Computer Physics Communications 278 (2022) 108421

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

HiPACE++: A portable, 3D quasi-static particle-in-cell code ☆,☆☆

S. Diederichs [a,b,c,*], C. Benedetti [b], A. Huebl [b], R. Lehe [b], A. Myers [b], A. Sinn [a], J.-L. Vay [b], W. Zhang [b], M. Thévenet [a]

[a] Deutsches Elektronen-Synchrotron DESY, Notkestr. 85, 22607 Hamburg, Germany
[b] Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA
[c] University of Hamburg, Institute of Experimental Physics, Luruper Chaussee 149, 22607 Hamburg, Germany

ARTICLE INFO

Article history:
Received 21 September 2021
Received in revised form 29 April 2022
Accepted 17 May 2022
Available online 24 May 2022

Keywords:
Particle-in-cell
Plasma acceleration
GPU computing
Quasi-static approximation

ABSTRACT

Modeling plasma accelerators is a computationally challenging task and the quasi-static particle-in-cell algorithm is a method of choice in a wide range of situations. In this work, we present the first performance-portable, quasi-static, three-dimensional particle-in-cell code HiPACE++. By decomposing all the computation of a 3D domain in successive 2D transverse operations and choosing appropriate memory management, HiPACE++ demonstrates orders-of-magnitude speedups on modern scientific GPUs over CPU-only implementations. The 2D transverse operations are performed on a single GPU, avoiding time-consuming communications. The longitudinal parallelization is done through temporal domain decomposition, enabling near-optimal strong scaling from 1 to 512 GPUs. HiPACE++ is a modular, open-source code enabling efficient modeling of plasma accelerators from laptops to state-of-the-art supercomputers.

Program summary
Program Title: HiPACE++
CPC Library link to program files: https://doi.org/10.17632/zh3rc7hvrm.1
Developer's repository link: HiPACE++ GitHub repository
Licensing provisions: BSD 3-clause
Programming language: C++
Nature of problem: Modeling plasma accelerators is a computationally challenging task requiring nanometer-scale resolutions over meter-scale propagation distances. The quasi-static particle-in-cell method enables high-fidelity simulations of this strongly non-linear process, but these simulations can be very expensive.
Solution method: The quasi-static particle-in-cell algorithm is modified to enable efficient utilization of accelerated hardware, in particular with GPU computing, reducing the cost of simulations by orders of magnitude. A novel longitudinal parallelization enables excellent strong scaling of this method up to hundreds of GPUs.
Reference: 10.5281/zenodo.5358483

1. Introduction

Plasma accelerators [1,2] enable the acceleration of charged particles over short distances due to their multi-GeV/m field gradients. Although great progress in terms of beam quality and stability has recently been achieved [3–6], significant advance is still required to make plasma-accelerator-driven applications feasible. The Particle-in-Cell (PIC) method [7,8] is a reliable tool to simulate plasma acceleration, and PIC simulations play a major role in understanding, exploring and improving plasma accelerators [9–11].

☆ The review of this paper was arranged by Prof. David W. Walker.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author at: Deutsches Elektronen-Synchrotron DESY, Notkestr. 85, 22607 Hamburg, Germany.
E-mail addresses: severin.diederichs@desy.de (S. Diederichs), maxence.thevenet@desy.de (M. Thévenet).

https://doi.org/10.1016/j.cpc.2022.108421

Simulation of a multi-GeV plasma-based accelerator typically requires modeling sub-micron-scale structures propagating over meter-scale distances, hence full electromagnetic PIC simulations require millions of time steps due to the Courant-Friedrichs-Lewy (CFL) condition [12], which makes them unpractical. Several methods were developed to circumvent this limitation and enable larger time steps, including running PIC in a Lorentz-boosted frame [13] or using a quasi-static approximation [14–17], both of which have proved performant for modeling of high-energy plasma accelerator stages [18–26].

Besides algorithmic improvements, further speedup can be accomplished from hardware improvement. Accelerated computing is growing in popularity in the supercomputer landscape [27], and in particular using GPUs (Graphics Processing Units) as accelerators enabled significant speedup in High-Performance Computing (HPC) applications including PIC [28–31].

The heterogeneity of processor architectures in HPC makes it difficult to maintain a portable codebase but, following modern HPC practices, this challenge can be efficiently addressed with performance-portability layers [32–34].

In this article, we present the portable, three-dimensional, open-source, quasi-static PIC code HiPACE++[1] [35]. HiPACE++ is written in C++ and is built on top of the AMReX [36] framework, which provides field data structure, Message Passing Interface (MPI) communications, and a performance-portability layer. In particular, the quasi-static PIC algorithm is adapted to accelerated computing, and HiPACE++ demonstrates orders-of-magnitude speedup over CPU implementations as well as near-optimal scaling up to hundreds of cutting-edge GPUs. These performances enable realistic simulations of $1024 \times 1024 \times 1024$ cells for 1000 time steps in less than two minutes on modern GPU-accelerated supercomputers. HiPACE++ is a new software, combining the algorithm from the legacy C code HiPACE [22] with a portability layer and specific modifications enabling GPU computing.

The article is organized as follows: Section 2 summarizes the well-known quasi-static PIC algorithm. The GPU-porting strategy is introduced in Sec. 3. Correctness of the code is demonstrated in Sec. 4. Sec. 5 presents performance results and a novel parallelization strategy improving scalability on accelerated platforms. Additional code features are highlighted in Sec. 6.

2. The quasi-static particle-in-cell algorithm

In a plasma accelerator, a driver perturbs the plasma electrons (the ions, heavier, can generally be considered immobile) and drives an electron plasma wave. While the driver can be a laser pulse or a particle beam, we hereafter focus on the case of a particle beam (beam-driven wakefield acceleration) for simplicity, as this is what is currently implemented in HiPACE++. In the wake of the driver, a witness beam of charged particles can be accelerated with a high field gradient. In most conditions (with the notable exception of witness beam self-injection), the driver and witness beams evolve on a time scale much longer than the plasma response [37]. The quasi-static approximation (QSA) treats the beams as rigid when computing the plasma response at a given beam location, hence decoupling the beam and plasma evolutions. Under this approximation, the Maxwell equations take the form of Poisson equations, and the scheme is not subject to a CFL condition. Then, the time step is determined by the smallest betatron period of the beams (the betatron period is the characteristic time scale over that a beam evolves), making it possible to use time steps orders of magnitude larger than those in conventional electromagnetic PIC [22]. The algorithm has two main parts: first, from the distributions of the beams, compute the plasma response (computationally expensive). Second, from the plasma fields, advance the beams by one time step (computationally cheap).

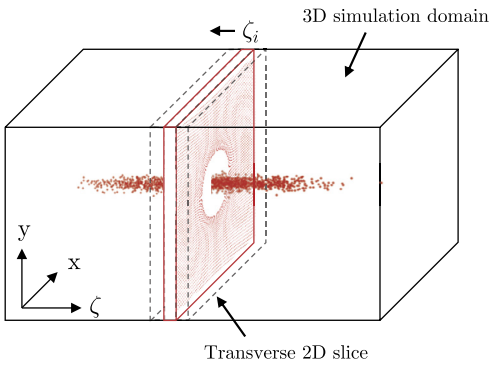

Fig. 1. Snapshot of the quasi-static PIC algorithm. The 3D simulation domain is calculated slice-by-slice in a loop over the longitudinal grid points from the head of the box to its tail. Only the beam particles, a 2D slice of plasma particles, and a few 2D slices of fields are required to determine the wake in the 3D simulation domain.

For a given distribution of the beams, the plasma response is computed in the co-moving frame defined by $\zeta = z - ct$, with c being the speed of light in vacuum (the beams propagate in the $+z$ direction). A slice of unperturbed plasma is initialized ahead of the beams and pushed backwards along the ζ coordinate. At each longitudinal position, the wakefields are calculated as a 2D problem in the transverse plane. The 3D problem is then solved as n_ζ 2D transverse problems (called *slices*), with n_ζ being the number of longitudinal grid points in the simulation domain. Fig. 1 illustrates the algorithm. In the standard algorithm, the fields are calculated in the whole 3D domain before the beams are advanced by one time step. In this work, we propose to integrate the beam advance in the loop over slices, hence pushing beam particles slice by slice. With this change, all parts of the simulation are done per slice, which is a crucial condition for our performance-portability strategy, in particular on GPUs.

