ISBN: 978-3-95450-232-5 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2022-TUYE2

NEXT GENERATION COMPUTATIONAL TOOLS FOR THE MODELING AND DESIGN OF PARTICLE ACCELERATORS AT EXASCALE[*]

A. Huebl[†], R. Lehe, C. E. Mitchell, J. Qiang, R. D. Ryne, R. T. Sandberg, J.-L. Vay
Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Abstract

Particle accelerators are among the largest, most complex devices. To meet the challenges of increasing energy, intensity, accuracy, compactness, complexity and efficiency, increasingly sophisticated computational tools are required for their design and optimization. It is key that contemporary software take advantage of the latest advances in computer hardware and scientific software engineering practices, delivering speed, reproducibility and feature composability for the aforementioned challenges. A new open source software stack is being developed at the heart of the Beam pLasma Accelerator Simulation Toolkit (BLAST) by LBNL and collaborators, providing new particle-in-cell modeling codes capable of exploiting the power of GPUs on Exascale supercomputers. Combined with advanced numerical techniques, such as mesh-refinement, and intrinsic support for machine learning, these codes are primed to provide ultrafast to ultraprecise modeling for future accelerator design and operations.

INTRODUCTION

Large-scale computer simulations of charged particle motion inside of particle accelerators play a crucial role in accelerator design and operation. In order to quickly simulate charged particle dynamics, including collective effects, advanced software must be developed to take advantage of state-of-the-art computer hardware.

With the onset of the Exascale supercomputing era, omnipresent GPU-accelerated machines require multi-level parallelism, multi-paradigm programming and dynamic load balancing. The U.S. Department of Energy Exascale Computing project addressed this need by co-developing applications and software for Exascale computing acquisitions. The laser-plasma modeling code `WarpX` [1], e.g., used in plasma-based particle acceleration, is a result of this project and recently provided the first full-scale runs at the first demonstrated Exascale machine, Frontier [2].

Beam, Plasma & Accelerator Simulation Toolkit

`WarpX` is a code in the Beam, Plasma & Accelerator Simulation Toolkit (`BLAST`, `https://blast.lbl.gov`), a suite of open source particle accelerator modeling codes. Originally developed under the name Berkeley Lab Accelerator Simulation Toolkit, included codes achieved compatibility through a common meta-data standard in I/O, `openPMD` [3], yet were implemented in disjoint code bases. `BLAST` has been renamed in 2021 to reflect international contributions from LIDYL (CEA, France), SLAC (USA), LLNL (USA), DESY (Germany), UHH (Germany), HZDR (Germany), Radiasoft (USA), CERN (Switzerland) and more; `BLAST` development involves deep collaboration among physicists, applied mathematicians, and computer scientists.

With the emergence of the first Exascale Computing supercomputers, modeling codes that were originally designed for parallel CPU-powered machines need to undergo a fundamental modernization effort. This became necessary, as compute nodes are now equipped with accelerator hardware such as GPUs (and potentially FPGAs in the future). Selected as application for the Department of Energy Exascale Computing Project, the `BLAST` code `WARP` [4,5] underwent a complete rewrite from Fortran to modern C++ resulting in its successor `WarpX` [1]. Building on the momentum of this transition to form a more cohesive Accelerator Toolkit, the specialized plasma wakefield acceleration code `HiPACE++` [6] and beam dynamics code `ImpactX` [7], presented herein, are developed.

Software Design

A central goal of the modernization of `BLAST` is modularity for efficient code reuse and tight integration for coupling, i.e., in hybrid particle accelerators with conventional and advanced (plasma) elements. Figure 1 shows the design of `BLAST`'s software dependencies, with upper components depending and sharing lower blocks in the schema. Shared code, common application programming interfaces (APIs) and data standards ensure composability and connection to the AI/ML and data science ecosystems. Performance-critical routines are implemented and reused in modern C++, using a single-source approach to program both CPUs and GPUs via a performance-portability layer in `AMReX` [8]. The newly introduced `ABLASTR` library collects common particle-in-cell (PIC) routines.

