
// corect treatment of coherent and incoherent radiation from macroparticles
// 1 = on (slower and more memory, but correct quantitative treatment)
// 0 = off (faster but macroparticles are treated as highly charged, point-like␣
↪particle)
#define __COHERENTINCOHERENTWEIGHTING__ 0
```

If switched on, one can select between different macro particle shapes. Currently three shapes are implemented. A shape can be selected by choosing one of the available namespaces:

```
/* choosing the 3D CIC-like macro particle shape */
namespace radFormFactor_selected = radFormFactor_CIC_3D;
```

Namespace	Description
radFormFactor_CIC_3D	3D Cloud-In-Cell shape
radFormFactor_CIC_1Cy	Cloud-In-Cell shape in y-direction, dot like in the other directions
radFormFactor_incoherent	forces a completely incoherent emission by scaling the macro particle charge with the square root of the weighting

Reducing the particle sample

In order to save computation time, only a random subset of all macro particles can be used to compute the emitted radiation. In order to do that, the radiating particle species needs the attribute `radiationMask` (which is initialized as `false`) which further needs to be manipulated, to set to true for specific (random) particles.

Note: The reduction of the total intensity is not considered in the output. The intensity will be (in the incoherent case) by the fraction of marked marticles smaller than in the case of selecting all particles.

Note: The radiation mask is only added to particles, if not all particles should be considered for radiation calculation. Adding the radiation flag costs memory.

Note: In future updates, the radiation will only be computed using an extra particle species. Therefore, this setup will be subject to further changes.

Gamma filter

In order to consider the radiation only of particles with a gamma higher than a specific threshold, the radiating particle species needs the attribute `radiationMask` (which is initialized as `false`). Using a filter functor as:

```
using RadiationParticleFilter = picongpu::particles::manipulators::FreeImpl<
    GammaFilterFunctor
 >;
```

(see Bunch or Kelvin Helmholtz example for details) sets the flag to true is a particle fulfills the gamma condition.

Note: More sophisticated filters might come in the near future. Therefore, this part of the code might be subject to changes.

Window function filter

A window function can be added to the simulation area to reduce ringing artifacts due to sharp transition from radiating regions to non-radiating regions at the boundaries of the simulation box. This should be applied to simulation setups where the entire volume simulated is radiating (e.g. Kelvin-Helmholtz Instability).

In `radiationConfig.param` the precompiler variable `PIC_RADWINDOWFUNCTION` defines if the window function filter should be used or not.

```
// add a window function weighting to the radiation in order
// to avoid ringing effects from sharp boundaries
// 1 = on (slower but with noise/ringing reduction)
// 0 = off (faster but might contain ringing)
#define PIC_RADWINDOWFUNCTION 0
```

If set to `1`, the window function filter is used.

There are several different window function available:

```
/* Choose different window function in order to get better ringing reduction
 * radWindowFunctionRectangle
 * radWindowFunctionTriangle
 * radWindowFunctionHamming
 * radWindowFunctionTriplett
 * radWindowFunctionGauss
 */
namespace radWindowFunctionRectangle { }
namespace radWindowFunctionTriangle { }
namespace radWindowFunctionHamming { }
namespace radWindowFunctionTriplett { }
namespace radWindowFunctionGauss { }

namespace radWindowFunction = radWindowFunctionTriangle;
```

By setting `radWindowFunction` a specific window function is selected.

.cfg file

For a specific (charged) species `<species>` e.g. `e`, the radiation can be computed by the following commands.

Command line option	Description
`--radiation_<species>_period`	Gives the number of time steps between which the radiation should be calculated. Default is `0`, which means that the radiation in never calculated and therefor off. Using *1* calculates the radiation constantly. Any value `>=2` is currently producing nonsense.
`--radiation_<species>_dump`	Period, after which the calculated radiation data should be dumped to the file system. Default is `0`, therefor never. In order to store the radiation data, a value *>=1* should be used.
`--radiation_<species>_lastRadiation`	If set, the radiation spectra summed between the last and the current dump-time-step are stored. Used for a better evaluation of the temporal evolution of the emitted radiation.
`--radiation_<species>_folderLastRad`	Name of the folder, in which the summed spectra for the simulation time between the last dump and the current dump are stored. Default is `lastRad`.
`--radiation_<species>_totalRadiation`	If set the spectra summed from simulation start till current time step are stored.
`--radiation_<species>_folderTotalRad`	Folder name in which the total radiation spectra, integrated from the beginning of the simulation, are stored. Default `totalRad`.
`--radiation_<species>_start`	Time step, at which PIConGPU starts calculating the radiation. Default is `2` in order to get enough history of the particles.
`--radiation_<species>_end`	Time step, at which the radiation calculation should end. Default: ‘0‘(stops at end of simulation).
`--radiation_<species>_omegaList`	In case the frequencies for the spectrum are coming from a list stored in a file, this gives the path to this list. Default: *_noPath_* throws an error. *This does not switch on the frequency calculation via list.*
`--radiation_<species>_radPerGPU`	If set, each GPU additionally stores its own spectra without summing over the entire simulation area. This allows for a localization of specific spectral features.
`--radiation_<species>_folderRadPerGPU`	Name of the folder, where the GPU specific spectra are stored. Default: `radPerGPU`
`--radiation_<species>_compression`	If set, the hdf5 output is compressed.

Output

Depending on the command line options used, there are different output files.

Command line flag	Output description
`--radiation_<species> totalRadiation`	Contains *ASCII* files that have the total spectral intensity until the timestep specified by the filename. Each row gives data for one observation direction (same order as specified in the `observer.py`). The values for each frequency are separated by *tabs* and have the same order as specified in `radiationConfig.param`. The spectral intensity is stored in the units **[J s]**.
`--radiation_<species> lastRadiation`	has the same format as the output of *totalRadiation*. The spectral intensity is only summed over the last radiation *dump* period.
`--radiation_<species> radPerGPU`	Same output as *totalRadiation* but only summed over each GPU. ecause each GPU specifies a spatial region, the origin of radiation signatures can be distinguished.
radiationHDF5	In the folder `radiationHDF5`, hdf5 files for each radiation dump and species are stored. These are complex amplitudes in units used by *PIConGPU*. These are for restart purposes and for more complex data analysis.

Analysing tools

In `picongp/src/tools/bin`, there are tools to analyze the radiation data after the simulation.

Tool	Description
`plotRadiation`	Reads *ASCII* radiation data and plots spectra over angles as color plots. This is a python script that has its own help. Run `plotRadiation --help` for more information.
`radiationSyntheticDetector`	Reads *ASCII* radiation data and statistically analysis the spectra for a user specified region of observation angles and frequencies. This is a python script that has its own help. Run `radiationSyntheticDetector --help` for more informations.
smooth.py	Python module needed by *plotRadiation*.

Known Issues

Currently, the radiation plugin does not support 2D simulations. This should be fixed with issue #289 . The plugin supports multiple radiation species but spectra (frequencies and observation directions) are the same for all species.

References

- Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics, Diploma thesis on the radiation plugin
- How to test and verify radiation diagnostics simulations within particle-in-cell frameworks, Some tests that have been performed to validate the code

2.5.18 Resource Log

Writes resource information such as rank, position, current simulation step, particle count, and cell count as json or xml formatted string to output streams (file, stdout, stderr).

.cfg file

Run the plugin for each nth time step: `--resourceLog.period n`

The following table will describes the settings for the plugin:

Command line option	Description
`--resourceLog.properties`	Selects properties to write [rank, position, currentStep, particleCount, cellCount]
`--resourceLog.format`	Selects output format [json, jsonpp, xml, xmlpp]
`--resourceLog.stream`	Selects output stream [file, stdout, stderr]
`--resourceLog.prefix`	Selects the prefix for the file stream name

Output / Example

Using the options

```
--resourceLog.period 1 \
--resourceLog.stream stdout \
--resourceLog.properties rank position currentStep particleCount cellCount \
--resourceLog.format jsonpp
```

will write resource objects to stdout such as:

```
[1,1]<stdout>:    "resourceLog": {
[1,1]<stdout>:        "rank": "1",
[1,1]<stdout>:        "position": {
[1,1]<stdout>:            "x": "0",
[1,1]<stdout>:            "y": "1",
[1,1]<stdout>:            "z": "0"
[1,1]<stdout>:        },
[1,1]<stdout>:        "currentStep": "357",
[1,1]<stdout>:        "cellCount": "1048576",
[1,1]<stdout>:        "particleCount": "2180978"
[1,1]<stdout>:    }
[1,1]<stdout>:}
```

For each format there exists always a non pretty print version to simplify further processing:

```
[1,3]<stdout>:{"resourceLog":{"rank":"3","position":{"x":"1","y":"1","z":"0"},
↪"currentStep":"415","cellCount":"1048576","particleCount":"2322324"}}
```

2.5.19 Slice Field Printer

Outputs a 2D slice of the **electric, magnetic and/or current field** in SI units. The slice position and the field can be specified by the user.

.cfg file

The plugin works on **electric**, **magnetic**, and **current** fields. For the electric field, the prefix `--E_slice.` for all command line arguments is used. For the magnetic field, the prefix `--B_slice.` is used. For the current field, the prefix `--J_slice.` is used.

The following table will describe the setup for the electric field. The same applied to the magnetic field. Only the prefix has to be adjusted.

Command line option	Description
`--E_slice.period`	The periodicity of the slice print out. If set to a non-zero value, e.g. to `--E_slice.period 100`, the slices are generated for every 100th simulation time step.
`--E_slice.fileName`	Name of the output file. Setting *–E_slice.fileName myName* will result in output files like `myName_100.dat`.
`--E_slice.plane`	Defines the plane that the slice will be parallel to. The plane is defined by its orthogonal axis. By using `0` for the x-axis, `1` for the y-axis and `2` for the z-axis, all standard planes can be selected. E.g. choosing the x-y-plane is done by setting the orthogonal axis to the z-axis by giving the command line argument `--E_slice.plane 2`.
`--E_slice.slicePoint`	Defines the position of the slice on the orthogonal axis. E.g. when the x-y-plane was selected, the slice position in z-direction has to be set. This is done using a value between `0.0` and `1.0`. E.g. by setting `--E_slice.slicePoint 0.5`, the slice is centered.

This plugin **supports using multiple slices**. By setting the command line arguments multiple times, multiple slices are printed to file. As an example, the following command line will create two slices:

```
picongpu # [...]
  --E_slice.period 100 --E_slice.fileName slice1 --E_slice.plane 2 --E_slice.
↪slicePoint 0.5
  --E_slice.period  50 --E_slice.fileName slice2 --E_slice.plane 0 --E_slice.
↪slicePoint 0.25
```

The first slice is a cut along the x-y axis. It is printed every 100th step. It cuts through the middle of the z-axis and the data is stored in files like *slice1_100.dat*. The second slice is a cut along the y-z axis. It is printed every 50th step. It cuts through the first quarter of the x-axis and the data is stored in files like *slice2_100.dat*.

2D fields

In the case of 2D fields, the plugin outputs a 1D slice. Be aware that `--E_slice.plane` still refers to the orthogonal axis, i.e. `--E_slice.plane 1` outputs a line along the **x-axis** and `--E_slice.plane 0` along the **y-axis**.

Output

The output is stored in an ASCII file for every time step selected by `.period` (see *How to set it up?*). The 2D slice is stored as lines and rows of the ASCII file. Spaces separate rows and newlines separate lines. Each entry is of the format `{1.1e-1,2.2e-2,3.3e.3}` giving each value of the vector field separately e.g. `{E_x,E_y,E_z}`.

In order to read this data format, there is a python module in `lib/python/picongpu/plugins/sliceFieldReader.py`. The function `readFieldSlices` needs a data file (file or filename) with data from the plugin and returns the data as numpy-array of size `(N_y, N_x, 3)`

Known Issues

See issue #348.

Should be solved with pull request #548.

2.5.20 Sum Currents

This plugin computes the total current integrated/added over the entire volume simulated.

.cfg file

The plugin can be activated by setting a non-zero value with the command line flag `--sumcurr.period`. The value set with `--sumcurr.period` is the periodicity, at which the total current is computed. E.g. `--sumcurr.period 100` computes and prints the total current for time step *0, 100, 200, ...*

Output

The result is printed to *standard output*. Therefore, it goes both to `./simOutput/output` as well as to the output file specified by the machine used (usually the `stdout` file in the main directory of the simulation). The output is ASCII-text only. It has the following format:

```
[ANALYSIS] [_rank] [COUNTER] [SumCurrents] [_currentTimeStep] {_current.x _current.
↪y _current.z} Abs:_absCurrent
```

Value	Description	Unit
`_rank`	MPI rank at which prints the particle position	*none*
`_currentTimeStep`	simulation time step = number of PIC cycles	*none*
`_current.x _current.y _current.z`	electric current	Ampere per second
`_absCurrent`	magnitude of current	Ampere per second

In order to extract only the total current information from the output stored in *stdout*, the following command on a bash command line could be used:

```
grep SumCurrents stdout > totalCurrent.dat
```

The plugin data is then stored in `totalCurrent.dat`.

Known Issues

Currently, both `output` and `stdout` are overwritten at restart. All data from the plugin is lost, if these file are not backuped manually.

2.6 TBG

Section author: Axel Huebl, Richard Pausch

Module author: René Widera

todo: explain idea and use case

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

2.6.1 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"] [-f] [-h]
           [projectPath] destinationPath

-c | --cfg      [file]         - Configuration file to set up batch file.
                                 Default: [cfgFile] via export TBG_CFGFILE
-s | --submit   [command]      - Submit command (qsub, "qsub -h", sbatch, ...)
                                 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
-f | --force                   - Override if 'destinationPath' exists.
-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
```

2.6.2 .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


# version information on startup
TBG_version="--versionOnce"


# 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"

# Enable macro particle merging
```

```
TBG_<species>_merger="--<species>_merger.period 100 --<species>_merger.
↪minParticlesToMerge 8 --<species>_merger.posSpreadThreshold 0.2 --<species>_
↪merger.absMomSpreadThreshold 0.01"


# Notification period of position plugin (single-particle debugging)
TBG_<species>_pos_dbg="--<species>_position.period 1"


# Create a particle-energy histogram [in keV] per species for every .period steps
TBG_<species>_histogram="--<species>_energyHistogram.period 500 --<species>_
↪energyHistogram.binCount 1024    \
                        --<species>_energyHistogram.minEnergy 0 --<species>_
↪energyHistogram.maxEnergy 500000"


# 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"

# Write out slices of field data for every .period step
TBG_EField_slice="--E_slice.period 100 --E_slice.fileName sliceE --E_slice.plane 2␣
↪--E_slice.slicePoint 0.5"
TBG_BField_slice="--B_slice.period 100 --B_slice.fileName sliceB --B_slice.plane 2␣
↪--B_slice.slicePoint 0.5"
TBG_JField_slice="--J_slice.period 100 --J_slice.fileName sliceJ --J_slice.plane 2␣
↪--J_slice.slicePoint 0.5"

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


# Count the number of macro particles per species for every .period steps
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 --checkpoint.period steps
#   http://git.io/PToFYg
TBG_checkpoint="--checkpoint.period 1000"
# Select the backend for the checkpoint, available are hdf5 and adios
#    --checkpoint.backend adios
#                         hdf5

# Restart the simulation from checkpoint created using TBG_checkpoint
TBG_restart="--checkpoint.restart"
# Select the backend for the restart (must fit the created checkpoint)
#    --checkpoint.restart.backend adios
#                                 hdf5
# By default, the last checkpoint is restarted if not specified via
#   --checkpoint.restart.step 1000
# To restart in a new run directory point to the old run where to start from
#   --checkpoint.restart.directory /path/to/simOutput/checkpoints

# Presentation mode: loop a simulation via restarts
#   does either start from 0 again or from the checkpoint specified with
#   --checkpoint.restart.step as soon as the simulation reached the last time step;
#   in the example below, the simulation is run 5000 times before it shuts down
# Note: does currently not work with `Radiation` plugin
TBG_restartLoop="--checkpoint.restart.loop 5000"

# Live in situ visualization using ISAAC
#   Initial period in which a image shall be rendered
#     --isaac.period PERIOD
#   Name of the simulation run as seen for the connected clients
#     --isaac.name NAME
#   URL of the server
#     --isaac.url URL
#   Number from 1 to 100 decribing the quality of the transceived jpeg image.
#   Smaller values are faster sent, but of lower quality
#     --isaac.quality QUALITY
#   Resolution of the rendered image. Default is 1024x768
#     --isaac.width WIDTH
#     --isaac.height HEIGHT
#   Pausing directly after the start of the simulation
#     --isaac.directPause
#   By default the ISAAC Plugin tries to reconnect if the sever is not available
#   at start or the servers crashes. This can be deactivated with this option
#     --isaac.reconnect false
TBG_isaac="--isaac.period 1 --isaac.name !TBG_jobName --isaac.url <server_url>"
TBG_isaac_quality="--isaac.quality 90"
TBG_isaac_resolution="--isaac.width 1024 --isaac.height 768"
TBG_isaac_pause="--isaac.directPause"
TBG_isaac_reconnect="--isaac.reconnect false"