From Maxwell's equations and the QSA, the following field equations can be derived [21]. The wake potential $\psi = \phi - c\,A_z$, with ϕ and $\boldsymbol{A}$ being the scalar and vector potential respectively, is obtained from

$$\nabla_\perp^2 \psi = -\frac{1}{\epsilon_0}\left(\rho - \frac{1}{c} J_z\right), \tag{1}$$

where ϵ_0 is the vacuum permittivity and ρ and $\boldsymbol{J}$ are the total (beams + plasma) charge and current densities, respectively. The transverse wakefields $E_x - c\,B_y$ and $E_y + c\,B_x$ are calculated from the transverse derivatives of ψ:

$$E_x - c\,B_y = -\frac{\partial}{\partial x}\psi\,, \tag{2a}$$

$$E_y + c\,B_x = -\frac{\partial}{\partial y}\psi. \tag{2b}$$

The longitudinal field E_z is obtained from

$$\nabla_\perp^2 E_z = \frac{1}{\epsilon_0 c}\nabla_\perp \cdot J_\perp\,. \tag{3}$$

The components of the magnetic field are given by

$$\nabla_\perp^2 B_x = \mu_0(-\partial_y J_z + \partial_\zeta J_y)\,, \tag{4a}$$

$$\nabla_\perp^2 B_y = \mu_0(\partial_x J_z - \partial_\zeta J_x)\,, \tag{4b}$$

and

[1] https://github.com/Hi-PACE/hipace.

$$\nabla_\perp^2 B_z = \mu_0(\partial_y J_x - \partial_x J_y), \tag{5}$$

where μ_0 denotes the vacuum permeability. All quantities except the longitudinal derivatives $\partial_\zeta J_x$ and $\partial_\zeta J_y$ are directly accessible after the current deposition. These derivatives can be obtained with a predictor-corrector loop [20,22] or by explicit integration [38,39]. Both options are available in HiPACE++, hereafter referred to as predictor-corrector or explicit method respectively, and their implementations are described in Sec. 3.2.

In the QSA PIC algorithm presented here, the fields at slice ζ are advanced in space (from $\zeta + \Delta\zeta$ to ζ) and the beam particles are advanced in time (from t to $t + \Delta t$) together, in the following sequence:

1. Gather fields and push plasma particles backwards from $\zeta + \Delta\zeta$ to ζ;
2. Deposit plasma currents and densities;
3. Deposit beam currents and densities;
4. Solve equation (1) for ψ to calculate $E_x - c\,B_y$ and $E_y + c\,B_x$ with equation (2);
5. Solve equation (3) for E_z;
6. Solve equation (5) for B_z;
7. Solve equations (4) for $B_{x/y}$.
8. Gather fields and push beam particles located in slice ζ from t to $t + \Delta t$.

In the standard QSA PIC algorithm (see [20,22]), the beam operations (current deposition, field gather and particle push) are separated from the loop over slices: at the end of the loop over slices, once the fields are computed on the whole 3D domain, the 3D fields are used to advance the beam. Here, we integrate the beam operations into the 2D slice routine, which removes the necessity to allocate the memory of the 3D simulation domain. This modification is a key requirement to harness the full compute potential of a GPU, as explained in the next section. Additionally, the 3D field arrays are never used for computation and therefore do not need to be allocated, which allows for fitting high-resolution simulations on a single GPU. This enables production-quality simulations even with modest GPU resources.

3. Porting quasi-static PIC to GPU

3.1. Porting strategy

HiPACE++ is written considering modern GPU architectures with tens-of-GB global memory, relatively slow transfers between host (CPU) and device (GPU) memories, and fast atomic operations. As illustrated in Fig. 1, the data that needs to be allocated for computation is modest, and only consists in the beam particles, a 2D slice of plasma particles, and a 2D slice of grid quantities. In a vast majority of practical cases, these quantities fit in the global memory of a single GPU. Thus, these are directly allocated in the GPU memory, reducing the need for host-device transfers during computation to its minimum. Host-device transfers are only used for I/O and communication during the longitudinal parallelization, although both can be in principle circumvented by using buffering methods combined with optimized transfers, such as using NVIDIA GPUDirect. Additionally, keeping all required data directly on the GPU makes it possible to use single-GPU Fast Fourier Transforms (FFTs) that are considerably faster than single- or multi-CPU FFTs, which accounts for a significant fraction of the observed speedup.

A practical consequence of this strategy is that the full 3D domain is not needed for computation and thus never allocated on GPU, leading to much reduced memory utilization. The GPU memory limits the transverse resolution, though high (2048×2048 grid points) up to extreme (8192×8192 grid points) resolutions are achievable with small GPUs (8 GB GPU-memory) and state-of-the-art GPUs (80 GB GPU-memory), respectively.

This has another important consequence: Since the 3D domain is not allocated on the GPU, the fields on a slice overwrite the previous values and are not known at the end of the loop over slices. For this reason, the beam operations (field gather, particle push and current deposition) must be performed per slice within the loop over slices. To that end, beam particles are sorted per slice at the beginning of each time step. Although this results in many small and inefficient kernels (a slice of beam particles contains ~1000 particles for typical simulation parameters), the beam operations take a negligible amount of time overall. Finally, field data in the full 3D domain can be stored for the purpose of diagnostics. In that case, the required data is stacked in host (CPU) memory until the last slice is computed, and then flushed to disk.

In summary, the porting strategy combines two elements: (i) fit the 2D transverse problem in GPU memory, so it can be solved without excessive communications and (ii) reduce host-device transfers to a minimum. Space is saved on device memory by not allocating 3D field arrays, thus enabling the computation of large domains on a single GPU.

3.2. Implementation

As part of the main loop over slices, the most time-consuming functions in the quasi-static PIC method are the field solver, the plasma particle pusher, and the plasma current deposition. Performance-portability is achieved via the AMReX framework [36], and the same methods as in Ref. [30] are applied: low-level loops (over all particles or over all grid points in a slice) are written in an abstract form (through a function `amrex::ParallelFor`), which is compiled into a vectorized loop (for CPU) or a GPU kernel depending on the target platform, enabling portability of a unique source code. The GPU implementation exploits fast single-GPU FFTs as well as fast atomic operations on modern GPUs (in particular for the current deposition, the most expensive particle operation). The implementation of the core functions is described in the next paragraphs.

For the **plasma particle push**, HiPACE++ uses a fifth-order Adams-Bashforth particle pusher, as described in [22]. The transverse beam position $x_\perp$, the transverse normalized momentum $u_\perp = \frac{1}{Mc}(p_x, p_y)$, and the normalized plasma wake potential $\Psi_p = \frac{e}{m_e c^2}\psi_p$ of each plasma particle (with M being the mass of the pushed plasma particle, m_e the mass of an electron, and e the elementary charge) are updated as follows:

$$\partial_\zeta x_\perp = -\frac{u_\perp}{1+\Psi_p}, \tag{6}$$

$$\partial_\zeta u_\perp = -\frac{q}{M}\left[\frac{\gamma_p}{1+\Psi_p}\begin{pmatrix} E_x - cB_y \\ E_y + cB_x \end{pmatrix} + \begin{pmatrix} cB_y \\ -cB_x \end{pmatrix} + \frac{cB_z}{1+\Psi_p}\begin{pmatrix} u_y \\ -u_x \end{pmatrix}\right], \tag{7}$$

$$\partial_\zeta \psi_p = -\frac{qm_e}{Me}\left[\frac{1}{1+\Psi_p}\begin{pmatrix} u_x \\ u_y \end{pmatrix} \cdot \begin{pmatrix} E_x - cB_y \\ E_y + cB_x \end{pmatrix} - E_z\right], \tag{8}$$

with q being the charge of the particle and γ_p the Lorentz factor given by

$$\gamma_p = \frac{1 + u_\perp^2 + (1+\Psi_p)^2}{2(1+\Psi_p)}. \tag{9}$$

The beam particles are advanced by a second-order symplectic integrator. The field gather and particle push are embarrassingly parallel operations well-suited to GPU computing.