Python high-level interfaces are used for user efficiency and to provide standardized APIs to data science and AI/ML frameworks, which are mostly driven from the same language. Documentation and examples are developed in lock-step with documentation and published on `https:`

[*] Work supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. 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. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award HEP-ERCAP0020744.
[†] axelhuebl@lbl.gov

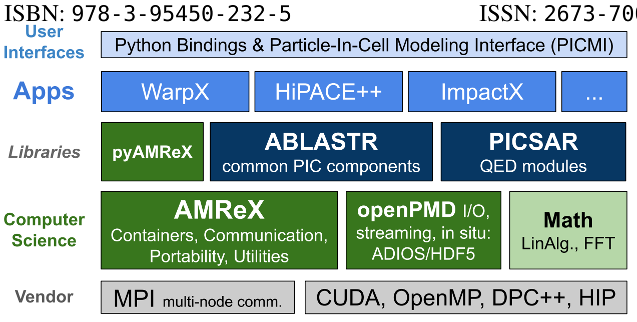

Figure 1: Design of the BLAST software stack. Modularization enables code sharing and tight coupling.

//impactx.readthedocs.io. Examples and test cases are continuously run against expected results.

All development is carried out in the open using open source licenses, contributable code repositories, code reviews, regular releases and change logs [7]. The community reports open "issues" for feature requests, bug reports, etc. Installation for users and developers is supported by package managers and HPC modules.

IMPACTX

ImpactX is an s-based beam dynamics code. Leveraging expertise and models in IMPACT-Z [9] and MaryLie [10, 11], this new simulation code is built from the ground up to take GPU-accelerated computing, mesh-refinement for space-charge effects, load balancing, and coupling AI/ML frameworks into consideration. ImpactX design relies on open community standards for I/O and data interfaces.

As of version 22.08, many features are still under active development. Generalized and reused from WarpX via the ABLASTR library are GPU-accelerated routines for charge deposition, beam statistics, Poisson solve, profiling, warning logging, Unix signal handling, build/installation logic, among others.

Model Assumptions

Tracking through lattice elements is performed by pushing particles in s using a symplectic map. Currently, each map applied during tracking is accurate through linear order (with respect to the reference orbit). The linearization implies that, when solving for 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.

The space charge fields are treated as electrostatic in the bunch rest frame. In particular, no retardation or radiation effects are included, and we solve the Poisson equation in the bunch rest frame. The effect of space charge is included in tracking using a map-based split-operator approach [9].

Usage Example

ImpactX can be executed in two ways: a traditional executable reading a textual input file or driven from Python. The latter approach is compact and expressive for simulation design, since Python's well-known syntax can be used to design complex beamlines with lines and segments. From

```python
from impactx import ImpactX, RefPart, \
                    distribution, elements

sim = ImpactX()  # simulation object

# set numerical parameters and IO control
sim.set_particle_shape(2)  # B-spline order
sim.set_slice_step_diagnostics(True)
sim.set_space_charge(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 = 2.0e3  # reference energy
charge_C = 1.0e-9  # used with space charge
mass_MeV = 0.510998950  # mass
qm_qeeV = -1.0e-6/mass_MeV  # charge/mass
npart = 10000  # number of macro particles

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(
    qm_qeeV, charge_C, distr, npart)

# set the energy in the reference particle
sim.particle_container().ref_particle() \
    .set_energy_MeV(energy_MeV, mass_MeV)

# design the accelerator lattice
ns = 25  # steps slicing through ds
fodo = [
    elements.Drift(ds=0.25, nslice=ns),
    elements.Quad(ds=1.0, k=1.0, nslice=ns),
    elements.Drift(ds=0.5, nslice=ns),
    elements.Quad(ds=1.0, k=-1.0, nslice=ns),
    elements.Drift(ds=0.25, nslice=ns)
]
# assign a fodo segment
sim.lattice.extend(fodo)

# run simulation
sim.evolve()
```

Listing 1: An example Python script FODO.py that can be used to design an FODO cell setup with ImpactX v22.08. This script can equally drive CPU and GPU simulations and executes over multiple nodes if started with MPI.

a performance viewpoint, both are equivalent since computational C++ kernels are compiled and just wrapped from Python. Listing 1 shows such a script-driven simulation.

