

Multi-GPU Hybrid Particle-in-Cell Monte Carlo Simulations for Exascale Computing Systems

Jeremy J. Williams¹, Jordy Trilaksono², Stefan Costea³, Yi Ju⁶, Luca Pennati¹, Jonah Ekelund¹, David Tskhakaya⁴, Leon Kos³, Ales Podolnik⁴, Jakub Hromadka⁴, Allen D. Malony⁵, Sameer Shende⁵, Tilman Dannert⁶, Frank Jenko², Erwin Laure⁶, and Stefano Markidis¹

¹ KTH Royal Institute of Technology, Stockholm, Sweden

² Max Planck Institute for Plasma Physics, Garching, Germany

³ Faculty of Mechanical Engineering, University of Ljubljana, Ljubljana, Slovenia

⁴ Institute of Plasma Physics of the CAS, Prague, Czech Republic

⁵ University of Oregon, Eugene, Oregon, USA

⁶ Max Planck Computing and Data Facility, Garching, Germany

Abstract. Particle-in-Cell (PIC) Monte Carlo (MC) simulations are central to plasma physics but face increasing challenges on heterogeneous HPC systems due to excessive data movement, synchronization overheads, and inefficient utilization of multiple accelerators. In this work, we present a portable, multi-GPU hybrid MPI+OpenMP implementation of BIT1 that enables scalable execution on both Nvidia and AMD accelerators through OpenMP target tasks with explicit dependencies to overlap computation and communication across devices. Portability is achieved through persistent device-resident memory, an optimized contiguous one-dimensional data layout, and a transition from unified to pinned host memory to improve large data-transfer efficiency, together with GPU Direct Memory Access (DMA) and runtime interoperability for direct device-pointer access. Standardized and scalable I/O is provided using openPMD and ADIOS2, supporting high-performance file I/O, in-memory data streaming, and in-situ analysis and visualization. Performance results on pre-exascale and exascale systems, including Frontier (OLCF-5) for up to 16,000 GPUs, demonstrate significant improvements in run time, scalability, and resource utilization for large-scale PIC MC simulations.

Keywords: Heterogeneous Computing · Hybrid MPI+OpenMP · BIT1 · Nvidia · AMD · Persistent GPU Memory · Asynchronous Multi-GPU Execution · Plasma Edge Modeling · Large-Scale PIC MC Simulations

1 Introduction

High-performance computing (HPC) is rapidly moving toward heterogeneous architectures [8], where efficient orchestration of computation, memory management, and overlap of communication and computation are essential for large-scale particle-based plasma simulations on pre-exascale and exascale systems [18].

BIT1 is a massively parallel Particle-in-Cell (PIC) Monte Carlo (MC) code for plasma simulations and plasma–material interactions. While earlier MPI-only versions scaled well on CPU clusters [17], moving to GPU supercomputers exposed limitations in on-node communication, memory usage, and efficient multi-GPU utilization. To address this, BIT1 was extended with hybrid MPI+OpenMP and MPI+OpenACC parallelization for shared-memory execution and initial GPU acceleration, which also identified key bottlenecks such as the particle mover and arranger [20,22,24], and was complemented by scalable data management and I/O based on the openPMD standard with parallel [21], streaming [25], and in-situ workflows [23] to reduce storage overhead and post-processing costs.

In this work, we present a portable multi-GPU hybrid MPI+OpenMP BIT1 implementation with persistent device-resident memory, optimized data transfers, GPU runtime interoperability for seamless integration between CUDA, HIP, and OpenMP runtimes, and asynchronous multi-GPU execution. By keeping data resident on the GPU, minimizing redundant transfers, and coordinating kernels asynchronously across devices, BIT1 efficiently exploits modern GPU accelerated systems while maintaining large-scale MPI scalability and portability across Nvidia and AMD GPU architectures.