Due to the handling per slice, the **current deposition** is so far limited to zeroth order longitudinally for both plasma and beam particles, while orders 0-3 are available in the transverse direction. On GPU, the otherwise expensive current deposition is performed using fast atomic operations to global device memory.

As can be seen in Eqs. (2), (3) and (5), most fields are computed by **solving a transverse Poisson equation** and applying finite-difference operators. The Poisson equation with Dirichlet boundary conditions is solved by means of fast Poisson solvers [40], which are based on a Discrete Sine Transform (DST) of the first type. The DST is provided by the FFTW [41] library on CPU and by a custom implementation using FFTs [42] on GPU. The FFTs on GPU are provided by vendor libraries. The capability to run 2D FFTs on a single GPU instead of parallel FFTs on many CPUs is critical to provide good performance, considering that parallel FFTs require large amount of communications. Special care is needed to compute $B_{x/y}$, because of the longitudinal derivatives $\partial_\zeta J_x$ and $\partial_\zeta J_y$ in Eq. (4). The $B_{x/y}$ field solver is usually the most expensive part of the 3D QSA PIC method.

Two options are implemented for the $B_{x/y}$ field solver algorithm. The first option is a **predictor-corrector field solver**, as implemented in the legacy code HiPACE. The longitudinal derivatives $\partial_\zeta J_{x/y}$ are evaluated on slice ζ from the previously-computed slice $\zeta + \Delta\zeta$ and slice $\zeta - \Delta\zeta$ still to be computed. An initial guess is made for $B_{x/y}$, with which particles are pushed from slice ζ to $\zeta - \Delta\zeta$ where their current is deposited. The current on slice $\zeta - \Delta\zeta$ is used to calculate $B_{x/y}$ at ζ, and the procedure is repeated until a convergence criterion is reached or a maximum number of iterations is attained. Each iteration involves all PIC operations for plasma particles as well as several Poisson solves.

The second option for the $B_{x/y}$ field solver is an **explicit field solver** using analytic integration, as done in Refs. [38,39]. A 2D non-homogeneous Helmholtz-like equation must be solved (see equation (19) in Ref. [39]), for which HiPACE++ uses the GPU-capable multigrid solver provided by AMReX. The multigrid solver is an expensive operation relying on an iterative solver, but does not require multiple iterations of PIC operations.

4. Correctness

The reference setup used throughout this article consists in a typical beam-driven wakefield acceleration simulation containing a driver beam and witness beam with Gaussian distributions with rms sizes $k_p\sigma_{\perp,d} = 0.3$, $k_p\sigma_{\zeta,d} = 1.41$ and $k_p\sigma_{\perp,w} = 0.1$, $k_p\sigma_{\zeta,w} = 0.2$, where $k_p = \omega_p/c$ is the plasma wavenumber, $\omega_p = \sqrt{n_0 e^2/(m_e\epsilon_0)}$ is the plasma frequency, and n_0 the ambient plasma density (subscripts d and w stand for driver and witness, respectively).

The driver beam is located at the origin and has a peak density of $n_{b,d}/n_0 = 10$. The witness beam is centered around longitudinal position $k_p\zeta_{0,witness} = -5$ and has a peak density of $n_{b,w}/n_0 = 100$. The electron plasma is modeled with 4 particles per cell, and the background ions are assumed to be immobile. The simulation domain in x, y, and ζ is, in units of k_p^{-1}, $(-8, 8)$, $(-8, 8)$, and $(-7, 5)$ and uses $1024 \times 1024 \times 1024$ grid points. All simulation parameters and the used software are listed in the Appendix.

Fig. 2 shows a comparison between HiPACE++, the legacy code HiPACE, and the full GPU-capable 3D electromagnetic PIC code WarpX [25]. The accelerating field E_z/E_0, where $E_0 = cm_e\omega_p/e$ is commonly referred to as the cold non-relativistic wave breaking limit [37], shows excellent agreement between these three codes. For HiPACE++, both the predictor-corrector and the explicit solver were used, and both demonstrate again very good agreement, as shown in Fig. 2(b). For WarpX, the rigid beams propagated in a uniform plasma long enough for the wake to reach a steady state. Minor differences in the witness beam region and in the spike at the back of the bubble can be attributed to different physical models, numerical methods, and initialization.

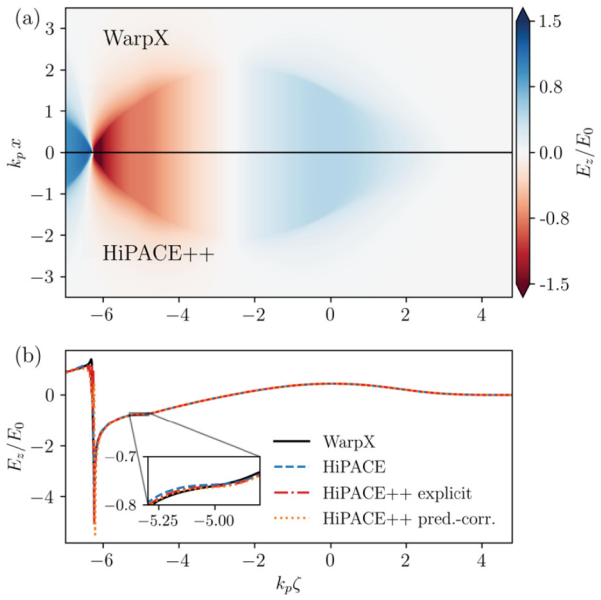

Fig. 2. (a) x-ζ snapshot of the electric field in a beam-driven wakefield acceleration simulation using WarpX (top) and HiPACE++ (bottom). (b) Lineout of the accelerating field from WarpX, HiPACE, and HiPACE++. The inset shows a zoom on the witness beam region, where flattening of the accelerating field due to beam loading is visible. Both the predictor-corrector (pred.-corr.) and the explicit field solvers are shown. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

5. Performance and parallelization

In this section, the performance of HiPACE++ is evaluated. Although QSA PIC codes are considered fast due to their large time steps, full 3D simulations remain computationally expensive. To reduce the runtime, the QSA PIC loop can be parallelized in two independent ways: First, for so-called *transverse parallelization*, the computation of a 2D slice can be performed by multiple processing units via transverse domain decomposition. Second, in the *longitudinal parallelization*, the domain is decomposed longitudinally, and different processing units compute different parts of this domain. Due to intrinsic dependencies of the QSA PIC method (at a given time step, the head of the domain *must* be computed before the tail), this longitudinal parallelization takes the form of a pipeline [43,44], where different ranks compute different time steps.

Transversely, the computation of individual slices in HiPACE++ is performed on 1 GPU (when running on GPU, see Sec. 5.1) or using multiple OpenMP threads (when running on CPU). Longitudinally, the code is parallelized with MPI through a novel pipeline algorithm, see Sec. 5.2.

Unless stated otherwise, all simulations in this section ran on the JUWELS Booster, where each node is equipped with 2 AMD EPYC 7402 processors with 24 cores each and 4 NVIDIA A100 GPUs (40 GB, NVLink3) per node.