# Print the maximum charge deviation between particles and div E to textfile
↪'chargeConservation.dat':
```

```
TBG_chargeConservation="--chargeConservation.period 100"

# Particle calorimeter: (virtually) propagates and collects particles to infinite␣
↪distance
TBG_<species>_calorimeter="--<species>_calorimeter.period 100 --<species>_
↪calorimeter.openingYaw 90 --<species>_calorimeter.openingPitch 30
                        --<species>_calorimeter.numBinsEnergy 32 --<species>_
↪calorimeter.minEnergy 10 --<species>_calorimeter.maxEnergy 1000
                        --<species>_calorimeter.logScale"

# 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 ))"
```

2.6.3 Batch System Examples

Slurm

Slurm is a modern batch system, e.g. installed on the Taurus cluster at TU Dresden.

Job Submission

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

- `tbg -s sbatch -c etc/picongpu/0008gpus.cfg -t etc/picongpu/taurus-tud/k80.tpl $SCRATCH/runs/test-001`

Job Control

- interactive job:
  - `salloc --time=1:00:00 --nodes=1 --ntasks-per-node=2 --cpus-per-task=8 --partition gpu-interactive`
  - e.g. `srun "hostname"`
  - GPU allocation on taurus requires an additional flag, e.g. for two GPUs `--gres=gpu:2`
- details for my jobs:
  - `scontrol -d show job 12345` all details for job with <job id> `12345`
  - `squeue -u $(whoami) -l` all jobs under my user name
- details for queues:
  - `squeue -p queueName -l` list full queue
  - `squeue -p queueName --start` (show start times for pending jobs)
  - `squeue -p queueName -l -t R` (only show running jobs in queue)
  - `sinfo -p queueName` (show online/offline nodes in queue)
  - `sview` (alternative on taurus: `module load llview` and `llview`)
  - `scontrol show partition queueName`
- communicate with job:
  - `scancel <job id>` abort job
  - `scancel -s <signal number> <job id>` send signal or signal name to job
  - `scontrol update timelimit=4:00:00 jobid=12345` change the walltime of a job
  - `scontrol update jobid=12345 dependency=afterany:54321` only start job `12345` after job with id `54321` has finished
  - `scontrol hold <job id>` prevent the job from starting
  - `scontrol release <job id>` release the job to be eligible for run (after it was set on hold)

PBS

PBS (for *Portable Batch System*) is a widely distributed batch system that comes in several implementations (open, professional, etc.). It is used, e.g. on Hypnos at HZDR.

Job Submission

PIConGPU job submission on the *Hypnos* cluster at *HZDR*:

- `tbg -s qsub -c etc/picongpu/0008gpus.cfg -t etc/picongpu/hypnos-hzdr/k20.tpl /bigdata/hplsim/<...>/test-001`

Where `<...>` is one of:

- `external/$(whoami)`
- internal:
  - `scratch/$(whoami)`
  - `development/$(whoami)`
  - `production/<project name>`

Job Control

- interactive job:
  - `qsub -I -q k20 -lwalltime=12:00:00 -lnodes=1:ppn=8`
- details for my jobs:
  - `qstat -f 12345` all details for job with <job id> `12345`
  - `qstat -u $(whoami)` all jobs under my user name
- details for queues:
  - `qstat -a queueName` show all jobs in a queue
  - `pbs_free -l` compact view on free and busy nodes
  - `pbsnodes` list all nodes and their detailed state (free, busy/job-exclusive, offline)
- communicate with job:
  - `qdel <job id>` abort job
  - `qsig -s <signal number> <job id>` send signal or signal name to job
  - `qalter -lwalltime=12:00:00 <job id>` change the walltime of a job
  - `qalter -Wdepend=afterany:54321 12345` only start job `12345` after job with id `54321` has finished
  - `qhold <job id>` prevent the job from starting
  - `qrls <job id>` release the job to be eligible for run (after it was set on hold)

2.7 Example Setups

2.7.1 Bremsstrahlung: Emission of Bremsstrahlung from Laser-Foil Interaction

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

Module author: 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 kinetic energy in favor of emission of Bremsstrahlung-photons. Passing over the back surface hot electrons are eventually reflected and re-enter the foil in opposite direction. Because of the ultra-relativistic energy Bremsstrahlung (BS) is continuously emitted mainly along the direction of motion of the electron. The BS-module models the electron-ion scattering as three single processes, including electron deflection, electron deceleration and photon creation with respect to the emission angle. Details of the implementation and the numerical model can be found in *[BurauDipl]*. Details of the theoretical description can be found in *[Jackson]* and *[Salvat]*.

This 2D test simulates a laser pulse of a_0=40, lambda=0.8µm, w0=1.5µm in head-on collision with a fully pre-ionized gold foil of 2µm thickness.

Checks

- check 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

2.7.2 Bunch: Thomson scattering from laser electron-bunch interaction

Section author: Richard Pausch <r.pausch (at) hzdr.de>

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

This is a simulation of an electron bunch that collides head-on with a laser pulse. Depending on the number of electrons in the bunch, their momentum and their distribution and depending on the laser wavelength and intensity, the emitted radiation differs. A general description of this simulation can be found in *[PauschDipl]*. A detailed analysis of this bunch simulation can be found in *[Pausch13]*. A theoretical study of the emitted radiation in head-on laser electron collisions can be found in *[Esarey93]*.

This test simulates an electron bunch with a relativistic gamma factor of gamma=5.0 and with a laser with a_0=1.0. The resulting radiation should scale with the number of real electrons (incoherent radiation).

References

2.7.3 Empty: Default PIC Algorithm

Section author: Axel Huebl <a.huebl (at) hzdr.de>

This is an "empty" example, initializing a default particle-in-cell cycle with default algorithms *[BirdsallLangdon]* *[HockneyEastwood]* but without a specific test case. When run, it iterates a particle-in-cell algorithm on a vacuum without particles or electro-magnetic fields initialized, which are the default *.param* files in *include/picongpu/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

2.7.4 FoilLCT: Ion Acceleration from a Liquid-Crystal Target

Section author: Axel Huebl

Module author: Axel Huebl, T. Kluge

The following example models a laser-ion accelerator in the *[TNSA]* regime. An optically over-dense target ($n_{\mathrm{max}} = 192 n_{\mathrm{c}}$) consisting of a liquid-crystal material *8CB* (4-octyl-4'-cyanobiphenyl) $C_{21}H_{25}N$ is used.

Irradiated with a high-power laser pulse with $a_0 = 5$ the target is assumed to be once pre-ionized due to realistic laser contrast and pre-pulses to C^+, H^+ and N^+ while being slightly expanded on its surfaces (modeled as exponential density slope). The overall target is assumed to be initially quasi-neutral and the *8CB* ion components are are not demixed in the surface regions. Surface contamination with, e.g. water vapor is neglected.

The laser is assumed to be in-focus and approximated as a plane wave with temporally Gaussian intensity envelope of $\tau_I^{\mathrm{FWHM}} = 25$ fs.

This example is used to demonstrate:

- an ion acceleration setup with
- *composite, multi ion-species target material*
- *quasi-neutral initial conditions*
- ionization models for *field ionization* and *collisional ionization*

with PIConGPU.

References

2.7.5 KelvinHelmholtz: Kelvin-Helmholtz Instability

Section author: Axel Huebl <a.huebl (at) hzdr.de>

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

This example simulates a shear-flow instability known as the Kelvin-Helmholtz Instability in a near-relativistic setup as studied in *[Alves12]*, *[Grismayer13]*, *[Bussmann13]*. The default setup uses a pre-ionized quasi-neutral hydrogen plasma. Modifiying the ion species' mass to resample positrons instead is a test we perform regularly to control numerical heating and charge conservation.

References

2.7.6 LaserWakefield: Laser Electron Acceleration

Section author: Axel Huebl <a.huebl (at) hzdr.de>

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

Setup for a laser-driven electron accelerator *[TajimaDawson]* in the blowout regime of an underdense plasma *[Modena] [PukhovMeyerterVehn]*. A short (fs) laser beam with ultra-high intensity (a_0 >> 1), modeled as a finite Gaussian beam is focussed in a hydrogen gas target. The target is assumed to be pre-ionized with negligible temperature. The relevant area of interaction is followed by a co-moving window, in whose time span the movement of ions is considered irrelevant which allows us to exclude those from our setup.

This is a demonstration setup to get a visible result quickly and test available methods and I/O. The plasma gradients are unphysically high, the resolution of the laser wavelength is seriously bad, the laser parameters (e.g. pulse length, focusing) are 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

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

Section author: Axel Huebl <a.huebl (at) hzdr.de>

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

This setup initializes a homogenous, non-moving, copper block irradiated by a laser with 10^18 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

2.8 Workflows

This section contains typical user workflows and best practices.

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

2.8.1 Setting the Number of Cells

Section author: Axel Huebl

Together with the grid resolution in *grid.param*, the number of cells in our *.cfg files* determine the overall size of a simulation (box). The following rules need to be applied when setting the number of cells:

Each 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.

2.8.2 Changing the Resolution with a Fixed Target

Section author: Axel Huebl

One often wants to refine an already existing resolution in order to model a setup more precisely or to be able to model a higher density.

1. change cell sizes and time step in *grid.param*
2. change number of GPUs in *.cfg file*
3. change number of *number of cells and distribution over GPUs* in *.cfg file*
4. adjust (transveral) positioning of targets in *density.param*
5. *recompile*

2.8.3 Setting the Laser Initialization Cut-Off

Section author: Axel Huebl

Laser profiles for simulation are modeled with a temporal envelope. A common model assumes a Gaussian intensity distribution over time which by definition never sets to zero, so it needs to be cut-off to a reasonable range.

In *laser.param* each profile implements the cut-off to start (and end) initializing the laser profile via a parameter `PULSE_INIT` t_{init} (sometimes also called `RAMP_INIT`). t_{init} is given in units of the `PULSE_LENGTH` τ which is implemented *laser-profile dependent* (but usually as σ_I of the standard Gaussian of intensity $I = E^2$).

For a fixed target in distance d to the lower $y = 0$ boundary of the simulation box, the maximum intensity arrives at time:

$$t_{\text{laserPeakOnTarget}} = \frac{t_{\text{init}} \cdot \tau}{2} + \frac{d}{c_0}$$

or in terms of discrete time steps Δt:

$$\text{step}_{\text{laserPeakOnTarget}} = \frac{t_{\text{laserPeakOnTarget}}}{\Delta t}.$$

Note: Moving the spatial plane of initialization of the laser pulse via `initPlaneY` does not change the formula above. The implementation covers this spatial offset during initialization.

2.8.4 Definition of Composite Materials

Section author: Axel Huebl

The easiest way to define a composite material in PIConGPU is starting relative to an idealized full-ionized electron density. As an example, lets use $C_{21}H_{25}N$ (*"8CB"*) with a plasma density of $n_{e,max} = 192\, n_c$ contributed by the individual ions relatively as:

- Carbon: $21 \cdot 6/N_{\Sigma e\text{-}}$
- Hydrogen: $25 \cdot 1/N_{\Sigma e\text{-}}$
- Nitrogen: $1 \cdot 7/N_{\Sigma e\text{-}}$

and $N_{\Sigma e\text{-}} = 21_C \cdot 6_{C^{6+}} + 25_H \cdot 1_{H^{+}} + 1_N \cdot 7_{N^{7+}} = 158$.

Set the idealized electron density in *density.param* as a reference and each species' relative `densityRatio` from the list above accordingly in *speciesDefinition.param* (see the input files in the *FoilLCT example* for details).

In order to initialize the electro-magnetic fields self-consistently, read *quasi-neutral initialization*.

2.8.5 Quasi-Neutral Initialization

Section author: Axel Huebl

In order to initialize the electro-magnetic fields self-consistently, one needs to fulfill Gauss's law $\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0}$ (and $\vec{\nabla} \cdot \vec{B} = 0$). The trivial solution to this equation is to start *field neutral* by microscopically placing a charge-compensating amount of free electrons on the same position as according ions.

Fully Ionized Ions

For fully ionized ions, just use `ManipulateDeriveSpecies` in *speciesInitialization.param* and derive macro-electrons $1:1$ from macro-ions but increase their weighting by $1:Z$ of the ion.

