ImpactX Documentation

Release 23.04

ImpactX collaboration

Apr 03, 2023

CONTENTS

ImpactX is an s-based beam dynamics code including space charge effects. This is the next generation of the IMPACT-Z code.

Note: ImpactX development is in beta status. Please contact us with any questions on it or if you like to contribute to its development.

CHAPTER

ONE

CONTACT US

If you are starting using ImpactX, or if you have a user question, please pop in our Gitter chat room and get in touch with the community.

The ImpactX GitHub repo is the main communication platform. Have a look at the action icons on the top right of the web page: feel free to watch the repo if you want to receive updates, or to star the repo to support the project. For bug reports or to request new features, you can also open a new issue.

We also have a discussion page on which you can find already answered questions, add new questions, get help with installation procedures, discuss ideas or share comments.

1.1 Code of Conduct

1.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

1.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

1.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

1.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

1.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at warpx-coc@lbl.gov. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

1.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.2 Acknowledge ImpactX

Please acknowledge the role that ImpactX played in your research.

1.2.1 In presentations

Note: TODO :-)

1.2.2 In publications

Please add the following sentence to your publications, it helps contributors keep in touch with the community and promote the project.

Plain text:

> This research used the open-source particle-in-cell code ImpactX https://github.com/ECP-WarpX/impactx. We acknowledge all ImpactX contributors.

Latex:

```
\usepackage{hyperref}
This research used the open-source particle-in-cell code ImpactX \url{https://github.com/
↪ECP-WarpX/impactx}.
We acknowledge all ImpactX contributors.
```

1.2.3 Main ImpactX reference

If your project leads to a scientific publication, please consider citing the paper below.

- Huebl A, Lehe R, Mitchell C E, Qiang J, Ryne R D, Sandberg R T, Vay JL. **Next Generation Computational Tools for the Modeling and Design of Particle Accelerators at Exascale**. 2022 North American Particle Accelerator Conference (NAPAC'22), TUYE2, pp. 302-306, 2022. arXiv:2208.02382, DOI:10.18429/JACoW-NAPAC2022-TUYE2

CHAPTER

TWO

INSTALLATION

2.1 Users

Our community is here to help. Please report installation problems in case you should get stuck.

Choose **one** of the installation methods below to get started:

2.1.1 HPC Systems

If want to use ImpactX on a specific high-performance computing (HPC) systems, jump directly to our *HPC system-specific documentation*.

2.1.2 Using the Conda Package

A package for ImpactX is available via the Conda package manager.

```
conda create -n impactx -c conda-forge impactx
conda activate impactx
```

Note: the `impactx` conda package does not yet provide GPU support.

2.1.3 Using the Spack Package

Note: Coming soon.

2.1.4 Using the PyPI Package

Note: Coming soon.

2.1.5 Using the Brew Package

Note: Coming soon.

2.1.6 From Source with CMake

After installing the *ImpactX dependencies*, you can also install ImpactX from source with CMake:

```
# get the source code
git clone https://github.com/ECP-WarpX/impactx.git $HOME/src/impactx
cd $HOME/src/impactx

# configure
cmake -S . -B build

# optional: change configuration
ccmake build

# compile
#   on Windows:          --config Release
cmake --build build -j 4

# executables for ImpactX are now in build/bin/
```

We document the details in the *developer installation*.

2.1.7 Tips for macOS Users

Tip: Before getting started with package managers, please check what you manually installed in `/usr/local`. If you find entries in `bin/`, `lib/` et al. that look like you manually installed MPI, HDF5 or other software in the past, then remove those files first.

If you find software such as MPI in the same directories that are shown as symbolic links then it is likely you brew installed software before. If you are trying annother package manager than `brew`, run brew unlink ... on such packages first to avoid software incompatibilities.

See also: A. Huebl, Working With Multiple Package Managers, Collegeville Workshop (CW20), 2020

2.2 Developers

CMake is our primary build system. If you are new to CMake, this short tutorial from the HEP Software foundation is the perfect place to get started. If you just want to use CMake to build the project, jump into sections 1. Introduction, 2. Building with CMake and 9. Finding Packages.

2.2.1 Dependencies

Before you start, you will need a copy of the ImpactX source code:

```
git clone https://github.com/ECP-WarpX/impactx.git $HOME/src/impactx
cd $HOME/src/impact
```

ImpactX depends on popular third party software.

- On your development machine, *follow the instructions here*.
- If you are on an HPC machine, *follow the instructions here*.

Dependencies

ImpactX depends on the following popular third party software. Please see installation instructions below.

- a mature C++17 compiler, e.g., GCC 7, Clang 7, NVCC 11.0, MSVC 19.15 or newer
- CMake 3.15.0+
- Git 2.18+
- AMReX: we automatically download and compile a copy
- WarpX: we automatically download and compile a copy

Optional dependencies include:

- MPI 3.0+: for multi-node and/or multi-GPU execution
- CUDA Toolkit 11.0+: for Nvidia GPU support (see matching host-compilers)
- OpenMP 3.1+: for threaded CPU execution
- FFTW3: for spectral solver support
- openPMD-api 0.15.1+: we automatically download and compile a copy of openPMD-api for openPMD I/O support
  - see optional I/O backends
- CCache: to speed up rebuilds (needs 3.7.9+ for CUDA)
- Ninja: for faster parallel compiles

Install

Pick *one* of the installation methods below to install all dependencies for ImpactX development in a consistent manner.

Conda (Linux/macOS/Windows)

With MPI (only Linux/macOS):

```
conda create -n impactx-dev -c conda-forge ccache cmake compilers git "openpmd-api=*=mpi_
↪mpich*" python mpich numpy scipy yt "fftw=*=mpi_mpich*" matplotlib mamba ninja numpy␣
↪pandas pytest scipy
conda activate impactx-dev
```

Without MPI:

```
conda create -n impactx-nompi-dev -c conda-forge ccache cmake compilers git openpmd-api␣
↪python numpy scipy yt fftw matplotlib mamba ninja numpy pandas scipy
conda activate impactx-nompi-dev

# compile ImpactX with -DImpactX_MPI=OFF
```

Note: A general option to deactivate that conda self-activates its base environment. This avoids interference with the system and other package managers.

```
conda config --set auto_activate_base false
```

Spack (macOS/Linux)

```
spack env create impactx-dev
spack env activate impactx-dev
spack add adios2        # for openPMD
spack add ccache
spack add cmake
spack add fftw
spack add hdf5          # for openPMD
spack add mpi
spack add pkgconfig     # for fftw
spack add python
spack add py-pip
spack add py-setuptools
spack add py-wheel

# OpenMP support on macOS
[[ $OSTYPE == 'darwin'* ]] && spack add llvm-openmp

# optional: Linux only
#spack add cuda

spack install
python3 -m pip install matplotlib numpy openpmd-api pandas pytest scipy
```

In new terminals, re-activate the environment with `spack env activate impactx-dev` again.

Brew (macOS/Linux)

```
brew update
brew install adios2      # for openPMD
brew install ccache
brew install cmake
brew install fftw
brew install git
brew install hdf5-mpi    # for openPMD
brew install libomp      # for OpenMP
brew install pkg-config  # for fftw
brew install open-mpi
brew install python
python3 -m pip install matplotlib yt scipy numpy openpmd-api
```

Apt (Debian/Ubuntu)

```
sudo apt update
sudo apt install build-essential ccache cmake g++ git libfftw3-mpi-dev libfftw3-dev␣
↪libhdf5-openmpi-dev libopenmpi-dev pkg-config python3 python3-matplotlib python3-numpy␣
↪python3-pandas python3-scipy
```

Note: Preparation: make sure you work with up-to-date Python tooling.

```
python3 -m pip install -U pip setuptools wheel pytest
python3 -m pip install -r examples/requirements.txt
```

2.2.2 Compile

From the base of the ImpactX source directory, execute:

```
# find dependencies & configure
#   see additional options below, e.g.
#                   -DCMAKE_INSTALL_PREFIX=$HOME/sw/impactX
cmake -S . -B build -DImpactX_PYTHON=ON

# compile, here we use four threads
cmake --build build -j 4
```

That's all! ImpactX binaries are now in `build/bin/`. Most people execute these binaries directly or copy them out.

If you want to install the executables in a programmatic way, run this:

```
# for default install paths, you will need administrator rights, e.g. with sudo:
#   this installs the application
cmake --build build --target install

#   this installs the Python bindings via "python3 -m pip install ..."
cmake --build build --target pip_install -j 4
```

You can inspect and modify build options after running `cmake -S . -B build` with either

```
ccmake build
```

or by adding arguments with `-D<OPTION>=<VALUE>` to the first CMake call, e.g.:

```
cmake -S . -B build -DImpactX_PYTHON=ON -DImpactX_COMPUTE=CUDA
```

That's it! You can now *run a first example*.

Developers could now change the ImpactX source code and then call the install lines again to refresh the installation.

Tip: If you do *not* develop with *a user-level package manager*, e.g., because you rely on a HPC system's environment modules, then consider to set up a virtual environment via Python venv. Otherwise, without a virtual environment, you likely need to add the CMake option `-DPYINSTALLOPTIONS="--user"`.

2.2.3 Build Options

CMake Option	Default & Values	Description
`BUILD_TESTING`	**ON**/OFF	Build tests
`CMAKE_BUILD_TYPE`	RelWithDebInfo/**Release**/Debug	Type of build, symbols & optimizations
`CMAKE_INSTALL_PREFIX`	system-dependent path	Install path prefix
`CMAKE_VERBOSE_MAKEFILE`	ON/**OFF**	Print all compiler commands to the terminal during build
`ImpactX_APP`	**ON**/OFF	Build the ImpactX executable application
`ImpactX_COMPUTE`	NOACC/**OMP**/CUDA/SYCL/HIP	On-node, accelerated computing backend
`ImpactX_IPO`	ON/**OFF**	Compile ImpactX with interprocedural optimization (aka LTO)
`ImpactX_LIB`	ON/**OFF**	Build ImpactX as a library (shared or static)
`ImpactX_MPI`	**ON**/OFF	Multi-node support (message-passing)
`ImpactX_MPI_THREAD_MULTIPLE`	**ON**/OFF	MPI thread-multiple support, i.e. for `async_io`
`ImpactX_OPENPMD`	**ON**/OFF	openPMD I/O (HDF5, ADIOS)
`ImpactX_PRECISION`	SINGLE/**DOUBLE**	Floating point precision (single/double)
`ImpactX_PYTHON`	ON/**OFF**	Python bindings
`Python_EXECUTABLE`	(newest found)	Path to Python executable

ImpactX can be configured in further detail with options from AMReX, which are documented in the AMReX manual.

Developers might be interested in additional options that control dependencies of ImpactX. By default, the most important dependencies of ImpactX are automatically downloaded for convenience:

CMake Option	Default & Values	Description
BUILD_SHARED_LIBS	ON/**OFF**	Build shared libraries for dependencies
CCACHE_PROGRAM	First found ccache executable.	Set to -DCCACHE_PROGRAM=NO to disable CCache.
ImpactX_ablastr_src	*None*	Path to ABLASTR source directory (preferred if set)
ImpactX_ablastr_repo	https://github.com/ECP-WarpX/WarpX.git	Repository URI to pull and build ABLASTR from
ImpactX_ablastr_branch	*we set and maintain a compatible commit*	Repository branch for ImpactX_ablastr_repo
ImpactX_ablastr_internal	**ON**/OFF	Needs a pre-installed ABLASTR library if set to OFF
ImpactX_amrex_src	*None*	Path to AMReX source directory (preferred if set)
ImpactX_amrex_repo	https://github.com/AMReX-Codes/amrex.git	Repository URI to pull and build AMReX from
ImpactX_amrex_branch	*we set and maintain a compatible commit*	Repository branch for ImpactX_amrex_repo
ImpactX_amrex_internal	**ON**/OFF	Needs a pre-installed AMReX library if set to OFF
ImpactX_openpmd_src	*None*	Path to openPMD-api source directory (preferred if set)
ImpactX_openpmd_repo	https://github.com/openPMD/openPMD-api.git	Repository URI to pull and build openPMD-api from
ImpactX_openpmd_branch	*we set and maintain a compatible commit*	Repository branch for ImpactX_openpmd_repo
ImpactX_openpmd_internal	**ON**/OFF	Needs a pre-installed openPMD-api library if set to OFF
ImpactX_pyamrex_src	*None*	Path to AMReX source directory (preferred if set)
ImpactX_pyamrex_repo	https://github.com/AMReX-Codes/pyamrex.git	Repository URI to pull and build pyAMReX from
ImpactX_pyamrex_branch	*we set and maintain a compatible commit*	Repository branch for ImpactX_pyamrex_repo
ImpactX_pyamrex_internal	**ON**/OFF	Needs a pre-installed pyAMReX module if set to OFF

For example, one can also build against a local AMReX copy. Assuming AMReX' source is located in $HOME/src/amrex, add the cmake argument -DImpactX_amrex_src=$HOME/src/amrex. Relative paths are also supported, e.g. -DImpactX_amrex_src=../amrex.

Or build against an AMReX feature branch of a colleague. Assuming your colleague pushed AMReX to https://github.com/WeiqunZhang/amrex/ in a branch new-feature then pass to cmake the arguments: -DImpactX_amrex_repo=https://github.com/WeiqunZhang/amrex.git -DImpactX_amrex_branch=new-feature.

If you want to develop against local versions of ABLASTR (from WarpX) and AMReX at the same time, pass for instance -DImpactX_ablastr_src=$HOME/src/warpx -DImpactX_amrex_src=$HOME/src/amrex.

You can speed up the install further if you pre-install these dependencies, e.g. with a package manager. Set -DImpactX_<dependency-name>_internal=OFF and add installation prefix of the dependency to the environment variable CMAKE_PREFIX_PATH. Please see the *introduction to CMake* if this sounds new to you.

If you re-compile often, consider installing the Ninja build system. Pass -G Ninja to the CMake configuration call to

speed up parallel compiles.

2.2.4 Configure Your Compiler

If you don't want to use your default compiler, you can set the following environment variables. For example, using a Clang/LLVM:

```
export CC=$(which clang)
export CXX=$(which clang++)
```

If you also want to select a CUDA compiler:

```
export CUDACXX=$(which nvcc)
export CUDAHOSTCXX=$(which clang++)
```

Note: Please clean your build directory with `rm -rf build/` after changing the compiler. Now call `cmake -S . -B build` (+ further options) again to re-initialize the build configuration.

2.2.5 Run

The ImpactX Python bindings, which provide the imports `impactx` and `amrex` (from pyAMReX), are automatically packaged and installed when calling the `pip_install` *CMake target*.

An executable ImpactX application binary with the current compile-time options encoded in its file name will be created in `build/bin/`. Additionally, a symbolic link named `impactx` can be found in that directory, which points to the last built ImpactX executable.

2.3 HPC

On selected high-performance computing (HPC) systems, ImpactX has documented or even pre-build installation routines. Follow the guide here instead of the generic installation routines for optimal stability and best performance.

2.3.1 impactx.profile

Use a `impactx.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:

```
source $HOME/impactx.profile
```

We list example `impactx.profile` files below, which can be used to set up ImpactX on various HPC systems.

2.3.2 HPC Systems

Cori (NERSC)

The Cori cluster is located at NERSC.

If you are new to this system, please see the following resources:

- GPU nodes
- Cori user guide
- Batch system: Slurm
- Jupyter service
- Production directories:
  - `$SCRATCH`: per-user production directory (20TB)
  - `/global/cscratch1/sd/m3239`: shared production directory for users in the project `m3239` (50TB)
  - `/global/cfs/cdirs/m3239/`: community file system for users in the project `m3239` (100TB)

Installation

Use the following commands to download the ImpactX source code and switch to the correct branch:

```
git clone https://github.com/ECP-WarpX/impactx.git $HOME/src/impactx
```

KNL

We use the following modules and environments on the system (`$HOME/knl_impactx.profile`).

```
module swap craype-haswell craype-mic-knl
module swap PrgEnv-intel PrgEnv-gnu
module load cmake/3.22.1
module load cray-hdf5-parallel/1.10.5.2
module load cray-fftw/3.3.8.10
module load cray-python/3.9.7.1

export PKG_CONFIG_PATH=$FFTW_DIR/pkgconfig:$PKG_CONFIG_PATH
export CMAKE_PREFIX_PATH=$HOME/sw/knl/adios2-2.7.1-install:$CMAKE_PREFIX_PATH

if [ -d "$HOME/sw/knl/venvs/impactx" ]
then
  source $HOME/sw/knl/venvs/impactx/bin/activate
fi

export CXXFLAGS="-march=knl"
export CFLAGS="-march=knl"
```

For PICMI and Python workflows, also install a virtual environment:

```
# establish Python dependencies
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv

python3 -m venv $HOME/sw/knl/venvs/impactx
source $HOME/sw/knl/venvs/impactx/bin/activate

python3 -m pip install --upgrade pip
MPICC="cc -shared" python3 -m pip install -U --no-cache-dir -v mpi4py
python3 -m pip install --upgrade pytest
python3 -m pip install -r $HOME/src/impactx/requirements.txt
python3 -m pip install -r $HOME/src/impactx/examples/requirements.txt
```

Haswell

We use the following modules and environments on the system (`$HOME/haswell_impactx.profile`).

```
module swap PrgEnv-intel PrgEnv-gnu
module load cmake/3.22.1
module load cray-hdf5-parallel/1.10.5.2
module load cray-fftw/3.3.8.10
module load cray-python/3.9.7.1

export PKG_CONFIG_PATH=$FFTW_DIR/pkgconfig:$PKG_CONFIG_PATH
export CMAKE_PREFIX_PATH=$HOME/sw/haswell/adios2-2.7.1-install:$CMAKE_PREFIX_PATH

if [ -d "$HOME/sw/haswell/venvs/impactx" ]
then
  source $HOME/sw/haswell/venvs/impactx/bin/activate
fi
```

For PICMI and Python workflows, also install a virtual environment:

```
# establish Python dependencies
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv

python3 -m venv $HOME/sw/haswell/venvs/impactx
source $HOME/sw/haswell/venvs/impactx/bin/activate

python3 -m pip install --upgrade pip
MPICC="cc -shared" python3 -m pip install -U --no-cache-dir -v mpi4py
python3 -m pip install -r $HOME/src/impactx/requirements.txt
```

GPU (V100)

Cori provides a partition with 18 nodes that include V100 (16 GB) GPUs. We use the following modules and environments on the system (`$HOME/gpu_impactx.profile`).

```
export proj="m1759"

module purge
module load modules
module load cgpu
module load esslurm
module load gcc/8.3.0 cuda/11.4.0 cmake/3.22.1
module load openmpi

export CMAKE_PREFIX_PATH=$HOME/sw/cori_gpu/adios2-2.7.1-install:$CMAKE_PREFIX_PATH

if [ -d "$HOME/sw/cori_gpu/venvs/impactx" ]
then
  source $HOME/sw/cori_gpu/venvs/impactx/bin/activate
fi

# compiler environment hints
export CC=$(which gcc)
export CXX=$(which g++)
export FC=$(which gfortran)
export CUDACXX=$(which nvcc)
export CUDAHOSTCXX=$(which g++)

# optimize CUDA compilation for V100
export AMREX_CUDA_ARCH=7.0

# allocate a GPU, e.g. to compile on
#   10 logical cores (5 physical), 1 GPU
function getNode() {
    salloc -C gpu -N 1 -t 30 -c 10 --gres=gpu:1 -A $proj
}
```

For PICMI and Python workflows, also install a virtual environment:

```
# establish Python dependencies
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv

python3 -m venv $HOME/sw/cori_gpu/venvs/impactx
source $HOME/sw/cori_gpu/venvs/impactx/bin/activate

python3 -m pip install --upgrade pip
python3 -m pip install -U --no-cache-dir -v mpi4py
python3 -m pip install -r $HOME/src/impactx/requirements.txt
```

Building ImpactX

We recommend to store the above lines in individual `impactx.profile` files, as suggested above. If you want to run on either of the three partitions of Cori, open a new terminal, log into Cori and *source* the environment you want to work with:

```
# KNL:
source $HOME/knl_impactx.profile

# Haswell:
#source $HOME/haswell_impactx.profile

# GPU:
#source $HOME/gpu_impactx.profile
```

Warning: Consider that all three Cori partitions are *incompatible*.

Do not *source* multiple `...impactx.profile` files in the same terminal session. Open a new terminal and log into Cori again, if you want to switch the targeted Cori partition.

If you re-submit an already compiled simulation that you ran on another day or in another session, *make sure to source* the corresponding `...impactx.profile` again after login!

Then, `cd` into the directory `$HOME/src/impactx` and use the following commands to compile:

```
cd $HOME/src/impactx
rm -rf build

#                       append if you target GPUs:    -DImpactX_COMPUTE=CUDA
cmake -S . -B build -DImpactX_OPENPMD=ON -DImpactX_DIMS=3
cmake --build build -j 16
```

Testing

To run all tests (here on KNL), do:

```
srun -C knl -N 1 -t 30 -q debug ctest --test-dir build --output-on-failure
```

Running

Navigate (i.e. `cd`) into one of the production directories (e.g. `$SCRATCH`) before executing the instructions below.

KNL

The batch script below can be used to run a ImpactX simulation on 2 KNL nodes on the supercomputer Cori at NERSC. Replace descriptions between chevrons <> by relevant values, for instance `<job name>` could be `laserWakefield`.

Do not forget to first `source $HOME/knl_impactx.profile` if you have not done so already for this terminal session.

For PICMI Python runs, the `<path/to/executable>` has to read `python3` and the `<input file>` is the path to your PICMI input script.

```
#!/bin/bash -l

# Copyright 2019-2023 Maxence Thevenet
#
# This file is part of ImpactX.
#
# License: BSD-3-Clause-LBNL


#SBATCH -N 2
#SBATCH -t 01:00:00
#SBATCH -q regular
#SBATCH -C knl
#SBATCH -S 4
#SBATCH -J <job name>
#SBATCH -A <allocation ID>
#SBATCH -e ImpactX.e%j
#SBATCH -o ImpactX.o%j

export OMP_PLACES=threads
export OMP_PROC_BIND=spread

# KNLs have 4 hyperthreads max
export CORI_MAX_HYPETHREAD_LEVEL=4
# We use 64 cores out of the 68 available on Cori KNL,
# and leave 4 to the system (see "#SBATCH -S 4" above).
export CORI_NCORES_PER_NODE=64

# Typically use 8 MPI ranks per node without hyperthreading,
# i.e., OMP_NUM_THREADS=8
export IMPACTX_NMPI_PER_NODE=8
export IMPACTX_HYPERTHREAD_LEVEL=1

# Compute OMP_NUM_THREADS and the thread count (-c option)
export CORI_NHYPERTHREADS_MAX=$(( ${CORI_MAX_HYPETHREAD_LEVEL} * ${CORI_NCORES_PER_NODE}␣
↪))
export IMPACTX_NTHREADS_PER_NODE=$(( ${IMPACTX_HYPERTHREAD_LEVEL} * ${CORI_NCORES_PER_
↪NODE} ))
export OMP_NUM_THREADS=$(( ${IMPACTX_NTHREADS_PER_NODE} / ${IMPACTX_NMPI_PER_NODE} ))
```

(continues on next page)

(continued from previous page)

```
export IMPACTX_THREAD_COUNT=$(( ${CORI_NHYPERTHREADS_MAX} / ${IMPACTX_NMPI_PER_NODE} ))

# for async_io support: (optional)
export MPICH_MAX_THREAD_SAFETY=multiple

srun --cpu_bind=cores -n $(( ${SLURM_JOB_NUM_NODES} * ${IMPACTX_NMPI_PER_NODE} )) -c $
↪{IMPACTX_THREAD_COUNT} \
  <path/to/executable> <input file> \
  > output.txt
```

To run a simulation, copy the lines above to a file `batch_cori.sh` and run

```
sbatch batch_cori.sh
```

to submit the job.