Furthermore, such scripts can be extended since `ImpactX` exposes the underlying `AMReX` data structures to Python via `pyAMReX`, implementing AMReX data storage for zero-copy access via a standardized *Array Interface* [12]. For instance, beam particles and generated fields can be manipulated and analyzed in memory, combined with solvers from other `BLAST` or third party software packages and additional computational routines can be added, even with GPU support, using packages such as `cupy` or `numba`.

NUMERICAL EXPERIMENTS

Correctness tests are performed continuously, for every code change, on `ImpactX`. The goal of these tests is to cover all implemented functionality and verify that computed results are within expected precision, independently of the compute hardware used. Following such *test-driven development* eases the entry burden for accelerator scientists adding new functionality to the project, since automated testing will inform them if unexpected side-effects of changed code would change benchmarked physics results. Tests and examples also add a solid body of documented examples to the project.

As of version 22.08, the following benchmarks and examples are implemented: a FODO cell, a magnetic bunch compression chicane [13], a stationary beam in a constant focusing channel, a Kurth-distribution beam in a periodic isotropic focusing channel [14], a stable FODO cell with short RF (buncher) cavities added for longitudinal focusing [15], a chain of thin multipoles, a nonlinear focusing channel based on the IOTA nonlinear lens, and a model of the Fermilab IOTA storage ring (linear optics) [16]. Detailed numerical parameters are archived online [7, 17].

Benchmark: FODO Cell

In this benchmark, a stable FODO lattice with a zero-current phase advance of 67.8 degrees per cell is modeled. An rms-matched 2 GeV electron beam with initial unnormalized rms emittance of 2 nm propagates through a single cell, with the beam size evolution as shown in Fig. 2. In Fig. 3, the evolution of the transverse phase space is shown. The benchmark verifies for every code change that the beam second moments remain matched and that the emittances remain constant.

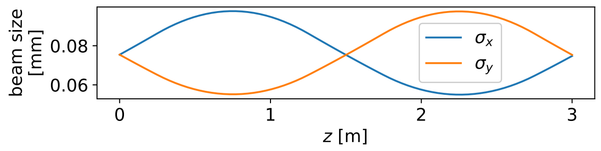

Figure 2: Evolution of the horizontal and vertical rms beam sizes in the FODO benchmark.

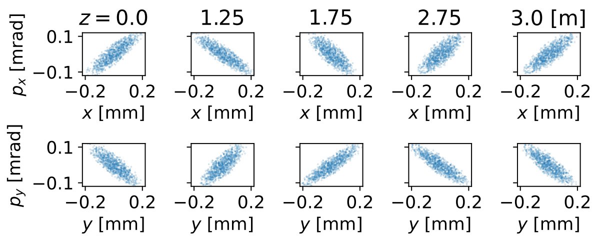

Figure 3: Transverse phase space projections in the FODO cell benchmark.

Benchmark: Berlin-Zeuthen Chicane

This benchmark is a simple, unshielded four-bend chicane with parameters similar to the ones required for the compression stages at LCLS (at 5 GeV) or TESLA XFEL (at 500 MeV) [13]. Here, a 5 GeV electron bunch with normalized transverse rms emittance of 1 µm is used. Figure 4 shows the longitudinal beam size and transverse emittance evolution. The former is compressed 10×, while the initial emittance is recovered at the end of transport. The longitudinal phase space evolution during the transport is shown in Fig. 5.

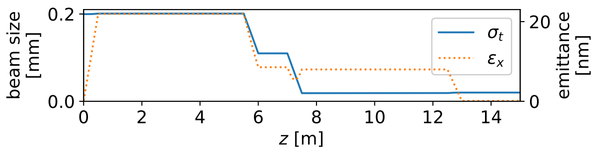

Figure 4: Evolution of longitudinal beam size (rms) and transverse emittance in the chicane benchmark.

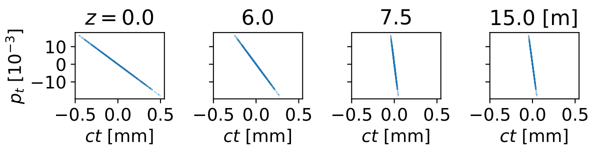

Figure 5: Evolution of the longitudinal phase space in the chicane benchmark, illustrating 10× compression.