```
using InitPipeline = mpl::vector<
    // density profile from density.param and
    //     start position from particle.param
    CreateDensity<
        densityProfiles::YourSelectedProfile,
        startPosition::YourStartPosition,
        Carbon
    >,
    // create a macro electron for each macro carbon but increase its
    //     weighting by the ion's proton number so it represents all its
    //     electrons after an instantanous ionization
    ManipulateDeriveSpecies<
        manipulators::ProtonTimesWeighting,
        Carbon,
        Electrons
    >
>;
```

Partly Ionized Ions

For partial pre-ionization, the *FoilLCT example* shows a detailed setup. First, define a functor that manipulates the number of bound electrons in *particle.param*, e.g. to *once ionized*. Then again in *speciesInitialization.param* set your initialization routines to:

```
using InitPipeline = mpl::vector<
    // density profile from density.param and
    //     start position from particle.param
```

```
    CreateDensity<
        densityProfiles::YourSelectedProfile,
        startPosition::YourStartPosition,
        Carbon
    >,
    // partially pre-ionize the carbons by manipulating the carbon's
    //     `boundElectrons` attribute,
    //     functor defined in particle.param: set to C1+
    Manipulate<
        manipulators::OnceIonized,
        Carbon
    >,
    // does not manipulate the weighting while deriving the electrons
    //     ("once pre-ionized") since we set carbon as C1+
    DeriveSpecies<
        Carbon,
        Electrons
    >
>;
```

If you want to initialize an arbitrary ionization state, just add your own weighting manipulating functor in *particle.param* for the electrons and use it with `ManipulateDeriveSpecies` as in the first example.

In the first example, which does not manipulate the `boundElectrons` attribute of the carbon species (optional, see *speciesAttributes.param*), the default is used (zero bound electrons, fully ionized).

CHAPTER 3

Models

3.1 The Particle-in-Cell Algorithm

Section author: Axel Huebl

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

3.1.1 References

3.2 Landau-Lifschitz Radiation Reaction

Module author: Richard Pausch, Marija Vranic

To do

3.2.1 References

3.3 Field Ionization

Section author: Marco Garten

Module author: Marco Garten

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

PIConGPU features an adaptable ionization framework for arbitrary and combinable ionization models.

Note: Most of the calculations and formulae in this section of the docs are done in the **Atomic Units (AU)** system.

$$\hbar = \mathrm{e} = m_\mathrm{e} = 1$$

Table 3.1: **Atomic Unit System**

AU	SI
length	$5.292 \cdot 10^{-11}\,\mathrm{m}$
time	$2.419 \cdot 10^{-17}\,\mathrm{s}$
energy	$4.360 \cdot 10^{-18}\,\mathrm{J}$ (= 27.21 eV = 1 Rydberg)
electrical field	$5.142 \cdot 10^{11}\,\frac{\mathrm{V}}{\mathrm{m}}$

3.3.1 Overview: Implemented Models

ionization regime	implemented model	reference
Multiphoton	None, yet	
Tunneling	• `Keldysh` • `ADKLinPol` • `ADKCircPol`	• *[BauerMulser1999]* • *[DeloneKrainov]* • *[DeloneKrainov]*
Barrier Suppression	• `BSI` • `BSIEffectiveZ` • `BSIStarkShifted` (R&D)	• *[MulserBauer2010]* • *[ClementiRaimondi1963]* *[ClementiRaimondi1967]* • *[BauerMulser1999]*

Attention: Models marked with "(R&D)" are under *research and development* and should be used with care.

3.3.2 Usage

Input for ionization models is defined in *speciesDefinition.param*, *ionizer.param and ionizationEnergies.param*.

3.3.3 Barrier Suppression Ionization

The so-called barrier-suppression ionization regime is reached for strong fields where the potential barrier binding an electron is completely suppressed.

3.3.4 Tunneling Ionization

Tunneling ionization describes the process during which an initially bound electron quantum-mechanically tunnels through a potential barrier of finite height.

Keldysh

$$\Gamma_\mathrm{K} = \frac{(6\pi)^{1/2}}{2^{5/4}} E_\mathrm{ip} \left(\frac{F}{(2E_\mathrm{ip})^{3/2}} \right)^{1/2} \exp\left(-\frac{2\,(2E_\mathrm{ip})^{3/2}}{3F} \right)$$

The Keldysh ionization rate has been implemented according to the equation (9) in *[BauerMulser1999]*. See also *[Keldysh]* for the original work.

Note: Assumptions:

- low field - perturbation theory
- $\omega_\mathrm{laser} \ll E_\mathrm{ip}$

- $F \ll F_{\mathrm{BSI}}$
- tunneling is instantaneous

Ammosov-Delone-Krainov (ADK)

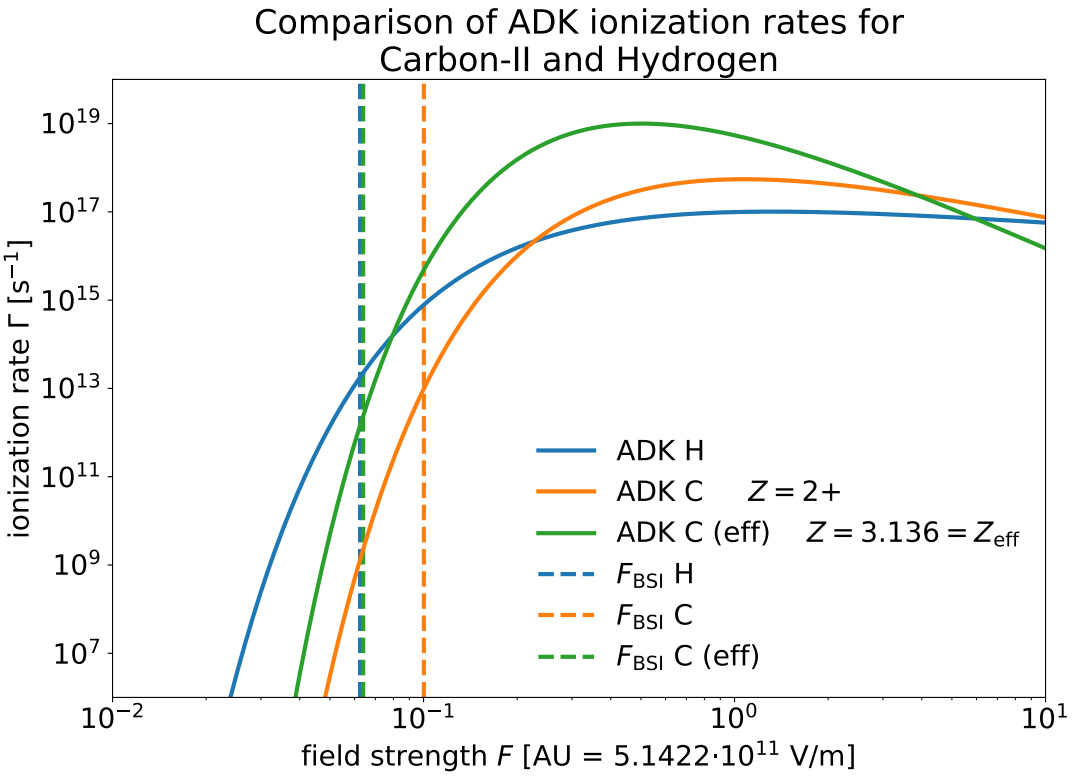
$$\Gamma_{\mathrm{ADK}} = \underbrace{\sqrt{\frac{3{n^*}^3 F}{\pi Z^3}}}_{\text{lin. pol.}} \frac{F D^2}{8\pi Z} \exp\left(-\frac{2Z}{3{n^*}^3 F}\right) \tag{3.1}$$

$$D \equiv \left(\frac{4\mathrm{e}Z^3}{F{n^*}^4}\right)^{n^*} \qquad n^* \equiv \frac{Z}{\sqrt{2E_{\mathrm{ip}}}} \tag{3.2}$$

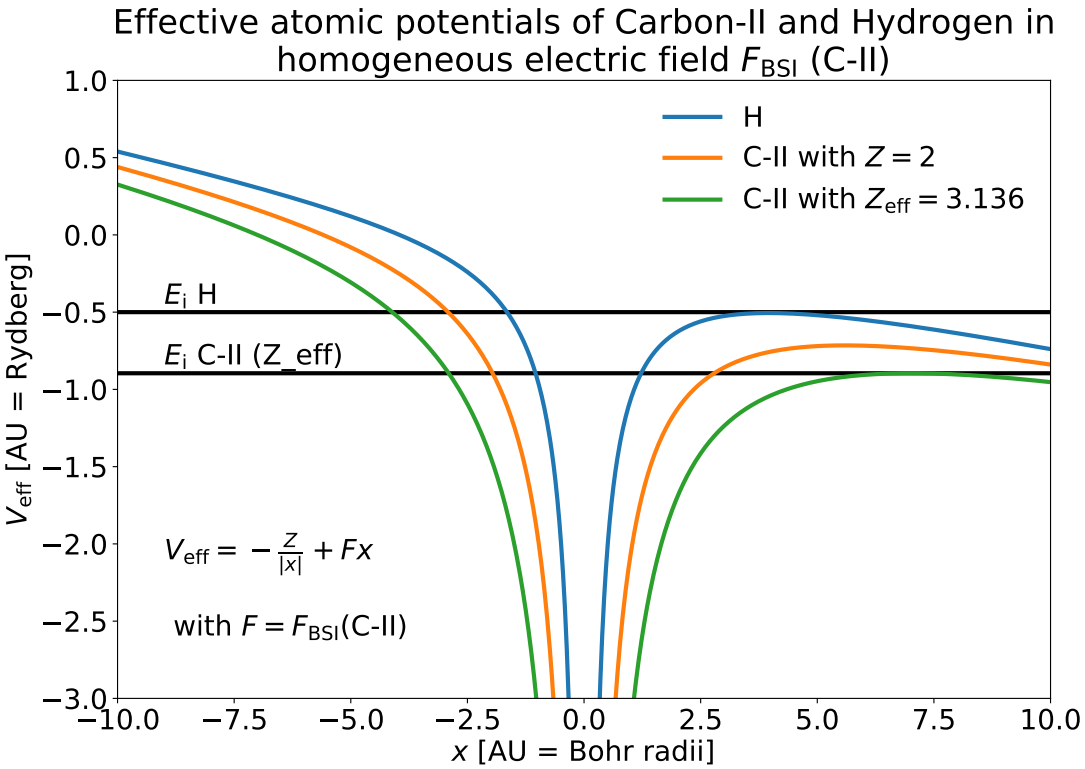
We implemented equation (7) from *[DeloneKrainov]* which is a *simplified result assuming s-states* (since we have no atomic structure implemented, yet). Leaving out the pre-factor distinguishes `ADKCircPol` from `ADKLinPol`. `ADKLinPol` results from replacing an instantaneous field strength F by $F\cos(\omega t)$ and averaging over one laser period.

> **Attention:** Be aware that Z denotes the **residual ion charge** and not the proton number of the nucleus!

In the following comparison one can see the `ADKLinPol` ionization rates for the transition from Carbon II to III (meaning 1+ to 2+). For a reference the rates for Hydrogen as well as the barrier suppression field strengths F_{BSI} have been plotted. They mark the transition from the tunneling to the barrier suppression regime.



When we account for orbital structure in shielding of the ion charge Z according to *[ClementiRaimondi1963]* in `BSIEffectiveZ` the barrier suppression field strengths of Hydrogen and Carbon-II are very close to one another. One would expect much earlier ionization of Hydrogen due to lower ionization energy. The following image shows how this can be explained by the shape of the ion potential that is assumed in this model.



3.3.5 References

3.4 Collisional Ionization

3.4.1 LTE Models

Module author: Marco Garten

Implemented LTE Model: Thomas-Fermi Ionization according to *[More1985]*

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

The implementation of the Thomas-Fermi model takes the following input quantities.

- ion proton number Z
- ion species mass density ρ
- electron "temperature" T

Due to the nature of our simulated setups it is also used in non-equilibrium situations. We therefore implemented additional conditions to mediate unphysical behavior but introduce arbitrariness.

1. Super-thermal electron cutoff

   We calculate the temperature according to $T_{\mathrm{e}} = \frac{2}{3} E_{\mathrm{kin,e}}$ in units of electron volts. We thereby assume an *ideal electron gas*. Via the variable `CUTOFF_MAX_ENERGY_KEV` in `ionizer.param` the user can exclude electrons with kinetic energy above this value from average energy calculation. That is motivated by a lower interaction cross section of particles with high relative velocities.

2. Low ion-density cutoff