For a 3D simulation with a few (1-4) particles per cell using FDTD Maxwell solver on Cori KNL for a well load-balanced problem (in our case laser wakefield acceleration simulation in a boosted frame in the quasi-linear regime), the following set of parameters provided good performance:

- `amr.max_grid_size=64` and `amr.blocking_factor=64` so that the size of each grid is fixed to `64**3` (we are not using load-balancing here).
- **8 MPI ranks per KNL node**, with `OMP_NUM_THREADS=8` (that is 64 threads per KNL node, i.e. 1 thread per physical core, and 4 cores left to the system).
- **2 grids per MPI**, *i.e.*, 16 grids per KNL node.

Haswell

The batch script below can be used to run a ImpactX simulation on 1 Haswell node on the supercomputer Cori at NERSC.

Do not forget to first `source $HOME/haswell_impactx.profile` if you have not done so already for this terminal session.

```
#!/bin/bash -l

# Just increase this number of you need more nodes.
#SBATCH -N 1
#SBATCH -t 03:00:00
#SBATCH -q regular
#SBATCH -C haswell
#SBATCH -J <job name>
#SBATCH -A <allocation ID>
#SBATCH -e ImpactX.e%j
#SBATCH -o ImpactX.o%j
# one MPI rank per half-socket (see below)
#SBATCH --tasks-per-node=4
# request all logical (virtual) cores per half-socket
#SBATCH --cpus-per-task=16


# each Cori Haswell node has 2 sockets of Intel Xeon E5-2698 v3
```

(continues on next page)

(continued from previous page)

```
# each Xeon CPU is divided into 2 bus rings that each have direct L3 access
export IMPACTX_NMPI_PER_NODE=4

# each MPI rank per half-socket has 8 physical cores
#   or 16 logical (virtual) cores
# over-subscribing each physical core with 2x
#   hyperthreading leads to a slight (3.5%) speedup
# the settings below make sure threads are close to the
#   controlling MPI rank (process) per half socket and
#   distribute equally over close-by physical cores and,
#   for N>8, also equally over close-by logical cores
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=16

# for async_io support: (optional)
export MPICH_MAX_THREAD_SAFETY=multiple

EXE="<path/to/executable>"

srun --cpu_bind=cores -n $(( ${SLURM_JOB_NUM_NODES} * ${IMPACTX_NMPI_PER_NODE} )) \
  ${EXE} <input file> \
  > output.txt
```

To run a simulation, copy the lines above to a file `batch_cori_haswell.sh` and run

```
sbatch batch_cori_haswell.sh
```

to submit the job.

For a 3D simulation with a few (1-4) particles per cell using FDTD Maxwell solver on Cori Haswell for a well load-balanced problem (in our case laser wakefield acceleration simulation in a boosted frame in the quasi-linear regime), the following set of parameters provided good performance:

- **4 MPI ranks per Haswell node** (2 MPI ranks per Intel Xeon E5-2698 v3), with `OMP_NUM_THREADS=16` (which uses 2x hyperthreading)

GPU (V100)

Do not forget to first `source $HOME/gpu_impactx.profile` if you have not done so already for this terminal session.

Due to the limited amount of GPU development nodes, just request a single node with the above defined `getNode` function. For single-node runs, try to run one grid per GPU.

A multi-node batch script template can be found below:

```
#!/bin/bash -l

# Copyright 2021-2023 Axel Huebl
# This file is part of ImpactX.
# License: BSD-3-Clause-LBNL
#
# Ref:
```

(continues on next page)

(continued from previous page)

```
# - https://docs-dev.nersc.gov/cgpu/hardware/
# - https://docs-dev.nersc.gov/cgpu/access/
# - https://docs-dev.nersc.gov/cgpu/usage/#controlling-task-and-gpu-binding

# Just increase this number of you need more nodes.
#SBATCH -N 2
#SBATCH -t 03:00:00
#SBATCH -J <job name>
#SBATCH -A m1759
#SBATCH -q regular
#SBATCH -C gpu
# 8 V100 GPUs (16 GB) per node
#SBATCH --gres=gpu:8
#SBATCH --exclusive
# one MPI rank per GPU (a quarter-socket)
#SBATCH --tasks-per-node=8
# request all logical (virtual) cores per quarter-socket
#SBATCH --cpus-per-task=10
#SBATCH -e ImpactX.e%j
#SBATCH -o ImpactX.o%j


# each Cori GPU node has 2 sockets of Intel Xeon Gold 6148 ('Skylake') @ 2.40 GHz
export IMPACTX_NMPI_PER_NODE=8

# each MPI rank per half-socket has 10 physical cores
#   or 20 logical (virtual) cores
# we split half-sockets again by 2 to have one MPI rank per GPU
# over-subscribing each physical core with 2x
#   hyperthreading leads to often to slight speedup on Intel
# the settings below make sure threads are close to the
#   controlling MPI rank (process) per half socket and
#   distribute equally over close-by physical cores and,
#   for N>20, also equally over close-by logical cores
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=10

# for async_io support: (optional)
export MPICH_MAX_THREAD_SAFETY=multiple

EXE="<path/to/executable>"

srun --cpu_bind=cores --gpus-per-task=1 --gpu-bind=map_gpu:0,1,2,3,4,5,6,7 \
  -n $(( ${SLURM_JOB_NUM_NODES} * ${IMPACTX_NMPI_PER_NODE} )) \
  ${EXE} <input file> \
  > output.txt
```

Post-Processing

For post-processing, most users use Python via NERSC's Jupyter service (Docs).

As a one-time preparatory setup, create your own Conda environment as described in NERSC docs. In this manual, we often use this `conda create` line over the officially documented one:

```
conda create -n myenv -c conda-forge python mamba ipykernel ipympl matplotlib numpy␣
↪pandas yt openpmd-viewer openpmd-api h5py fast-histogram
```

We then follow the Customizing Kernels with a Helper Shell Script section to finalize the setup of using this conda-environment as a custom Jupyter kernel.

When opening a Jupyter notebook, just select the name you picked for your custom kernel on the top right of the notebook.

Additional software can be installed later on, e.g., in a Jupyter cell using `!mamba install -c conda-forge ...`. Software that is not available via conda can be installed via `!python -m pip install ...`.

Perlmutter (NERSC)

Warning: Perlmutter is still in acceptance testing. This page documents our internal testing workflow only.

The Perlmutter cluster is located at NERSC.

If you are new to this system, please see the following resources:

- NERSC user guide
- Batch system: Slurm
- Jupyter service
- Production directories:
  - `$PSCRATCH`: per-user production directory (<TBD>TB)
  - `/global/cscratch1/sd/m3239`: shared production directory for users in the project `m3239` (50TB)
  - `/global/cfs/cdirs/m3239/`: community file system for users in the project `m3239` (100TB)

Installation

Use the following commands to download the ImpactX source code and switch to the correct branch:

```
git clone https://github.com/ECP-WarpX/impactx.git $HOME/src/impactx
```

We use the following modules and environments on the system (`$HOME/perlmutter_impactx.profile`).

```
# please set your project account
export proj=<yourProject>  # LBNL/AMP: m3906_g

# required dependencies
module load cmake/3.22.0
module load cray-hdf5-parallel/1.12.2.1
```

(continues on next page)

(continued from previous page)

```
# optional: just an additional text editor
#module load nano  # TODO: request from support

# optional: CCache for faster rebuilds
export PATH=/global/common/software/spackecp/perlmutter/e4s-22.05/78535/spack/opt/spack/
↪cray-sles15-zen3/gcc-11.2.0/ccache-4.5.1-ybl7xefvggn6hov4dsdxxnztji74tolj/bin:$PATH

# Python
module load cray-python/3.9.13.1
if [ -d "$HOME/sw/perlmutter/venvs/impactx" ]
then
    source $HOME/sw/perlmutter/venvs/impactx/bin/activate
fi

# an alias to request an interactive batch node for one hour
#   for parallel execution, start on the batch node: srun <command>
alias getNode="salloc -N 1 --ntasks-per-gpu=1 -t 1:00:00 -q interactive -C gpu --gpu-
↪bind=single:1 -c 32 -G 4 -A $proj"
# an alias to run a command on a batch node for up to 30min
#   usage: runNode <command>
alias runNode="srun -N 1 --ntasks-per-gpu=1 -t 0:30:00 -q interactive -C gpu --gpu-
↪bind=single:1 -c 32 -G 4 -A $proj"

# GPU-aware MPI
export MPICH_GPU_SUPPORT_ENABLED=1

# necessary to use CUDA-Aware MPI and run a job
export CRAY_ACCEL_TARGET=nvidia80

# optimize CUDA compilation for A100
export AMREX_CUDA_ARCH=8.0

# compiler environment hints
export CC=cc
export CXX=CC
export FC=ftn
export CUDACXX=$(which nvcc)
export CUDAHOSTCXX=CC
```

We recommend to store the above lines in a file, such as `$HOME/perlmutter_impactx.profile`, and load it into your shell after a login:

```
source $HOME/perlmutter_impactx.profile
```

For Python workflows & tests, also install a virtual environment:

```
# establish Python dependencies
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv

python3 -m venv $HOME/sw/perlmutter/venvs/impactx
```

(continues on next page)

(continued from previous page)

```
source $HOME/sw/perlmutter/venvs/impactx/bin/activate

python3 -m pip install --upgrade pip
MPICC="cc -target-accel=nvidia80 -shared" python3 -m pip install -U --no-cache-dir -v␣
↪mpi4py
python3 -m pip install --upgrade pytest
python3 -m pip install -r $HOME/src/impactx/requirements.txt
python3 -m pip install -r $HOME/src/impactx/examples/requirements.txt
```

Then, cd into the directory `$HOME/src/impactx` and use the following commands to compile:

```
cd $HOME/src/impactx
rm -rf build

cmake -S . -B build_perlmutter -DImpactX_COMPUTE=CUDA -DImpactX_PYTHON=ON
cmake --build build_perlmutter -j 32
```

To run all tests, do:

```
# work on an interactive node
salloc -N 1 --ntasks-per-node=4 -t 1:00:00 -C gpu --gpu-bind=single:1 -c 32 -G 4 --
↪qos=interactive -A m3906_g

# test
ctest --test-dir build_perlmutter -E AMReX --output-on-failure
```

The general *cmake compile-time options* apply as usual.

Running

A100 GPUs (40 GB)

The batch script below can be used to run a ImpactX simulation on multiple nodes (change `-N` accordingly) on the supercomputer Perlmutter at NERSC. Replace descriptions between chevrons <> by relevant values, for instance `<input file>` could be `plasma_mirror_inputs`. Note that we run one MPI rank per GPU.

```
#!/bin/bash -l

# Copyright 2021-2023 Axel Huebl, Kevin Gott
#
# This file is part of WarpX.
#
# License: BSD-3-Clause-LBNL

#SBATCH -t 01:00:00
#SBATCH -N 4
#SBATCH -J WarpX
#    note: <proj> must end on _g
#SBATCH -A <proj>
#SBATCH -q regular
#SBATCH -C gpu
```

(continues on next page)

(continued from previous page)

```
#SBATCH -c 32
#SBATCH --ntasks-per-gpu=1
#SBATCH --gpus-per-node=4
#SBATCH -o WarpX.o%j
#SBATCH -e WarpX.e%j

# GPU-aware MPI
export MPICH_GPU_SUPPORT_ENABLED=1

# expose one GPU per MPI rank
export CUDA_VISIBLE_DEVICES=$SLURM_LOCALID

EXE=./impactx
#EXE=../build/bin/impactx.3d.MPI.CUDA.DP
INPUTS=inputs_small

srun ${EXE} ${INPUTS} \
  > output.txt
```

To run a simulation, copy the lines above to a file `batch_perlmutter.sh` and run

```
sbatch batch_perlmutter.sh
```

to submit the job.

Post-Processing

For post-processing, most users use Python via NERSC's Jupyter service (Docs).

Please follow the same guidance as for *NERSC Cori post-processing*.

The Perlmutter `$PSCRATCH` filesystem is currently not yet available on Jupyter. Thus, store or copy your data to Cori's `$SCRATCH` or use the Community FileSystem (CFS) for now.

Tip: Your HPC system is not in the list? Open an issue and together we can document it!

CHAPTER

THREE

USAGE

3.1 Run ImpactX

In order to run a new simulation:

1. create a **new directory**, where the simulation will be run
2. make sure the ImpactX **executable** is either copied into this directory or in your `PATH` environment variable
3. add an **inputs file** and on *HPC systems* a **submission script** to the directory
4. run

3.1.1 1. Run Directory

On Linux/macOS, this is as easy as this

```
mkdir -p <run_directory>
```

Where `<run_directory>` by the actual path to the run directory.

3.1.2 2. Executable

If you installed ImpactX with a *package manager*, a `impactx`-prefixed executable will be available as a regular system command to you. Depending on the choosen build options, the name is suffixed with more details. Try it like this:

```
impactx<TAB>
```

Hitting the `<TAB>` key will suggest available ImpactX executables as found in your `PATH` environment variable.

If you *compiled the code yourself*, the ImpactX executable is stored in the source folder under `build/bin`. We also create a symbolic link that is just called `impactx` that points to the last executable you built, which can be copied, too. Copy the **executable** to this directory:

```
cp build/bin/<impactx_executable> <run_directory>/
```

where `<impactx_executable>` should be replaced by the actual name of the executable (see above) and `<run_directory>` by the actual path to the run directory.

3.1.3 3. Inputs

Add an **input file** in the directory (see *examples* and *parameters*). This file contains the numerical and physical parameters that define the situation to be simulated.

On *HPC systems*, also copy and adjust a submission script that allocated computing nodes for you. Please *reach out to us* if you need help setting up a template that runs with ideal performance.

3.1.4 4. Run

Run the executable, e.g. with MPI:

```
cd <run_directory>

# run with an inputs file:
mpirun -np <n_ranks> ./impactx <input_file>
```

or

```
# run with a PICMI input script:
mpirun -np <n_ranks> python <python_script>
```

Here, `<n_ranks>` is the number of MPI ranks used, and `<input_file>` is the name of the input file (`<python_script>` is the name of the *PICMI* script). Note that the actual executable might have a longer name, depending on build options.

We used the copied executable in the current directory (`./`); if you installed with a package manager, skip the `./` because ImpactX is in your `PATH`.

On an *HPC system*, you would instead submit the *job script* at this point, e.g. `sbatch <submission_script>` (SLURM on Cori/NERSC) or `bsub <submission_script>` (LSF on Summit/OLCF).

Tip: In the *next sections*, we will explain parameters of the `<input_file>`. You can overwrite all parameters inside this file also from the command line, e.g.:

```
mpirun -np 4 ./impactx <input_file> max_step=10 impactx.numprocs=1 2 2
```

3.2 Parameters: Python

This documents on how to use ImpactX as a Python script (`python3 run_script.py`).

3.2.1 General

class impactx.**ImpactX**

This is the central simulation class.

property particle_shape

Control the particle B-spline order.

The order of the shape factors (splines) for the macro-particles along all spatial directions: *1* for linear, *2* for quadratic, *3* for cubic. Low-order shape factors result in faster simulations, but may lead to more

noisy results. High-order shape factors are computationally more expensive, but may increase the overall accuracy of the results. For production runs it is generally safer to use high-order shape factors, such as cubic order.

Parameters
order (*int*) – B-spline order 1, 2, or 3

property n_cell

The number of grid points along each direction on the coarsest level.

property domain

The physical extent of the full simulation domain, relative to the reference particle position, in meters. When set, turns `dynamic_size` to `False`.

Note: particles that move outside the simulation domain are removed.

property prob_relative

The field mesh is expanded beyond the physical extent of particles by this factor. When set, turns `dynamic_size` to `True`.

property dynamic_size

Use dynamic (`True`) resizing of the field mesh or static sizing (`False`).

property space_charge

Enable (`True`) or disable (`False`) space charge calculations (default: `True`).

Whether to calculate space charge effects. This is in-development. At the moment, this flag only activates coordinate transformations and charge deposition.

property diagnostics

Enable (`True`) or disable (`False`) diagnostics generally (default: `True`). Disabling this is mostly used for benchmarking.

property slice_step_diagnostics

Enable (`True`) or disable (`False`) diagnostics every slice step in elements (default: `True`).

By default, diagnostics is performed at the beginning and end of the simulation. Enabling this flag will write diagnostics every step and slice step.

property diag_file_min_digits

The minimum number of digits (default: 6) used for the step number appended to the diagnostic file names.

init_grids()

Initialize AMReX blocks/grids for domain decomposition & space charge mesh.

This must come first, before particle beams and lattice elements are initialized.

add_particles(*charge_C*, *distr*, *npart*)

Generate and add n particles to the particle container. Note: Set the reference particle properties (charge, mass, energy) first.

Will also resize the geometry based on the updated particle distribution's extent and then redistribute particles in according AMReX grid boxes.

Parameters

- **charge_C** (*float*) – bunch charge (C)
- **distr** – distribution function to draw from (object from *impactx.distribution*)
- **npart** (*int*) – number of particles to draw

particle_container()

Access the beam particle container (*impactx.ParticleContainer*).

property lattice

Access the elements in the accelerator lattice. See *impactx.elements* for lattice elements.

property abort_on_warning_threshold

(optional) Set to “low”, “medium” or “high”. Cause the code to abort if a warning is raised that exceeds the warning threshold.

property abort_on_unused_inputs

Set to 1 to cause the simulation to fail *after* its completion if there were unused parameters. (default: `0` for false) It is mainly intended for continuous integration and automated testing to check that all tests and inputs are adapted to API changes.

property always_warn_immediately

If set to 1, ImpactX immediately prints every warning message as soon as it is generated. (default: `0` for false) It is mainly intended for debug purposes, in case a simulation crashes before a global warning report can be printed.

evolve()

Run the main simulation loop for a number of steps.

class impactx.Config

Configuration information on ImpactX that were set at compile-time.

property have_mpi

Indicates multi-process/multi-node support via the message-passing interface (MPI). Possible values: `True/False`

Note: Particle beam particles are not yet dynamically load balanced. Please see the progress in issue 198.

property have_gpu

Indicates GPU support. Possible values: `True/False`

property gpu_backend

Indicates the available GPU support. Possible values: `None`, `"CUDA"` (for Nvidia GPUs), `"HIP"` (for AMD GPUs) or `"SYCL"` (for Intel GPUs).

property have_omp

Indicates multi-threaded CPU support via OpenMP. Possible values: `True/False`\`

Set the environment variable `OMP_NUM_THREADS` to control the number of threads.

Note: Not yet implemented. Please see the progress in issue 195.

Warning: By default, OpenMP spawns as many threads as there are available virtual cores on a host. When MPI and OpenMP support are used at the same time, it can easily happen that one over-subscribes the available physical CPU cores. This will lead to a severe slow-down of the simulation.

By setting appropriate environment variables for OpenMP, ensure that the number of MPI processes (ranks) per node multiplied with the number of OpenMP threads is equal to the number of physical (or

virtual) CPU cores. Please see our examples in the *high-performance computing (HPC)* on how to run efficiently in parallel environments such as supercomputers.

3.2.2 Particles

class impactx.**ParticleContainer**

Beam Particles in ImpactX.

This class stores particles, distributed over MPI ranks.

add_n_particles(*lev*, *x*, *y*, *z*, *px*, *py*, *pyz*, *qm*, *bchchg*)

Add new particles to the container

Note: This can only be used *after* the initialization (grids) have
been created, meaning after the call to `ImpactX.init_grids()` has been made in the ImpactX class.

Parameters

- **lev** – mesh-refinement level
- **x** – positions in x
- **y** – positions in y
- **z** – positions in z
- **px** – momentum in x
- **py** – momentum in y
- **pz** – momentum in z
- **qm** – charge over mass in 1/eV
- **bchchg** – total charge within a bunch in C

ref_particle()

Access the reference particle (*impactx.RefPart*).

Returns
return a data reference to the reference particle

Return type
impactx.RefPart

set_ref_particle(*refpart*)

Set reference particle attributes.

Parameters
refpart (impactx.RefPart) – a reference particle to copy all attributes from

class impactx.**RefPart**

This struct stores the reference particle attributes stored in *impactx.ParticleContainer*.

property s

integrated orbit path length, in meters

property x

horizontal position x, in meters

property y

vertical position y, in meters

property z

longitudinal position y, in meters

property t

clock time * c in meters

property px

momentum in x, normalized to proper velocity

property py

momentum in y, normalized to proper velocity

property pz

momentum in z, normalized to proper velocity

property pt

energy deviation, normalized by rest energy

property gamma

Read-only: Get reference particle relativistic gamma.

property beta

Read-only: Get reference particle relativistic beta.

property beta_gamma

Read-only: Get reference particle beta*gamma

property qm_qeeV

Read-only: Get reference particle charge to mass ratio (elementary charge/eV)

set_charge_qe(*charge_qe*)

Write-only: Set reference particle charge in (positive) elementary charges.

set_mass_MeV(*massE*)

Write-only: Set reference particle rest mass (MeV/c^2).

set_energy_MeV(*energy_MeV*)

Write-only: Set reference particle energy.

load_file(*madx_file*)

Load reference particle information from a MAD-X file.

Parameters

madx_file – file name to MAD-X file with a `BEAM` entry

3.2.3 Initial Beam Distributions

This module provides particle beam distributions that can be used to initialize particle beams in an *impactx.ParticleContainer*.

class impactx.distribution.**Gaussian**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A 6D Gaussian distribution.

Parameters

- **sigx** – for zero correlation, these are the related RMS sizes (in meters)
- **sigy** – see sigx
- **sigt** – see sigx
- **sigpx** – RMS momentum
- **sigpy** – see sigpx
- **sigpt** – see sigpx
- **muxpx** – correlation length-momentum
- **muypy** – see muxpx
- **mutpt** – see muxpx

class impactx.distribution.**Kurth4D**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A 4D Kurth distribution transversely + a uniform distribution it t + a Gaussian distribution in pt.

class impactx.distribution.**Kurth6D**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A 6D Kurth distribution.

R. Kurth, Quarterly of Applied Mathematics vol. 32, pp. 325-329 (1978) C. Mitchell, K. Hwang and R. D. Ryne, IPAC2021, WEPAB248 (2021)

class impactx.distribution.**KVdist**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A K-V distribution transversely + a uniform distribution it t + a Gaussian distribution in pt.

class impactx.distribution.**None**

This distribution does nothing.

class impactx.distribution.**Semigaussian**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A 6D Semi-Gaussian distribution (uniform in position, Gaussian in momentum).

class impactx.distribution.**Waterbag**(*sigx*, *sigy*, *sigt*, *sigpx*, *sigpy*, *sigpt*, *muxpx=0.0*, *muypy=0.0*, *mutpt=0.0*)

A 6D Waterbag distribution.

3.2.4 Lattice Elements

This module provides elements for the accelerator lattice.

class impactx.elements.**KnownElementsList**

An iterable, `list`-like type of elements.

clear()

Clear the list to become empty.

extend(*list*)

Add a list of elements to the list.

append(*element*)

Add a single element to the list.

load_file(*madx_file*, *nslice=1*)

Load and append an accelerator lattice description from a MAD-X file.

Parameters

- **madx_file** – file name to MAD-X file with beamline elements
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**ConstF**(*ds*, *kx*, *ky*, *kt*, *nslice=1*)

A linear Constant Focusing element.

