ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP101

TOWARDS DIFFERENTIABLE BEAM DYNAMICS MODELING IN BLAST/IMPACTX*

A. Huebl†, C.E. Mitchell, R. Lehe, A. Formenti, G. Charleux, A. Myers, W. Zhang, J. Qiang, J.-L. Vay, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
J. Kaiser, C. Hespe, Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany
J.P. Gonzalez-Aguilera, University of Chicago, Chicago, IL, USA
C. Xu, Argonne National Laboratory, Lemont, IL, USA
A. Santamaria Garcia, University of Liverpool, Liverpool, United Kingdom
R. Roussel, A. Edelen, SLAC National Accelerator Laboratory, Menlo Park, CA, USA
W.S. Moses, University of Illinois Urbana-Champaign, Champaign, IL, USA

Abstract

Differentiable beam dynamics simulations are in demand, e.g., to solve inverse and optimization problems in accelerator and beam physics. Here, the two modern open source codes Cheetah and ImpactX are compared in their approach to performance-portable programming and automatic differentiation. A first auto-differentiable prototype of ImpactX explores how the C++ codes in the Beam, Plasma & Accelerator Simulation Toolkit (BLAST) could introduce differentiable modeling.

INTRODUCTION

Differentiable simulations are in demand in accelerator physics, demonstrating order-of-magnitude improvements for complex tasks such as many-parameter optimization for beamline calibrations [1, 2], reconstruction of hard-to-measure quantities [3–6], or uncertainty quantification.

Calculating gradients in simulations with respect to inputs requires more calculations and memory than commonly used, gradient-free simulations. At present, it is an open question which programming approach might be most effective and performant to implement the many advanced numerical methods of a (differentiable) beam dynamics simulation. Desirable is an approach where the physics models only need to be implemented once ("single-source") and automation ensures gradient-calculations (auto-differentiation) and performance on various computing platforms (CPU/GPU).

This work makes use of - and enhances - two fast, modern codes: ImpactX (25.08) [7] and Cheetah (0.7.5) [1, 2]. Both codes are actively developed in open source, well documented and support CPUs and GPUs. Historically, Cheetah implemented only linear tracking models [1], but recently added nonlinear paraxial methods from merging with the BMAD-X [2] code. ImpactX implements linear, nonlinear paraxial and exact Hamiltonian element models. Both codes implement the same 3D space charge model [8].

ImpactX approaches accelerator modeling from a need for symplectic modeling with collective effects, supporting high-order, multi-node scalable, long-term stable, simulations. It is written in C++ with Python bindings and reuses modular exascale methods from the WarpX [7, 9] code. Cheetah follows a differentiability-first approach, implementing exclusively on the PyTorch (2.7.1) library in its auto-differentiable array ("tensor") programming model [10]. Cheetah is limited to a single compute node (shared-memory system).

This paper is organized in two parts. First, we investigate the latest performance in an anecdotal example, where both codes have functional overlap – gradient-free modeling of chromatic element models with up to few-million particles per bunch and 3D space charge effects. Second, we show progress in introducing single-source differentiability in ImpactX using a modern compiler plugin, producing executables for gradient-based and gradient-free modeling without a rewrite away from performant C++.

GRADIENT-FREE BENCHMARKS

Achieving high performance for nonlinear beam dynamics simulations is a moving target, because programming languages, compilation techniques and hardware evolve constantly. The two codes in this paper use a modern approach of portable, "single-source" programming, where a generalized implementation of computational routines aims to perform well on CPUs and at least three different vendors of GPUs. While both codes approach accelerator modeling from different needs and implementations, comparing a snapshot of performance can be an instructive exercise to inform future implementations.

This section compares and tunes performance of commonly used, gradient-free nonlinear paraxial particle tracking and 3D space charge on a single node. Not studied here are features exclusively in ImpactX, e.g., exact Hamiltonian models or distributed-memory scalability, or only in Cheetah, e.g., concurrent runs of multiple small, independent simulations. The performance baseline in this paper is the current state-of-the-art performance, see comparisons in Refs. [1, 2]. Benchmarks use 1 node of the NERSC Perlmutter machine with and AMD EPYC 7763 CPU and Nvidia A100 80 GB GPU. The best-of-5 measurements is shown.