   The Thomas-Fermi model displays unphysical behaviour for low ion densities in that it predicts an increasing charge state for decreasing ion densities. This occurs already for electron tempera-

tures of 10 eV and the effect increases as the temperature increases. For instance in pre-plasmas of solid density targets the charge state would be overestimated where

- on average electron energies are not large enough for collisional ionization of a respective charge state
- ion density is not large enough for potential depression
- electron-ion interaction cross sections are small due to small ion density

It is strongly suggested to do approximations for **every** setup or material first. To that end, a parameter scan with *[FLYCHK]* can help in choosing a reasonable value.

3.4.2 NLTE Models

Module author: Axel Huebl

in development

3.5 Photons

Section author: Axel Huebl

Module author: Heiko Burau

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

3.5.1 References

CHAPTER 4

Post-Processing

4.1 Python

Section author: Axel Huebl

If you are new to python, get your hands on the tutorials of the following important libraries to get started.

- https://www.python.org/about/gettingstarted/
- https://docs.python.org/3/tutorial/index.html

4.1.1 Numpy

Numpy is the universal swiss army knife for working on ND arrays in python.

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

4.1.2 Matplotlib

One common way to visualize plots:

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

4.1.3 Jupyter

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

https://jupyter.readthedocs.io

4.1.4 openPMD-viewer

A library that reads and visualizes data in our HDF5 files thanks to their *openPMD markup*. 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.

- Project Homepage
- Tutorial

4.1.5 yt-project

With yt 3.4 or newer, our HDF5 output, which uses the *openPMD markup*, can be read, processed and visualized with yt.

- Project Homepage
- Data Loading
- Data Tutorial

4.1.6 pyDive (experimental)

pyDive provides numpy-style array and file processing on distributed memory systems ("numpy on MPI" for data sets that are much larger than your local RAM). pyDive is currently not ready to interpret *openPMD* directly, but can work on generated raw ADIOS and HDF5 files.

https://github.com/ComputationalRadiationPhysics/pyDive

4.2 openPMD

Section author: Axel Huebl

Module author: Axel Huebl

Our *HDF5* and *ADIOS* use a specific internal markup to structure physical quantities called **openPMD**. If you hear of it for the first time you can find a quick online tutorial on it here.

As a user of PIConGPU, you will be mainly interested in our *python tools* and readers, that can read openPMD, e.g. into:

- Numpy reader and Jupyter notebook GUI via openPMD-viewer (tutorial)
- yt-project (tutorial)
- *ParaView*
- VisIt
- full list of projects using openPMD

If you intend to write your own post-processing routines, make sure to check out our example files, the formal, open standard on openPMD and a list of projects that already support openPMD.

4.3 ParaView

Section author: Axel Huebl

Module author: Axel Huebl

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

CHAPTER 5

Development

5.1 How to Participate as a Developer

5.1.1 Contents

5.1.2 Code - Version Control

If you are familiar with git, feel free to jump to our *github workflow* section.

install git

Debian/Ubuntu:

- `sudo apt-get install git`

- make sure `git --version` is at least at version 1.7.10

Optional *one* of these. There are nice GUI tools available to get an overview on your repository.

- `gitk git-gui qgit gitg`

Mac:

- see here

Windows:

- see here
- just kidding, it's this link
- please use ASCII 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.)

5.1.3 GitHub Workflow

Welcome to github! We will try to explain our coordination strategy (I am out of here!) and our development workflow in this section.

In a Nutshell

Create a *GitHub* account and prepare your *basic git config*.

Prepare your *forked* copy of our repository:

- fork picongpu on *GitHub*
- `git clone git@github.com:<YourUserName>/picongpu.git` (create local copy)
- `git remote add mainline git@github.com:ComputationalRadiationPhysics/picongpu.git` (add our main repository for updates)
- `git checkout dev` (switch to our, its now *your*, dev branch to start from)

Start a *topic/feature branch*:

- `git checkout -b <newFeatureName>` (start a new branch from dev and check it out)
- *hack hack*
- `git add <yourChangedFiles>` (add changed and new files to index)
- `git commit` (commit your changes to your *local* repository)
- `git pull --rebase mainline dev` (update with our *remote dev* updates and avoid a merge commit)

Optional, *clean up* your feature branch. That can be *dangerous*:

- `git pull` (if you pushed your branch already to your public repository)
- `git pull --rebase mainline dev` (apply the mainline updates to your feature branch)
- `git log ..mainline/dev`, `git log --oneline --graph --decorate --all` (check for related commits and ugly merge commits)
- `git rebase mainline/dev` (re-apply your changes after a fresh update to the `mainline/dev`, see here)
- `git rebase -i mainline/dev` (squash related commits to reduce the complexity of the features history during a pull request)

Publish your feature and start a *pull request*:

- `git push -u origin <newFeatureName>` (push your local branch to your github profile)
- Go to your *GitHub* page and open a *pull request*, e.g. by clicking on *compare & review*
- Select `ComputationalRadiationPhysics:dev` instead of the default `master` branch
- Add additional updates (if requested to do so) by `push`-ing to your branch again. This will update the *pull request*.

How to fork from us

To keep our development fast and conflict free, we recomment you to fork our repository and start your work from our **dev** (development) branch in your private repository. Simply click the *Fork* button above to do so.

Afterwards, `git clone` **your** repository to your local machine. But that is not it! To keep track of the original **dev** repository, add it as another remote.

- `git remote add mainline https://github.com/ComputationalRadiationPhysics/picongpu.git`
- `git checkout dev` (go to branch **dev**)

Well done so far! Just start developing. Just like this? No! As always in git, start a *new branch* with `git checkout -b topic-<yourFeatureName>` and apply your changes there.

Keep track of updates

We consider it a **best practice** *not to modify* neither your **master** nor your **dev** branch at all. Instead you can use it to `pull --ff-only` new updates from the original repository. Take care to **switch to dev** by `git checkout dev` to start **new feature branches** from **dev**.

So, if you like to do so, you can even keep track of the *original dev* branch that way. Just start your new branch with `git branch --track <yourFeatureName> mainline/dev` instead. This allows you to immediatly pull or fetch from **our dev** and avoids typing (during `git pull --rebase`). Nevertheless, if you like to `push` to *your* forked (== `origin`) repository, you have to say e.g. `git push origin <branchName>` explicitly.