Parameters

- **ds** – Segment length in m.
- **kx** – Focusing strength for x in 1/m.
- **ky** – Focusing strength for y in 1/m.
- **kt** – Focusing strength for t in 1/m.
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**DipEdge**(*psi*, *rc*, *g*, *K2*)

Edge focusing associated with bend entry or exit

This model assumes a first-order effect of nonzero gap. Here we use the linear fringe field map, given to first order in g/rc (gap / radius of curvature).

References: * K. L. Brown, SLAC Report No. 75 (1982). * K. Hwang and S. Y. Lee, PRAB 18, 122401 (2015).

Parameters

- **psi** – Pole face angle in rad
- **rc** – Radius of curvature in m
- **g** – Gap parameter in m
- **K2** – Fringe field integral (unitless)

class impactx.elements.**Drift**(*ds*, *nslice=1*)

A drift.

Parameters

- **ds** – Segment length in m
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**Multipole**(*multipole*, *K_normal*, *K_skew*)

A general thin multipole element.

Parameters

- **multipole** – index m (m=1 dipole, m=2 quadrupole, m=3 sextupole etc.)
- **K_normal** – Integrated normal multipole coefficient (1/meter^m)
- **K_skew** – Integrated skew multipole coefficient (1/meter^m)

class impactx.elements.**NonlinearLens**(*knll*, *cnll*)

Single short segment of the nonlinear magnetic insert element.

A thin lens associated with a single short segment of the nonlinear magnetic insert described by V. Danilov and S. Nagaitsev, PRSTAB 13, 084002 (2010), Sect. V.A. This element appears in MAD-X as type NLLENS.

Parameters

- **knll** – integrated strength of the nonlinear lens (m)
- **cnll** – distance of singularities from the origin (m)

class impactx.elements.**BeamMonitor**(*name*, *backend='default'*, *encoding='g'*)

A beam monitor, writing all beam particles at fixed `s` to openPMD files.

If the same element `name` is used multiple times, then an output series is created with multiple outputs.

The I/O backend for openPMD data dumps. `bp` is the ADIOS2 I/O library, `h5` is the HDF5 format, and `json` is a simple text format. `json` only works with serial/single-rank jobs. By default, the first available backend in the order given above is taken.

openPMD iteration encoding determines if multiple files are created for individual output steps or not. Variable based is an experimental feature with ADIOS2.

Parameters

- **name** – name of the series
- **backend** – I/O backend, e.g., `bp`, `h5`, `json`
- **encoding** – openPMD iteration encoding: (v)ariable based, (f)ile based, (g)roup based (default)

class impactx.elements.**Programmable**

A programmable beam optics element.

This element can be programmed to receive callback hooks into Python functions.

property **beam_particles**

This is a function hook for pushing all beam particles. This accepts a function or lambda with the following arguments:

user_defined_function(*pti: ImpactXParIter*, *refpart:* RefPart)

This function is called repeatedly for all particle tiles or boxes in the beam particle container. Particles can be pushed and are relative to the reference particle

property **ref_particle**

This is a function hook for pushing the reference particle. This accepts a function or lambda with the following argument:

another_user_defined_function(*refpart:* RefPart)

This function is called for the reference particle as it passes through the element. The reference particle is updated *before* the beam particles are pushed.

class impactx.elements.**Quad**(*ds*, *k*, *nslice=1*)

A Quadrupole magnet.

Parameters

- **ds** – Segment length in m.
- **k** – Quadrupole strength in m^(-2) (MADX convention) = (gradient in T/m) / (rigidity in T-m) k > 0 horizontal focusing k < 0 horizontal defocusing
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**RFCavity**(*ds*, *escale*, *freq*, *phase*, *mapsteps*, *nslice*)

A radiofrequency cavity.

Parameters

- **ds** – Segment length in m.
- **escale** – scaling factor for on-axis RF electric field in 1/m = (peak on-axis electric field Ez in MV/m) / (particle rest energy in MeV)
- **freq** – RF frequency in Hz
- **phase** – RF driven phase in degrees
- **cos_coefficients** – array of `float` cosine coefficients in Fourier expansion of on-axis electric field Ez (optional); default is a 9-cell TESLA superconducting cavity model from DOI:10.1103/PhysRevSTAB.3.092001
- **cos_coefficients** – array of `float` sine coefficients in Fourier expansion of on-axis electric field Ez (optional); default is a 9-cell TESLA superconducting cavity model from DOI:10.1103/PhysRevSTAB.3.092001
- **mapsteps** – number of integration steps per slice used for map and reference particle push in applied fields
- **nslice** – number of slices used for the application of space charge

`class impactx.elements.Sbend(ds, rc, nslice=1)`

An ideal sector bend.

Parameters

- **ds** – Segment length in m.
- **rc** – Radius of curvature in m.
- **nslice** – number of slices used for the application of space charge

`class impactx.elements.ShortRF(V, k)`

A short RF cavity element at zero crossing for bunching.

Parameters

- **V** – Normalized RF voltage drop V = Emax*L/(c*Brho)
- **k** – Wavenumber of RF in 1/m

`class impactx.elements.SoftSolenoid(ds, bscale, cos_coefficients, sin_coefficients, nslice=1)`

A soft-edge solenoid.

Parameters

- **ds** – Segment length in m.
- **bscale** – Scaling factor for on-axis magnetic field Bz in inverse meters
- **cos_coefficients** – array of `float` cosine coefficients in Fourier expansion of on-axis magnetic field Bz (optional); default is a thin-shell model from DOI:10.1016/J.NIMA.2022.166706
- **sin_coefficients** – array of `float` sine coefficients in Fourier expansion of on-axis magnetic field Bz (optional); default is a thin-shell model from DOI:10.1016/J.NIMA.2022.166706
- **mapsteps** – number of integration steps per slice used for map and reference particle push in applied fields
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**Sol**(*ds*, *ks*, *nslice=1*)

An ideal hard-edge Solenoid magnet.

Parameters

- **ds** – Segment length in m.
- **ks** – Solenoid strength in m^(-1) (MADX convention) in (magnetic field Bz in T) / (rigidity in T-m)
- **nslice** – number of slices used for the application of space charge

class impactx.elements.**SoftQuadrupole**(*ds*, *gscale*, *cos_coefficients*, *sin_coefficients*, *nslice=1*)

A soft-edge quadrupole.

Parameters

- **ds** – Segment length in m.
- **gscale** – Scaling factor for on-axis field gradient in inverse meters
- **cos_coefficients** – array of `float` cosine coefficients in Fourier expansion of on-axis field gradient (optional); default is a tanh fringe field model based on http://www.physics.umd.edu/dsat/docs/MaryLieMan.pdf
- **sin_coefficients** – array of `float` sine coefficients in Fourier expansion of on-axis field gradient (optional); default is a tanh fringe field model based on http://www.physics.umd.edu/dsat/docs/MaryLieMan.pdf
- **mapsteps** – number of integration steps per slice used for map and reference particle push in applied fields
- **nslice** – number of slices used for the application of space charge

3.3 Parameters: Inputs File

This documents on how to use ImpactX with an inputs file (`impactx input_file.in`).

Note: The AMReX parser (see *Math parser and user-defined constants*) is used for the right-hand-side of all input parameters that consist of one or more integers or floats, so expressions like `<species_name>.density_max = "2.+1."` and/or using user-defined constants are accepted.

3.3.1 Overall simulation parameters

- **`max_step` (`integer`)**
  The number of PIC cycles to perform.
- **`stop_time` (`float`; in seconds)**
  The maximum physical time of the simulation. Can be provided instead of `max_step`. If both `max_step` and `stop_time` are provided, both criteria are used and the simulation stops when the first criterion is hit.
- **`amrex.abort_on_out_of_gpu_memory` (`0` or `1`; default is `1` for true)**
  When running on GPUs, memory that does not fit on the device will be automatically swapped to host memory when this option is set to `0`. This will cause severe performance drops. Note that even with this set to `1` ImpactX will not catch all out-of-memory events yet when operating close to maximum device memory. Please also see the documentation in AMReX.

- **amrex.abort_on_unused_inputs (0 or 1; default is 0 for false)**
  When set to 1, this option causes the simulation to fail *after* its completion if there were unused parameters. It is mainly intended for continuous integration and automated testing to check that all tests and inputs are adapted to API changes.
- **impactx.always_warn_immediately (0 or 1; default is 0 for false)**
  If set to 1, ImpactX immediately prints every warning message as soon as it is generated. It is mainly intended for debug purposes, in case a simulation crashes before a global warning report can be printed.
- **impactx.abort_on_warning_threshold (string: low, medium or high) optional**
  Optional threshold to abort as soon as a warning is raised. If the threshold is set, warning messages with priority greater than or equal to the threshold trigger an immediate abort. It is mainly intended for debug purposes, and is best used with `impactx.always_warn_immediately=1`. For more information on the warning logger, see this section of the WarpX documentation.

3.3.2 Setting up the field mesh

- **amr.n_cell (3 integers) optional (default: 1 blocking_factor per MPI process)**
  The number of grid points along each direction (on the **coarsest level**)
- **amr.max_level (integer, default: 0)**
  When using mesh refinement, the number of refinement levels that will be used.

  Use `0` in order to disable mesh refinement.
- **amr.ref_ratio (integer per refined level, default: 2)**
  When using mesh refinement, this is the refinement ratio per level. With this option, all directions are fined by the same ratio.
- **amr.ref_ratio_vect (3 integers for x,y,z per refined level)**
  When using mesh refinement, this can be used to set the refinement ratio per direction and level, relative to the previous level.

  Example: for three levels, a value of `2 2 4 8 8 16` refines the first level by 2-fold in x and y and 4-fold in z compared to the coarsest level (level 0/mother grid); compared to the first level, the second level is refined 8-fold in x and y and 16-fold in z.

Note: Field boundaries for space charge calculation are located at the outer ends of the field mesh. We currently assume Dirichlet boundary conditions with zero potential (a mirror charge). Thus, to emulate open boundaries, consider adding enough vacuum padding to the beam. This will be improved in future versions.

Note: Particles that move outside the simulation domain are removed.

- **geometry.dynamic_size (boolean) optional (default: true for dynamic)**
  Use dynamic (`true`) resizing of the field mesh, via `geometry.prob_relative`, or static sizing (`false`), via `geometry.prob_lo`/`geometry.prob_hi`.
- **geometry.prob_relative (positive float, unitless) optional (default: 1.0)**
  By default, we dynamically extract the minimum and maximum of the particle positions in the beam. The field mesh is expanded, per direction, beyond the physical extent of particles by this factor. For instance, `0.1` means 10% more cells above and below the beam for vacuum; `1.0` means twice as many cells as covered by the beam are used, per direction, for vacuum padding.

- **geometry.prob_lo and geometry.prob_hi (3 floats, in meters) optional (required if geometry.dynamic_size is false)**
  The extent of the full simulation domain relative to the reference particle position. This can be used to explicitly size the simulation box and ignore `geometry.prob_relative`.

  This box is rectangular, and thus its extent is given here by the coordinates of the lower corner (`geometry.prob_lo`) and upper corner (`geometry.prob_hi`). The first axis of the coordinates is x and the last is z.

3.3.3 Domain Boundary Conditions

Note: TODO :-)

3.3.4 Initial Beam Distributions

- **<distribution>.type (string)**
  Indicates the initial distribution type. This should be one of:
  - `waterbag` for initial Waterbag distribution. With additional parameters:
    - <distribution>.sigmaX (float, in meters) rms X
    - <distribution>.sigmaY (float, in meters) rms Y
    - <distribution>.sigmaT (float, in radian) rms normalized time difference T
    - <distribution>.sigmaPx (float, in momentum) rms Px
    - <distribution>.sigmaPy (float, in momentum) rms Py
    - <distribution>.sigmaPt (float, in energy deviation) rms Pt
    - <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
    - <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
    - <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt
  - `kurth6d` for initial 6D Kurth distribution. With additional parameters:
    - <distribution>.sigmaX (float, in meters) rms X
    - <distribution>.sigmaY (float, in meters) rms Y
    - <distribution>.sigmaT (float, in radian) rms normalized time difference T
    - <distribution>.sigmaPx (float, in momentum) rms Px
    - <distribution>.sigmaPy (float, in momentum) rms Py
    - <distribution>.sigmaPt (float, in energy deviation) rms Pt
    - <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
    - <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
    - <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt
  - `gaussian` for initial 6D Gaussian (normal) distribution. With additional parameters:
    - <distribution>.sigmaX (float, in meters) rms X

    * <distribution>.sigmaY (float, in meters) rms Y
    * <distribution>.sigmaT (float, in radian) rms normalized time difference T
    * <distribution>.sigmaPx (float, in momentum) rms Px
    * <distribution>.sigmaPy (float, in momentum) rms Py
    * <distribution>.sigmaPt (float, in energy deviation) rms Pt
    * <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
    * <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
    * <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt
  - kvdist for initial K-V distribution in the transverse plane. The distribution is uniform in t and Gaussian in pt. With additional parameters:
    * <distribution>.sigmaX (float, in meters) rms X
    * <distribution>.sigmaY (float, in meters) rms Y
    * <distribution>.sigmaT (float, in radian) rms normalized time difference T
    * <distribution>.sigmaPx (float, in momentum) rms Px
    * <distribution>.sigmaPy (float, in momentum) rms Py
    * <distribution>.sigmaPt (float, in energy deviation) rms Pt
    * <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
    * <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
    * <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt
  - kurth4d for initial 4D Kurth distribution in the transverse plane. The distribution is uniform in t and Gaussian in pt. With additional parameters:
    * <distribution>.sigmaX (float, in meters) rms X
    * <distribution>.sigmaY (float, in meters) rms Y
    * <distribution>.sigmaT (float, in radian) rms normalized time difference T
    * <distribution>.sigmaPx (float, in momentum) rms Px
    * <distribution>.sigmaPy (float, in momentum) rms Py
    * <distribution>.sigmaPt (float, in energy deviation) rms Pt
    * <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
    * <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
    * <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt
  - semigaussian for initial Semi-Gaussian distribution. The distribution is uniform within a cylinder in (x,y,z) and Gaussian in momenta (px,py,pt). With additional parameters:
    * <distribution>.sigmaX (float, in meters) rms X
    * <distribution>.sigmaY (float, in meters) rms Y
    * <distribution>.sigmaT (float, in radian) rms normalized time difference T
    * <distribution>.sigmaPx (float, in momentum) rms Px
    * <distribution>.sigmaPy (float, in momentum) rms Py

* <distribution>.sigmaPt (float, in energy deviation) rms Pt
* <distribution>.muxpx (float, dimensionless, default: 0) correlation X-Px
* <distribution>.muypy (float, dimensionless, default: 0) correlation Y-Py
* <distribution>.mutpt (float, dimensionless, default: 0) correlation T-Pt

3.3.5 Lattice Elements

- **lattice.elements (list of strings) optional (default: no elements)**
  A list of names (one name per lattice element), in the order that they appear in the lattice.
- **lattice.nslice (integer) optional (default: 1)**
  A positive integer specifying the number of slices used for the application of space charge in all elements; overwritten by element parameter "nslice"
- **<element_name>.type (string)**
  Indicates the element type for this lattice element. This should be one of:
  - drift for free drift. This requires these additional parameters:
    * <element_name>.ds (float, in meters) the segment length
    * <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
  - quad for a quadrupole. This requires these additional parameters:
    * <element_name>.ds (float, in meters) the segment length
    * <element_name>.k (float, in inverse meters squared) the quadrupole strength
      = (magnetic field gradient in T/m) / (magnetic rigidity in T-m)
      · k > 0 horizontal focusing
      · k < 0 horizontal defocusing
    * <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
  - quadrupole_softedge for a soft-edge quadrupole. This requires these additional parameters:
    * <element_name>.ds (float, in meters) the segment length
    * <element_name>.gscale (float, in inverse meters) Scaling factor for on-axis magnetic field gradient
    * <element_name>.cos_coefficients (array of float) cos coefficients in Fourier expansion of the on-axis field gradient (optional); default is a tanh fringe field model from MaryLie 3.0
    * <element_name>.sin_coefficients (array of float) sin coefficients in Fourier expansion of the on-axis field gradient (optional); default is a tanh fringe field model from MaryLie 3.0
    * **<element_name>.mapsteps (integer) number of integration steps per slice used for map and reference particle push in applied fields**
      (default: 1)
    * <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
  - sbend for a bending magnet. This requires these additional parameters:
    * <element_name>.ds (float, in meters) the segment length

    - <element_name>.rc (float, in meters) the bend radius
    - <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
- solenoid for an ideal hard-edge solenoid magnet. This requires these additional parameters:
    - <element_name>.ds (float, in meters) the segment length
    - <element_name>.ks (float, in meters) Solenoid strength in m^(-1) (MADX convention)

      = (magnetic field Bz in T) / (rigidity in T-m)
    - <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
- solenoid_softedge for a soft-edge solenoid. This requires these additional parameters:
    - <element_name>.ds (float, in meters) the segment length
    - <element_name>.bscale (float, in inverse meters) Scaling factor for on-axis magnetic field Bz
    - <element_name>.cos_coefficients (array of float) cos coefficients in Fourier expansion of the on-axis magnetic field Bz (optional); default is a thin-shell model from DOI:10.1016/J.NIMA.2022.166706
    - <element_name>.sin_coefficients (array of float) sin coefficients in Fourier expansion of the on-axis magnetic field Bz (optional); default is a thin-shell model from DOI:10.1016/J.NIMA.2022.166706
    - <element_name>.mapsteps (integer) number of integration steps per slice used for map and reference particle push in applied fields (default: 1)
    - <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
- dipedge for dipole edge focusing. This requires these additional parameters:
    - <element_name>.psi (float, in radians) the pole face rotation angle
    - <element_name>.rc (float, in meters) the bend radius
    - <element_name>.g (float, in meters) the gap size
    - <element_name>.K2 (float, dimensionless) normalized field integral for fringe field
- constf for a constant focusing element. This requires these additional parameters:
    - <element_name>.ds (float, in meters) the segment length
    - <element_name>.kx (float, in 1/meters) the horizontal focusing strength
    - <element_name>.ky (float, in 1/meters) the vertical focusing strength
    - <element_name>.kt (float, in 1/meters) the longitudinal focusing strength
    - <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)
- rfcavity a radiofrequency cavity. This requires these additional parameters:
    - <element_name>.ds (float, in meters) the segment length
    - <element_name>.escale (float, in 1/m) scaling factor for on-axis RF electric field

      = (peak on-axis electric field Ez in MV/m) / (particle rest energy in MeV)

    * <element_name>.freq (float, in Hz) RF frequency
    * <element_name>.phase (float, in degrees) RF driven phase
    * <element_name>.cos_coefficients (array of float) cosine coefficients in Fourier expansion of on-axis electric field Ez (optional); default is a 9-cell TESLA superconducting cavity model from DOI:10.1103/PhysRevSTAB.3.092001
    * <element_name>.cos_coefficients (array of float) sine coefficients in Fourier expansion of on-axis electric field Ez (optional); default is a 9-cell TESLA superconducting cavity model from DOI:10.1103/PhysRevSTAB.3.092001
    * <element_name>.mapsteps (integer) number of integration steps per slice used for map and reference particle push in applied fields (default: 1)
    * <element_name>.nslice (integer) number of slices used for the application of space charge (default: 1)

- shortrf for a short RF (bunching) cavity element. This requires these additional parameters:
    * <element_name>.V (float, dimensionless) normalized voltage drop across the cavity

      = (maximum voltage drop in Volts) / (speed of light in m/s * magnetic rigidity in T-m)
    * <element_name>.k (float, in 1/meters) the RF wavenumber

      = 2*pi/(RF wavelength in m)
- multipole for a thin multipole element. This requires these additional parameters:
    * <element_name>.multipole (integer, dimensionless) order of multipole

      (m = 1) dipole, (m = 2) quadrupole, (m = 3) sextupole, etc.
    * <element_name>.k_normal (float, in 1/meters^m) integrated normal multipole coefficient (MAD-X convention)

      = 1/(magnetic rigidity in T-m) * (derivative of order m-1 of By with respect to x)
    * <element_name>.k_skew (float, in 1/meters^m) integrated skew multipole strength (MAD-X convention)
- nonlinear_lens for a thin IOTA nonlinear lens element. This requires these additional parameters:
    * <element_name>.knll (float, in meters) integrated strength of the lens segment (MAD-X convention)

      = dimensionless lens strength * c parameter**2 * length / Twiss beta
    * <element_name>.cnll (float, in meters) distance of the singularities from the origin (MAD-X convention)

      = c parameter * sqrt(Twiss beta)
- beam_monitor a beam monitor, writing all beam particles at fixed s to openPMD files. If the same element name is used multiple times, then an output series is created with multiple outputs.
    * <element_name>.name (string, default value: <element_name>)

      The output series name to use. By default, output is created under diags/openPMD/<element_name>.<backend>.
    * <element_name>.backend (string, default value: default)

      I/O backend for openPMD data dumps. bp is the ADIOS2 I/O library, h5 is the HDF5 format, and json is a simple text format. json only works with serial/single-rank jobs. By default, the first available backend in the order given above is taken.

* `<element_name>.encoding` (`string`, default value: `g`)

  openPMD iteration encoding: (v)ariable based, (f)ile based, (g)roup based (default) variable based is an experimental feature with ADIOS2.

3.3.6 Distribution across MPI ranks and parallelization

- **`amr.max_grid_size` (`integer`) optional (default: `128`)**
  Maximum allowable size of each **subdomain** (expressed in number of grid points, in each direction). Each subdomain has its own ghost cells, and can be handled by a different MPI rank ; several OpenMP threads can work simultaneously on the same subdomain.

  If `max_grid_size` is such that the total number of subdomains is **larger** that the number of MPI ranks used, than some MPI ranks will handle several subdomains, thereby providing additional flexibility for **load balancing**.

  When using mesh refinement, this number applies to the subdomains of the coarsest level, but also to any of the finer level.

3.3.7 Math parser and user-defined constants

ImpactX uses AMReX's math parser that reads expressions in the input file. It can be used in all input parameters that consist of one or more integers or floats. Integer input expecting boolean, 0 or 1, are not parsed. Note that when multiple values are expected, the expressions are space delimited. For integer input values, the expressions are evaluated as real numbers and the final result rounded to the nearest integer. See this section of the AMReX documentation for a complete list of functions supported by the math parser.

ImpactX constants

ImpactX will provide a few pre-defined constants, that can be used for any parameter that consists of one or more floats.

Note: Develop, such as:

q_e	elementary charge
m_e	electron mass
m_p	proton mass
m_u	unified atomic mass unit (Dalton)
epsilon0	vacuum permittivity
mu0	vacuum permeability
clight	speed of light
pi	math constant pi

See in WarpX the file `Source/Utils/WarpXConst.H` for the values.

User-defined constants

Users can define their own constants in the input file. These constants can be used for any parameter that consists of one or more integers or floats. User-defined constant names can contain only letters, numbers and the character _. The name of each constant has to begin with a letter. The following names are used by ImpactX, and cannot be used as user-defined constants: `x`, `y`, `z`, `X`, `Y`, `t`. The values of the constants can include the predefined ImpactX constants listed above as well as other user-defined constants. For example:

- `my_constants.a0 = 3.0`
- `my_constants.z_plateau = 150.e-6`
- `my_constants.n0 = 1.e22`
- `my_constants.wp = sqrt(n0*q_e**2/(epsilon0*m_e))`

Coordinates

Besides, for profiles that depend on spatial coordinates (the plasma momentum distribution or the laser field, see below `Particle initialization` and `Laser initialization`), the parser will interpret some variables as spatial coordinates. These are specified in the input parameter, i.e., `density_function(x,y,z)` and `field_function(X,Y,t)`.