* Supported by the CAMPA collaboration, a project of the U.S. DOE, Office of Science SciDAC program. Supported by the LDRD Program of LBNL under U.S. DOE Contract No. DE-AC02-05CH11231. This research used resources of NERSC, a U.S. DOE Office of Science User Facility located at LBNL, using NERSC award HEP-ERCAP0031191.
† axelhuebl@lbl.gov

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP101

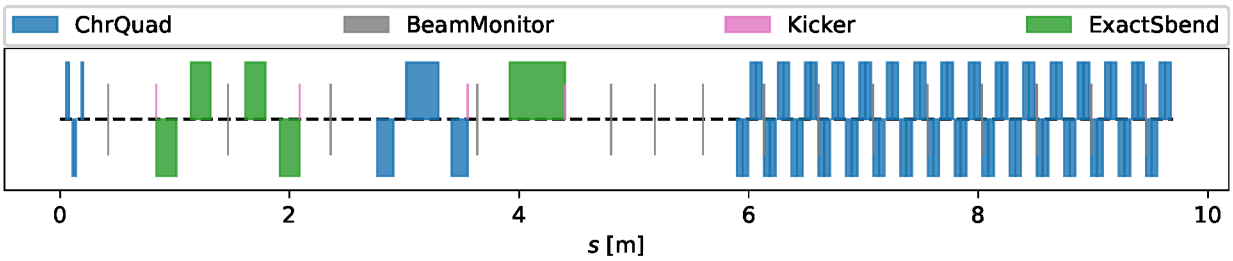

Figure 1: Survey plot of the BELLA Hundred-Terawatt Undulator (HTU) beamline at LBNL.

HTU Beamline

As a first benchmark, particle tracking for the BELLA Hundred-Terawatt Undulator (HTU) beamline at LBNL is studied [11]. The HTU beamline (Fig. 1) is a compact free electron laser (FEL), using a short transport line and undulator, fed by a laser-wakefield electron source of 100 MeV electrons. The laser-wakefield source has a moderate energy spread of approximately 2.5% and a relatively low 2 mrad transverse divergence, which requires adequate modeling of chromatic effects using the paraxial approximation.

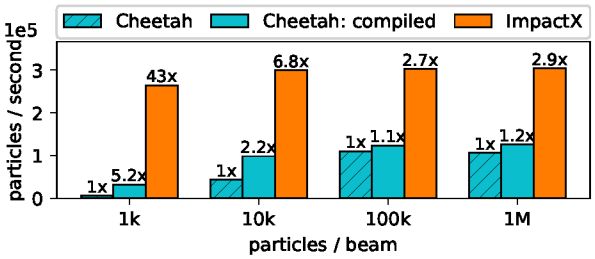

Figure 2: Single precision CPU performance (1 core) grouped into modeled number of particles. Labels above the bars are the relative speedup per group to Cheetah performance before this work.

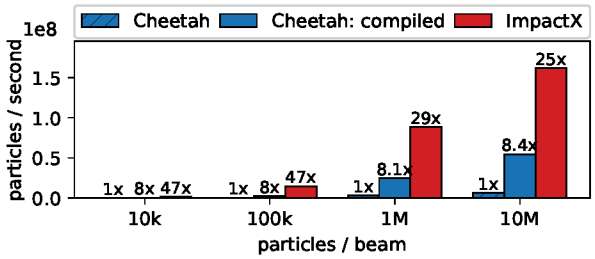

Figure 3: Single precision GPU performance grouped into modeled number of particles. Labels above the bars are the relative speedup per group to Cheetah performance before this work.

The dominating cost of this benchmark is the modeling of chromatic quadrupole magnets in the beamline. For the HTU beamline, single precision modeling was found to be sufficiently accurate and both codes were benchmarked to agree in their predictions.