The main contributions of this work are:

- We design hybrid MPI+OpenMP versions of BIT1 to improve performance on a single node and in both strong and weak scaling tests, employing a task-based approach to mitigate load imbalance and optimize resource utilization.
- We extend OpenMP GPU offloading for Nvidia and AMD, enabling portability using OpenMP target tasks with `nowait` and `depend` clauses, overlapping computation and communication, flattening 3D arrays into contiguous 1D layouts, and maintaining device-resident memory across time steps using OpenMP target `enter/exit data` regions, and switching from unified to pinned host memory for large arrays to reduce data movement overheads.
- We implement vendor-specific GPU optimizations to improve data-transfer efficiency, using compile-time pinned memory on Nvidia and OpenMP pinned allocators on AMD, enabling higher throughput and lower-latency host-to-device (HtoD) transfers for large arrays.
- We integrate openPMD and ADIOS2 (BP4 and SST backends) to remove I/O bottlenecks, enabling high-throughput I/O and in-memory streaming, with in-situ analysis and visualization for real-time insight and reduced post-processing on Nvidia and AMD GPUs architectures at scale.

2 Background

The PIC method is one of the most widely used computational approaches for plasma simulations and is applied across a broad range of plasma environments, including space and astrophysical plasmas, laboratory experiments, industrial processes, and fusion devices [16,20].

As seen in Fig. 1, the classical PIC method begins with an initialization stage, in which the simulation setup and initial conditions are defined. The main

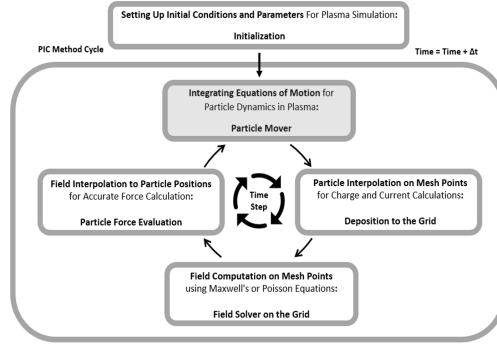


Fig. 1: A diagram representing the PIC Method on HPC architectures. After initialization, the PIC method repeats at each time step. In gray, we highlight the particle mover step that we parallelize in the portable multi-GPU hybrid BIT1.

simulation loop then executes four phases: (i) the particle mover, which updates particle positions and velocities; (ii) deposition to the grid, which maps particle charges and currents to the mesh; (iii) the field solver, which computes electromagnetic or electrostatic fields by solving Maxwell's or Poisson's equations; and finally (iv) particle force evaluation, which interpolates the grid fields back to the particle positions. This cycle is repeated for each time step, with each step representing a small fraction of the characteristic plasma timescale in order to accurately capture plasma dynamics [20,24].

In this work, we use the Berkeley Innsbruck Tbilisi 1D3V (BIT1) code for large-scale plasma simulations on HPC systems, targeting plasma edge and tokamak divertor physics [15,17]. BIT1 is a 1D3V electrostatic PIC MC code derived from Birdsall's XPDP1 code (UC Berkeley, 1990s) [19]. It is written in C (31,600 lines of code) and uses native Poisson solver, particle mover, and smoothing operators without external numerical libraries.

BIT1 employs the "natural sorting" method [15], which accelerates collision operators by up to a factor of five, but increases memory use and complicating GPU offloading and load balancing in high-density regions [20,24]. To address these limitations and improve portability, BIT1 is evolving toward a hybrid MPI + OpenMP model that reduces communication overhead and improves scalability on multi-core and heterogeneous architectures [22,24].

2.1 OpenMP Memory Allocators, Unified & Pinned Host Memory

OpenMP (Open Multi-Processing) provides a standardized mechanism for controlling memory allocation through *memory allocators* (Table 1), which return logically contiguous memory from a specified memory space and guarantee that allocations do not overlap. The behavior of an allocator is configured using *allocator traits* (Table 2) that define synchronization expectations, alignment, accessibility, fallback behavior, pinning and memory placement, providing a portable abstraction for heterogeneous memory hierarchies without relying on vendor-specific API capabilities.

OpenMP Allocator Name	Associated Memory Space	Primary Use
omp_default_mem_alloc	omp_default_mem_space	Default system storage

Table 1: Predefined OpenMP allocator and memory space. [10,12]

OpenMP Allocator Trait	Allowed Values	Default Setting
pinned	true, false	false

Table 2: OpenMP allocator trait, accepted values, and default setting. [10,12]

Efficient data movement is critical for particle-based PIC codes on heterogeneous systems. Unified (managed) memory (UM) offers a single virtual address space with on-demand page migration between CPU and GPU, simplifying programming but potentially incurring overheads due to page faults and runtime migrations for irregular or latency-sensitive access patterns [9]. In contrast, pinned (page-locked) host memory (PinM) enables higher-bandwidth and lower-latency transfers, supports asynchronous copies and GPU DMA, but must be managed explicitly and can increase pressure on host memory resources. For performance critical PIC MC kernels, such as particle movers and collision operators, PinM is therefore commonly preferred when explicit data movement and overlap of communication and computation are required [2,11].