The parser reads python-style expressions between double quotes, for instance `"a0*x**2 * (1-y*1.e2) * (x>0)"` is a valid expression where `a0` is a user-defined constant (see above) and `x` and `y` are spatial coordinates. The names are case sensitive. The factor `(x>0)` is `1` where `x>0` and `0` where `x<=0`. It allows the user to define functions by intervals. Alternatively the expression above can be written as `if(x>0, a0*x**2 * (1-y*1.e2), 0)`.

3.3.8 Numerics and algorithms

- **`algo.particle_shape` (integer; 1, 2, or 3)**
  The order of the shape factors (splines) for the macro-particles along all spatial directions: *1* for linear, *2* for quadratic, *3* for cubic. Low-order shape factors result in faster simulations, but may lead to more noisy results. High-order shape factors are computationally more expensive, but may increase the overall accuracy of the results. For production runs it is generally safer to use high-order shape factors, such as cubic order.
- **`algo.space_charge` (boolean, optional, default: `true`)**
  Whether to calculate space charge effects. This is in-development. At the moment, this flag only activates coordinate transformations and charge deposition.

3.3.9 Diagnostics and output

- `diag.enable` (`boolean`, optional, default: `true`) Enable or disable diagnostics generally. Disabling this is mostly used for benchmarking.

  This option is ignored for the openPMD output elements (remove them from the lattice to disable).
- `diag.slice_step_diagnostics` (`boolean`, optional, default: `false`) By default, diagnostics is performed at the beginning and end of the simulation. Enabling this flag will write diagnostics every step and slice step
- **`diag.file_min_digits` (integer, optional, default: 6)**
  The minimum number of digits used for the step number appended to the diagnostic file names.

Reduced Diagnostics

Reduced diagnostics allow the user to compute some reduced quantity (invariants of motion, particle temperature, max of a field, . . .) and write a small amount of data to text files. Reduced diagnostics are run *in situ* with the simulation.

Diagnostics related to integrable optics in the IOTA nonlinear magnetic insert element:

- `diag.alpha` (`float`, unitless) Twiss alpha of the bare linear lattice at the location of output for the nonlinear IOTA invariants H and I. Horizontal and vertical values must be equal.
- `diag.beta` (`float`, meters) Twiss beta of the bare linear lattice at the location of output for the nonlinear IOTA invariants H and I. Horizontal and vertical values must be equal.
- `diag.tn` (`float`, unitless) dimensionless strength of the IOTA nonlinear magnetic insert element used for computing H and I.
- `diag.cn` (`float`, meters^(1/2)) scale factor of the IOTA nonlinear magnetic insert element used for computing H and I.

In-situ visualization

Note: TODO :-)

Note: TODO :-)

3.3.10 Checkpoints and restart

Note: ImpactX will support checkpoints/restart via AMReX. The checkpoint capability can be turned with regular diagnostics: `<diag_name>.format = checkpoint`.

- **`amr.restart`** (*string*)
  Name of the checkpoint file to restart from. Returns an error if the folder does not exist or if it is not properly formatted.

3.3.11 Intervals parser

Note: TODO :-)

ImpactX can parse time step interval expressions of the form `start:stop:period`, e.g. `1:2:3, 4::, 5:6, :, ::10`. A comma is used as a separator between groups of intervals, which we call slices. The resulting time steps are the union set of all given slices. White spaces are ignored. A single slice can have 0, 1 or 2 colons `:`, just as numpy slices, but with inclusive upper bound for `stop`.

- For 0 colon the given value is the period
- For 1 colon the given string is of the type `start:stop`
- For 2 colons the given string is of the type `start:stop:period`

Any value that is not given is set to default. Default is `0` for the start, `std::numeric_limits<int>::max()` for the stop and 1 for the period. For the 1 and 2 colon syntax, actually having values in the string is optional (this means that `::5`, `100 ::10` and `100 :` are all valid syntaxes).

All values can be expressions that will be parsed in the same way as other integer input parameters.

Examples

- `something_intervals = 50` -> do something at timesteps 0, 50, 100, 150, etc. (equivalent to `something_intervals = ::50`)
- `something_intervals = 300:600:100` -> do something at timesteps 300, 400, 500 and 600.
- `something_intervals = 300::50` -> do something at timesteps 300, 350, 400, 450, etc.
- `something_intervals = 105:108,205:208` -> do something at timesteps 105, 106, 107, 108, 205, 206, 207 and 208. (equivalent to `something_intervals = 105 : 108 : , 205 : 208 :`)
- `something_intervals = :` or `something_intervals = ::` -> do something at every timestep.
- `something_intervals = 167:167,253:253,275:425:50` do something at timesteps 167, 253, 275, 325, 375 and 425.

This is essentially the python slicing syntax except that the stop is inclusive (`0:100` contains 100) and that no colon means that the given value is the period.

Note that if a given period is zero or negative, the corresponding slice is disregarded. For example, `something_intervals = -1` deactivates `something` and `something_intervals = ::-1,100:1000:25` is equivalent to `something_intervals = 100:1000:25`.

3.4 Examples

This section allows you to **download input files** that correspond to different physical situations.

3.4.1 FODO Cell

Stable FODO cell with a zero-current phase advance of 67.8 degrees.

The matched Twiss parameters at entry are:

- $\beta_\mathrm{x} = 2.82161941$ m
- $\alpha_\mathrm{x} = -1.59050035$
- $\beta_\mathrm{y} = 2.82161941$ m
- $\alpha_\mathrm{y} = 1.59050035$

We use a 2 GeV electron beam with initial unnormalized rms emittance of 2 nm.

The second moments of the particle distribution after the FODO cell should coincide with the second moments of the particle distribution before the FODO cell, to within the level expected due to noise due to statistical sampling.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_fodo.py`) or with an app with an input file (`impactx input_fodo.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.1: You can copy this file from `examples/fodo/run_fodo.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV electron beam with an initial
# unnormalized rms emittance of 2 nm
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=3.9984884770e-5,
    sigmaY=3.9984884770e-5,
    sigmaT=1.0e-3,
    sigmaPx=2.6623538760e-5,
    sigmaPy=2.6623538760e-5,
    sigmaPt=2.0e-3,
    muxpx=-0.846574929020762,
    muypy=0.846574929020762,
```

(continues on next page)

(continued from previous page)

```
    mutpt=0.0,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice)
ns = 25  # number of slices per ds in the element
fodo = [
    monitor,
    elements.Drift(ds=0.25, nslice=ns),
    monitor,
    elements.Quad(ds=1.0, k=1.0, nslice=ns),
    monitor,
    elements.Drift(ds=0.5, nslice=ns),
    monitor,
    elements.Quad(ds=1.0, k=-1.0, nslice=ns),
    monitor,
    elements.Drift(ds=0.25, nslice=ns),
    monitor,
]
# assign a fodo segment
sim.lattice.extend(fodo)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.2: You can copy this file from `examples/fodo/input_fodo.in`.

```
##########################################################################################
# Particle Beam(s)
##########################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-9
beam.particle = electron
beam.distribution = waterbag
beam.sigmaX = 3.9984884770e-5
beam.sigmaY = 3.9984884770e-5
beam.sigmaT = 1.0e-3
beam.sigmaPx = 2.6623538760e-5
beam.sigmaPy = 2.6623538760e-5
```

(continues on next page)

(continued from previous page)

```
beam.sigmaPt = 2.0e-3
beam.muxpx = -0.846574929020762
beam.muypy = 0.846574929020762
beam.mutpt = 0.0


##################################################################################
# Beamline: lattice elements and segments
##################################################################################
lattice.elements = monitor drift1 monitor quad1 monitor drift2 monitor quad2 monitor␣
↪drift3 monitor
lattice.nslice = 25

monitor.type = beam_monitor
monitor.backend = h5

drift1.type = drift
drift1.ds = 0.25

quad1.type = quad
quad1.ds = 1.0
quad1.k = 1.0

drift2.type = drift
drift2.ds = 0.5

quad2.type = quad
quad2.ds = 1.0
quad2.k = -1.0

drift3.type = drift
drift3.ds = 0.25


##################################################################################
# Algorithms
##################################################################################
algo.particle_shape = 2
algo.space_charge = false


##################################################################################
# Diagnostics
##################################################################################
diag.slice_step_diagnostics = true
```

Analyze

We run the following script to analyze correctness:

Script `analysis_fodo.py`

Listing 3.3: You can copy this file from `examples/fodo/analysis_fodo.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#


import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
```

(continues on next page)

(continued from previous page)

```
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        7.5451170454175073e-005,
        7.5441588239210947e-005,
        9.9775878164077539e-004,
        1.9959540393751392e-009,
        2.0175015289132990e-009,
        2.0013820193294972e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
```

(continues on next page)

(continued from previous page)

```
    [
        7.479011849622406e-005,
        7.535752516968014e-005,
        9.9775879288128088e-004,
        1.9959539836392703e-009,
        2.0175014668882125e-009,
        2.0013820380883801e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

Visualize

You can run the following script to visualize the beam evolution over time:

Script `plot_fodo.py`

Listing 3.4: You can copy this file from `examples/fodo/plot_fodo.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import argparse
import glob
import re

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import openpmd_api as io
import pandas as pd
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
```

(continues on next page)

(continued from previous page)

```
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


def read_file(file_pattern):
    for filename in glob.glob(file_pattern):
        df = pd.read_csv(filename, delimiter=r"\s+")
        if "step" not in df.columns:
            step = int(re.findall(r"[0-9]+", filename)[0])
            df["step"] = step
        yield df


def read_time_series(file_pattern):
    """Read in all CSV files from each MPI rank (and potentially OpenMP
    thread). Concatenate into one Pandas dataframe.

    Returns
    -------
    pandas.DataFrame
    """
    return pd.concat(
        read_file(file_pattern),
        axis=0,
        ignore_index=True,
    )  # .set_index('id')


# options to run this script
parser = argparse.ArgumentParser(description="Plot the FODO benchmark.")
parser.add_argument(
    "--save-png", action="store_true", help="non-interactive run: save to PNGs"
)
args = parser.parse_args()


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()
ref_particle = read_time_series("diags/ref_particle.*")

# scaling to units
millimeter = 1.0e3  # m->mm
mrad = 1.0e3  # ImpactX uses "static units": momenta are normalized by the magnitude of␣
↪the momentum of the reference particle p0: px/p0 (rad)
# mm_mrad = 1.e6
nm_rad = 1.0e9


# select a single particle by id
# particle_42 = beam[beam["id"] == 42]
# print(particle_42)


# steps & corresponding z
steps = list(series.iterations)

z = list(
    map(lambda step: ref_particle[ref_particle["step"] == step].z.values[0], steps)
)
# print(f"z={z}")


# beam transversal size & emittance over steps
moments = list(
    map(
        lambda step: (
            step,
            get_moments(series.iterations[step].particles["beam"].to_df()),
        ),
        steps,
    )
)
# print(moments)
sigx = list(map(lambda step_val: step_val[1][0] * millimeter, moments))
sigy = list(map(lambda step_val: step_val[1][1] * millimeter, moments))
emittance_x = list(map(lambda step_val: step_val[1][3] * nm_rad, moments))
emittance_y = list(map(lambda step_val: step_val[1][4] * nm_rad, moments))

# print(sigx, sigy)


# print beam transversal size over steps
f = plt.figure(figsize=(9, 4.8))
ax1 = f.gca()
im_sigx = ax1.plot(z, sigx, label=r"$\sigma_x$")
im_sigy = ax1.plot(z, sigy, label=r"$\sigma_y$")
ax2 = ax1.twinx()
```

(continues on next page)

(continued from previous page)

```
ax2._get_lines.prop_cycler = ax1._get_lines.prop_cycler
im_emittance_x = ax2.plot(z, emittance_x, ":", label=r"$\epsilon_x$")
im_emittance_y = ax2.plot(z, emittance_y, ":", label=r"$\epsilon_y$")

ax1.legend(
    handles=im_sigx + im_sigy + im_emittance_x + im_emittance_y, loc="lower center"
)
ax1.set_xlabel(r"$z$ [m]")
ax1.set_ylabel(r"$\sigma_{x,y}$ [mm]")
# ax2.set_ylabel(r"$\epsilon_{x,y}$ [mm-mrad]")
ax2.set_ylabel(r"$\epsilon_{x,y}$ [nm]")
ax2.set_ylim([1.5, 2.5])
ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.tight_layout()
if args.save_png:
    plt.savefig("fodo_sigma.png")
else:
    plt.show()


# beam transversal scatter plot over steps
num_plots_per_row = len(steps)
fig, axs = plt.subplots(
    3, num_plots_per_row, figsize=(9, 4.8), sharex="row", sharey="row"
)

ncol_ax = -1
for step in steps:
    # plot initial distribution & at exit of each element
    ncol_ax += 1

    # x-y
    ax = axs[(0, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_x.multiply(millimeter),
        beam_at_step.position_y.multiply(millimeter),
        s=0.01,
    )

    ax.set_title(f"$z={z[ncol_ax]}$ [m]")
    ax.set_xlabel(r"$x$ [mm]")

    # x-px
    ax = axs[(1, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_x.multiply(millimeter),
        beam_at_step.momentum_x.multiply(mrad),
        s=0.01,
    )
    ax.set_xlabel(r"$x$ [mm]")
```

(continues on next page)

(continued from previous page)

```
    # y-py
    ax = axs[(2, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_y.multiply(millimeter),
        beam_at_step.momentum_y.multiply(mrad),
        s=0.01,
    )
    ax.set_xlabel(r"$y$ [mm]")

axs[(0, 0)].set_ylabel(r"$y$ [mm]")
axs[(1, 0)].set_ylabel(r"$p_x$ [mrad]")
axs[(2, 0)].set_ylabel(r"$p_y$ [mrad]")
plt.tight_layout()
if args.save_png:
    plt.savefig("fodo_scatter.png")
else:
    plt.show()
```

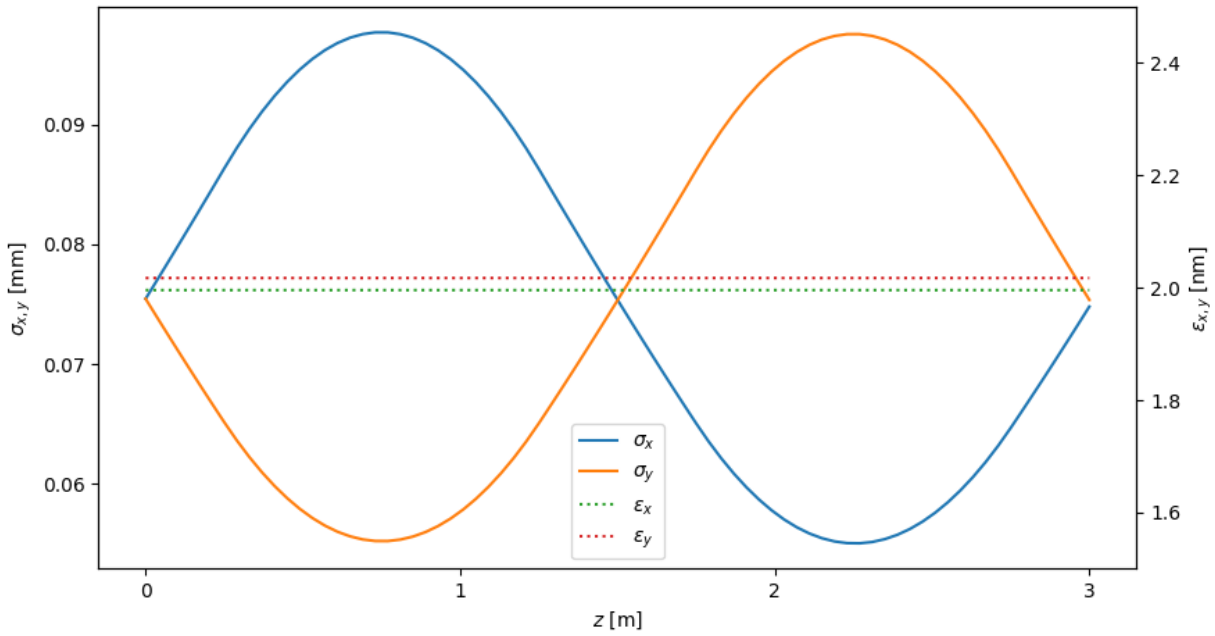

Fig. 3.1: FODO transversal beam width and emittance evolution

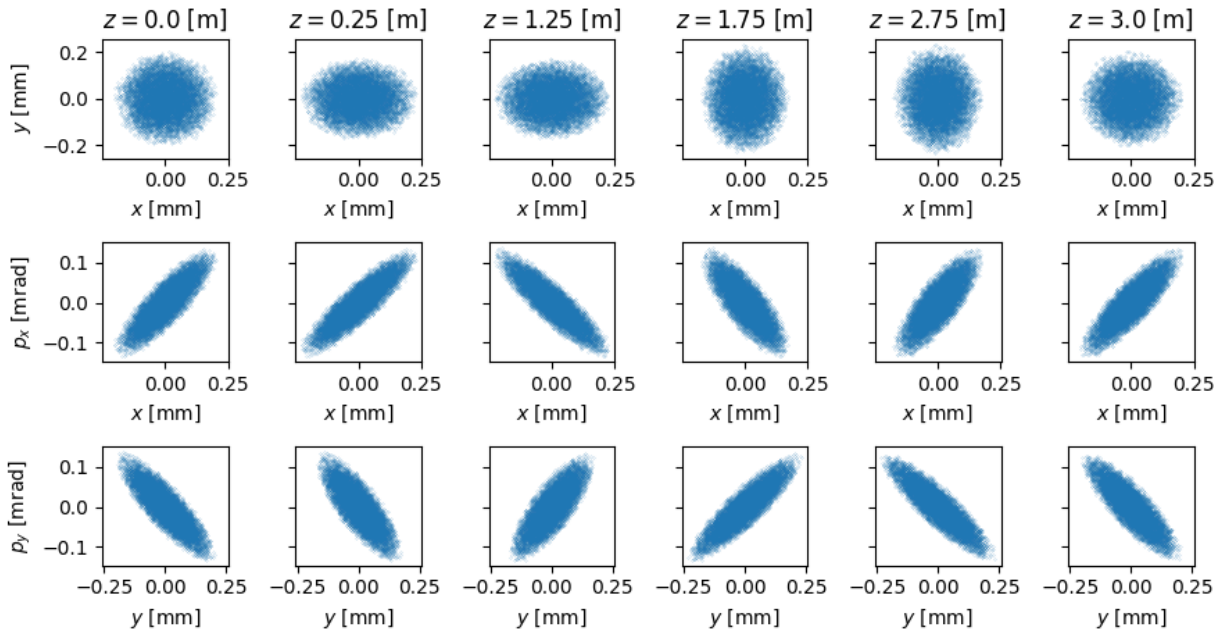

Fig. 3.2: FODO transversal beam width and phase space evolution

3.4.2 Chicane

Berlin-Zeuthen magnetic bunch compression chicane: https://www.desy.de/csr/

All parameters can be found online. A 5 GeV electron bunch with normalized transverse rms emittance of 1 um undergoes longitudinal compression by a factor of 10 in a standard 4-bend chicane.

The emittances should be preserved, and the rms pulse length should decrease by the compression factor (10).

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_chicane.py`) or with an app with an input file (`impactx input_chicane.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.5: You can copy this file from `examples/chicane/run_chicane.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-
```

(continues on next page)

(continued from previous page)

```
import amrex
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 5 GeV electron beam with an initial
# normalized transverse rms emittance of 1 um
energy_MeV = 5.0e3  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=2.2951017632e-5,
    sigmaY=1.3084093142e-5,
    sigmaT=5.5555553e-8,
    sigmaPx=1.598353425e-6,
    sigmaPy=2.803697378e-6,
    sigmaPt=2.000000000e-6,
    muxpx=0.933345606203060,
    muypy=0.933345606203060,
    mutpt=0.999999961419755,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
ns = 25  # number of slices per ds in the element
rc = 10.35  # bend radius (meters)
psi = 0.048345620280243  # pole face rotation angle (radians)

# Drift elements
dr1 = elements.Drift(ds=5.0058489435, nslice=ns)
dr2 = elements.Drift(ds=1.0, nslice=ns)
dr3 = elements.Drift(ds=2.0, nslice=ns)
```

(continues on next page)

(continued from previous page)

```
# Bend elements
sbend1 = elements.Sbend(ds=0.50037, rc=-rc, nslice=ns)
sbend2 = elements.Sbend(ds=0.50037, rc=rc, nslice=ns)

# Dipole Edge Focusing elements
dipedge1 = elements.DipEdge(psi=-psi, rc=-rc, g=0.0, K2=0.0)
dipedge2 = elements.DipEdge(psi=psi, rc=rc, g=0.0, K2=0.0)

lattice_half = [sbend1, dipedge1, dr1, dipedge2, sbend2]
# assign a segment with the first half of the lattice
sim.lattice.append(monitor)
sim.lattice.extend(lattice_half)
sim.lattice.append(dr2)
lattice_half.reverse()
# extend the lattice by a reversed half
sim.lattice.extend(lattice_half)
sim.lattice.append(dr3)
sim.lattice.append(monitor)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.6: You can copy this file from `examples/chicane/input_chicane.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 5.0e3
beam.charge = 1.0e-9
beam.particle = electron
beam.distribution = waterbag
beam.sigmaX = 2.2951017632e-5
beam.sigmaY = 1.3084093142e-5
beam.sigmaT = 5.5555553e-8
beam.sigmaPx = 1.598353425e-6
beam.sigmaPy = 2.803697378e-6
beam.sigmaPt = 2.000000000e-6
beam.muxpx = 0.933345606203060
beam.muypy = 0.933345606203060
beam.mutpt = 0.999999961419755
```

(continues on next page)

(continued from previous page)

```
################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor sbend1 dipedge1 drift1 dipedge2 sbend2 drift2
                   sbend2 dipedge2 drift1 dipedge1 sbend1 drift3 monitor
lattice.nslice = 25

sbend1.type = sbend
sbend1.ds = 0.50037       # projected length 0.5 m, angle 2.77 deg
sbend1.rc = -10.35

drift1.type = drift
drift1.ds = 5.0058489435  # projected length 5 m

sbend2.type = sbend
sbend2.ds = 0.50037       # projected length 0.5 m, angle 2.77 deg
sbend2.rc = 10.35

drift2.type = drift
drift2.ds = 1.0

drift3.type = drift
drift3.ds = 2.0

dipedge1.type = dipedge   # dipole edge focusing
dipedge1.psi = -0.048345620280243
dipedge1.rc = -10.35
dipedge1.g = 0.0
dipedge1.K2 = 0.0

dipedge2.type = dipedge
dipedge2.psi = 0.048345620280243
dipedge2.rc = 10.35
dipedge2.g = 0.0
dipedge2.K2 = 0.0

monitor.type = beam_monitor
monitor.backend = h5


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = false


################################################################################
# Diagnostics
################################################################################
diag.slice_step_diagnostics = true
```

Analyze

We run the following script to analyze correctness:

Script `analysis_chicane.py`

Listing 3.7: You can copy this file from `examples/chicane/analysis_chicane.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        6.4214719960819659e-005,
        3.6603372435649773e-005,
        1.9955175623579313e-004,
        1.0198263116327677e-010,
        1.0308359092878036e-010,
        4.0035161705244885e-010,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
```

(continues on next page)

(continued from previous page)