5.1. Single-GPU performance

As discussed in Sec. 3 and illustrated in Fig. 1, the amount of data that must be allocated for a 3D domain is relatively modest and consists of beam particles and 2D slices of plasma particles

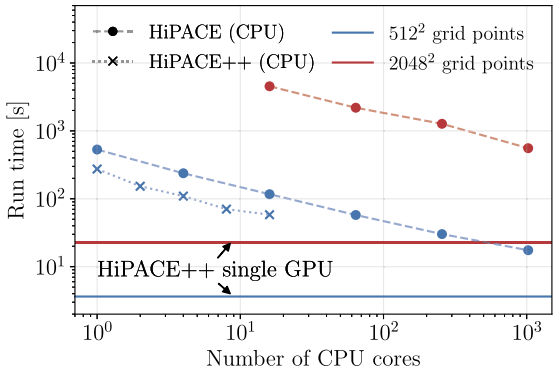

Fig. 3. Performance comparison between GPU and CPU, for the same setup as Sec. 4 for a single time step with medium ($512 \times 512 \times 1024$ cells, blue lines) and high ($2048 \times 2048 \times 1024$ cells, red lines) resolutions, with predictor-corrector field solver) on the JUWELS Booster. Simulations on CPU used HiPACE (MPI-parallel, dashed lines) and HiPACE++ (OpenMP-parallel, dotted line). Simulations on GPU used HiPACE++. The high resolution run with HiPACE does not fit on less than 16 nodes on CPU.

and fields on the grid. Due to the relatively small size of beam data, the amount of data virtually depends only on the transverse number of cells. For example, the total allocated data on the global memory of a NVIDIA A100 GPU with the explicit solver (respectively predictor-corrector method) with 2 million beam particles (accounting for ~ 230 MB) and 1 particle per cell for the plasma electrons is 2.0 GB (resp. 2.0 GB) for a problem of 128×128 cells transversely, 2.9 GB (resp. 2.6 GB) for a 1024×1024 problem and 19.2 GB (resp. 12.9 GB) for a 4096×4096 problem size (for details on all simulation parameters see the Appendix). Therefore, most practical problems fit on a single GPU, and the performance of HiPACE++ on a single NVIDIA A100 GPU is detailed below.

The benefit of fitting the problem on a single GPU is clearly demonstrated in Fig. 3. Typical CPU implementations of the 3D QSA PIC method [22,39,21] accelerate the calculation by decomposing the domain transversely, resulting in large amounts of communications (in particular in the Poisson solver) that dominate the runtime and cause non-ideal scaling. The CPU runs used only the 48 CPU cores on the nodes of the JUWELS Booster. The GPU runs also used the 4 GPUs. The CPU runs were parallelized in the transverse direction only. Longitudinal parallelization is an orthogonal problem, and is done in the exact same way on CPU and GPU (see Sec. 5.2). On GPU, the simulations at medium and high resolutions take 3.6 sec and 22.9 sec and cost 2.5×10^{-4} node-hours and 1.6×10^{-3} node-hours, respectively. For the same simulations using 1024 cores on CPU, HiPACE requires 17.5 sec and 556.1 sec for a cost of 0.10 node-hours and 3.3 node-hours. At medium resolution, the run on 1 (1024) CPU cores was 145× (4.7×) slower and cost 12× (630×) more node-hours than on 1 GPU. At high resolution, the run on 16 (1024) CPU cores was 197× (24×) slower and cost 261× (2050×) more node-hours than on 1 GPU. The number of node-hours was calculated as [number of CPU cores]/48 for CPU runs, and [number of GPUs]/4 for GPU runs, as each node has 48 CPU cores and 4 GPUs.

For CPU computing with no hardware accelerator, shared-memory parallelization with OpenMP is implemented to enable transverse parallelization when running on CPU only. In that case, tiling is implemented for plasma particle operations (field gather, particle push and current deposition), and the threaded version of FFTW can be called. As shown by the dotted line in Fig. 3, the transverse OpenMP parallelization of HiPACE++ gives a similar scaling as the pure MPI transverse parallelization of the legacy code HiPACE up to 16 threads (running on 16 cores of the 24-core JUWELS Booster CPUs). We attribute the performance improvement of HiPACE++ over HiPACE to better memory handling, but

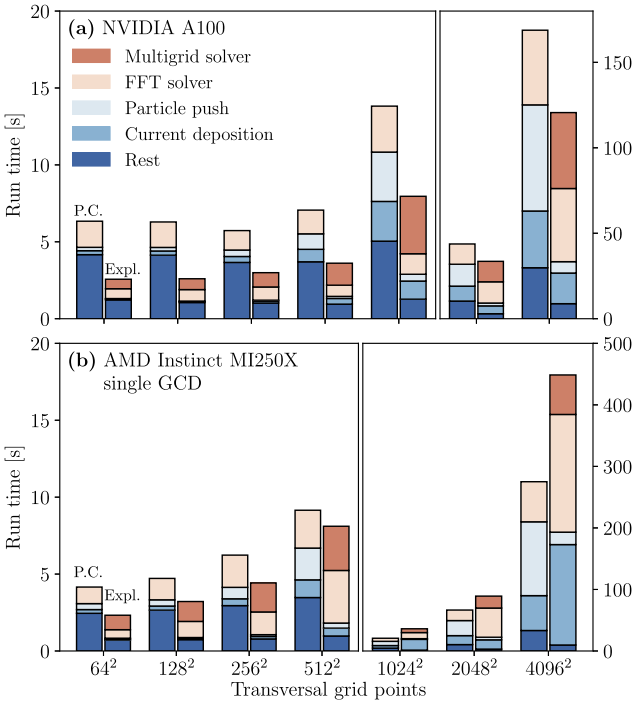

Fig. 4. Runtime for different transverse resolutions on (a) NVIDIA A100 GPUs and (b) a single Graphics Compute Die (GCD) of an AMD Instinct MI250X. Left bars: using the predictor-corrector loop. Right bars: using the explicit field solver. The runtimes of 1024×1024 for AMD Instinct MI250X only, 2048×2048 and 4096×4096 transverse grid points are plotted on a separate y-axis to improve readability of the figure.

detailed profiling of the legacy code HiPACE is out of scope of this article.

For further insight into the performance of HiPACE++, we ran the reference setup presented in Sec. 4 with increasing transverse resolution, keeping all other parameters constant (for more details see the Appendix). This scan uses 1024 longitudinal grid points, and performance data is given for both the predictor-corrector loop and the explicit field solver. The predictor-corrector loop used up to 5 iterations, which typically yields a comparable level of convergence between the two solvers in standard beam-driven plasma accelerator scenarios. We observed that the explicit solver converges faster than the predictor-corrector loop in challenging simulation settings, such as large transverse box sizes or abrupt beam current spikes.

The most time-consuming functions of the two solvers on an NVIDIA A100 are shown in Fig. 4 (a). In both cases a vast majority of the time is spent in solving for $B_{x/y}$. While both the fast Poisson solver and particle operations dominate the predictor-corrector solver at different resolutions, the multigrid solver is always the most expensive operation for the explicit solver. As a reminder, each iteration in the predictor-corrector loop involves all PIC operations for the plasma particles (field gather, particle push, current deposition and field solve) repeated up to 5 times per slice. Note that this study is not a comparison of the two field solvers, as they have different convergence properties, but rather a performance analysis of each solver separately.

The performance portability of HiPACE++ on ROCm-capable AMD GPUs is demonstrated by running the transverse scaling on a single Graphics Compute Die (GCD) of an AMD Instinct MI250X, shown in Fig. 4 (b). The scan was performed on the early-access test system Crusher at the Oak Ridge Leadership Facility, which is equipped with a 64-core AMD EPYC 7A53 “Optimized 3rd Gen EPYC” CPU and four AMD Instinct MI250X. Each MI250X contains two GCDs, which can be viewed as two separate GPUs from a program-

ming perspective. The run time differs roughly by $0.7-1.6\times$ (resp. $0.9-3.7\times$) for the predictor-corrector solver (resp. explicit solver) in comparison with the NVIDIA A100. Note that these results were obtained on a test platform to demonstrate the portability and significant performance improvements are to be expected, for example by optimizations of rocFFT, which currently suffers from performance penalties for FFTs with grid sizes that are not a power of two.