In Fig. 2 (CPU) and 3 (GPU) and following, performance numbers are presented as following: on the x-axis, groups of code benchmarks are performed for varying number of particles in the beam. On the y-axis is the performance, a higher number is better, showing the number of particles simulated divided by the computing time. Numbers over the bars show the improvement over the baseline, the best Cheetah performance before this work.

The three benchmarks per scenario group are: baseline `Cheetah` (in Python), `Cheetah` using `torch.compile` [12], and `ImpactX`. ImpactX runs a *pre-compiled* executable, relying on the C++ performance-portability layer in AMReX and C++ `std::simd` for efficient execution [13]. Cheetah is a Python code, and as such interpreted at runtime. Compiling Cheetah means, that at runtime the central tracking loop is not just-in-time interpreted as in regular Python, but *transpiled* to vectorized C++ code (for CPU), then *compiled* to machine-code and executed [12]. The time spent in transpilation/compilation is excluded from measurements for both codes.

Compiling nonlinear tracking in Cheetah provides a 1.1-5.2x speedup for CPU and 8x speedup on GPU, by vectorizing CPU code, eliminating intermediate variables and fusing GPU kernels. C++ code from ImpactX runs another 2.4-8.3x faster on CPU and 3-6.7x faster on GPU. Modeling only a small number of particles could benefit from (L2/L3) caching of the beam particles on CPU (not observed here), and will lead to under-utilization of a GPU, explaining the rise of particles/second for more particles/beam in Fig. 3. Cheetah performance struggles with small particle numbers, but provides the aforementioned concurrent-simulation mode, effectively tapping into the higher performance measured for more particles per beam.

3D Space Charge

Our teams implemented the same 3D space charge algorithm in Cheetah and ImpactX, based on an Integrated Green's Function (IGF) method (open boundaries) [8]. The ImpactX implementation supports again multiple compute nodes, while the Cheetah implementation is single-node and thus 1 node is used for comparison in both codes.

This test is performed in double precision for accurate results and uses 1M particles / beam, a 3D grid of 256^3 cells for space charge, and cloud-in-cell particle shape for deposi-

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP101

tion and force kick. The dominating cost of this benchmark is the calculation of the Green's function.

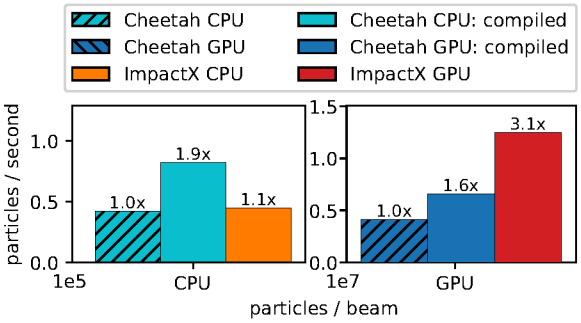

Figure 4: Double precision CPU performance (1 core) and double precision GPU performance. Labels above the bars in each group are with respect to the uncompiled Cheetah performance before this work.

The performance comparison in Fig. 4, while on first look comparable up to a factor 2x in performance, revealed potential for improvements in both codes. For ImpactX on CPU, the Green's function calculation in AMReX is not explicitly vectorized, while the PyTorch computations are. In Cheetah on GPU, even in the compiled version, the memory footprint does not fit 100M particles on a 80 GB GPU, while the ImpactX implementation fits. In both codes, the Green's function calculation could in some cases be cached: in segments where the energy of the reference particle does not change and the beam envelope evolves slowly, the space charge grid resolution could be fixed for N steps, revealing a potential for speedups in future versions.

DIFFERENTIABLE MODELING IN IMPACTX

In order to introduce gradient-based modeling in large, existing C++ code bases like the Beam, Plasma & Accelerator Simulation Toolkit (BLAST) [7, 9], support for the compiler plugin Enzyme [14, 15] was explored in ImpactX. By slightly restructuring the existing ImpactX code base, we developed a first prototype that supports both forward-mode and reverse-mode differentiation for envelope-based modeling, including space charge effects.