You should **add updates** from the original repository on a **regular basis** or *at least* when you *finished your feature*.

- commit your local changes in your *feature branch*: `git commit`

Now you *could* do a normal *merge* of the latest `mainline/dev` changes into your feature branch. That is indeed possible, but will create an ugly merge commit. Instead try to first update *the point where you branched from* and apply your changes *again*. That is called a **rebase** and is indeed less harmful as reading the sentence before:

- `git checkout <yourFeatureName>`
- `git pull --rebase mainline dev` (in case of an emergency, hit `git rebase --abort`)

Now solve your conflicts, if there are any, and you got it! Well done!

Pull requests or *being social*

How to propose that **your awesome feature** (we know it will be awesome!) should be included in the **mainline 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 publish you local branch via a *push* to *your* GitHub repository: `git push -u origin <yourLocalBranchName>`

Now let's start a *review*. Open the *GitHub* homepage, go to your repository and switch to your *pushed feature branch*. Select the green **compare & review** button. Now compare the changes between **your feature branch** and **our dev**.

Everything looks good? Submit it as a **pull request** (link in the header). Please take the time to write an **extensive description**.

- What did you implement and why?
- Is there an open issue that you try to address (please link it)?
- Do not be afraid to add images!

The description of the pull request is essential and will be referred to in the change log of the next release.

Please consider to change only **one aspect per pull request** (do not be afraid of follow-up pull requests!). For example, submit a pull request with a bug fix, another one with new math implementations and the last one with a new awesome implementation that needs both of them. You will see, that speeds up *review time* a lot!

Speaking of those, a fruitful (*wuhu, we love you - don't be scared*) *discussion* about your **submitted change set** will start at this point. If we find some things you could *improve* (*That looks awesome, all right!*), simply change your *local feature branch* and *push the changes back* to your GitHub repository, to **update the pull request**. (You can now rebase follow-up branches, too.)

One of our maintainers will pick up the pull request to coordinate the review. Other regular developers that are competent in the topic might assist.

Sharing is caring! Thank you for participating, **you are great**!

maintainer notes

- do not *push* to the main repository on a regular basis, use **pull request** for your features like everyone else
- **never** do a *rebase* on the mainline repositories (this causes heavy problems for everyone who pulls them)
- on the other hand try to use pull –rebase to **avoid merge commits** (in your *local/topic branches* **only**)
- do not vote on your *own pull requests*, wait for the other maintainers
- we try to follow the strategy of a-successful-git-branching-model

Last but not least, help.github.com has a very nice FAQ section.

More best practices.

5.1.4 Commit Rules

See our commit rules page

5.1.5 Test Suite Examples

You know a useful setting to validate our provided methods? Tell us about it or add it to our test sets in the examples/ folder!

5.2 Repository Structure

Section author: Axel Huebl

5.2.1 Branches

- `master`: the latest stable release, always tagged with a version
- `dev`: the development branch where all features start from and are merged to
- `release-X.Y.Z`: release candiate for version `X.Y.Z` with an upcoming release, receives updates for bug fixes and documentation such as change logs but usually no new features

5.2.2 Directory Structure

- `include/`
  - C++ header *and* source files
  - set `-I` here
  - prefixed with project name
- `lib/`
  - pre-compiled libraries
  - `python/`
    - modules, e.g. for RT interfaces, pre* & post-processing
    - set `PYTHONPATH` here
- `etc/`
  - (runtime) configuration files
  - `picongpu/`
    - `tbg` templates (as long as PIConGPU specific, later on to `share/tbg/`)
    - network configurations (e.g. infiniband)
    - score-p and vampir-trace filters
- `share/`
  - examples, documentation
  - `picongpu/`
    - `completions/`: bash completions
    - `examples/`: each with same structure as `/`
- `bin/`
  - core tools for the "PIConGPU framework"
  - set `PATH` here
- `docs/`

- currently for the documentation files
- might move, e.g. to `lib/picongpu/docs/` and its build artifacts to `share/{doc,man}/`,

5.3 Coding Guide Lines

Section author: Axel Huebl

See also:

Our coding guide lines are documented in this repository.

5.3.1 Source Style

For contributions, *an ideal patch blends in the existing coding style around it* without being noticed as an addition when applied. Nevertheless, please make sure *new files* follow the styles linked above as strict as possible from the beginning.

Unfortunately, we currently do not have tools available to auto-format all aspects of our style guidelines. Since we want to focus on the content of your contribution, we try to cover as much as possible by automated tests which you always have to pass. Nevertheless, we will not enforce the still uncovered, *non-semantic aspects* of style in a *pedantic* way until we find a way to automate it fully.

(That also means that we do not encourage manual style-only changes of our existing code base, since both you and us have better things to do than adding newlines and spaces manually. Doxygen and documentation additions are always welcome!)

5.3.2 License Header

Please **add the according license header** snippet to your *new files*:

- for PIConGPU (GPLv3+): `src/tools/bin/addLicense <FileName>`
- for libraries (LGPLv3+ & GPLv3+): `export PROJECT_NAME=PMacc && src/tools/bin/addLicense <FileName>`
- delete other headers: `src/tools/bin/deleteHeadComment <FileName>`
- add license to all `.hpp` files within a directory (recursive): `export PROJECT_NAME=PIConGPU && src/tools/bin/findAndDo <PATH> "*.hpp" src/tools/bin/addLicense`
- the default project name is `PIConGPU` (case sensitive!) and add the GPLv3+ only

Files in the directory `thirdParty/` are only imported from remote repositories. If you want to improve them, submit your pull requests there and open an issue for our **maintainers** to update to a new version of the according software.

5.4 Sphinx

Section author: Axel Huebl

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.

5.4.1 Build Locally

This document is build based on free open-source software, namely Sphinx, Doxygen (C++ APIs as XML) and Breathe (to include doxygen XML in Sphinx). A web-version is hosted on ReadTheDocs.

The following requirements need to be installed (once) to build our documentation successfully:

```
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
```

5.4.2 Useful Links

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

5.5 Doxygen

Section author: Axel Huebl

Although our main documentation is integrated via *Sphinx*, some developers might still want to read the plain HTML build of Doxygen. If you need these docs, this section explains how to build it locally.

5.5.1 Requirements

First, install Doxygen and its dependencies for graph generation.

```
# install requirements (Debian/Ubuntu)
sudo apt-get install doxygen graphviz

# enable HTML output in our Doxyfile
sed -i 's/GENERATE_HTML.*=.*NO/GENERATE_HTML     = YES/' docs/Doxyfile
```

5.5.2 Build

Now run the following commands to build the Doxygen HTML documentation locally.

```
cd docs/

# build the doxygen HTML documentation
doxygen

# open the generated HTML pages, e.g. with firefox
firefox html/index.html
```

5.6 Important PIConGPU Classes

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

5.6.1 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**()

5.6.2 FieldE

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

5.6.3 FieldB

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

5.6.4 FieldJ

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

5.6.5 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

5.6.6 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 **update**(uint32_t **const** *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**()

5.6.7 ComputeGridValuePerFrame

Warning: doxygenclass: Cannot find class "picongpu::particleToGrid::ComputeGridValuePerFrame" in doxygen xml output for project "PIConGPU" from directory: ../xml

5.7 Important pmacc Classes

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

Note: Please help adding more Doxygen doc strings to the classes described below. As an example, here is a listing of possible extensive docs that new developers find are missing: https://github.com/ComputationalRadiationPhysics/picongpu/issues/776

5.7.1 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 >

5.7.2 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**

5.7.3 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

5.7.4 Vector

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

5.7.5 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 *)

5.7.6 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**

5.7.7 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**

5.7.8 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 **copyGuardToExchange**(uint32_t *exchangeType*)
copy gard particles to intermediate exchange buffer

Copy all particles from the guard of a direction to the device exchange buffer.

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**

5.7.9 ParticleDescription

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

5.7.10 ParticleBox

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

5.7.11 Frame

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

5.7.12 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**

5.7.13 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`

5.7.14 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**

5.7.15 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 >

5.7.16 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 Types

template<>
using `KernelType` = T_KernelFunctor

Public Functions

HINLINE `Kernel` (T_KernelFunctor **const** &*kernelFunctor*, std::string **const** &*file* = std::string(), size_t **const** *line* = 0)

Return

Parameters

- `gridExtent`: grid extent configuration for the kernel
- `blockExtent`: block extent configuration for the kernel
- `sharedMemByte`: dynamic shared memory used by the kernel (in byte)

template <**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` (...)
create a kernel object out of a functor instance

this macro add the current filename and line number to the kernel object

Parameters

- `...`: instance of kernel functor

5.7.17 Struct Factory

Syntax to generate structs with all members inline. Allows to conveniently switch between variable and constant defined members without the need to declare or initialize them externally. See for example PIConGPU's *density.param* for usage.

PMACC_STRUCT (name, ...)
generate a struct with static and dynamic members

```
PMACC_STRUCT(StructAlice,
    // constant member variable
    (PMACC_C_VALUE(float, varFoo, -1.0))
    // lvalue member variable
    (PMACC_VALUE(float, varFoo, -1.0))
    // constant vector member variable
    (PMACC_C_VECTOR_DIM(double, 3, vectorBarC, 1.134e-5, 1.134e-5, 1.134e-5))
    // lvalue vector member variable
    (PMACC_VECTOR_DIM(double, 3, vectorBarC, 1.134e-5, 1.134e-5, 1.134e-5))
    // constant string member variable
    (PMACC_C_STRING(someString, "anythingYouWant: even spaces!"))
    // plain C++ member
    PMACC_EXTENT(
        using float_64 = double;
        static constexpr int varBar = 42;
    );
);
```

Note do not forget the surrounding parenthesize for each element of a sequence

Parameters

- `name`: name of the struct
- `...`: preprocessor sequence with TypeMemberPair's e.g. (*PMACC_C_VALUE(int,a,2)*)

PMACC_C_VECTOR_DIM (type, dim, name, ...)
create static const member vector that needs no memory inside of the struct

```
PMACC_C_VECTOR_DIM(float_64, simDim, center_SI, 1.134e-5, 1.134e-5, 1.134e-5);
// is syntactically equivalent to
static const Vector<float_64,simDim> center_SI = Vector<float_64,simDim>(1.
↪134e-5, 1.134e-5, 1.134e-5);
```

Parameters

- `type`: type of an element
- `dim`: number of vector components
- `name`: member variable name
- `...`: enumeration of init values (number of components must be greater or equal than dim)

PMACC_C_VALUE (type, name, value)
create static constexpr member

```
PMACC_C_VALUE(float_64, power_SI, 2.0);
// is syntactically equivalent to
static constexpr float_64 power_SI = float_64(2.0);
```

Parameters

- `type`: type of the member
- `name`: member variable name
- `value`: init value

PMACC_VALUE(type, name, initValue)
create changeable member

```
PMACC_VALUE(float_64, power_SI, 2.0);
// is the equivalent of
float_64 power_SI(2.0);
```

Parameters

- `type`: type of the member
- `name`: member variable name
- `value`: init value

PMACC_VECTOR(type, name, ...)
create changeable member vector

```
PMACC_VECTOR(float2_64, center_SI, 1.134e-5, 1.134e-5);
// is the equivalent of
float2_64 center_SI(1.134e-5, 1.134e-5);
```

Parameters

- `type`: type of an element
- `name`: member variable name
- `...`: enumeration of init values

PMACC_VECTOR_DIM(type, dim, name, ...)
create changeable member vector

```
PMACC_VECTOR_DIM(float_64, simDim, center_SI, 1.134e-5, 1.134e-5, 1.134e-5);
// is the equivalent of
Vector<float_64,3> center_SI(1.134e-5, 1.134e-5, 1.134e-5);
```

Parameters

- `type`: type of an element
- `dim`: number of vector components
- `name`: member variable name
- `...`: enumeration of init values (number of components must be equal to dim)

PMACC_C_STRING(name, initValue)
create static const character string