5.2. Longitudinal parallelization via temporal domain decomposition

As presented in Sec. 2, the computation of the fields in the full domain at a given time step relies on a loop from the headmost slice to the tail, and consequently cannot be parallelized longitudinally by standard domain decomposition. When computing multiple time steps, longitudinal parallelization can be achieved via pipelining algorithms [43,44], which were first realized in the form of a spatial decomposition [43]. Because of the combination of (i) faster computation of each slice and (ii) using a single rank per slice, the standard spatial decomposition demonstrates poor scaling with our GPU implementation. We hereafter present a temporal domain decomposition, more suitable to the GPU implementation in HiPACE++. This implementation has some similarities with the streaming pipeline presented in Ref. [44]. Both pipelines are summarized below, assuming the problem is decomposed longitudinally in as many sub-domains (*boxes*) as the number of ranks n_{ranks}, and runs for `nt` time steps.

In the spatial decomposition, each rank gets assigned one sub-domain, which it consecutively calculates for every time step. After the rank has calculated its sub-domain for time step t, it passes the plasma particles and slices required for computation via MPI to the next rank downstream. It then receives the plasma particles and slices at $t+\Delta t$ from the next rank upstream. A rank keeps its associated beam particles, unless they slip backwards out of the sub-domain due to longitudinal velocities smaller than the speed of light. The communication caused by beam particle slippage is usually negligible.

The algorithm (in pseudo-code) reads:

```
# Rank r computes box b for all time steps
for t in 0:nt-1:
  Receive last slice from box b+1 at time t
  Compute box b at time t
  Send last slice to box b-1 at time t
```

where a slice consists in field data and plasma particle data. This pipeline is represented on the left of Fig. 5. The number of scalars communicated per time step and per rank reads $N_s = n_x n_y (S_{cell} + n_{ppc} S_{plasma})$ where n_x (n_y) is the number of cells in the transverse direction x (y), S_{cell} is the number of scalars communicated per cell (in HiPACE++, $S_{cell} = 6$ for J_x and J_y of the previous slice, and $B_{x/y}$ of the two previous slices) and S_{plasma} is the number of scalars communicated per plasma particle (in HiPACE++, $S_{plasma} = 35$ due to fifth-order Adams-Bashforth pusher). Here, n_{ppc} is the number of plasma particles per cell. Each rank always communicates a full slice, so the amount of data communicated does not scale with n_{ranks}.

In the temporal decomposition, each rank computes the full domain for the subset of time steps t for which $t \equiv r \pmod{nt}$ (where r is the current rank). At each time step, the assigned rank computes the full domain in a loop over the boxes, from head to tail. After each box is calculated, the beam particles within that box are sent to the next rank downstream, which calculates the next time step for this box. Then, the rank receives the beam particles of the next box from the rank upstream and continues its calculation from head to tail. The rank keeps the plasma particles and 2D grid quantities and resets them at each new time step in the first box.

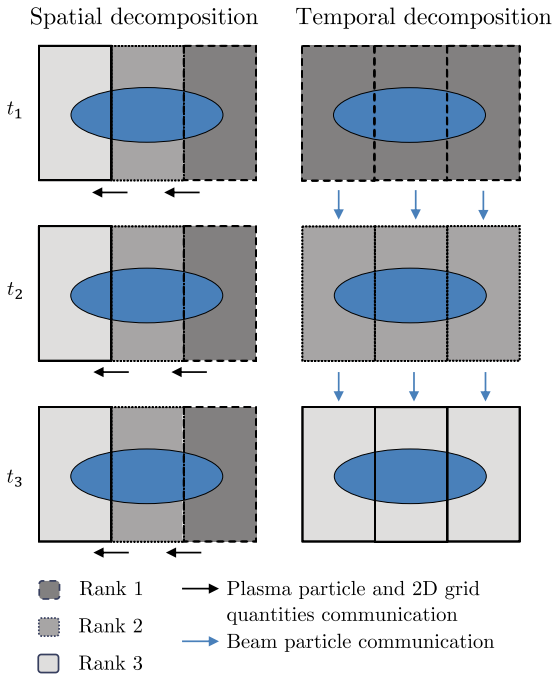

Fig. 5. Left: spatial domain decomposition. Each rank calculates a fixed sub-domain for all time steps. Plasma particles and 2D field slices need to be communicated. Right: temporal domain decomposition. Each rank calculates the full domain for a sub-set of time steps. The beam particles of a sub-domain need to be communicated.

The algorithm (in pseudo-code) reads:

```
# Rank r computes all boxes for time step t
for b in nb-1:0:
  Receive beam from box b at time t-1
  Compute box b at time t
  Send beam to box b at time t+1
```

This pipeline is represented on the right of Fig. 5. The number of scalars communicated per time step and per rank reads $N_t = n_{beam,r} \times S_{beam}$ where $n_{beam,r}$ and S_{beam} denote the number of beam particles on that rank and number of scalars communicated per beam particle, respectively. As can be seen in Fig. 5, $n_{beam,r}$ scales with the number of ranks (*i.e.*, the number of sub-domains), so this pipeline should perform better for strong scaling.

Scalability is the key advantage of the temporal decomposition: in the spatial decomposition, the amount of data to send/receive is constant (one slice of fields and plasma particles) while, in the temporal decomposition, it scales with the number of ranks (only the beam particles within the sub-domain are communicated).

As an example, let us consider a typical problem with $n_x = n_y = 1024$, $n_{ppc} = 1$, and $n_{beam,total} = 2 \times 10^6$. Even in the most favorable case (exchanging as few scalars as possible), $S_{plasma} = 7$ for position, momentum and particle weight (HiPACE++ uses 35), $S_{beam} = 7$ and $S_{cell} = 6$ to calculate the initial guess, the amount of data (assuming IEEE 754 `double` precision) exchanged per rank and per time step is 110 MB for the spatial decomposition. Although this is usually not the case, we assume a load-balanced beam particle distribution across ranks for simplicity, so that $n_{beam,r} = n_{beam,total}/n_{ranks}$. The temporal decomposition exchanges roughly $110\,\text{MB}/n_{ranks}$ per rank and per time step. For $n_{ranks} = 256$ the temporal decomposition exchanges roughly two

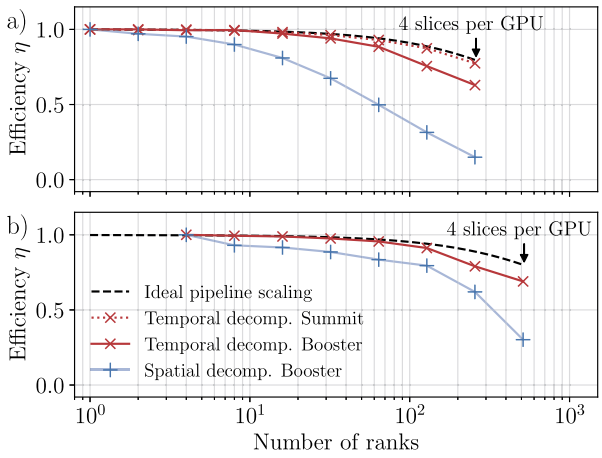

Fig. 6. Strong scaling for two different problems with a) $1024 \times 1024 \times 1024$ cells with 4 plasma particles per cell for 1000 time steps and b) $2048 \times 2048 \times 2048$ cells with 1 plasma particle per cell for 2048 time steps. Both settings used two beams with 10^6 beam particles each. The final run time is given for the maximum number of ranks used. In b) the scaling starts at 4 GPUs due to time limit restriction on the supercomputer. The problems are parallelized in the longitudinal direction only.