Benchmark: Fermilab IOTA Storage Ring

This benchmark is a model of the bare (linear) lattice of the Fermilab IOTA storage ring (v. 8.4) [16], with optics configured for operation with a 2.5 MeV proton beam. An rms-matched proton beam with an unnormalized emittance of 4.5 µm propagates over a single turn. The second moments of the particle distribution after a single turn are checked to coincide with the initial second moments of the particle distribution, to within the level expected due to numerical particle noise.

In Fig. 6, the reference orbit indicating the global beam position within the ring is shown. Figure 7 shows the rms beam size evolution as a function of path length over a single turn. The thin dark lines are from `ImpactX`, while the light bold lines in the background are from `IMPACT-Z`. The results of the two codes are in excellent agreement.

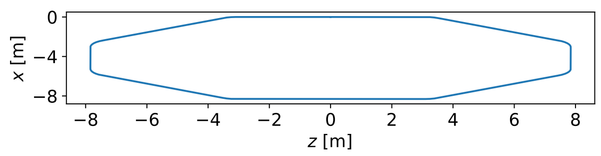

Figure 6: The reference orbit in the IOTA lattice benchmark, as produced by `ImpactX`, showing the storage ring floorplan.

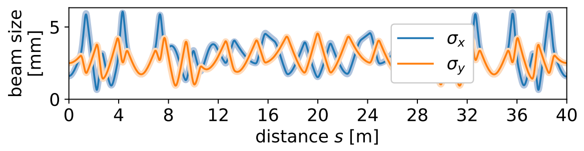

Figure 7: Evolution of the rms beam size in the IOTA lattice benchmark. Thin dark lines: `ImpactX`, light bold lines: `IMPACT-Z` results.

Performance

Although `ImpactX` in version 22.08 is in an early development state, initial performance comparisons can be made between conventional (CPU) and accelerated compute hardware (GPU). In the following, the above IOTA lattice benchmark is used as a computing benchmark (without I/O or space charge solvers) to measure the performance of `ImpactX` on a node of the NERSC Perlmutter Phase 2 supercomputer. Prototypical for future large-scale simulations, but not strictly needed for this benchmark to converge, 10^8 beam macro particles are used.

Figure 8 shows the `ImpactX` strong-scaling speedup in time-to-solution relative to a 16 CPU core run (146 sec). Compilation uses `-O3` and fast-math enabled; compilers are GNU 11.2.0 and NVCC 11.7.64, respectively. Values above 1.0 are faster, below are slower. Dynamic load-balancing is not yet used.

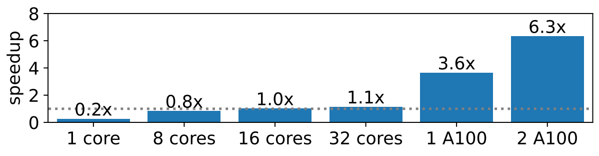

Figure 8: Relative performance between CPU and GPU runs on Perlmutter Phase 2 (NERSC). 16 CPU cores (w/o hyperthreading) and one A100 GPU are each 1/4th of a node's resources: one AMD EPYC 7763 (Milan) processor and four NVIDIA Ampere A100 SXM2 (40 GB) GPUs.

Data Availability

Simulation code and documentation are openly developed in Ref. [7]. Numerical experiments and performance results are archived in Ref. [17].

CONCLUSION

This paper presents computational tools for the modeling and design of particle accelerators, readying codes up for next generation machines in the Exascale era. The open source software toolkit `BLAST` provides modeling tools to model hybrid accelerators, containing both plasma and conventional beamline elements. `ABLASTR` is a modern C++17 library used to share particle-in-cell routines between simulation codes. Based on this, `ImpactX` is developed to succeed `IMPACT-Z` as a new, *s*-based beam dynamics code with intrinsic GPU, mesh-refinement and tight coupling to time-based codes and AI/ML capabilities.

`ImpactX` is in an early state, but already able to model significantly larger particle ensembles than its predecessor codes. Ongoing and future developments will add capabilities for: space charge effects, scalable I/O, non-linear elements, RF-modeling, wakefield effects, surface methods and support to read accelerator lattices from MAD-X files, among others.

ACKNOWLEDGEMENTS

We acknowledge the community of contributors to the open source projects `ABLASTR`, `AMReX`, `ImpactX`, `IMPACT-Z`, `openPMD`, `pybind11` and `WarpX`.