```
        2.3928429374387210e-005,
        8.4424535301423173e-005,
        1.9976426324802290e-005,
        1.0198281373761584e-010,
        1.0308356090529235e-010,
        4.0027996099961315e-010,
    ],
    rtol=rtol,
    atol=atol,
)
```

Visualize

You can run the following script to visualize the beam evolution over time:

Script `plot_chicane.py`

Listing 3.8: You can copy this file from `examples/chicane/plot_chicane.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import argparse
import glob
import re

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import openpmd_api as io
import pandas as pd
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
```

(continues on next page)

(continued from previous page)

```
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


def read_file(file_pattern):
    for filename in glob.glob(file_pattern):
        df = pd.read_csv(filename, delimiter=r"\s+")
        if "step" not in df.columns:
            step = int(re.findall(r"[0-9]+", filename)[0])
            df["step"] = step
        yield df


def read_time_series(file_pattern):
    """Read in all CSV files from each MPI rank (and potentially OpenMP
    thread). Concatenate into one Pandas dataframe.

    Returns
    -------
    pandas.DataFrame
    """
    return pd.concat(
        read_file(file_pattern),
        axis=0,
        ignore_index=True,
    )  # .set_index('id')


# options to run this script
parser = argparse.ArgumentParser(description="Plot the chicane benchmark.")
parser.add_argument(
    "--save-png", action="store_true", help="non-interactive run: save to PNGs"
)
args = parser.parse_args()


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
```

(continues on next page)

(continued from previous page)

```
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()
ref_particle = read_time_series("diags/ref_particle.*")

# scaling to units
millimeter = 1.0e3  # m->mm
# for "t": the time coordinate is scaled by c, and therefore has units of length (m) by␣
↪default, so we can label the axis ct (mm)
mrad = 1.0e3  # ImpactX uses "static units": momenta are normalized by the magnitude of␣
↪the momentum of the reference particle p0: px/p0 (rad)
# mm_mrad = 1.e6
nm_rad = 1.0e9


# select a single particle by id
# particle_42 = beam[beam["id"] == 42]
# print(particle_42)


# steps & corresponding z
steps = list(series.iterations)

z = list(
    map(lambda step: ref_particle[ref_particle["step"] == step].z.values[0], steps)
)
x = list(
    map(lambda step: ref_particle[ref_particle["step"] == step].x.values[0], steps)
)
# print(f"z={z}")


# beam transversal size & emittance over steps
moments = list(
    map(
        lambda step: (
            step,
            get_moments(series.iterations[step].particles["beam"].to_df()),
        ),
        steps,
    )
)
# print(moments)
sigx = list(map(lambda step_val: step_val[1][0] * millimeter, moments))
sigt = list(map(lambda step_val: step_val[1][2] * millimeter, moments))
emittance_x = list(map(lambda step_val: step_val[1][3] * nm_rad, moments))
emittance_t = list(map(lambda step_val: step_val[1][5] * nm_rad, moments))

# print(sigx, sigt)


# print beam transversal size over steps
f, axs = plt.subplots(
```

(continues on next page)

(continued from previous page)

```
    2, 1, figsize=(9, 4.8), sharex=True, gridspec_kw={"height_ratios": [1, 2]}
)
ax0 = axs[0]
im_xz = ax0.plot(z, x, "--", lw=3, label=r"$x$")
ax0.legend(loc="upper right")
ax0.set_ylim([0, None])
ax0.set_ylabel(r"$x$ [m]")

ax1 = axs[1]
im_sigx = ax1.plot(z, sigx, label=r"$\sigma_x$")
im_sigt = ax1.plot(z, sigt, label=r"$\sigma_t$")
ax2 = ax1.twinx()
ax2._get_lines.prop_cycler = ax1._get_lines.prop_cycler
im_emittance_x = ax2.plot(z, emittance_x, ":", label=r"$\epsilon_x$")
im_emittance_t = ax2.plot(z, emittance_t, ":", label=r"$\epsilon_t$")

ax1.legend(
    handles=im_sigx + im_sigt + im_emittance_x + im_emittance_t, loc="upper right"
)
ax1.set_xlabel(r"$z$ [m]")
ax1.set_ylabel(r"$\sigma_{x,t}$ [mm]")
# ax2.set_ylabel(r"$\epsilon_{x,y}$ [mm-mrad]")
ax2.set_ylabel(r"$\epsilon_{x,t}$ [nm]")
ax1.set_ylim([0, None])
ax2.set_ylim([0, None])
ax1.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.tight_layout()
if args.save_png:
    plt.savefig("chicane_sigma.png")
else:
    plt.show()


# beam transversal scatter plot over steps
num_plots_per_row = len(steps)
fig, axs = plt.subplots(
    4, num_plots_per_row, figsize=(9, 6.4), sharex="row", sharey="row"
)

ncol_ax = -1
for step in steps:
    # plot initial distribution & at exit of each element
    ncol_ax += 1

    # t-pt
    ax = axs[(0, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_ct.multiply(millimeter),
        beam_at_step.momentum_t.multiply(mrad),
        s=0.01,
    )
```

(continues on next page)

(continued from previous page)

```
    ax.set_xlabel(r"$ct$ [mm]")
    z_unit = ""
    if ncol_ax == num_plots_per_row - 1:
        z_unit = " [m]"
    ax.set_title(f"$z={z[ncol_ax]:.1f}${z_unit}")

    # x-px
    ax = axs[(1, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_x.multiply(millimeter),
        beam_at_step.momentum_x.multiply(mrad),
        s=0.01,
    )
    ax.set_xlabel(r"$x$ [mm]")

    # t-x
    ax = axs[(2, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_ct.multiply(millimeter),
        beam_at_step.position_x.multiply(millimeter),
        s=0.01,
    )
    ax.set_xlabel(r"$ct$ [mm]")

    # t-px
    ax = axs[(3, ncol_ax)]
    beam_at_step = series.iterations[step].particles["beam"].to_df()
    ax.scatter(
        beam_at_step.position_ct.multiply(millimeter),
        beam_at_step.momentum_x.multiply(mrad),
        s=0.01,
    )
    ax.set_xlabel(r"$ct$ [mm]")

axs[(0, 0)].set_ylabel(r"$p_t$ [mrad]")
axs[(1, 0)].set_ylabel(r"$p_x$ [mrad]")
axs[(2, 0)].set_ylabel(r"$x$ [mm]")
axs[(3, 0)].set_ylabel(r"$p_x$ [mrad]")
plt.tight_layout()
if args.save_png:
    plt.savefig("chicane_scatter.png")
else:
    plt.show()
```

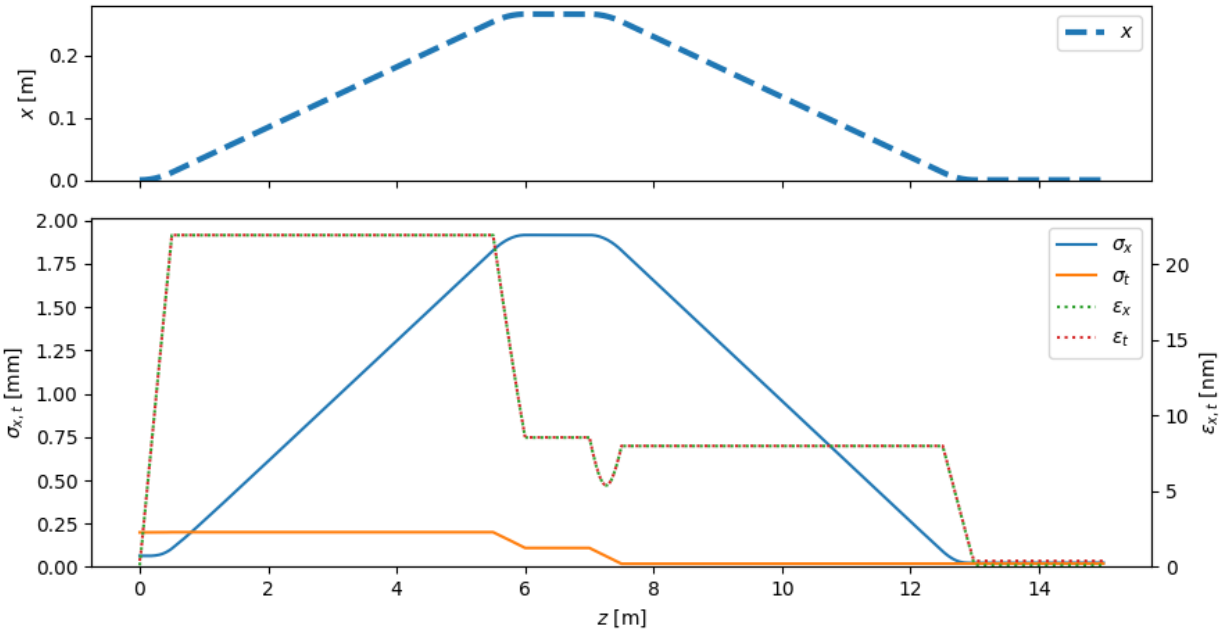

Fig. 3.3: (top) Chicane floorplan. (bottom) Chicane beam width and emittance evolution.

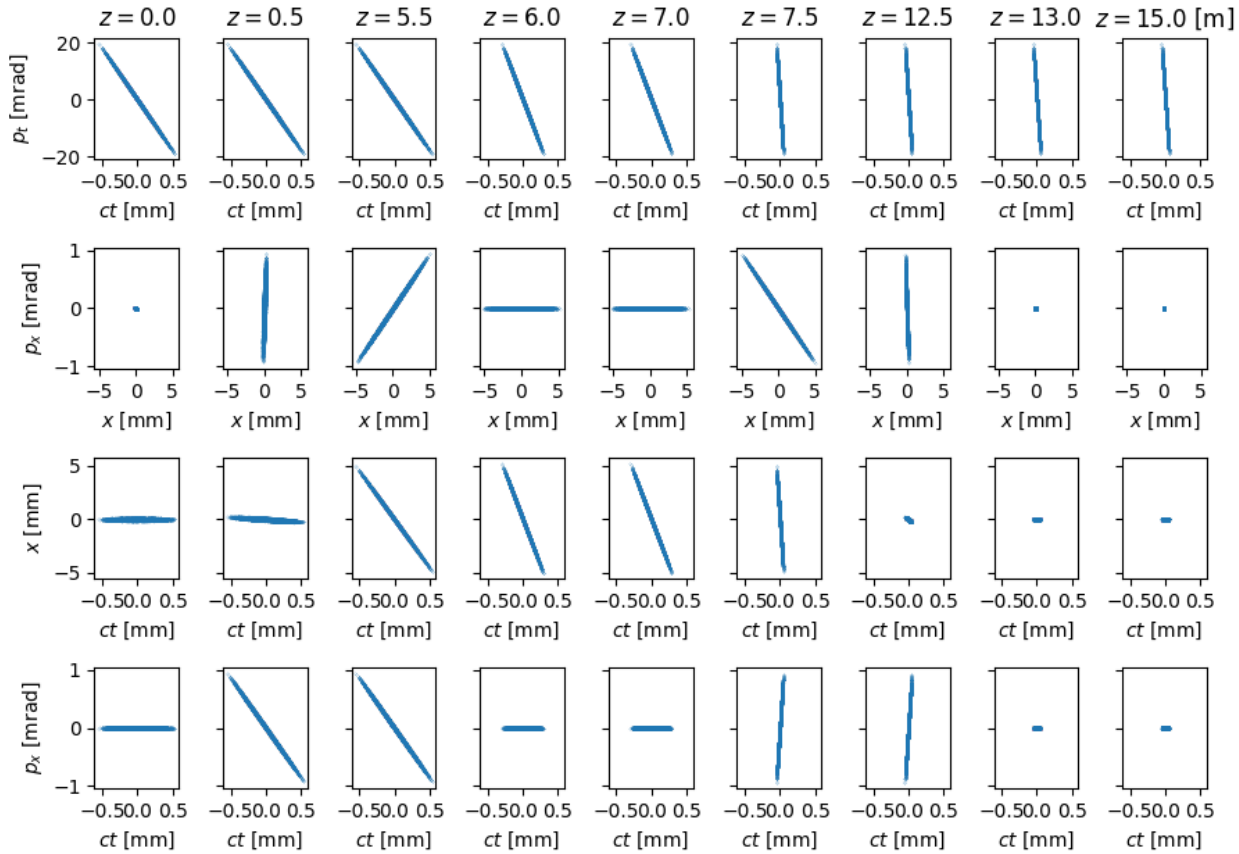

Fig. 3.4: Chicane beam width and emittance evolution

3.4.3 Constant Focusing Channel

Stationary beam in a constant focusing channel (without space charge).

The matched Twiss parameters at entry are:

- $\beta_x = 1.0$ m
- $\alpha_x = 0.0$
- $\beta_y = 1.0$ m
- $\alpha_y = 0.0$

We use a 2 GeV proton beam with initial unnormalized rms emittance of 1 um. The longitudinal beam parameters are chosen so that the bunch has radial symmetry when viewed in the beam rest frame.

The particle distribution should remain unchanged, to within the level expected due to numerical particle noise. This fact is independent of the length of the channel. This is tested using the second moments of the distribution.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_cfchannel.py`) or with an app with an input file (`impactx input_cfchannel.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.9: You can copy this file from `examples/cfchannel/run_cfchannel.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV proton beam with an initial
```

(continues on next page)

(continued from previous page)

```
# normalized transverse rms emittance of 1 um
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=1.0e-3,
    sigmaY=1.0e-3,
    sigmaT=3.369701494258956e-4,
    sigmaPx=1.0e-3,
    sigmaPy=1.0e-3,
    sigmaPt=2.9676219145931020e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
sim.lattice.extend(
    [
        monitor,
        elements.ConstF(ds=2.0, kx=1.0, ky=1.0, kt=1.0),
        monitor,
    ]
)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.10: You can copy this file from `examples/cfchannel/input_cfchannel.in`.

```
##################################################################################
# Particle Beam(s)
##################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-9
```

(continues on next page)

(continued from previous page)

```
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 1.0e-3
beam.sigmaY = 1.0e-3
beam.sigmaT = 3.369701494258956e-4
beam.sigmaPx = 1.0e-3
beam.sigmaPy = 1.0e-3
beam.sigmaPt = 2.9676219145931020e-3
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


##################################################################################
# Beamline: lattice elements and segments
##################################################################################
lattice.elements = monitor constf1 monitor

monitor.type = beam_monitor
monitor.backend = h5

constf1.type = constf
constf1.ds = 2.0
constf1.kx = 1.0
constf1.ky = 1.0
constf1.kt = 1.0


##################################################################################
# Algorithms
##################################################################################
algo.particle_shape = 2
algo.space_charge = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_cfchannel.py`

Listing 3.11: You can copy this file from `examples/cfchannel/analysis_cfchannel.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
```

(continues on next page)

(continued from previous page)

```
)

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.0e-003,
        1.0e-003,
        3.369701494258956e-4,
        1.0e-006,
        1.0e-006,
        1.0e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.0e-003,
        1.0e-003,
        3.369701494258956e-4,
        1.0e-006,
        1.0e-006,
        1.0e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.4 Constant Focusing Channel with Space Charge

RMS-matched beam in a constant focusing channel with space charge.

The matched Twiss parameters at entry are:

- $\beta_x = 1.477305$ m
- $\alpha_x = 0.0$
- $\beta_y = 1.477305$ m
- $\alpha_y = 0.0$

We use a 2 GeV proton beam with initial unnormalized rms emittance of 1 um. The longitudinal beam parameters are chosen so that the bunch has radial symmetry when viewed in the beam rest frame. The bunch charge is set to 10 nC, resulting in a transverse tune depression ratio of 0.67. The initial distribution used is a 6D waterbag.

The beam second moments should remain nearly unchanged, except for some small emittance growth due to nonlinear space charge. This is tested using the second moments of the distribution.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and :math:`

Run

This example can be run as a Python script (`python3 run_cfchannel_10nC.py`) or as an app with an input file (`impactx input_cfchannel_10nC.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.12: You can copy this file from `examples/cfchannel/run_cfchannel_10nC.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

pp_amr = amrex.ParmParse("amr")
pp_amr.addarr("n_cell", [48, 48, 40])  # [72, 72, 64] for increased precision

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = True
sim.prob_relative = 1.0
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True
```

(continues on next page)

(continued from previous page)

```
# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV proton beam with an initial
# normalized transverse rms emittance of 1 um
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-8  # used with space charge
npart = 10000  # number of macro particles; use 1e5 for increased precision

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=1.2154443728379865788e-3,
    sigmaY=1.2154443728379865788e-3,
    sigmaT=4.0956844276541331005e-4,
    sigmaPx=8.2274435782286157175e-4,
    sigmaPy=8.2274435782286157175e-4,
    sigmaPt=2.4415943602685364584e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
nslice = 50  # use 1e5 for increased precision

# design the accelerator lattice
sim.lattice.extend(
    [
        monitor,
        elements.ConstF(ds=2.0, kx=1.0, ky=1.0, kt=1.0, nslice=nslice),
        monitor,
    ]
)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.13: You can copy this file from `examples/cfchannel/input_cfchannel_10nC.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
#beam.npart = 100000  # optional for increased precision
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-8
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 1.2154443728379865788e-3
beam.sigmaY = 1.2154443728379865788e-3
beam.sigmaT = 4.0956844276541331005e-4
beam.sigmaPx = 8.2274435782286157175e-4
beam.sigmaPy = 8.2274435782286157175e-4
beam.sigmaPt = 2.4415943602685364584e-3


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor constf1 monitor
lattice.nslice = 50
#lattice.nslice = 60 # optional for increased precision

monitor.type = beam_monitor
monitor.backend = h5

constf1.type = constf
constf1.ds = 2.0
constf1.kx = 1.0
constf1.ky = 1.0
constf1.kt = 1.0


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = true

amr.n_cell = 48 48 40
#amr.n_cell = 72 72 64 #optional for increased precision
geometry.prob_relative = 1.0
```

Analyze

We run the following script to analyze correctness:

Script `analysis_cfchannel_10nC.py`

Listing 3.14: You can copy this file from `examples/cfchannel/analysis_cfchannel_10nC.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.2154443728379865788e-003,
        1.2154443728379865788e-003,
        4.0956844276541331005317e-004,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
```

(continues on next page)

(continued from previous page)

```
        1.2154443728379865788e-003,
        1.2154443728379865788e-003,
        4.0956844276541331005317e-004,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.5 Expanding Beam in Free Space

A coasting bunch expanding freely in free space under its own space charge.

We use a cold (zero emittance) 250 MeV electron bunch whose initial distribution is a uniformly-populated 3D ball of radius R0 = 1 mm when viewed in the bunch rest frame.

In the laboratory frame, the bunch is a uniformly-populated ellipsoid, which expands to twice its original size. This is tested using the second moments of the distribution.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_expanding.py`) or with an app with an input file (`impactx input_expanding.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.15: You can copy this file from `examples/expanding/run_expanding.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

pp_amr = amrex.ParmParse("amr")
pp_amr.addarr("n_cell", [56, 56, 48])

sim = ImpactX()

# set numerical parameters and IO control
```

(continues on next page)

(continued from previous page)

```
sim.particle_shape = 2  # B-spline order
sim.space_charge = True
sim.dynamic_size = True
sim.prob_relative = 1.0

# beam diagnostics
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = False

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV electron beam with an initial
# unnormalized rms emittance of 2 nm
energy_MeV = 250  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles (outside tests, use 1e5 or more)

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Kurth6D(
    sigmaX=4.472135955e-4,
    sigmaY=4.472135955e-4,
    sigmaT=9.12241869e-7,
    sigmaPx=0.0,
    sigmaPy=0.0,
    sigmaPt=0.0,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
sim.lattice.extend([monitor, elements.Drift(ds=6.0, nslice=40), monitor])

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.16: You can copy this file from `examples/expanding/input_expanding.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000  # outside tests, use 1e5 or more
beam.units = static
beam.energy = 250.0
beam.charge = 1.0e-9
beam.particle = electron
beam.distribution = kurth6d
beam.sigmaX = 4.472135955e-4
beam.sigmaY = 4.472135955e-4
beam.sigmaT = 9.12241869e-7
beam.sigmaPx = 0.0
beam.sigmaPy = 0.0
beam.sigmaPt = 0.0


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor drift1 monitor
lattice.nslice = 40

drift1.type = drift
drift1.ds = 6.0

monitor.type = beam_monitor
monitor.backend = h5


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = true

amr.n_cell = 56 56 48
geometry.prob_relative = 1.0
```

Analyze

We run the following script to analyze correctness:

Script `analysis_expanding.py`

Listing 3.17: You can copy this file from `examples/expanding/analysis_expanding.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        4.4721359550e-004,
        4.4721359550e-004,
        9.1224186858e-007,
        0.0e-006,
        0.0e-006,
        0.0e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt],
    [
```

(continues on next page)

(continued from previous page)

```
        8.9442719100e-004,
        8.9442719100e-004,
        1.8244837370e-006,
    ],
    rtol=rtol,
    atol=atol,
)
atol = 1.0e-8
rtol = 0.0  # ignored
assert np.allclose(
    [emittance_x, emittance_y, emittance_t],
    [
        0.0,
        0.0,
        0.0,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.6 Kurth Distribution in a Periodic Focusing Channel

Matched Kurth distribution in a periodic focusing channel (without space charge).

The distribution is radially symmetric in (x,y,t) space, and matched to a radially symmetric periodic linear focusing lattice with a phase advance of 121 degrees.

We use a 2 GeV proton beam with initial unnormalized rms emittance of 1 um in all three phase planes.

The particle distribution should remain unchanged, to within the level expected due to numerical particle noise. This is tested using the second moments of the distribution.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_kurth_periodic.py`) or with an app with an input file (`impactx input_kurth_periodic.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.18: You can copy this file from `examples/kurth/run_kurth_periodic.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Ryan Sandberg, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
```

(continues on next page)

(continued from previous page)

```
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV proton beam with an initial
# unnormalized rms emittance of 1 um in each
# coordinate plane
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-8  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Kurth6D(
    sigmaX=1.11e-3,
    sigmaY=1.11e-3,
    sigmaT=3.74036839224568e-4,
    sigmaPx=9.00900900901e-4,
    sigmaPy=9.00900900901e-4,
    sigmaPt=2.6735334467940146e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
constf1 = elements.ConstF(ds=2.0, kx=0.7, ky=0.7, kt=0.7)
drift1 = elements.Drift(ds=1.0)
sim.lattice.extend([monitor, drift1, constf1, drift1, monitor])

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.19: You can copy this file from `examples/kurth/input_kurth_periodic.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-8
beam.particle = proton
beam.distribution = kurth6d
beam.sigmaX = 1.11e-3
beam.sigmaY = 1.11e-3
beam.sigmaT = 3.74036839224568e-4
beam.sigmaPx = 9.00900900901e-4
beam.sigmaPy = 9.00900900901e-4
beam.sigmaPt = 2.6735334467940146e-3


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor drift1 constf1 drift1 monitor

monitor.type = beam_monitor
monitor.backend = h5

drift1.type = drift
drift1.ds = 1.0

constf1.type = constf
constf1.ds = 2.0
constf1.kx = 0.7
constf1.ky = 0.7
constf1.kt = 0.7


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = false

amr.n_cell = 40 40 32
geometry.prob_relative = 1.0
```

Analyze

We run the following script to analyze correctness:

Script `analysis_kurth_periodic.py`

Listing 3.20: You can copy this file from `examples/kurth/analysis_kurth_periodic.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.11e-03,
        1.11e-03,
        3.74036839224568e-04,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
```

(continues on next page)

(continued from previous page)

```
        1.11e-03,
        1.11e-03,
        3.74036839224568e-04,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.7 Kurth Distribution in a Periodic Focusing Channel with Space Charge

Matched Kurth distribution in a periodic focusing channel with space charge.