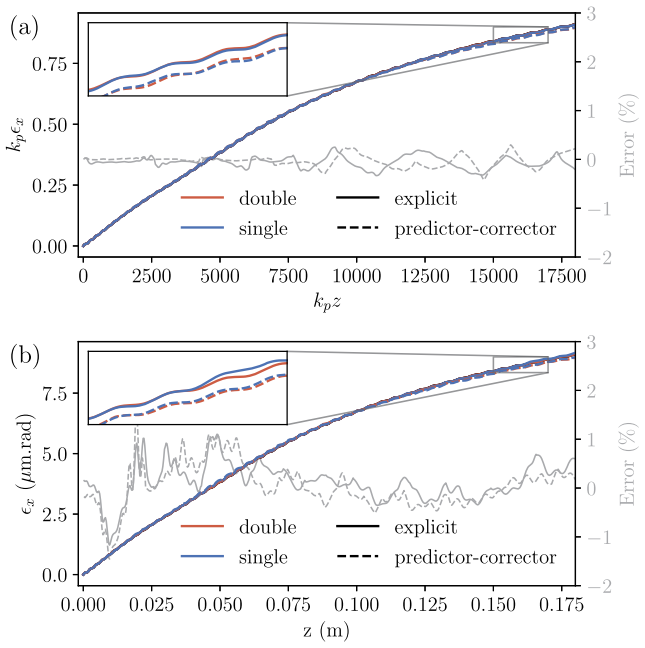

Fig. 7. (a) Evolution of the emittance during propagation over 3000 time steps of the witness beam presented in Sec. 4 with an initial transverse offset of the witness bunch centroid of $x_b = \sigma_x$, for the two field solvers, in normalized units. The error due to single precision is computed for each field solver with respect to the double-precision simulation; (b) Same for a simulation running in SI units, where $k_p^{-1} = 10\,\mu$m.

orders of magnitude less data than the spatial decomposition, and is hence expected to show better scalability.

The performance of the temporal decomposition pipeline is assessed via a strong scaling of two different setups. One setup is the reference simulation setup from Sec. 4 with $n_{steps} = 1000$ time steps and the other uses $2048 \times 2048 \times 2048$ cells and 2048 time steps (for more details see the Appendix). The efficiency η is given by $\eta(n_{ranks}) = t(1)/[n_{ranks}t(n_{ranks})]$ where $t(i)$ is the run time on i ranks. Due to the filling and emptying of the pipeline, the ideal efficiency for both pipelines is not identically 1 but rather given by

$$\eta_{ideal}(n_{ranks}) = \left(1 + \frac{n_{ranks} - 1}{n_{steps}}\right)^{-1}. \tag{10}$$

An efficiency of 1 is obtained in the limit of $n_{steps} \gg n_{ranks}$. The results are shown in Fig. 6. The temporal domain decomposition (red lines) shows an efficiency close to the ideal pipeline scaling (black dashed line). The spatial decomposition (blue lines) suffers from efficiency degradation above 8 ranks. Both scalings were performed on the JUWELS Booster and the reference setup was also run on Summit (red dotted line), which is equipped with 6 NVIDIA V100 GPUs per node. The maximum number of ranks is chosen so that only 4 slices remain per sub-domain, which was the case at 256 ranks (= 256 GPUs) for the reference setup and 512 ranks for the higher-resolution case.

Note, that the temporal domain decomposition outperforms the spatial decomposition even though it is at a disadvantage: due to performance enhancements unrelated to the parallelization, the absolute run time is reduced, causing the communications to take up a larger fraction of the total run time.

6. Software practice and additional features

HiPACE++ is a versatile, open-source, 3D, quasi-static PIC software with an object-oriented design to invite the integration of new numerical methods or physics packages. HiPACE++ uses the cross-platform build system CMake and can be installed, as well as its dependencies, with software package managers, such as Spack [45].

HiPACE++ complies with the openPMD standard [46] and uses the openPMD-api [47] for I/O, allowing for interoperability and simple benchmarking with other codes. Both HDF5 [48] and ADIOS2 [49] file formats are supported (a feature inherited from the openPMD-api), and the capability to read an external beam from file at the openPMD format is available.

Two unit systems, SI units and normalized units, are available as a runtime parameter. In normalized units, all lengths are re-scaled to the plasma skin depth k_p^{-1}, the fields to the cold, non-relativistic wave breaking limit E_0, and all densities to the background plasma density n_0. All operations are performed in the unit system chosen by the user. An advanced parser makes it possible to write the input file in a unit system and run the simulation in the other one, allowing to use the advantages of both unit systems (numerical accuracy, interoperability with other codes, convenience for multi-physics implementations, etc.) in a flexible manner.

The code can be compiled in either double (C++ `double`) or single (C++ `float`) precision, a feature inherited from AMReX. The effect of the precision on the simulation accuracy is investigated by comparing the evolution of the emittance of the witness beam of the reference setup with an initial emittance of $\epsilon_{x,0} = 0$ when an initial transverse offset of the bunch centroid $x_b = \sigma_x$ is present in Fig. 7. For both predictor-corrector and explicit field solvers, the error attributed to using single precision remains well below 1% (2%) in normalized units (SI units) after 3000 time steps. As expected, the difference between single and double precision is higher for SI units than for normalized units, although both remain on the percent level.

Table 1 shows the runtime in single and double precisions for the two solvers on two different architectures, a cutting-edge HPC GPU (NVIDIA A100) and a typical consumer-grade ("*gaming*") GPU (NVIDIA RTX2070), easily available on a laptop. As anticipated, double-precision calculations are much faster on the HPC GPU than on the gaming GPU. However, with the capability to run high-resolution production simulations on a gaming GPU with comparable accuracy and performance as on an HPC GPU in single precision, HiPACE++ provides useful scalability from laptops to the largest supercomputers.

Table 1
Runtime for 1 time step of the simulation presented in Fig. 7 on NVIDIA A100 and NVIDIA RTX2070 GPUs. P-C stands for the predictor-corrector loop.

GPU	Solver	T_{double}	T_{single}	speedup
A100	P-C	14.9 s	11.4 s	1.3×
A100	explicit	8.0 s	5.8 s	1.4×
RTX2070	P-C	96.9 s	33.7 s	2.9×
RTX2070	explicit	52.4 s	17.0 s	3.1×

Though in active development, HiPACE++ features numerous capabilities useful for production simulations including multiple beams and plasma with different species and profiles (driver and witness beam, ion motion, etc.), the possibilities to load external beams and apply external fields as well as specialized mesh refinement capabilities [50]. Field ionization using the ADK-model [51] is available in SI units and could readily be extended to normalized units. HiPACE++ proposes different field solvers, and uses modern software practices to make it a user-friendly and stable code (continuous integration, open development repository, extensive documentation and comments). Planned upgrades include a laser envelope model [52,17,53], full mesh refinement, and the support of more GPU architectures from other providers. The code is fully operational on CPU, on modern NVIDIA GPUs, and modern, ROCm-capable AMD GPUs.

7. Conclusion

This paper presented the open-source, performance-portable, 3D quasi-static PIC code HiPACE++. The main adjustments required to port the quasi-static PIC loop to accelerated computing consist in (i) ensuring that all operations, including the beam operations, are performed within the loop over slices so little data needs to be stored on device memory and (ii) proposing a different longitudinal parallelization (pipeline) with which the amount of data communicated per rank scales down with the number of ranks, enabling excellent scalability for production simulations up to hundreds of GPUs.

Focusing on runtime rather than scalability, HiPACE++ is not MPI-parallel transversally: each slice is computed on a single GPU, enabling orders-of-magnitude speedup with respect to CPU implementations. Transverse parallelization will be considered if it provides significant speedup without impacting the code clarity. Benchmarks show excellent agreement with legacy code HiPACE and full electromagnetic PIC code WarpX.

HiPACE++ is built on top of cutting-edge libraries (in particular AMReX and openPMD-api) to harness top performance-portability and encourage open science, while improving sustainability. It enables production simulations of plasma acceleration from laptops to supercomputers.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors gratefully acknowledge helpful discussions with T.J. Mehrling, C.B. Schroeder, J. Osterhoff, T. Wetzel, and B. Diederichs. We gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS Booster at Jülich Supercomputing Centre (JSC). We acknowledge the Funding by the Helmholtz Matter and Technologies Accelerator Research and Development Program. This research was also supported by the Exascale Computing Project (No. 17-SC-20-SC), a collaborative effort of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration. We acknowledge the support of the Director, Office of Science, Office of High Energy Physics, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Appendix A. Simulation parameters

Table A.2
Simulation parameters of the presented studies in the respective figure. P-C stands for predictor-corrector solver, n_{ppc} refers to the number of plasma particles per cell, the *beam* column indicates whether the beam was initialized as a random beam with a fixed number of particles and a fixed weight or by a beam with fixed particles per cell and a variable weight, and n_{steps} denotes the number of time steps. The maximum number of iterations in the predictor-corrector solver is given between brackets in the Solver column.