REFERENCES

[1] A. Myers *et al*, "Porting WarpX to GPU-accelerated platforms", *J. Parallel Comput.*, vol. 108, p. 102833, 2021. `doi:10.1016/j.parco.2021.102833`

[2] L. Fedeli, A. Huebl, *et al.*, "Pushing the frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined Particle-In-Cell simulations on Exascale-class supercomputers", *SC'22: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, to be published.

[3] A. Huebl *et al*, "openPMD 1.0.0: A meta data standard for particle and mesh based data", 2015. `doi:10.5281/zenodo.33624`

[4] J.-L. Vay, D. P. Grote, R. H. Cohen, and A. Friedman, "Novel methods in the Particle-In-Cell accelerator Code-Framework Warp", *Comput. Sci. & Disc.*, vol. 5, p. 014019, 2012. `doi:10.1088/1749-4699/5/1/014019`

[5] A. Friedman, R. H. Cohen, D. P. Grote, S. M. Lund, W. M. Sharp, J.-L. Vay, I. Haber, Irving and R. A. Kishek, "Computational Methods in the Warp Code Framework for Kinetic Simulations of Particle Beams and Plasmas", *IEEE Trans. Plasma Sci.*, vol. 42, pp. 1321-1334, 2014. `doi:10.1109/TPS.2014.2308546`

[6] S. Diederichs *et al*, "HiPACE++: A portable, 3D quasi-static particle-in-cell code", *Computer Physics Communications*, vol. 278, p. 108421, 2022. `doi:10.1016/j.cpc.2022.108421`

[7] A. Huebl, C. E. Mitchell, R. Lehe, J. Qiang, *et al*, "ECP-WarpX/impactx: 22.08", 2022. doi:10.5281/zenodo.6954923

[8] W. Zhang *et al*, "AMReX: a framework for block-structured adaptive mesh refinement", *Journal of Open Source Software*, vol. 4, no. 37, p. 1370, 2019. doi:10.21105/joss.01370

[9] J. Qiang, R. Ryne, S. Habib, and V. Decyk, "An Object-Oriented Parallel Particle-in-Cell Code for Beam Dynamics Simulation in Linear Accelerators", *J. Comput. Phys.*, vol. 163, pp. 434-451, 2000. doi:10.1006/jcph.2000.6570

[10] A. Dragt *et al*, "MARYLIE 3.0 User's Manual", Department of Physics and Astronomy, University of Maryland, College Park, MD; https://www.physics.umd.edu/dsat/

[11] R. D. Ryne *et al*, "Recent Progress on the MaryLie/IMPACT Beam Dynamics Code", in *Proc. ICAP'06*, Chamonix, France, 2006, paper TUAPMP03, pp. 157–159. https://accelconf.web.cern.ch/icap06/papers/tuapmp03.pdf

[12] Consortium for Python Data API Standards, array interface, 2021, https://data-apis.org/array-api

[13] Berlin-Zeuthen benchmark chicane, https://www.desy.de/csr/

[14] C.E. Mitchell, K. Hwang, and R.D. Ryne, "Kurth Vlasov-Poisson Solution for a Beam in the Presence of Time-Dependent Isotropic Focusing", in *Proc. IPAC'21*, Campinas, SP, Brazil, May 2021, pp. 3213–3216. doi:10.18429/JACoW-IPAC2021-WEPAB248

[15] R. D. Ryne *et al*, "A Test Suite of Space-charge Problems for Code Benchmarking", in *Proc. EPAC'04*, Lucerne, Switzerland, 2004, paper WEPLT047, pp. 1942-1945. https://accelconf.web.cern.ch/e04/papers/weplt047.pdf

[16] S. Antipov *et al*, "IOTA (Integrable Optics Test Accelerator): facility and experimental beam physics program", *JINST*, vol. 12, p. T03002, 2017. doi:10.1088/1748-0221/12/03/T03002

[17] A. Huebl, C. E. Mitchell, *et al*, "Supplementary Materials: Next Generation Computational Tools for the Modeling and Design of Particle Accelerators at Exascale", NAPAC'22 data artifact for paper TUYE2, 2022. doi:10.5281/zenodo.6954898