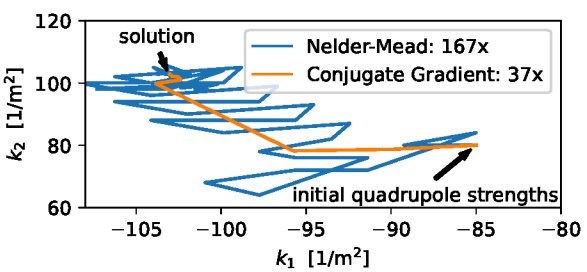

Figure 5: Gradient-free (Nelder-Mead) and gradient-based (conjugate gradient) optimization of quadrupole strengths and necessary number of simulations to perform.

Comparing the existing gradient-free and new gradient-based ImpactX simulation, detuned quadrupole strengths k_1, k_2 of a FODO cell were numerically optimized to match the transverse Twiss parameters of a 0.5 A proton beam with a kinetic energy of 6.7 MeV energy from FODO entrance to exit [16, 17]. Figure 5 shows the required number of simulations to convergence to the analytical solution $k_{1,2} = [-103.12, 103.12]/\mathrm{m}^2$ starting from initial values $k_{1,2} = [-85, 80]/\mathrm{m}^2$ using optimizers in `scipy.minimize`. In this simple example, gradient-aided optimization using the auto-differentiated simulation gradients reduces the required number of simulations from 167 to 37, while individual gradient-based simulation CPU runtime is higher by approximately the same ratio, for both forward and reverse-mode differentiation.

CONCLUSION

The community is pursuing implementations of beam dynamics codes capable of CPU/GPU-acceleration, differentiability, and high performance. This work studied the latest snapshot of the two actively developed codes ImpactX (25.08) and Cheetah (0.7.5), both showing high performance on modern hardware for a set of selected, nonlinear paraxial methods. Compiling Cheetah's PyTorch (2.7.1) code now provides up to 8.4x speedup over traditional Python execution, approaching performance portable ImpactX C++ code up to a factor 2-8x.

Leveraging the high performance of ImpactX, a first prototype introducing differentiability in C++ was presented, based on the Enzyme compiler plugin. While currently based on an envelope tracking model, it shows a potential avenue to introduce performant differentiability in large existing C++ code bases, such as the Beam, Plasma & Accelerator Simulation Toolkit (BLAST). Crucially for ML coupling tasks requiring backpropagation, backward-differentiation is supported, improving over approaches in Refs. [15, 18]

Future studies will need to investigate performance (runtime and memory usage) of the studied approaches with respect to: a) AD code generation, optimization and compilation, and b) more complex methods, like exact Hamiltonian integrators, collisional methods, or beam-wall interaction.

ACKNOWLEDGEMENTS

We acknowledge the community of contributors to the open source projects `ABLASTR`, `AMReX`, `Cheetah`, `Enzyme`, `ImpactX`, `pybind11`, `PyTorch`, `scipy`, and `WarpX`. All presented source code is open source, actively developed, and together with presented benchmarks and data available or linked in Ref. [19].

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP101

REFERENCES

[1] J. Kaiser *et al.*, "Bridging the gap between machine learning and particle accelerator physics with high-speed, differentiable simulations", *Phys. Rev. Accel. Beams*, vol. 27, p. 054601, 2024. doi:10.1103/PhysRevAccelBeams.27.054601

[2] J.-P. Gonzalez-Aguilera *et al.*, "Towards fully differentiable accelerator modeling", in *Proc. IPAC2023*, Venice, Italy, paper WEPA065, pp. 2797-–2800, 2023. doi:10.18429/JACoW-IPAC2023-WEPA065

[3] R. Roussel *et al.*, "Phase space reconstruction from accelerator beam measurements using neural networks and differentiable simulations", *PRL*, vol. 130, p. 145001, 2023. doi:10.1103/PhysRevLett.130.145001