Fig.	Solver	$n_{x/y} \times n_\zeta$	n_{ppc}	Beam	n_{steps}
2	P-C (5)	1024×1024	4	fixed ppc	1
	explicit	1024×1024	4	fixed ppc	1
3	P-C (1)	512×1024	1	random	1
		2048×1024	1	random	1
4	P-C (5)	$2^n \times 1024$	1	random	1
	explicit	$2^n \times 1024$	1	random	1
6	P-C (1)	1024×1024	4	random	1000
	P-C (1)	2048×2048	1	random	2048
7	P-C (5)	1024×1024	1	random	3000

In all simulations, second-order current deposition was used in the transverse direction. The reference setup consists of a drive and a witness beam. Both beams are Gaussian with rms sizes $k_p\sigma_{\perp,d} = 0.3$, $k_p\sigma_{\zeta,d} = 1.41$ and $k_p\sigma_{\perp,w} = 0.1$, $k_p\sigma_{\zeta,w} = 0.2$. The peak densities are $n_{b,d}/n_0 = 10$ and $n_{b,w}/n_0 = 100$. The drive beam has an initial energy of 10 GeV and 0.1% rms energy spread, the witness beam has an initial energy of 1 GeV and no initial energy spread. The drive beam is located at the origin, the witness beam is centered around longitudinal position $k_p\zeta_{0,w} = -5$. The beams are initialized at waist, either with a fixed number of particles per cell with a variable weight or with random positions and fixed weights. The beams initialized by a variable weight use 1 particle per cell. The randomly initialized beams use 10^6 fixed weight particles per beam. For all simulations, the domain is, in units of k_p^{-1}, $(-8, 8)$, $(-8, 8)$, and $(-7, 5)$ in x, y, and ζ. The varying simulation parameters for the presented studies are listed in Table A.2. The time step in all simulations is $dt = 6\,\omega_p^{-1}$.

For the HiPACE++ and WarpX simulations on the JUWELS Booster, we used GCC 9.3.0, CUDA 11.0, OpenMPI 4.1.0rc1, CMake 3.18.0, and FFTW 3.3.8, except for the strong scaling using the temporal domain decomposition on the JUWELS Booster, which used GCC 10.3.0, CUDA 11.3, and OpenMPI 4.1.1. The GPU runs were compiled with nvcc 11.0.221 using the following flags: `-O3 -gencode=arch=compute_80,code=sm_80 -gencode=arch=compute_80,code=compute_80 -maxrregcount=255 --use_fast_math`.

The HiPACE++ CPU runs were compiled using the following flags: `-O3 -DNDEBUG -pthread -fopenmp -Werror=return-type`.

The legacy code HiPACE was compiled with ICC 19.1.2.254 using the flags: `-std=c99 -march=native -O3 -Os`.

On the laptop, we used GCC 8.4.0, CUDA 11.0, MPI 3.1, and CMake 3.20.3. The code was compiled with nvcc 11.0.194 using the following flags: `-O3 -DNDEBUG --expt-relaxed-constexpr`

```
--expt-extended-lambda -Xcudafe
--diag_suppress=esa_on_defaulted_function_ignored
-maxrregcount=255 -Xcudafe --display_error_number
--Wext-lambda-captures-this --use_fast_math
-Xcompiler -pthread.
```

Throughout the studies, we used AMReX v21.05 to v22.04 and HiPACE++ from commit 3f2f4e15a607 to v22.04, except for the simulation using spatial domain decomposition, which was conducted on commit 11523c24c0f7c73ce3fe8d3424ede54565f58d50.

References

[1] T. Tajima, J.M. Dawson, Phys. Rev. Lett. 43 (1979) 267–270, https://doi.org/10.1103/PhysRevLett.43.267.

[2] P. Chen, J.M. Dawson, R.W. Huff, T. Katsouleas, Phys. Rev. Lett. 54 (1985) 693–696, https://doi.org/10.1103/PhysRevLett.54.693.

[3] C.A. Lindstrøm, J.M. Garland, S. Schröder, L. Boulton, G. Boyle, J. Chappell, R. D'Arcy, P. Gonzalez, A. Knetsch, V. Libov, G. Loisch, A. Martinez de la Ossa, P. Niknejadi, K. Põder, L. Schaper, B. Schmidt, B. Sheeran, S. Wesch, J. Wood, J. Osterhoff, Phys. Rev. Lett. 126 (2021) 014801, https://doi.org/10.1103/PhysRevLett.126.014801.

[4] J. Couperus, R. Pausch, A. Köhler, O. Zarini, J. Krämer, M. Garten, A. Huebl, R. Gebhardt, U. Helbig, S. Bock, et al., Nat. Commun. 8 (1) (2017) 1–7, https://doi.org/10.1038/s41467-017-00592-7.

[5] M. Kirchen, S. Jalas, P. Messner, P. Winkler, T. Eichner, L. Hübner, T. Hülsenbusch, L. Jeppe, T. Parikh, M. Schnepp, A.R. Maier, Phys. Rev. Lett. 126 (2021) 174801, https://doi.org/10.1103/PhysRevLett.126.174801.

[6] W. Wang, K. Feng, L. Ke, C. Yu, Y. Xu, R. Qi, Y. Chen, Z. Qin, Z. Zhang, M. Fang, et al., Nature 595 (7868) (2021) 516–520, https://doi.org/10.1038/s41586-021-03678-x.

[7] R. Hockney, J. Eastwood, Computer Simulation Using Particles, Advanced Book Program: Addison-Wesley, McGraw-Hill, 1981, https://doi.org/10.1201/9780367806934.

[8] C. Birdsall, A. Langdon, Plasma Physics via Computer Simulation, The Adam Hilger Series on Plasma Physics, McGraw-Hill, 1985, https://doi.org/10.1201/9781315275048.

[9] R. Zgadzaj, T. Silva, V. Khudyakov, A. Sosedkin, J. Allen, S. Gessner, Z. Li, M. Litos, J. Vieira, K. Lotov, et al., Nat. Commun. 11 (1) (2020) 1–11, https://doi.org/10.1038/s41467-020-18490-w.

[10] A. Caldwell, K. Lotov, A. Pukhov, F. Simon, Nat. Phys. 5 (5) (2009) 363–367, https://doi.org/10.1038/nphys1248.

[11] S. Schröder, C. Lindstrøm, S. Bohlen, G. Boyle, R. D'Arcy, S. Diederichs, M. Garland, P. Gonzalez, A. Knetsch, V. Libov, et al., Nat. Commun. 11 (1) (2020) 1–6, https://doi.org/10.1038/s41467-020-19811-9.

[12] R. Courant, K. Friedrichs, H. Lewy, Math. Ann. 100 (1) (1928) 32–74, https://doi.org/10.1007/BF01448839.

[13] J.-L. Vay, Phys. Rev. Lett. 98 (2007) 130405, https://doi.org/10.1103/PhysRevLett.98.130405.

[14] P. Sprangle, E. Esarey, A. Ting, Phys. Rev. A 41 (1990) 4463–4469, https://doi.org/10.1103/PhysRevA.41.4463.

[15] P. Sprangle, E. Esarey, A. Ting, Phys. Rev. Lett. 64 (1990) 2011–2014, https://doi.org/10.1103/PhysRevLett.64.2011.

[16] P. Mora, T.M. Antonsen Jr., Phys. Rev. E 53 (1996) R2068–R2071, https://doi.org/10.1103/PhysRevE.53.R2068.