2.2 openPMD Standard, openPMD-api Integration & ADIOS2

The openPMD (open Particle-Mesh Data) standard defines a portable, extensible data model for particle and mesh data, supporting multiple storage backends such as HDF5, ADIOS1, ADIOS2, and JSON in serial and MPI workflows [3]. The openPMD-api library provides a unified interface, storing physical quantities as multi-dimensional records for mesh fields or particle data, with time-dependent data organized as iterations [4].

ADIOS2 (Adaptable Input/Output System version 2) is an open-source parallel I/O framework for scalable HPC data movement, offering a unified API for multi-dimensional variables, attributes, and time steps, with modular engines including BP4 (high-throughput file I/O) and SST (low-latency streaming for in-situ MPI-parallel workflows) [25]. In hybrid BIT1, we use BP4 for high-throughput file I/O [21,23] and SST for in-memory data streaming [25] independently, enabling in-situ analysis and visualization without interrupting asynchronous multi-GPU execution on pre-exascale and exascale systems.

3 Related Work

The rapid transition of scientific simulations toward exascale has driven a shift from traditional CPU implementations to heterogeneous, accelerator focused, and hybrid programming models capable of efficiently leveraging multiple GPU devices. Early work in hybrid parallelization for particle-based simulations explored OpenMP and MPI for scalable performance on HPC clusters [1]. GPU

programming models have evolved significantly, with OpenMP offloading emerging as a promising framework for portability across Nvidia, AMD, and Intel GPUs [7]. Studies have evaluated OpenMP offloading for computational kernels and data-transfer strategies, demonstrating performance comparable to vendor optimized numerical libraries on H100 and MI250X GPUs [6]. Other investigations have focused on portable GPU runtimes, with Tian et al. [13] showing that OpenMP 5.1, with minor compiler extensions, can replace vendor-specific runtimes such as CUDA or HIP without performance loss, enabling efficient compilation across LLVM/Clang toolchains for multiple GPU architectures. Similarly, portable implementations of MC particle transport codes, such as OpenMC, have been successfully ported to heterogeneous GPUs using OpenMP target offloading, with large-scale benchmarks confirming efficiency across AMD, Nvidia, and Intel accelerators [14].

4 Methodology & Experimental Setup

In this work, we focus on a portable, multi-GPU hybrid MPI+OpenMP implementation of BIT1 for PIC MC simulations on exascale systems, leveraging persistent GPU memory, contiguous 1D data layouts, runtime interoperability, asynchronous multi-GPU execution, and OpenMP target offloading with vendor-specific optimizations, while integrating high-performance file and streaming I/O, and in-situ workflows using openPMD and ADIOS2.

4.1 Portable Multi-GPU Hybrid BIT1

OpenMP is a widely adopted programming model for shared-memory parallelism in HPC, supported by major compilers including GCC, LLVM, and Intel, providing portability across platforms and simplifying the development of parallel applications through directives and runtime routines.

OpenMP Target Parallelization, Data Structure & Memory Layout. Previously, Williams et al. [5,22,24] developed an OpenMP target tasks implementation for the particle mover in hybrid BIT1. In this function, particle positions and velocities are stored in `x[species][cell][particle]` and `vx[species][cell][particle]`, where `nsp`, `nc` and `np[species][cell]` denote the number of species, cells and particles per cell, respectively [15]. Such 3D data layouts cause non-contiguous GPU accesses [26], reducing memory efficiency and bandwidth. To improve GPU runtime performance, hybrid BIT1 adopts a revised data layout after identifying limitations in large 3D arrays, such as `x[species][cell][particle]`. These large arrays were restructured into contiguous 1D arrays (`x_GPU[species*cell*particle]` or `x_GPU[:LenA]`) to enable sequential memory access, simpler indexing and higher sustained throughput in GPU kernels. HtoD data movement is optimized using PinM to ensure fixed physical data residency, supporting efficient asynchronous transfers and GPU DMA. On Nvidia GPUs, PinM is selected at compile-time, while on AMD GPUs, where compile-time UM/PinM is unavailable, OpenMP memory allocators explicitly