[4] A. Hoover *et al.*, "High-dimensional maximum-entropy phase space tomography using normalizing flows", *Phys. Rev. Research*, vol. 6, p. 033163, 2023. doi:10.1103/PhysRevResearch.6.033163

[5] R. Roussel *et al.*, "Efficient six-dimensional phase space reconstructions from experimental measurements using generative machine learning", *Phys. Rev. Accel. Beams*, vol. 27, p. 094601, 2024. doi:10.1103/PhysRevAccelBeams.27.094601

[6] S. Kim *et al.*, "Four-dimensional phase-space reconstruction of flat and magnetized beams using neural networks and differentiable simulations", *Phys. Rev. Accel. Beams*, vol. 27, no. 7, p. 074601, Jul. 2024. doi:10.1103/physrevaccelbeams.27.074601

[7] A. Huebl *et al.*, "Next generation computational tools for the modeling and design of particle accelerators at exascale", in *Proc. NAPAC'22*, Albuquerque, NM, USA, Aug. 2022, pp. 302–306. doi:10.18429/JACoW-NAPAC2022-TUYE2

[8] J. Qiang *et al.*, "Implementation of the integrated Green's function method for 3D Poisson's equation in a large aspect ratio computational Domain", *J. Software Eng. Appl.*, vol. 17, pp. 740-749, 2024. doi:10.4236/jsea.2024.179039

[9] L. Fedeli *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", in *SC22: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2022, pp. 1–12. doi:10.1109/sc41404.2022.00008

[10] A Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library", in *Proc. NeurIPS2019*, Vancouver, Canada, 2019. doi:10.48550/arXiv.1912.01703

[11] S.K. Barber *et al.*, "Greater than 1000-fold gain in a free-electron laser driven by a laser-plasma accelerator with high reliability", *Phys. Rev. Lett.*, vol. 135, no. 5, p. 055001, 2025. doi:10.1103/vh62-gz1p

[12] J. Gong *et al.*, *TorchInductor Update 9: Harden Vectorization Support and Enhance Loop Optimizations in TorchInductor CPP Backend*, 2024. https://docs.pytorch.org/tutorials/intermediate/torch_compile_tutorial.html

[13] A. Myers *et al.*, "AMReX and pyAMReX: Looking beyond the exascale computing project", *Int. J. High Perform. Comput. Appl.*, vol. 38, no. 6, pp. 599–611, Aug. 2024. doi:10.1177/10943420241271017

[14] W. S. Moses *et al.*, "Scalable automatic differentiation of multiple parallel paradigms through compiler augmentation", in *SC22: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2022, pp. 1–18. doi:10.1109/sc41404.2022.00065

[15] J. Wan, H. Alamprese, C. Ratcliff, J. Qiang, and Y. Hao, "JuTrack: A Julia package for auto-differentiable accelerator modeling and particle tracking", *Comput. Phys. Commun.*, vol. 309, p. 109497, Apr. 2025. doi:10.1016/j.cpc.2024.109497

[16] C. Mitchell and A. Huebl, *ImpactX example: FODO Cell with 2D Space Charge using Envelope Tracking*, 2025. https://impactx.readthedocs.io/en/latest/usage/examples/fodo_space_charge/README.html

[17] A. Adelmann, J. Amundson, J. Qiang, R. D. Ryne, and P. Spentzouris, "A Test Suite of Space-charge Problems for Code Benchmarking", in *Proc. EPAC'04*, Lucerne, Switzerland, Jul. 2004, paper WEPLT047, pp. 942–1944.

[18] J. Qiang, "Differentiable self-consistent space-charge simulation for accelerator design", *Phys. Rev. Accel. Beams*, vol. 26, no. 2, p. 024601, Feb. 2023. doi:10.1103/physrevaccelbeams.26.024601

[19] A. Huebl *et al.*, *Supplementary Materials: Towards Differentiable Beam Dynamics Modeling in BLAST/ImpactX*, 2025. doi:10.5281/zenodo.16783369