[17] P. Mora, T.M. Antonsen Jr., Phys. Plasmas 4 (1) (1997) 217–229, https://doi.org/10.1063/1.872134.

[18] K.V. Lotov, Phys. Plasmas 5 (3) (1998) 785–791, https://doi.org/10.1063/1.872765.

[19] K.V. Lotov, Phys. Rev. Spec. Top., Accel. Beams 6 (2003) 061301, https://doi.org/10.1103/PhysRevSTAB.6.061301.

[20] C. Huang, V. Decyk, C. Ren, M. Zhou, W. Lu, W. Mori, J. Cooley, T.M. Antonsen Jr., T. Katsouleas, J. Comput. Phys. 217 (2) (2006) 658–679, https://doi.org/10.1016/j.jcp.2006.01.039.

[21] W. An, V.K. Decyk, W.B. Mori, T.M. Antonsen Jr., J. Comput. Phys. 250 (2013) 165–177, https://doi.org/10.1016/j.jcp.2013.05.020.

[22] T.J. Mehrling, C. Benedetti, C.B. Schroeder, J. Osterhoff, Plasma Phys. Control. Fusion 56 (8) (2014) 084012, https://doi.org/10.1088/0741-3335/56/8/084012.

[23] C. Benedetti, C.B. Schroeder, E. Esarey, C.G.R. Geddes, W.P. Leemans, AIP Conf. Proc. 1812 (1) (2017) 050005, https://doi.org/10.1063/1.4975866.

[24] A. Pukhov, CERN Yellow Rep. 1 (2016) 181, https://doi.org/10.5170/CERN-2016-001.181.

[25] J.-L. Vay, A. Huebl, A. Almgren, L. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. Grote, M. Hogan, et al., Phys. Plasmas 28 (2) (2021) 023105, https://doi.org/10.1063/5.0028512.

[26] P. Yu, X. Xu, V.K. Decyk, W. An, J. Vieira, F.S. Tsung, R.A. Fonseca, W. Lu, L.O. Silva, W.B. Mori, J. Comput. Phys. 266 (2014) 124–138, https://doi.org/10.1016/j.jcp.2014.02.016.

[27] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, Top 500 supercomputers, http://www.top500.org/lists/2021/11/, 2021.

[28] H. Burau, R. Widera, W. Hönig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T.E. Cowan, R. Sauerbrey, M. Bussmann, IEEE Trans. Plasma Sci. 38 (10) (2010) 2831–2839, https://doi.org/10.1109/TPS.2010.2064310.

[29] M. Bussmann, H. Burau, T.E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W.E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, R. Widera, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, 5, https://doi.org/10.1145/2503210.2504564.

[30] A. Myers, A. Almgren, L. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. Grote, M. Hogan, A. Huebl, R. Jambunathan, R. Lehe, C. Ng, M. Rowan, O. Shapoval, M. Thévenet, J.-L. Vay, H. Vincenti, E. Yang, N. Zaïm, W. Zhang, Y. Zhao, E. Zoni, Parallel Comput. 108 (2021) 102833, https://doi.org/10.1016/j.parco.2021.102833.

[31] R. Bird, N. Tan, S.V. Luedtke, S. Harrell, M. Taufer, B. Albright, IEEE Trans. Parallel Distrib. Syst. 33 (04) (2022) 952–963, https://doi.org/10.1109/TPDS.2021.3084795.

[32] H.C. Edwards, C.R. Trott, D. Sunderland, in: Domain-Specific Languages and High-Level Frameworks for High-Performance Computing, J. Parallel Distrib. Comput. 74 (12) (2014) 3202–3216, https://doi.org/10.1016/j.jpdc.2014.07.003.

[33] D.A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A.J. Kunen, O. Pearce, P. Robinson, B.S. Ryujin, T.R. Scogland, in: 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2019, pp. 71–81, https://doi.org/10.1109/P3HPC49587.2019.00012.

[34] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W.E. Nagel, M. Bussmann, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2016, pp. 631–640, https://doi.org/10.1109/IPDPSW.2016.50.

[35] M. Thévenet, S. Diederichs, A. Huebl, A. Myers, A. Sinn, R. Lehe, J.-L. Vay, W. Zhang, Hi-pace/hipace: v21.09, https://doi.org/10.5281/zenodo.5358484, Sep. 2021.

[36] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, M. Zingale, J. Open Sour. Softw. 4 (37) (2019) 1370, https://doi.org/10.21105/joss.01370.

[37] E. Esarey, C.B. Schroeder, W.P. Leemans, Rev. Mod. Phys. 81 (2009) 1229–1285, https://doi.org/10.1103/RevModPhys.81.1229.

[38] T. Wang, V. Khudik, B. Breizman, G. Shvets, Phys. Plasmas 24 (10) (2017) 103117, https://doi.org/10.1063/1.4999629.

[39] T. Wang, V. Khudik, G. Shvets, WAND-PIC: a three-dimensional quasi-static particle-in-cell code with parallel multigrid solver and without predictor-corrector, https://doi.org/10.48550/arXiv.2012.00881, 2020.

[40] C. Van Loan, Computational Frameworks for the Fast Fourier Transform, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, 1992, https://doi.org/10.1137/1.9781611970999.

[41] M. Frigo, S. Johnson, Proc. IEEE 93 (2) (2005) 216–231, https://doi.org/10.1109/JPROC.2004.840301.

[42] J.W. Cooley, P. Lewis, P. Welch, J. Sound Vib. 12 (3) (1970) 315–337, https://doi.org/10.1016/0022-460X(70)90075-1.

[43] B. Feng, C. Huang, V. Decyk, W. Mori, P. Muggli, T. Katsouleas, J. Comput. Phys. 228 (15) (2009) 5340–5348, https://doi.org/10.1016/j.jcp.2009.04.019.

[44] A. Sosedkin, K.V. Lotov, Nucl. Instrum. Methods Phys. Res., Sect. A, Accel. Spectrom. Detect. Assoc. Equip. 829 (2016) 350–352, https://doi.org/10.1016/j.nima.2015.12.032.

[45] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. de Supinski, S. Futral, in: SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2015, pp. 1–12, https://doi.org/10.1145/2807591.2807623.

[46] A. Huebl, R. Lehe, J.-L. Vay, D.P. Grote, I.F. Sbalzarini, S. Kuschel, D. Sagan, C. Mayes, F. Perez, F. Koller, M. Bussmann, openPMD: a meta data standard for particle and mesh based data, https://doi.org/10.5281/zenodo.591699, 2015.

[47] A. Huebl, F. Poeschel, F. Koller, J. Gu, openPMD-api: C++ & Python API for Scientific I/O with openPMD, https://doi.org/10.14278/rodare.27, 2018.

[48] The HDF Group, Hierarchical data format version 5, http://www.hdfgroup.org, 2000–2019.

[49] W.F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, et al., SoftwareX 12 (2020) 100561, https://doi.org/10.1016/j.softx.2020.100561.

[50] T.J. Mehrling, C. Benedetti, C.B. Schroeder, E. Esarey, in: 2018 IEEE Advanced Accelerator Concepts Workshop (AAC), IEEE, 2018, https://doi.org/10.1109/AAC.2018.8659404.

[51] M.V. Ammosov, N.B. Delone, V.P. Krainov, in: J.A. Alcock (Ed.), High Intensity Laser Processes, in: International Society for Optics and Photonics, vol. 0664, SPIE, 1986, pp. 138–141, https://doi.org/10.1117/12.938695.

[52] P. Sprangle, E. Esarey, J. Krall, G. Joyce, Phys. Rev. Lett. 69 (15) (1992) 2200, https://doi.org/10.1103/PhysRevLett.69.2200.

[53] C. Benedetti, C.B. Schroeder, C.G.R. Geddes, E. Esarey, W.P. Leemans, Plasma Phys. Control. Fusion 60 (1) (2017) 014002, https://doi.org/10.1088/1361-6587/aa8977.