The distribution is radially symmetric in (x,y,t) space, and matched to a radially symmetric constant linear focusing.

We use a 2 GeV proton beam with initial unnormalized rms emittance of 1 um in all three phase planes. The bunch charge is set to 10 nC, depressing the phase advance from 121 degrees to 74 degrees.

The particle distribution should remain unchanged, to within the level expected due to numerical particle noise. This is tested using the second moments of the distribution.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and :math:`

Run

This example can be run as a Python script (`python3 run_kurth_10nC_periodic.py`) or as an app with an input file (`impactx input_kurth_10nC_periodic.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.21: You can copy this file from `examples/kurth/run_kurth_10nC_periodic.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Ryan Sandberg, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

pp_amr = amrex.ParmParse("amr")
pp_amr.addarr("n_cell", [48, 48, 40])  # use [72, 72, 72] for increased precision

sim = ImpactX()
```

(continues on next page)

(continued from previous page)

```
# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = True
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV proton beam with an initial
# unnormalized rms emittance of 1 um in each
# coordinate plane
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-8  # used with space charge
npart = 10000  # number of macro particles; use 1e5 for increased precision

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Kurth6D(
    sigmaX=1.46e-3,
    sigmaY=1.46e-3,
    sigmaT=4.9197638312420749e-4,
    sigmaPx=6.84931506849e-4,
    sigmaPy=6.84931506849e-4,
    sigmaPt=2.0326178944803812e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
nslice = 20  # use 30 for increased precision
constf1 = elements.ConstF(ds=2.0, kx=0.7, ky=0.7, kt=0.7, nslice=nslice)
drift1 = elements.Drift(ds=1.0, nslice=nslice)
sim.lattice.extend([monitor, drift1, constf1, drift1, monitor])

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.22: You can copy this file from `examples/kurth/input_kurth_10nC_periodic.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
#beam.npart = 100000 #optional for increased precision
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-8
beam.particle = proton
beam.distribution = kurth6d
beam.sigmaX = 1.46e-3
beam.sigmaY = 1.46e-3
beam.sigmaT = 4.9197638312420749e-4
beam.sigmaPx = 6.84931506849e-4
beam.sigmaPy = 6.84931506849e-4
beam.sigmaPt = 2.0326178944803812e-3


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor drift1 constf1 drift1 monitor
lattice.nslice = 20
#lattice.nslice = 30 #optional for increased precision

monitor.type = beam_monitor
monitor.backend = h5

drift1.type = drift
drift1.ds = 1.0

constf1.type = constf
constf1.ds = 2.0
constf1.kx = 0.7
constf1.ky = 0.7
constf1.kt = 0.7


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = true

amr.n_cell = 48 48 40
#amr.n_cell = 72 72 72 #optional for increased precision
geometry.prob_relative = 1.0
```

Analyze

We run the following script to analyze correctness:

Script `analysis_kurth_10nC_periodic.py`

Listing 3.23: You can copy this file from `examples/kurth/analysis_kurth_10nC_periodic.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 2.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.46e-3,
        1.46e-3,
        4.9197638312420749e-4,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 2.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
```

(continues on next page)

(continued from previous page)

```
        1.46e-3,
        1.46e-3,
        4.9197638312420749e-4,
        1.000000000e-006,
        1.000000000e-006,
        1.000000000e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.8 Acceleration by RF Cavities

Beam accelerated through a sequence of 4 RF cavities (without space charge).

We use a 230 MeV electron beam with initial normalized rms emittance of 1 um.

The lattice and beam parameters are based on Example 2 of the IMPACT-Z examples folder:

https://github.com/impact-lbl/IMPACT-Z/tree/master/examples/Example2

The final target beam energy and beam moments are based on simulation in IMPACT-Z, without space charge.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_rfcavity.py`) or with an app with an input file (`impactx input_rfcavity.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.24: You can copy this file from `examples/rfcavity/run_rfcavity.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
```

(continues on next page)

(continued from previous page)

```
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = False

# domain decomposition & space charge mesh
sim.init_grids()

# load a 230 MeV electron beam with an initial
# unnormalized rms emittance of 1 mm-mrad in all
# three phase planes
energy_MeV = 230.0  # reference energy
bunch_charge_C = 1.0e-10  # used with space charge
npart = 10000  # number of macro particles (outside tests, use 1e5 or more)

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=0.352498964601e-3,
    sigmaY=0.207443478142e-3,
    sigmaT=0.70399950746e-4,
    sigmaPx=5.161852770e-6,
    sigmaPy=9.163582894e-6,
    sigmaPt=0.260528852031e-3,
    muxpx=0.5712386101751441,
    muypy=-0.514495755427526,
    mutpt=-5.05773e-10,
)
sim.add_particles(bunch_charge_C, distr, npart)

# design the accelerator lattice

#   Drift elements
dr1 = elements.Drift(ds=0.4, nslice=1)
dr2 = elements.Drift(ds=0.032997, nslice=1)
#   RF cavity element
rf = elements.RFCavity(
    ds=1.31879807,
    escale=62.0,
    freq=1.3e9,
    phase=85.5,
    cos_coefficients=[
        0.1644024074311037,
        -0.1324009958969339,
        4.3443060026047219e-002,
        8.5602654094946495e-002,
        -0.2433578169042885,
        0.5297150596779437,
        0.7164884680963959,
        -5.2579522442877296e-003,
        -5.5025369142193678e-002,
```

(continues on next page)

(continued from previous page)

```
        4.6845673335028933e-002,
        -2.3279346335638568e-002,
        4.0800777539657775e-003,
        4.1378326533752169e-003,
        -2.5040533340490805e-003,
        -4.0654981400000964e-003,
        9.6630592067498289e-003,
        -8.5275895985990214e-003,
        -5.8078747006425020e-002,
        -2.4044337836660403e-002,
        1.0968240064697212e-002,
        -3.4461179858301418e-003,
        -8.1201564869443749e-004,
        2.1438992904959380e-003,
        -1.4997753525697276e-003,
        1.8685171825676386e-004,
    ],
    sin_coefficients=[
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
    ],
    mapsteps=100,
    nslice=4,
)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

sim.lattice.extend(
```

(continues on next page)

(continued from previous page)

```
    [
        monitor,
        dr1,
        dr2,
        rf,
        dr2,
        dr2,
        rf,
        dr2,
        dr2,
        rf,
        dr2,
        dr2,
        rf,
        dr2,
        monitor,
    ]
)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.25: You can copy this file from `examples/rfcavity/input_rfcavity.in`.

```
##########################################################################################
# Particle Beam(s)
##########################################################################################
beam.npart = 10000  # outside tests, use 1e5 or more
beam.units = static
beam.energy = 230
beam.charge = 1.0e-10
beam.particle = electron
beam.distribution = waterbag
beam.sigmaX = 0.352498964601e-3
beam.sigmaY = 0.207443478142e-3
beam.sigmaT = 0.70399950746e-4
beam.sigmaPx = 5.161852770e-6
beam.sigmaPy = 9.163582894e-6
beam.sigmaPt = 0.260528852031e-3
beam.muxpx = 0.5712386101751441
beam.muypy = -0.514495755427526
beam.mutpt = -5.05773e-10
```

(continues on next page)

(continued from previous page)

```
################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor dr1 dr2 rf dr2 dr2 rf dr2 dr2 rf dr2 dr2 rf dr2 monitor

monitor.type = beam_monitor
monitor.backend = h5

dr1.type = drift
dr1.ds = 0.4
dr1.nslice = 1

dr2.type = drift
dr2.ds = 0.032997
dr1.nslice = 1

rf.type = rfcavity
rf.ds = 1.31879807
rf.escale = 62.0
rf.freq = 1.3e9
rf.phase = 85.5
rf.mapsteps = 100
rf.nslice = 4
rf.cos_coefficients =
                0.1644024074311037
                -0.1324009958969339
                4.3443060026047219e-002
                8.5602654094946495e-002
                -0.2433578169042885
                0.5297150596779437
                0.7164884680963959
                -5.2579522442877296e-003
                -5.5025369142193678e-002
                4.6845673335028933e-002
                -2.3279346335638568e-002
                4.0800777539657775e-003
                4.1378326533752169e-003
                -2.5040533340490805e-003
                -4.0654981400000964e-003
                9.6630592067498289e-003
                -8.5275895985990214e-003
                -5.8078747006425020e-002
                -2.4044337836660403e-002
                1.0968240064697212e-002
                -3.4461179858301418e-003
                -8.1201564869443749e-004
                2.1438992904959380e-003
                -1.4997753525697276e-003
                1.8685171825676386e-004
rf.sin_coefficients = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                0 0 0 0 0 0 0
```

(continues on next page)

(continued from previous page)

```
################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = false


################################################################################
# Diagnostics
################################################################################
diag.slice_step_diagnostics = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_rfcavity.py`

Listing 3.26: You can copy this file from `examples/rfcavity/analysis_rfcavity.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
```

(continues on next page)

(continued from previous page)

```
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        4.29466150443e-4,
        2.41918588389e-4,
        7.0399951912e-5,
        2.21684103818e-9,
        2.21684103818e-9,
        1.83412186547e-8,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
```

(continues on next page)

(continued from previous page)

```
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        3.52596000000e-4,
        2.41775000000e-4,
        7.0417917357e-5,
        1.70893497973e-9,
        1.70893497973e-9,
        1.413901564889e-8,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.9 FODO Cell with RF

Stable FODO cell with short RF (buncher) cavities added for longitudinal focusing. The phase advance in all three phase planes is between 86-89 degrees.

The matched Twiss parameters at entry are:

- $\beta_x = 9.80910407$ m
- $\alpha_x = 0.0$
- $\beta_y = 1.31893788$ m
- $\alpha_y = 0.0$
- $\beta_t = 4.6652668782$ m
- $\alpha_t = 0.0$

We use a 250 MeV proton beam with initial unnormalized rms emittance of 1 mm-mrad in all three phase planes.

The second moments of the particle distribution after the FODO cell should coincide with the second moments of the particle distribution before the FODO cell, to within the level expected due to noise due to statistical sampling.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_fodo_rf.py`) or with an app with an input file (`impactx input_fodo_rf.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.27: You can copy this file from `examples/fodo_rf/run_fodo_rf.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 250 MeV proton beam with an initial
# unnormalized rms emittance of 1 mm-mrad in all
# three phase planes
energy_MeV = 250.0  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=3.131948925200e-3,
    sigmaY=1.148450209423e-3,
    sigmaT=2.159922887089e-3,
    sigmaPx=3.192900088357e-4,
    sigmaPy=8.707386631090e-4,
    sigmaPt=4.62979491526e-4,
```

(continues on next page)

(continued from previous page)

```
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
sim.lattice.append(monitor)
#   Quad elements
quad1 = elements.Quad(ds=0.15, k=2.5)
quad2 = elements.Quad(ds=0.3, k=-2.5)
#   Drift element
drift1 = elements.Drift(ds=1.0)
#   Short RF cavity element
shortrf1 = elements.ShortRF(V=0.01, k=15.0)

lattice_no_drifts = [quad1, shortrf1, quad2, shortrf1, quad1]
#   set first lattice element
sim.lattice.append(lattice_no_drifts[0])
#   intersperse all remaining elements of the lattice with a drift element
for element in lattice_no_drifts[1:]:
    sim.lattice.extend([drift1, element])

sim.lattice.append(monitor)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.28: You can copy this file from `examples/fodo_rf/input_fodo_rf.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 250.0
beam.charge = 1.0e-9
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 3.131948925200e-3
beam.sigmaY = 1.148450209423e-3
beam.sigmaT = 2.159922887089e-3
beam.sigmaPx = 3.192900088357e-4
beam.sigmaPy = 8.707386631090e-4
```

(continues on next page)

(continued from previous page)

```
beam.sigmaPt = 4.62979491526e-4
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


##################################################################################
# Beamline: lattice elements and segments
##################################################################################
lattice.elements = monitor quad1 drift1 shortrf1 drift1 quad2 drift1
                   shortrf1 drift1 quad1 monitor

monitor.type = beam_monitor
monitor.backend = h5

quad1.type = quad
quad1.ds = 0.15
quad1.k = 2.5

drift1.type = drift
drift1.ds = 1.0

shortrf1.type = shortrf
shortrf1.V = 0.01
shortrf1.k = 15.0

quad2.type = quad
quad2.ds = 0.3
quad2.k = -2.5


##################################################################################
# Algorithms
##################################################################################
algo.particle_shape = 2
algo.space_charge = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_fodo_rf.py`

Listing 3.29: You can copy this file from `examples/fodo_rf/analysis_fodo_rf.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
```

(continues on next page)

(continued from previous page)

```
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        3.145694e-03,
        1.153344e-03,
        2.155082e-03,
        9.979770e-07,
        1.008751e-06,
        1.000691e-06,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        3.112318e-03,
        1.153322e-03,
        2.166501e-03,
        9.979770e-07,
        1.008751e-06,
```

(continues on next page)

(continued from previous page)

```
        1.000691e-06,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.10 Chain of thin multipoles

A series of thin multipoles (quad, sext, oct) with both normal and skew coefficients.

We use a 2 GeV electron beam.

The second moments of x, y, and t should be unchanged, but there is large emittance growth in the x and y phase planes.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_multipole.py`) or with an app with an input file (`impactx input_multipole.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.30: You can copy this file from `examples/multipole/run_multipole.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Ryan Sandberg, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV electron beam with an initial
# unnormalized rms emittance of  nm
```

(continues on next page)

(continued from previous page)

```
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-9  # used without space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=4.0e-3,
    sigmaY=4.0e-3,
    sigmaT=1.0e-3,
    sigmaPx=3.0e-4,
    sigmaPy=3.0e-4,
    sigmaPt=2.0e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
multipole = [
    monitor,
    elements.Multipole(multiple=2, K_normal=3.0, K_skew=0.0),
    elements.Multipole(multiple=3, K_normal=100.0, K_skew=-50.0),
    elements.Multipole(multiple=4, K_normal=65.0, K_skew=6.0),
    monitor,
]
# assign a fodo segment
sim.lattice.extend(multipole)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.31: You can copy this file from `examples/multipole/input_multipole.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.0e3
```

(continues on next page)

(continued from previous page)

```
beam.charge = 1.0e-9
beam.particle = electron
beam.distribution = waterbag
beam.sigmaX = 4.0e-3
beam.sigmaY = 4.0e-3
beam.sigmaT = 1.0e-3
beam.sigmaPx = 3.0e-4
beam.sigmaPy = 3.0e-4
beam.sigmaPt = 2.0e-3
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor thin_quadrupole thin_sextupole thin_octupole monitor

monitor.type = beam_monitor
monitor.backend = h5

thin_quadrupole.type = multipole
thin_quadrupole.multipole = 2      //Thin quadrupole
thin_quadrupole.k_normal = 3.0
thin_quadrupole.k_skew = 0.0

thin_sextupole.type = multipole
thin_sextupole.multipole = 3      //Thin sextupole
thin_sextupole.k_normal = 100.0
thin_sextupole.k_skew = -50.0

thin_octupole.type = multipole
thin_octupole.multipole = 4     //Thin octupole
thin_octupole.k_normal = 65.0
thin_octupole.k_skew = 6.0


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_multipole.py`

Listing 3.32: You can copy this file from `examples/multipole/analysis_multipole.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
```

(continues on next page)

(continued from previous page)

```
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        4.017554e-03,
        4.017044e-03,
        9.977588e-04,
        1.197572e-06,
        1.210501e-06,
        2.001382e-06,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
```

(continues on next page)

(continued from previous page)

```
        4.017554e-03,
        4.017044e-03,
        9.977588e-04,
        6.532644e-06,
        6.630912e-06,
        2.001382e-06,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.11 A nonlinear focusing channel based on the IOTA nonlinear lens

A constant focusing channel with nonlinear focusing, using a string of thin IOTA nonlinear lens elements alternating with constant focusing elements.

We use a 2.5 MeV proton beam, corresponding to the nominal IOTA proton energy.

The two functions H (Hamiltonian) and I (the second invariant) should remain unchanged for all particles.

In this test, the initial and final values of $\mu_H, \sigma_H, \mu_I, \sigma_I$ must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_iotalens.py`) or with an app with an input file (`impactx input_iotalens.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.33: You can copy this file from `examples/iota_lens/run_iotalens.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Ryan Sandberg, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True
```

(continues on next page)

(continued from previous page)

```
# domain decomposition & space charge mesh
sim.init_grids()

# load a 2.5 MeV proton beam
energy_MeV = 2.5  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=2.0e-3,
    sigmaY=2.0e-3,
    sigmaT=1.0e-3,
    sigmaPx=3.0e-4,
    sigmaPy=3.0e-4,
    sigmaPt=0.0,
)
sim.add_particles(bunch_charge_C, distr, npart)

# design the accelerator lattice
constEnd = elements.ConstF(ds=0.0025, kx=1.0, ky=1.0, kt=1.0e-12)
nllens = elements.NonlinearLens(knll=2.0e-7, cnll=0.01)
const = elements.ConstF(ds=0.005, kx=1.0, ky=1.0, kt=1.0e-12)

num_lenses = 10
nllens_lattice = [constEnd] + [nllens, const] * (num_lenses - 1) + [nllens, constEnd]

# add elements to the lattice segment
sim.lattice.extend(nllens_lattice)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.34: You can copy this file from `examples/iota_lens/input_iotalens.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.5
beam.charge = 1.0e-9
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 2.0e-3
beam.sigmaY = 2.0e-3
beam.sigmaT = 1.0e-3
beam.sigmaPx = 3.0e-4
beam.sigmaPy = 3.0e-4
beam.sigmaPt = 0.0
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = const_end nllens const nllens const nllens const nllens const
                   nllens const nllens const nllens const nllens const nllens
                   const nllens const_end

nllens.type = nonlinear_lens
nllens.knll = 2.0e-7
nllens.cnll = 0.01

const_end.type = constf
const_end.ds = 0.0025
const_end.kx = 1.0
const_end.ky = 1.0
const_end.kt = 1.0e-12

const.type = constf
const.ds = 0.005
const.kx = 1.0
const.ky = 1.0
const.kt = 1.0e-12


################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
```

(continues on next page)

(continued from previous page)

```
algo.space_charge = false


################################################################################
# Diagnostics
################################################################################
diag.alpha = 0.0
diag.beta = 1.0
diag.tn = 0.4
diag.cn = 0.01
```

Analyze

We run the following script to analyze correctness:

Script `analysis_iotalens.py`

Listing 3.35: You can copy this file from `examples/iota_lens/analysis_iotalens.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
import glob

import numpy as np
import pandas as pd
from scipy.stats import moment


def get_moments(beam):
    """Calculate mean and std dev of functions defining the IOTA invariants
    Returns
    -------
    meanH, sigH, meanI, sigI
    """
    meanH = np.mean(beam["H"])
    sigH = moment(beam["H"], moment=2) ** 0.5
    meanI = np.mean(beam["I"])
    sigI = moment(beam["I"], moment=2) ** 0.5

    return (meanH, sigH, meanI, sigI)


def read_all_files(file_pattern):
    """Read in all CSV files from each MPI rank (and potentially OpenMP
    thread). Concatenate into one Pandas dataframe.
```

(continues on next page)

(continued from previous page)

```
    Returns
    -------
    pandas.DataFrame
    """
    return pd.concat(
        (
            pd.read_csv(filename, delimiter=r"\s+")
            for filename in glob.glob(file_pattern)
        ),
        axis=0,
        ignore_index=True,
    ).set_index("id")


# initial/final beam
initial = read_all_files("diags/nonlinear_lens_invariants_000000.*")
final = read_all_files("diags/nonlinear_lens_invariants_final.*")

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
meanH, sigH, meanI, sigI = get_moments(initial)
print(f"  meanH={meanH:e} sigH={sigH:e} meanI={meanI:e} sigI={sigI:e}")

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [meanH, sigH, meanI, sigI],
    [4.122650e-02, 4.235181e-02, 7.356057e-02, 8.793753e-02],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
meanH, sigH, meanI, sigI = get_moments(final)
print(f"  meanH={meanH:e} sigH={sigH:e} meanI={meanI:e} sigI={sigI:e}")

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [meanH, sigH, meanI, sigI],
    [4.122704e-02, 4.230576e-02, 7.348275e-02, 8.783157e-02],
    rtol=rtol,
```

(continues on next page)

(continued from previous page)

```
    atol=atol,
)

# join tables on particle ID, so we can compare the same particle initial->final
beam_joined = final.join(initial, lsuffix="_final", rsuffix="_initial")
# add new columns: dH and dI
beam_joined["dH"] = (beam_joined["H_initial"] - beam_joined["H_final"]).abs()
beam_joined["dI"] = (beam_joined["I_initial"] - beam_joined["I_final"]).abs()
# print(beam_joined)

# particle-wise comparison of H & I initial to final
atol = 2.0e-3
rtol = 0.0  # large number
print()
print(f"  atol={atol} (ignored: rtol~={rtol})")

print(f"  dH_max={beam_joined['dH'].max()}")
assert np.allclose(beam_joined["dH"], 0.0, rtol=rtol, atol=atol)

atol = 3.0e-3
print(f"  atol={atol} (ignored: rtol~={rtol})")
print(f"  dI_max={beam_joined['dI'].max()}")
assert np.allclose(beam_joined["dI"], 0.0, rtol=rtol, atol=atol)
```

3.4.12 The “bare” linear lattice of the Fermilab IOTA storage ring

The linear lattice of the IOTA storage ring, configured for operation with a 2.5 MeV proton beam.

The drift regions available for insertion of the special nonlinear magnetic element for integrable optics experiments are denoted `dnll`.