```
PMACC_C_STRING(filename, "fooFile.txt");
// is syntactically equivalent to
static const char* filename = (char*)"fooFile.txt";
```

Parameters

- `name`: member variable name

- `char_string`: character string

PMACC_EXTENT(...)
create any code extension

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

Parameters

- `...`: any code

5.7.18 Identifier

Construct unique types, e.g. to name, access and assign default values to particle species' attributes. See for example PIConGPU's speciesAttributes.param for usage.

value_identifier(in_type, name, in_default)
define a unique identifier with name, type and a default value

The created identifier has the following options: getValue() - return the user defined value getName() - return the name of the identifier ::type - get type of the value

Parameters

- `in_type`: type of the value
- `name`: name of identifier
- `in_value`: user defined value of in_type (can be a constructor of a class)

e.g. value_identifier(float,length,0.0f) typedef length::type value_type; // is float value_type x = length::getValue(); //set x to 0.f printf("Identifier name: %s",length::getName()); //print Identifier name: length

to create a instance of this value_identifier you can use: `length()` or `length_`

alias(name)
create an alias

an alias is a unspecialized type of an identifier or a value_identifier

example: alias(aliasName); //create type varname

Parameters

- `name`: name of alias

to specialize an alias do: aliasName<valueIdentifierName> to create an instance of this alias you can use: aliasName(); or aliasName_

get type which is represented by the alias typedef typename traits::Resolve<name>::type resolved_type;

5.8 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

CHAPTER 6

Programming Patterns

See also:

In order to follow this section, you need to understand the CUDA programming model.

6.1 Lockstep Programming Model

Section author: René Widera, Axel Huebl

The *lockstep programming model* structures code that is evaluated collectively and independently by workers (physical threads). Actual processing is described by one-dimensional index domains of *virtual workers* which can even be changed within a kernel. Mathematically, index domains are none-injective, total functions on physical workers.

An index domain is **independent** from data but **can** be mapped to a data domain, e.g. one to one or with more complex mappings.

Code which is implemented by the *lockstep programming model* is free of any dependencies between the number of worker and processed data elements. To simplify the implementation, each index within a domain can be seen as a *virtual worker* which is processing one data element (like the common workflow to programming CUDA). Each *worker* i can be executed as N_i *virtual workers* $(1 : N_i)$.

6.1.1 pmacc helpers

template <uint32_t *T_domainSize*, uint32_t *T_workerSize*, uint32_t *T_simdSize* = 1u>
struct `pmacc::mappings::threads::IdxConfig`

describe a constant index domain

describe the size of the index domain and the number of workers to operate on the domain

Template Parameters

- `T_domainSize`: number of indices in the domain
- `T_workerSize`: number of worker working on `T_domainSize`
- `T_simdSize`: SIMD width

template <typename T_Type, **typename** T_IdxConfig>

struct pmacc::memory::**CtxArray**
Static sized array for a local variable.

The array is designed to hold context variables in lock step programming. A context variable is just a local variable of a virtual worker. Allocating and using a context array allows to propagate virtual worker states over subsequent lock steps. A context array for a set of virtual workers is owned by their (physical) worker.

The number of elements depends on the index domain size and the number of workers to process the indices.

Inherits from pmacc::memory::Array< T_Type, T_IdxConfig::numCollIter *T_IdxConfig::simdSize >, T_IdxConfig

template <typename T_IdxConfig>
struct pmacc::mappings::threads::**ForEachIdx**
execute a functor for each index

Distribute the indices even over all worker and execute a user defined functor. There is no guarantee in which order the indices will be processed.

Template Parameters

- T_IdxConfig: index domain description

Inherits from T_IdxConfig

6.1.2 Common Patterns

Collective Loop

- each worker needs to pass a loop N times
- in this example, there are more dates than workers that process them

```
// `frame` is a list which must be traversed collectively
while( frame.isValid() )
{
    uint32_t const workerIdx = threadIdx.x;
    using ParticleDomCfg = IdxConfig<
        frameSize,
        numWorker
    >;
    ForEachIdx< ParticleDomCfg > forEachParticle( workerIdx );
    forEachParticle(
       [&]( uint32_t const linearIdx, uint32_t const idx )
       {
           // independent work
       }
   );
}
```

Non-Collective Loop

- each *virtual worker* increments a private variable

```
uint32_t const workerIdx = threadIdx.x;
using ParticleDomCfg = IdxConfig<
    frameSize,
    numWorkers
>;
ForEachIdx< ParticleDomCfg > forEachParticle( workerIdx );
memory::CtxArray< int, ParticleDomCfg > vWorkerIdx( 0 );
forEachParticle(
```

```
    [&]( uint32_t const linearIdx, uint32_t const idx )
    {
        vWorkerIdx[ idx ] = linearIdx;
        for( int i = 0; i < 100; i++ )
            vWorkerIdx[ idx ]++;
    }
);
```

Create a Context Variable

- ... and initialize with the index of the virtual worker

```
uint32_t const workerIdx = threadIdx.x;
using ParticleDomCfg = IdxConfig<
    frameSize,
    numWorkers
>;
memory::CtxArray< int, ParticleDomCfg > vIdx(
    workerIdx,
    [&]( uint32_t const linearIdx, uint32_t const ) -> int32_t
    {
        return linearIdx;
    }
);

// is equal to

memory::CtxArray< int, ParticleDomCfg > vIdx;
ForEachIdx< ParticleDomCfg > forEachParticle{ workerIdx }(
    [&]( uint32_t const linearIdx, uint32_t const idx )
    {
        vIdx[ idx ] = linearIdx;
    }
);
```

Using a Master Worker

- only one *virtual worker* (called *master*) of all available `numWorkers` manipulates a shared data structure for all others

```
// example: allocate shared memory (uninitialized)
PMACC_SMEM(
    finished,
    bool
);

uint32_t const workerIdx = threadIdx.x;
ForEachIdx<
    IdxConfig<
        1,
        numWorkers
    >
> onlyMaster{ workerIdx };

// manipulate shared memory
onlyMaster(
    [&](
        uint32_t const,
        uint32_t const
```

```
    )
    {
        finished = true;
    }
);

/* important: synchronize now, in case upcoming operations (with
 * other workers) access that manipulated shared memory section
 */
__syncthreads();
```

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/spack/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*, https://cmake.org/

[Huebl2014] A. Huebl. *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, chapter 3.2, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), https://doi.org/10.5281/zenodo.15924

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

[Jackson] J.D. Jackson. *Electrodynamics*, Wiley-VCH Verlag GmbH & Co. KGaA (1975), https://dx.doi.org/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), https://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

[TNSA] S.C. Wilks, A.B. Langdon, T.E. Cowan, M. Roth, M. Singh, S. Hatchett, M.H. Key, D. Pennington, A. MacKinnon, and R.A. Snavely. *Energetic proton generation in ultra-intense laser-solid interactions*, Physics of Plasmas **8**, 542 (2001), https://dx.doi.org/10.1063/1.1333697

[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

[DeloneKrainov] N. B. Delone and V. P. Krainov. *Tunneling and barrier-suppression ionization of atoms and ions in a laser radiation field*, Phys. Usp. 41 469–485 (1998), http://dx.doi.org/10.1070/PU1998v041n05ABEH000393

[BauerMulser1999] 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

[MulserBauer2010] P. Mulser and D. Bauer. *High Power Laser-Matter Interaction*, Springer-Verlag Berlin Heidelberg (2010), https://dx.doi.org/10.1007/978-3-540-46065-7

[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

[ClementiRaimondi1963] E. Clementi and D. Raimondi. *Atomic Screening Constant from SCF Functions*, The Journal of Chemical Physics 38, 2686-2689 (1963) https://dx.doi.org/10.1063/1.1733573

[ClementiRaimondi1967] E. Clementi and D. Raimondi. *Atomic Screening Constant from SCF Functions. II. Atoms with 37 to 86 Electrons*, The Journal of Chemical Physics 47, 1300-1307 (1967) https://dx.doi.org/10.1063/1.1712084

[More1985] R. M. More. *Pressure Ionization, Resonances, and the Continuity of Bound and Free States*, Advances in Atomic, Molecular and Optical Physics Vol. 21 C, 305-356 (1985), https://dx.doi.org/10.1016/S0065-2199(08)60145-1

[FLYCHK] *FLYCHK: Generalized population kinetics and spectral model for rapid spectroscopic analysis for all elements*, H.-K. Chung, M.H. Chen, W.L. Morgan, Yu. Ralchenko, and R.W. Lee, *High Energy Density Physics* v.1, p.3 (2005) http://nlte.nist.gov/FLY/

[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