The second moments of the particle distribution after a single turn should coincide with the initial section moments of the particle distribution, to within the level expected due to numerical particle noise.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_iotalattice.py`) or with an app with an input file (`impactx input_iotalattice.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.36: You can copy this file from `examples/iota_lattice/run_iotalattice.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Chad Mitchell, Axel Huebl
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# init particle beam
energy_MeV = 2.5
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=1.588960728035e-3,
    sigmaY=2.496625268437e-3,
    sigmaT=1.0e-3,
    sigmaPx=2.8320397837724e-3,
    sigmaPy=1.802433091137e-3,
    sigmaPt=0.0,
)
sim.add_particles(bunch_charge_C, distr, npart)
```

(continues on next page)

(continued from previous page)

```
# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# init accelerator lattice
ns = 10  # number of slices per ds in the element

# Drift elements
dra1 = elements.Drift(ds=0.9125, nslice=ns)
dra2 = elements.Drift(ds=0.135, nslice=ns)
dra3 = elements.Drift(ds=0.725, nslice=ns)
dra4 = elements.Drift(ds=0.145, nslice=ns)
dra5 = elements.Drift(ds=0.3405, nslice=ns)
drb1 = elements.Drift(ds=0.3205, nslice=ns)
drb2 = elements.Drift(ds=0.14, nslice=ns)
drb3 = elements.Drift(ds=0.1525, nslice=ns)
drb4 = elements.Drift(ds=0.31437095, nslice=ns)
drc1 = elements.Drift(ds=0.42437095, nslice=ns)
drc2 = elements.Drift(ds=0.355, nslice=ns)
dnll = elements.Drift(ds=1.8, nslice=ns)
drd1 = elements.Drift(ds=0.62437095, nslice=ns)
drd2 = elements.Drift(ds=0.42, nslice=ns)
drd3 = elements.Drift(ds=1.625, nslice=ns)
drd4 = elements.Drift(ds=0.6305, nslice=ns)
dre1 = elements.Drift(ds=0.5305, nslice=ns)
dre2 = elements.Drift(ds=1.235, nslice=ns)
dre3 = elements.Drift(ds=0.8075, nslice=ns)

# Bend elements
rc30 = 0.822230996255981
sbend30 = elements.Sbend(ds=0.4305191429, rc=rc30)
edge30 = elements.DipEdge(psi=0.0, rc=rc30, g=0.058, K2=0.5)

rc60 = 0.772821121503940
sbend60 = elements.Sbend(ds=0.8092963858, rc=rc60)
edge60 = elements.DipEdge(psi=0.0, rc=rc60, g=0.058, K2=0.5)

# Quad elements
ds_quad = 0.21
qa1 = elements.Quad(ds=ds_quad, k=-8.78017699, nslice=ns)
qa2 = elements.Quad(ds=ds_quad, k=13.24451745, nslice=ns)
qa3 = elements.Quad(ds=ds_quad, k=-13.65151327, nslice=ns)
qa4 = elements.Quad(ds=ds_quad, k=19.75138652, nslice=ns)
qb1 = elements.Quad(ds=ds_quad, k=-10.84199727, nslice=ns)
qb2 = elements.Quad(ds=ds_quad, k=16.24844348, nslice=ns)
qb3 = elements.Quad(ds=ds_quad, k=-8.27411104, nslice=ns)
qb4 = elements.Quad(ds=ds_quad, k=-7.45719247, nslice=ns)
qb5 = elements.Quad(ds=ds_quad, k=14.03362243, nslice=ns)
qb6 = elements.Quad(ds=ds_quad, k=-12.23595641, nslice=ns)
qc1 = elements.Quad(ds=ds_quad, k=-13.18863768, nslice=ns)
qc2 = elements.Quad(ds=ds_quad, k=11.50601829, nslice=ns)
qc3 = elements.Quad(ds=ds_quad, k=-11.10445869, nslice=ns)
```

(continues on next page)

(continued from previous page)

```
qd1 = elements.Quad(ds=ds_quad, k=-6.78179218, nslice=ns)
qd2 = elements.Quad(ds=ds_quad, k=5.19026998, nslice=ns)
qd3 = elements.Quad(ds=ds_quad, k=-5.8586173, nslice=ns)
qd4 = elements.Quad(ds=ds_quad, k=4.62460039, nslice=ns)
qe1 = elements.Quad(ds=ds_quad, k=-4.49607687, nslice=ns)
qe2 = elements.Quad(ds=ds_quad, k=6.66737146, nslice=ns)
qe3 = elements.Quad(ds=ds_quad, k=-6.69148177, nslice=ns)

# build lattice: first half, qe3, then mirror
# fmt: off
lattice_half = [
    dra1, qa1, dra2, qa2, dra3, qa3, dra4, qa4, dra5,
    edge30, sbend30, edge30, drb1, qb1, drb2, qb2, drb2, qb3,
    drb3, dnll, drb3, qb4, drb2, qb5, drb2, qb6, drb4,
    edge60, sbend60, edge60, drc1, qc1, drc2, qc2, drc2, qc3, drc1,
    edge60, sbend60, edge60, drd1, qd1, drd2, qd2, drd3, qd3, drd2, qd4, drd4,
    edge30, sbend30, edge30, dre1, qe1, dre2, qe2, dre3
]
# fmt:on
sim.lattice.append(monitor)
sim.lattice.extend(lattice_half)
sim.lattice.append(qe3)
lattice_half.reverse()
sim.lattice.extend(lattice_half)
sim.lattice.append(monitor)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.37: You can copy this file from `examples/iota_lattice/input_iotalattice.in`.

```
##################################################################################
# Particle Beam(s)
##################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.5
beam.charge = 1.0e-9
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 1.588960728035e-3
beam.sigmaY = 2.496625268437e-3
beam.sigmaT = 1.0e-3
beam.sigmaPx = 2.8320397837724e-3
```

(continues on next page)

(continued from previous page)

```
beam.sigmaPy = 1.802433091137e-3
beam.sigmaPt = 0.0
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


###############################################################################
# Beamline: lattice elements and segments
###############################################################################
lattice.elements = monitor
                   dra1 qa1 dra2 qa2 dra3 qa3 dra4 qa4 dra5
                   edge30 sbend30 edge30 drb1 qb1 drb2 qb2 drb2 qb3
                   drb3 dnll drb3 qb4 drb2 qb5 drb2 qb6 drb4
                   edge60 sbend60 edge60 drc1 qc1 drc2 qc2 drc2 qc3 drc1
                   edge60 sbend60 edge60 drd1 qd1 drd2 qd2 drd3 qd3 drd2 qd4 drd4
                   edge30 sbend30 edge30 dre1 qe1 dre2 qe2 dre3 qe3
                   dre3 qe2 dre2 qe1 dre1 edge30 sbend30 edge30
                   drd4 qd4 drd2 qd3 drd3 qd2 drd2 qd1 drd1 edge60 sbend60 edge60
                   drc1 qc3 drc2 qc2 drc2 qc1 drc1 edge60 sbend60 edge60
                   drb4 qb6 drb2 qb5 drb2 qb4 drb3 dnll drb3
                   qb3 drb2 qb2 drb2 qb1 drb1 edge30 sbend30 edge30
                   dra5 qa4 dra4 qa3 dra3 qa2 dra2 qa1 dra1
                   monitor

lattice.nslice = 10


# Drift elements:

dra1.type = drift
dra1.ds = 0.9125

dra2.type = drift
dra2.ds = 0.135

dra3.type = drift
dra3.ds = 0.725

dra4.type = drift
dra4.ds = 0.145

dra5.type = drift
dra5.ds = 0.3405

drb1.type = drift
drb1.ds = 0.3205

drb2.type = drift
drb2.ds = 0.14

drb3.type = drift
```

(continues on next page)

(continued from previous page)

```
drb3.ds = 0.1525

drb4.type = drift
drb4.ds = 0.31437095

drc1.type = drift
drc1.ds = 0.42437095

drc2.type = drift
drc2.ds = 0.355

dnll.type = drift
dnll.ds = 1.8

drd1.type = drift
drd1.ds = 0.62437095

drd2.type = drift
drd2.ds = 0.42

drd3.type = drift
drd3.ds = 1.625

drd4.type = drift
drd4.ds = 0.6305

dre1.type = drift
dre1.ds = 0.5305

dre2.type = drift
dre2.ds = 1.235

dre3.type = drift
dre3.ds = 0.8075


# Bend elements:

sbend30.type = sbend
sbend30.ds = 0.4305191429
sbend30.rc = 0.822230996255981

edge30.type = dipedge
edge30.psi = 0.0
edge30.rc = 0.822230996255981
edge30.g = 0.058
edge30.K2 = 0.5

sbend60.type = sbend
sbend60.ds = 0.8092963858
sbend60.rc = 0.772821121503940
```

(continues on next page)

(continued from previous page)

```
edge60.type = dipedge
edge60.psi = 0.0
edge60.rc = 0.772821121503940
edge60.g = 0.058
edge60.K2 = 0.5


# Quad elements:

qa1.type = quad
qa1.ds = 0.21
qa1.k = -8.78017699

qa2.type = quad
qa2.ds = 0.21
qa2.k = 13.24451745

qa3.type = quad
qa3.ds = 0.21
qa3.k = -13.65151327

qa4.type = quad
qa4.ds = 0.21
qa4.k = 19.75138652

qb1.type = quad
qb1.ds = 0.21
qb1.k = -10.84199727

qb2.type = quad
qb2.ds = 0.21
qb2.k = 16.24844348

qb3.type = quad
qb3.ds = 0.21
qb3.k = -8.27411104

qb4.type = quad
qb4.ds = 0.21
qb4.k = -7.45719247

qb5.type = quad
qb5.ds = 0.21
qb5.k = 14.03362243

qb6.type = quad
qb6.ds = 0.21
qb6.k = -12.23595641

qc1.type = quad
qc1.ds = 0.21
qc1.k = -13.18863768
```

(continues on next page)

(continued from previous page)

```
qc2.type = quad
qc2.ds = 0.21
qc2.k = 11.50601829

qc3.type = quad
qc3.ds = 0.21
qc3.k = -11.10445869

qd1.type = quad
qd1.ds = 0.21
qd1.k = -6.78179218

qd2.type = quad
qd2.ds = 0.21
qd2.k = 5.19026998

qd3.type = quad
qd3.ds = 0.21
qd3.k = -5.8586173

qd4.type = quad
qd4.ds = 0.21
qd4.k = 4.62460039

qe1.type = quad
qe1.ds = 0.21
qe1.k = -4.49607687

qe2.type = quad
qe2.ds = 0.21
qe2.k = 6.66737146

qe3.type = quad
qe3.ds = 0.21
qe3.k = -6.69148177

# Beam Monitor: Diagnostics
monitor.type = beam_monitor
monitor.backend = h5


#################################################################################
# Algorithms
#################################################################################
algo.particle_shape = 2
algo.space_charge = false


#################################################################################
# Diagnostics
#################################################################################
```

(continues on next page)

(continued from previous page)

```
diag.slice_step_diagnostics = true
```

Analyze

We run the following script to analyze correctness:

Script `analysis_iotalattice.py`

Listing 3.38: You can copy this file from `examples/iota_lattice/analysis_iotalattice.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)
```

(continues on next page)

(continued from previous page)

```
# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [1.595934e-03, 2.507263e-03, 9.977588e-04, 4.490896e-06, 4.539378e-06, 0.000000e00],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # a big number
rtol = 1.5 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.579848e-03,
        2.510900e-03,
        1.208202e-02,
```

(continues on next page)

(continued from previous page)

```
        4.490897e-06,
        4.539378e-06,
        0.0,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.13 Solenoid channel

Proton beam undergoing 90 deg X-Y rotation in an ideal solenoid channel.

The matched Twiss parameters at entry are:

- $\beta_x = 2.4321374875$ m
- $\alpha_x = 0.0$
- $\beta_y = 2.4321374875$ m
- $\alpha_y = 0.0$

We use a 250 MeV proton beam with initial unnormalized rms emittance of 1 micron in the horizontal plane, and 2 micron in the vertical plane.

The solenoid magnetic field corresponds to B = 2 T.

The second moments of the particle distribution after the solenoid channel are rotated by 90 degrees: the final horizontal moments should coincide with the initial vertical moments, and vice-versa, to within the level expected due to noise due to statistical sampling.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_solenoid.py`) or with an app with an input file (`impactx input_solenoid.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.39: You can copy this file from `examples/solenoid/run_solenoid.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Marco Garten, Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
```

(continues on next page)

(continued from previous page)

```
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 250 MeV proton beam with an initial
# horizontal rms emittance of 1 um and an
# initial vertical rms emittance of 2 um
energy_MeV = 250.0  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=1.559531175539e-3,
    sigmaY=2.205510139392e-3,
    sigmaT=1.0e-3,
    sigmaPx=6.41218345413e-4,
    sigmaPy=9.06819680526e-4,
    sigmaPt=1.0e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
sim.lattice.extend(
    [
        monitor,
        elements.Sol(ds=3.820395, ks=0.8223219329893234),
        monitor,
    ]
)

# run simulation
sim.evolve()

# clean shutdown
del sim
```

(continues on next page)

(continued from previous page)

```
amrex.finalize()
```

App Input File

Listing 3.40: You can copy this file from `examples/solenoid/input_solenoid.in`.

```
##################################################################################
# Particle Beam(s)
##################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 250.0
beam.charge = 1.0e-9
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 1.559531175539e-3
beam.sigmaY = 2.205510139392e-3
beam.sigmaT = 1.0e-3
beam.sigmaPx = 6.41218345413e-4
beam.sigmaPy = 9.06819680526e-4
beam.sigmaPt = 1.0e-3
beam.muxpx = 0.0
beam.muypy = 0.0
beam.mutpt = 0.0


##################################################################################
# Beamline: lattice elements and segments
##################################################################################
lattice.elements = monitor sol1 monitor
lattice.nslice = 1

monitor.type = beam_monitor
monitor.backend = h5

sol1.type = solenoid
sol1.ds = 3.820395
sol1.ks = 0.8223219329893234


##################################################################################
# Algorithms
##################################################################################
algo.particle_shape = 2
algo.space_charge = false


##################################################################################
# Diagnostics
##################################################################################
```

(continues on next page)

(continued from previous page)

```
diag.slice_step_diagnostics = true
```

Analyze

We run the following script to analyze correctness:

Script `analysis_solenoid.py`

Listing 3.41: You can copy this file from `examples/solenoid/analysis_solenoid.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)
```

(continues on next page)

(continued from previous page)

```
# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.559531175539e-3,
        2.205510139392e-3,
        1.0e-3,
        1.0e-6,
        2.0e-6,
        1.0e-6,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")
```

(continues on next page)

(continued from previous page)

```
assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        2.205510139392e-3,
        1.559531175539e-3,
        6.404930308742e-3,
        2.0e-6,
        1.0e-6,
        1.0e-6,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.14 Soft-edge solenoid

Proton beam propagating through a 6 m region containing a soft-edge solenoid.

The solenoid model used is the default thin-shell model described in: P. Granum et al, "Efficient calculations of magnetic fields of solenoids for simulations," NIMA 1034, 166706 (2022) DOI:10.1016/j.nima.2022.166706

The solenoid is a cylindrical current sheet with a length of 1 m and a radius of 0.1667 m, corresponding to an aspect ratio diameter/length = 1/3. The peak magnetic field on-axis is 3 T.

We use a 250 MeV proton beam with initial unnormalized rms emittance of 1 micron in the horizontal plane, and 2 micron in the vertical plane.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_solenoid_softedge.py`) or with an app with an input file (`impactx input_solenoid_softedge.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.42: You can copy this file from `examples/solenoid_softedge/run_solenoid_softedge.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Chad Mitchell, Axel Huebl
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements
```

(continues on next page)

(continued from previous page)

```
sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = False

# domain decomposition & space charge mesh
sim.init_grids()

# load a 250 MeV proton beam with an initial
# horizontal rms emittance of 1 um and an
# initial vertical rms emittance of 2 um
energy_MeV = 250.0  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(1.0).set_mass_MeV(938.27208816).set_energy_MeV(energy_MeV)

#   particle bunch
distr = distribution.Waterbag(
    sigmaX=1.559531175539e-3,
    sigmaY=2.205510139392e-3,
    sigmaT=1.0e-3,
    sigmaPx=6.41218345413e-4,
    sigmaPy=9.06819680526e-4,
    sigmaPt=1.0e-3,
)
sim.add_particles(bunch_charge_C, distr, npart)

# design the accelerator lattice
sol = elements.SoftSolenoid(
    ds=6.0,
    bscale=1.233482899483985,
    cos_coefficients=[
        0.350807812299706,
        0.323554693720069,
        0.260320578919415,
        0.182848575294969,
        0.106921016050403,
        4.409581845710694e-002,
        -9.416427163897508e-006,
        -2.459452716865687e-002,
        -3.272762575737291e-002,
        -2.936414401076162e-002,
        -1.995780078926890e-002,
        -9.102893342953847e-003,
        -2.456410658713271e-006,
```

(continues on next page)

(continued from previous page)

```
        5.788233017324325e-003,
        8.040408292420691e-003,
        7.480064552867431e-003,
        5.230254569468851e-003,
        2.447614547094685e-003,
        -1.095525090532255e-006,
        -1.614586867387170e-003,
        -2.281365457438345e-003,
        -2.148709081338292e-003,
        -1.522541739363011e-003,
        -7.185505862719508e-004,
        -6.171194824600157e-007,
        4.842109305036943e-004,
        6.874508102002901e-004,
        6.535550288205728e-004,
        4.648795813759210e-004,
        2.216564722797528e-004,
        -4.100982995210341e-007,
        -1.499332112463395e-004,
        -2.151538438342482e-004,
        -2.044590946652016e-004,
        -1.468242784844341e-004,
    ],
    sin_coefficients=[
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
```

(continues on next page)

(continued from previous page)

```
        0,
        0,
        0,
        0,
        0,
        0,
        0,
    ],
    mapsteps=200,
    nslice=4,
)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

sim.lattice.extend(
    [
        monitor,
        sol,
        monitor,
    ]
)

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.43: You can copy this file from `examples/solenoid_softedge/input_solenoid_softedge.in`.

```
#################################################################################
# Particle Beam(s)
#################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 250.0
beam.charge = 1.0e-9
beam.particle = proton
beam.distribution = waterbag
beam.sigmaX = 1.559531175539e-3
beam.sigmaY = 2.205510139392e-3
beam.sigmaT = 1.0e-3
beam.sigmaPx = 6.41218345413e-4
beam.sigmaPy = 9.06819680526e-4
beam.sigmaPt = 1.0e-3
beam.muxpx = 0.0
```

(continues on next page)

(continued from previous page)

```
beam.muypy = 0.0
beam.mutpt = 0.0


##################################################################################
# Beamline: lattice elements and segments
##################################################################################
lattice.elements = monitor sol1 monitor
lattice.nslice = 1

monitor.type = beam_monitor
monitor.backend = h5

sol1.type = solenoid_softedge
sol1.ds = 6.0
sol1.bscale = 1.233482899483985
sol1.mapsteps = 800


##################################################################################
# Algorithms
##################################################################################
algo.particle_shape = 2
algo.space_charge = false


##################################################################################
# Diagnostics
##################################################################################
diag.slice_step_diagnostics = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_solenoid_softedge.py`

Listing 3.44: You can copy this file from `examples/solenoid_softedge/analysis_solenoid_softedge.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Chad Mitchell, Axel Huebl
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment
```

(continues on next page)

(continued from previous page)

```
def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5

    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 2.0 * num_particles**-0.5  # from random sampling of a smooth distribution
```

(continues on next page)

(continued from previous page)

```
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        1.559531175539e-3,
        2.205510139392e-3,
        1.0e-3,
        1.0e-6,
        2.0e-6,
        1.0e-6,
    ],
    rtol=rtol,
    atol=atol,
)


print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = 2.0 * num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        2.425578906459e-3,
        2.654015302646e-3,
        9.985897906860e-3,
        1.365357890e-6,
        1.634641555e-6,
        1.000000000e-6,
    ],
    rtol=rtol,
    atol=atol,
)
```

3.4.15 Soft-Edge Quadrupole

This is a modification of *the test for a matched electron beam propagating through a stable FODO cell*, in which the quadrupoles have been replaced with soft-edge quadrupole elements. The on-axis field profile in this example has been chosen to correspond to the hard-edge limit, so the two tests should coincide.

We use a 2 GeV electron beam with initial unnormalized rms emittance of 2 nm.

In this test, the initial and final values of σ_x, σ_y, σ_t, ϵ_x, ϵ_y, and ϵ_t must agree with nominal values.

Run

This example can be run as a Python script (`python3 run_quadrupole_softedge.py`) or with an app with an input file (`impactx input_quadrupole_softedge.in`). Each can also be prefixed with an MPI executor, such as `mpiexec -n 4 ...` or `srun -n 4 ...`, depending on the system.

Python Script

Listing 3.45: You can copy this file from `examples/quadrupole_softedge/run_quadrupole_softedge.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#
# -*- coding: utf-8 -*-

import amrex
from impactx import ImpactX, RefPart, distribution, elements

sim = ImpactX()

# set numerical parameters and IO control
sim.particle_shape = 2  # B-spline order
sim.space_charge = False
# sim.diagnostics = False  # benchmarking
sim.slice_step_diagnostics = True

# domain decomposition & space charge mesh
sim.init_grids()

# load a 2 GeV electron beam with an initial
# unnormalized rms emittance of 2 nm
energy_MeV = 2.0e3  # reference energy
bunch_charge_C = 1.0e-9  # used with space charge
npart = 10000  # number of macro particles

#   reference particle
ref = sim.particle_container().ref_particle()
ref.set_charge_qe(-1.0).set_mass_MeV(0.510998950).set_energy_MeV(energy_MeV)
```

(continues on next page)

(continued from previous page)

```
#   particle bunch
distr = distribution.Waterbag(
    sigmaX=3.9984884770e-5,
    sigmaY=3.9984884770e-5,
    sigmaT=1.0e-3,
    sigmaPx=2.6623538760e-5,
    sigmaPy=2.6623538760e-5,
    sigmaPt=2.0e-3,
    muxpx=-0.846574929020762,
    muypy=0.846574929020762,
    mutpt=0.0,
)
sim.add_particles(bunch_charge_C, distr, npart)

# add beam diagnostics
monitor = elements.BeamMonitor("monitor", backend="h5")

# design the accelerator lattice
ns = 1  # number of slices per ds in the element

quad1 = elements.SoftQuadrupole(
    ds=1.0,
    gscale=1.0,
    cos_coefficients=[2],
    sin_coefficients=[0],
    mapsteps=400,
    nslice=ns,
)

quad2 = elements.SoftQuadrupole(
    ds=1.0,
    gscale=-1.0,
    cos_coefficients=[2],
    sin_coefficients=[0],
    mapsteps=200,
    nslice=ns,
)

drift1 = elements.Drift(ds=0.25, nslice=ns)
drift2 = elements.Drift(ds=0.5, nslice=ns)

# assign a fodo segment
sim.lattice.extend([monitor, drift1, quad1, drift2, quad2, drift1, monitor])

# run simulation
sim.evolve()

# clean shutdown
del sim
amrex.finalize()
```

App Input File

Listing 3.46: You can copy this file from `examples/quadrupole_softedge/input_quadrupole_softedge.in`.

```
################################################################################
# Particle Beam(s)
################################################################################
beam.npart = 10000
beam.units = static
beam.energy = 2.0e3
beam.charge = 1.0e-9
beam.particle = electron
beam.distribution = waterbag
beam.sigmaX = 3.9984884770e-5
beam.sigmaY = 3.9984884770e-5
beam.sigmaT = 1.0e-3
beam.sigmaPx = 2.6623538760e-5
beam.sigmaPy = 2.6623538760e-5
beam.sigmaPt = 2.0e-3
beam.muxpx = -0.846574929020762
beam.muypy = 0.846574929020762
beam.mutpt = 0.0


################################################################################
# Beamline: lattice elements and segments
################################################################################
lattice.elements = monitor drift1 quad1 drift2 quad2 drift1 monitor
lattice.nslice = 1

monitor.type = beam_monitor
monitor.backend = h5

drift1.type = drift
drift1.ds = 0.25

quad1.type = quadrupole_softedge
quad1.ds = 1.0
quad1.gscale = 1.0
quad1.cos_coefficients = 2.0
quad1.sin_coefficients = 0.0
quad1.mapsteps = 400

drift2.type = drift
drift2.ds = 0.5

quad2.type = quadrupole_softedge
quad2.ds = 1.0
quad2.gscale = -1.0
quad2.cos_coefficients = 2.0
quad2.sin_coefficients = 0.0
quad2.mapsteps = 400
```

(continues on next page)

(continued from previous page)

```
################################################################################
# Algorithms
################################################################################
algo.particle_shape = 2
algo.space_charge = false


################################################################################
# Diagnostics
################################################################################
diag.slice_step_diagnostics = false
```

Analyze

We run the following script to analyze correctness:

Script `analysis_quadrupole_softedge.py`

Listing 3.47: You can copy this file from `examples/quadrupole_softedge/analysis_quadrupole_softedge.py`.

```
#!/usr/bin/env python3
#
# Copyright 2022-2023 ImpactX contributors
# Authors: Axel Huebl, Chad Mitchell
# License: BSD-3-Clause-LBNL
#

import numpy as np
import openpmd_api as io
from scipy.stats import moment


def get_moments(beam):
    """Calculate standard deviations of beam position & momenta
    and emittance values

    Returns
    -------
    sigx, sigy, sigt, emittance_x, emittance_y, emittance_t
    """
    sigx = moment(beam["position_x"], moment=2) ** 0.5  # variance -> std dev.
    sigpx = moment(beam["momentum_x"], moment=2) ** 0.5
    sigy = moment(beam["position_y"], moment=2) ** 0.5
    sigpy = moment(beam["momentum_y"], moment=2) ** 0.5
    sigt = moment(beam["position_ct"], moment=2) ** 0.5
    sigpt = moment(beam["momentum_t"], moment=2) ** 0.5
```

(continues on next page)

(continued from previous page)

```
    epstrms = beam.cov(ddof=0)
    emittance_x = (
        sigx**2 * sigpx**2 - epstrms["position_x"]["momentum_x"] ** 2
    ) ** 0.5
    emittance_y = (
        sigy**2 * sigpy**2 - epstrms["position_y"]["momentum_y"] ** 2
    ) ** 0.5
    emittance_t = (
        sigt**2 * sigpt**2 - epstrms["position_ct"]["momentum_t"] ** 2
    ) ** 0.5

    return (sigx, sigy, sigt, emittance_x, emittance_y, emittance_t)


# initial/final beam
series = io.Series("diags/openPMD/monitor.h5", io.Access.read_only)
last_step = list(series.iterations)[-1]
initial = series.iterations[1].particles["beam"].to_df()
final = series.iterations[last_step].particles["beam"].to_df()

# compare number of particles
num_particles = 10000
assert num_particles == len(initial)
assert num_particles == len(final)

print("Initial Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(initial)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        7.545117045417507 3e-005,
        7.5441588239210947e-005,
        9.9775878164077539e-004,
        1.9959540393751392e-009,
        2.0175015289132990e-009,
        2.0013820193294972e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

(continues on next page)

(continued from previous page)

```
print("")
print("Final Beam:")
sigx, sigy, sigt, emittance_x, emittance_y, emittance_t = get_moments(final)
print(f"  sigx={sigx:e} sigy={sigy:e} sigt={sigt:e}")
print(
    f"  emittance_x={emittance_x:e} emittance_y={emittance_y:e} emittance_t={emittance_
↪t:e}"
)

atol = 0.0  # ignored
rtol = num_particles**-0.5  # from random sampling of a smooth distribution
print(f"  rtol={rtol} (ignored: atol~={atol})")

assert np.allclose(
    [sigx, sigy, sigt, emittance_x, emittance_y, emittance_t],
    [
        7.4790118496224206e-005,
        7.5357525169680140e-005,
        9.9775879288128088e-004,
        1.9959539836392703e-009,
        2.0175014668882125e-009,
        2.0013820380883801e-006,
    ],
    rtol=rtol,
    atol=atol,
)
```

For every change of the ImpactX code base, each of these examples are continuously tested and benchmarked.

3.5 Workflows

This section collects typical user workflows and best practices for ImpactX.

Note: TODO: Add more workflows as in https://warpx.readthedocs.io/en/latest/usage/workflows.html

CHAPTER

FOUR

DATA ANALYSIS

4.1 Data Analysis

Note: TODO :-)

Please see https://warpx.readthedocs.io/en/latest/dataanalysis/formats.html for now.

CHAPTER

FIVE

THEORY

5.1 Introduction

5.1.1 Assumptions

This is a work-in-progress list of physical assumptions implemented in the numerics of ImpactX.

Tracking and Lattice Optics

- **tracking through lattice optics:** is treated through linear order with respect to the reference particle
  - **velocity spread:** the above linearization implies that, when solving space-charge effects, we assume that the relative spread of velocities of particles in the beam is negligible compared to the velocity of the reference particle

Space Charge (Poisson Solver)

- **electrostatic in the bunch frame:** we assume there are no retardation effects and we solve the Poisson equation in the bunch frame

CHAPTER

SIX

DEVELOPMENT

6.1 Contribute to ImpactX

We welcome new contributors! Here is how to participate to the ImpactX development.

6.1.1 Git workflow

The ImpactX project uses git for version control. If you are new to git, you can follow one of these tutorials:

- Learn git with bitbucket
- git - the simple guide

Configure your GitHub Account & Development Machine

First, let's setup your Git environment and GitHub account.

1. Go to https://github.com/settings/profile and add your real name and affiliation
2. Go to https://github.com/settings/emails and add & verify the professional e-mails you want to be associated with.
3. Configure `git` on the machine you develop on to *use the same spelling of your name and email*:
   - `git config --global user.name "FIRSTNAME LASTNAME"`
   - `git config --global user.email EMAIL@EXAMPLE.com`
4. Go to https://github.com/settings/keys and add the SSH public key of the machine you develop on. (Check out the GitHub guide to generating SSH keys or troubleshoot common SSH problems.)

Make your own fork

First, fork the ImpactX "mainline" repo on GitHub by pressing the *Fork* button on the top right of the page. A fork is a copy of ImpactX on GitHub, which is under your full control.

Then, we create local copies, for development:

```
# Clone the mainline ImpactX source code to your local computer.
# You cannot write to this repository, but you can read from it.
git clone git@github.com:ECP-WarpX/impactx.git
cd impactx
```

(continues on next page)

(continued from previous page)

```
# rename what we just cloned: call it "mainline"
git remote rename origin mainline

# Add your own fork. You can get this address on your fork's Github page.
# Here is where you will publish new developments, so that they can be
# reviewed and integrated into "mainline" later on.
# "myGithubUsername" needs to be replaced with your user name on GitHub.
git remote add myGithubUsername git@github.com:myGithubUsername/impactx.git
```

Now you are free to play with your fork (for additional information, you can visit the Github fork help page).

Note: We only need to do the above steps for the first time.

Let's Develop

You are all set! Now, the basic ImpactX development workflow is:

1. Implement your changes and push them on a new branch `branch_name` on your fork.
2. Create a Pull Request from branch `branch_name` on your fork to branch `development` on the main ImpactX repo.

Create a branch `branch_name` (the branch name should reflect the piece of code you want to add, like `fix-spectral-solver`) with

```
# start from an up-to-date development branch
git checkout development
git pull mainline development

# create a fresh branch
git checkout -b branch_name
```

and do the coding you want.

It is probably a good time to look at the AMReX documentation and at the Doxygen reference pages:

- ImpactX Doxygen: https://impactx.readthedocs.io/en/latest/_static/doxyhtml
- AMReX Doxygen: https://amrex-codes.github.io/amrex/doxygen
- WarpX Doxygen: https://warpx.readthedocs.io/en/latest/_static/doxyhtml

Once you are done developing, add the files you created and/or modified to the `git` *staging area* with

```
git add <file_I_created> <and_file_I_modified>
```

Build your changes

If you changed C++ files, then now is a good time to test those changes by compiling ImpactX locally. Follow the developer instructions in our manual to set up a local development environment, then compile and run ImpactX.

Commit & push your changes

Periodically commit your changes with

```
git commit
```

The commit message (between quotation marks) is super important in order to follow the developments during code-review and identify bugs. A typical format is:

```
This is a short, 40-character title

After a newline, you can write arbitray paragraphs. You
usually limit the lines to 70 characters, but if you don't, then
nothing bad will happen.

The most important part is really that you find a descriptive title
and add an empty newline after it.
```

For the moment, commits are on your local repo only. You can push them to your fork with

```
git push -u myGithubUsername branch_name
```

If you want to synchronize your branch with the `development` branch (this is useful when the `development` branch is being modified while you are working on `branch_name`), you can use

```
git pull mainline development
```

and fix any conflict that may occur.

Submit a Pull Request

A Pull Request (PR) is the way to efficiently visualize the changes you made and to propose your new feature/improvement/fix to the ImpactX project. Right after you push changes, a banner should appear on the Github page of your fork, with your `branch_name`.

- Click on the `compare & pull request` button to prepare your PR.
- It is time to communicate your changes: write a title and a description for your PR. People who review your PR are happy to know
  - what feature/fix you propose, and why
  - how you made it (added new/edited files, created a new class than inherits from…)
  - how you tested it and what was the output you got
  - and anything else relevant to your PR (attach images and scripts, link papers, *etc.*)
- Press `Create pull request`. Now you can navigate through your PR, which highlights the changes you made.

Please DO NOT write large pull requests, as they are very difficult and time-consuming to review. As much as possible, split them into small, targeted PRs. For example, if find typos in the documentation open a pull request that only fixes typos. If you want to fix a bug, make a small pull request that only fixes a bug.

If you want to implement a feature and are not too sure how to split it, just open an issue about your plans and ping other ImpactX developers on it to chime in. Generally, write helper functionality first, test it and then write implementation code. Submit tests, documentation changes and implementation of a feature together for pull request review.

Even before your work is ready to merge, it can be convenient to create a PR (so you can use Github tools to visualize your changes). In this case, please put the `[WIP]` tag (for Work-In-Progress) at the beginning of the PR title. You can also use the GitHub project tab in your fork to organize the work into separate tasks/PRs and share it with the ImpactX community to get feedback.

Include a test to your PR

A new feature is great, a **working** new feature is even better! Please test your code and add your test to the automated test suite. It's the way to protect your work from adventurous developers.

Note: TOOD: Write a workflow how to add a test.

Include documentation about your PR

Now, let users know about your new feature by describing its usage in the ImpactX documentation. Our documentation uses Sphinx, and it is located in `docs/source/`.

Note: TODO: For instance, if you introduce a new runtime parameter in the input file, you can add it to *Docs/source/running_cpp/parameters.rst*.

If Sphinx is installed on your computer, you should be able to generate the html documentation with

```
make html
```

in `docs/`. Then open `docs/build/html/index.html` with your favorite web browser and look for your changes.

Once your code is ready with documentation and automated test, congratulations! You can create the PR (or remove the `[WIP]` tag if you already created it). Reviewers will interact with you if they have comments/questions.

6.1.2 Style and conventions

- For indentation, ImpactX uses four spaces (no tabs)
- Some text editors automatically modify the files you open. We recommend to turn on to remove trailing spaces and replace Tabs with 4 spaces.
- The number of characters per line should be <100
- Exception: in documentation files (`.rst`/`.md`) use one sentence per line independent of its number of characters, which will allow easier edits.
- Space before and after assignment operator (=)
- To define a function , for e.g., `myfunction()` use a space between the name of the function and the paranthesis - `myfunction ()`. To call the function, the space is not required, i.e., just use `myfunction()`.

- The reason this is beneficial is that when we do a `git grep` to search for `myfunction ()`, we can clearly see the locations where `myfunction ()` is defined and where `myfunction()` is called.
- Also, using `git grep "myfunction ()"` searches for files only in the git repo, which is more efficient compared to the `grep "myfunction ()"` command that searches through all the files in a directory, including plotfiles for example.
- It is recommended that style changes are not included in the PR where new code is added. This is to avoid any errors that may be introduced in a PR just to do style change.
- ImpactX uses `CamelCase` convention for file names and class names, rather than `snake_case`.
- The names of all member variables should be prefixed with `m_`. This is particularly useful to avoid capturing member variables by value in a lambda function, which causes the whole object to be copied to GPU when running on a GPU-accelerated architecture. This convention should be used for all new piece of code, and it should be applied progressively to old code.
- `#include` directives in C++ have a distinct order to avoid bugs, see *the ImpactX repo structure* for details
- For all new code, we should avoid relying on `using namespace amrex;` and all amrex types should be prefixed with `amrex::`. Inside limited scopes, AMReX type literals can be included with `using namespace amrex::literals;`. Ideally, old code should be modified accordingly.

6.2 Testing

6.2.1 Preparation

Prepare for running tests of ImpactX by *building ImpactX from source*.

In order to run our tests, you need to have a few *Python packages installed*:

```
python3 -m pip install -U pip setuptools wheel pytest
python3 -m pip install -r examples/requirements.txt
```

6.2.2 Run

You can run all our tests with:

```
ctest --test-dir build --output-on-failure
```

6.2.3 Further Options

- help: `ctest --test-dir build --help`
- list all tests: `ctest --test-dir build -N`
- only run tests that have "FODO" in their name: `ctest --test-dir build -R FODO`

6.3 Documentation

6.3.1 Doxygen documentation

WarpX uses a Doxygen documentation. Whenever you create a new class, please document it where it is declared (typically in the header file):

```
/** A brief title
 *
 * few-line description explaining the purpose of my_class.
 *
 * If you are kind enough, also quickly explain how things in my_class work.
 * (typically a few more lines)
 */
class my_class
{ ... }
```

Doxygen reads this docstring, so please be accurate with the syntax! See Doxygen manual for more information. Similarly, please document functions when you declare them (typically in a header file) like:

```
/** A brief title
 *
 * few-line description explaining the purpose of my_function.
 *
 * \param[in,out] my_int a pointer to an integer variable on which
 *                       my_function will operate.
 * \return what is the meaning and value range of the returned value
 */
int my_class::my_function(int* my_int);
```

An online version of this documentation is *linked here*.

6.3.2 Breathe documentation

Your Doxygen documentation is not only useful for people looking into the code, it is also part of the ImpactX online documentation based on Sphinx! This is done using the Python module Breathe, that allows you to read Doxygen documentation dorectly in the source and include it in your Sphinx documentation, by calling Breathe functions. For instance, the following line will get the Doxygen documentation for `ImpactXParticleContainer` in `src/particles/ImpactXParticleContainer.H` and include it to the html page generated by Sphinx:

class **ImpactXParticleContainer** : public amrex::ParticleContainer<0, 0, RealSoA::nattribs, IntSoA::nattribs>

Beam Particles in ImpactX

This class stores particles, distributed over MPI ranks.

6.3.3 Building the documentation

To build the documentation on your local computer, you will need to install Doxygen as well as the Python module `breathe`. First, change into `docs/` and install the Python requirements:

```
cd docs/
pip install -r requirements.txt
```

You will also need Doxygen (macOS: `brew install doxygen`; Ubuntu: `sudo apt install doxygen`).

Then, to compile the documentation, use

```
make html
# This will first compile the Doxygen documentation (execute doxygen)
# and then build html pages from rst files using sphinx and breathe.
```

Open the created `build/html/index.html` file with your favorite browser. Rebuild and refresh as needed.

6.4 ImpactX Structure

6.4.1 Repo Organization

All the ImpactX source code is located in `src/`. All sub-directories have a pretty straightforward name.

Here is a visual representation of the repository structure.

6.4.2 Code organization

The main ImpactX class is ImpactX, implemented in `src/ImpactX.cpp`.

6.4.3 Build System

ImpactX uses the *CMake build system generator*. Each sub-folder contains a file `CMakeLists.txt` with the names of the source files (`.cpp`) that are added to the build. Do not list header files (`.H`) here.

6.4.4 C++ Includes

All ImpactX header files need to be specified relative to the `src/` directory.

- e.g. `#include "Utils/ImpactXConst.H"`
- files in the same directory as the including header-file can be included with `#include "FileName.H"`

By default, in a `MyName.cpp` source file we do not include headers already included in `MyName.H`. Besides this exception, if a function or a class is used in a source file, the header file containing its declaration must be included, unless the inclusion of a facade header is more appropriate. This is sometimes the case for AMReX headers. For instance `AMReX_GpuLaunch.H` is a façade header for `AMReX_GpuLaunchFunctsC.H` and `AMReX_GpuLaunchFunctsG.H`, which contain respectively the CPU and the GPU implemetation of some methods, and which should not be included directly. Whenever possible, forward declarations headers are included instead of the actual headers, in order to save compilation time (see dedicated section below). In ImpactX forward declaration headers have the suffix `*_fwd.H`, while in AMReX they have the suffix `*Fwd.H`. The include order (see PR #874 and PR #1947) and proper quotation marks are:

In a `MyName.cpp` file:

1. `#include "MyName.H"` (its header) then
2. (further) ImpactX header files `#include "..."` then
3. ImpactX forward declaration header files `#include "..._fwd.H"`
4. AMReX header files `#include <...>` then
5. AMReX forward declaration header files `#include <...Fwd.H>` then
6. PICSAR header files `#include <...>` then
7. other third party includes `#include <...>` then
8. standard library includes, e.g. `#include <vector>`

In a `MyName.H` file:

1. `#include "MyName_fwd.H"` (the corresponding forward declaration header, if it exists) then
2. ImpactX header files `#include "..."` then
3. ImpactX forward declaration header files `#include "..._fwd.H"`
4. AMReX header files `#include <...>` then
5. AMReX forward declaration header files `#include <...Fwd.H>` then
6. PICSAR header files `#include <...>` then
7. other third party includes `#include <...>` then
8. standard library includes, e.g. `#include <vector>`

Each of these groups of header files should ideally be sorted alphabetically, and a blank line should be placed between the groups.

For details why this is needed, please see PR #874, PR #1947, the LLVM guidelines, and include-what-you-use.

6.4.5 Forward Declaration Headers

Forward declarations can be used when a header file needs only to know that a given class exists, without any further detail (e.g., when only a pointer to an instance of that class is used). Forward declaration headers are a convenient way to organize forward declarations. If a forward declaration is needed for a given class `MyClass`, declared in `MyClass.H`, the forward declaration should appear in a header file named `MyClass_fwd.H`, placed in the same folder containing `MyClass.H`. As for regular header files, forward declaration headers must have include guards. Below we provide a simple example:

`MyClass_fwd.H`:

```
#ifndef MY_CLASS_FWD_H
#define MY_CLASS_FWD_H

class MyClass;

#endif //MY_CLASS_FWD_H
```

`MyClass.H`:

```
#ifndef MY_CLASS_H
#define MY_CLASS_H

#include "MyClass_fwd.H"
#include "someHeader.H"
class MyClass{/* stuff */};

#endif //MY_CLASS_H
```

`MyClass.cpp`:

```
#include "MyClass.H"
class MyClass{/* stuff */};
```

Usage: in `SimpleUsage.H`

```
#include "MyClass_fwd.H"
#include <memory>

/* stuff */
std::unique_ptr<MyClass> p_my_class;
/* stuff */
```

6.5 Implementation Details

Note: TODO :-)

6.6 C++ Objects & Functions

We generate the documentation of C++ objects and functions *from our C++ source code* by adding *Doxygen strings*.

Our Doxygen C++ API documentation in classic formatting is located here.

6.7 Python interface

Note: TODO :-)

6.8 Debugging the code

Sometimes, the code does not give you the result that you are expecting. This can be due to a variety of reasons, from misunderstandings or changes in the *input parameters*, system specific quirks, or bugs. You might also want to debug your code as you implement new features in ImpactX during development.

This section gives a step-by-step guidance on how to systematically check what might be going wrong.

6.8.1 Debugging Workflow

Try the following steps to debug a simulation:

1. Check the output text file, usually called `output.txt`: are there warnings or errors present?
2. On an HPC system, look for the job output and error files, usually called `ImpactX.e...` and `ImpactX.o...`. Read long messages from the top and follow potential guidance.
3. If your simulation already created output data files: Check if they look reasonable before the problem occurred; are the initial conditions of the simulation as you expected? Do you spot numerical artifacts or instabilities that could point to missing resolution or unexpected/incompatible numerical parameters?
4. Did the job output files indicate a crash? Check the `Backtrace.<mpirank>` files for the location of the code that triggered the crash. Backtraces are read from bottom (high-level) to top (most specific line that crashed).
5. In case of a crash, Backtraces can be more detailed if you *re-compile* with debug flags: for example, try compiling with `-DCMAKE_BUILD_TYPE=RelWithDebInfo` (some slowdown) or even `-DCMAKE_BUILD_TYPE=Debug` (this will make the simulation way slower) and rerun.
6. If debug builds are too costly, try instead compiling with `-DAMReX_ASSERTIONS=ON` to activate more checks and rerun.
7. If the problem looks like a memory violation, this could be from an invalid field or particle index access. Try compiling with `-DAMReX_BOUND_CHECK=ON` (this will make the simulation very slow), and rerun.
8. If the problem looks like a random memory might be used, try initializing memory with signaling Not-a-Number (NaN) values through the runtime option `fab.init_snan = 1`. Further useful runtime options are `amrex.fpe_trap_invalid`, `amrex.fpe_trap_zero` and `amrex.fpe_trap_overflow` (see details in the AMReX link below).
9. On Nvidia GPUs, if you suspect the problem might be a race condition due to a missing host / device synchronization, set the environment variable `export CUDA_LAUNCH_BLOCKING=1` and rerun.
10. Consider simplifying your input options and re-adding more options after having found a working baseline.

Fore more information, see also the AMReX Debugging Manual.

Last but not least: the community of ImpactX developers and users can help if you get stuck. Collect your above findings, describe where and what you are running and how you installed the code, describe the issue you are seeing with details and input files used and what you already tried. Can you reproduce the problem with a smaller setup (less parallelism and/or less resolution)? Report these details in a *ImpactX GitHub issue*.

6.8.2 Debuggers

See the AMReX debugger section on additional runtime parameters to

- disable backtraces
- rethrow exceptions
- avoid AMReX-level signal handling

You will need to set those runtime options to work directly with debuggers.

CHAPTER

SEVEN

MAINTENANCE

7.1 Dependencies & Releases

7.1.1 Update ImpactX' Core Dependencies

ImpactX has direct dependencies on AMReX and WarpX, which we periodically update.

The following scripts automate this workflow, in case one needs a newer commit of AMReX or WarpX between releases:

Note:

```
./Tools/Release/updateAMReX.py
./Tools/Release/updateWarpX.py
```

7.1.2 Create a new ImpactX release

ImpactX has one release per month. The version number is set at the beginning of the month and follows the format `YY.MM`.

In order to create a GitHub release, you need to:

1. Create a new branch from `development` and update the version number in all source files. We usually wait for the AMReX release to be tagged first, then we also point to its tag.

   There is a script for updating core dependencies of ImpactX and the ImpactX version:

   ```
   ./Tools/Release/updateAMReX.py
   ./Tools/Release/updateWarpX.py

   ./Tools/Release/newVersion.sh
   ```

   For a ImpactX release, ideally a *git tag* of AMReX & WarpX shall be used instead of an unnamed commit.

   Then open a PR, wait for tests to pass and then merge.

2. **Local Commit** (Optional): at the moment, `@ax3l` is managing releases and signs tags (naming: `YY.MM`) locally with his GPG key before uploading them to GitHub.

   Publish: On the GitHub Release page, create a new release via `Draft a new release`. Either select the locally created tag or create one online (naming: `YY.MM`) on the merged commit of the PR from step 1.

In the *release description*, please specify the compatible versions of dependencies (see previous releases), and provide info on the content of the release. In order to get a list of PRs merged since last release, you may run

```
git log <last-release-tag>.. --format='- %s'
```

3. Optional/future: create a `release-<version>` branch, write a changelog, and backport bug-fixes for a few days.

CHAPTER

EIGHT

EPILOGUE

8.1 Glossary

In daily communication, we tend to abbreviate a lot of terms. It is important to us to make it easy to interact with the ImpactX community and thus, this list shall help to clarify often used terms.

Please see: https://warpx.readthedocs.io/en/latest/glossary.html

8.2 Funding and Acknowledgements

This work was supported by the Laboratory Directed Research and Development Program of Lawrence Berkeley National Laboratory under U.S. Department of Energy Contract No. DE-AC02-05CH11231.

ImpactX is supported by the CAMPA collaboration, a project of the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of High Energy Physics, Scientific Discovery through Advanced Computing (SciDAC) program.

We acknowledge all the contributors and users of the ImpactX community who participate to the code quality with valuable code improvement and important feedback.

PYTHON MODULE INDEX

i

INDEX

L

M

N

P

Q

R

S

T

U

X

Y

Z