manage memory placement and selective pinning of performance critical arrays, minimizing overhead while maintaining portability across heterogeneous GPU architectures.

```

1 // Enter unstructured OpenMP target data region
2 #pragma omp target enter data map(to: chsp[:lenA], sn2d[:lenA], dinj[:lenA],
3 poff[:lenA], nstep[:lenA], maxL[:lenA], np_GPU[:lenA], x_GPU[:lenA],
4 y_GPU[:lenA], vx_GPU[:lenA], vy_GPU[:lenA])
5 // Persistent device-resident (GPU) memory allocation
6 {
7     while (tstep < last_step) {
8         ...
9         (*move_p)(); // Particle mover -> move0, moveb
10        ...
11        xarrj(); // Particle arranger -> arrj, narrj
12        ...
13        // I/O -> openPMD, ADIOS2, dumpstep, datfiles
14        ...
15        ...
16        ...
17        ...
18        ...
19    } // End of while(tstep < last_step)
20 }
21 ...
22 // Exit unstructured OpenMP target data region
23 #pragma omp target exit data map(delete: x_GPU[:lenA], y_GPU[:lenA])
24 // Release persistent device-resident (GPU) memory

```

Listing 1.1: Simplified C code snippet illustrating the Persistent Asynchronous GPU OpenMP Offloading

Persistent Device-Resident Memory Allocation. Building on the asynchronous GPU acceleration of the particle mover [24], hybrid BIT1 reduces GPU data movement overhead by using persistent device-resident memory allocation. In Listing 1.1, rather than repeatedly transferring and allocating large arrays (`x_GPU[:LenA]`, `vx_GPU[:LenA]`, `y_GPU[:LenA]`, `vy_GPU[:LenA]`) each time step, a single `#pragma omp target enter data region` persistently allocates all relevant arrays and control data (`chsp`, `sn2d`, `dinj`, `poff`, `nstep`, `maxL`, `np_GPU`) on the GPU. The main simulation loop then iterates over time steps, executing the particle mover (`move0`, `moveb`), particle arranger (`arrj`, `narrj`), and performing in-situ I/O without repeated data transfers. After the simulation, `#pragma omp target exit data map(delete: ...)` releases the device-resident memory.

```

1 /* Initialize OMP Pinned Allocator */
2 omp_allocator_handle_t pinned_alloc = omp_null_allocator;
3 ...
4 // Create Pinned OpenMP Memory Allocator
5 if (pinned_alloc == omp_null_allocator) {
6     omp_allocator_trait_t traits[1];
7     ...
8     pinned_alloc = omp_init_allocator(omp_default_mem_space, 1, traits);
9     ...
10 }
11 // Pinned Allocation
12 x_GPU = (float *) omp_alloc(lenA * sizeof(float), pinned_alloc);
13 ...
14 // Persistent Device-Resident (GPU) Memory Allocation
15 ...
16 // GPU OpenMP Offloading, Sorting, Computation, I/O
17 ...
18 // Release Persistent Device-Resident (GPU) Memory
19 ...
20 // Final I/O -> openPMD, ADIOS2, dumpstep, datfiles
21 ...
22 // Destroy and Cleanup OpenMP Pinned Memory
23 if (pinned_alloc != omp_null_allocator) {
24     omp_free(x_GPU, pinned_alloc);
25     ...
26     omp_destroy_allocator(pinned_alloc);
27     pinned_alloc = omp_null_allocator;
28 }
29 ...

```

Listing 1.2: Simplified C code snippet illustrating OpenMP memory allocators with the PinM trait enabled.

Predefined OpenMP Memory Allocators. Efficient data movement is critical in GPU accelerated hybrid BIT1, especially for large particle arrays. Listing 1.2 shows how hybrid BIT1 reduces repeated allocations and transfers (par-

ticularly on AMD GPUs) by using OpenMP PinM memory allocators, which provide fixed residency, high-throughput asynchronous transfers, and GPU DMA while preserving portability. The allocator is initialized once at startup, allocating large arrays (`x_GPU`, `y_GPU`, `vx_GPU`, `vy_GPU`) persistently on the GPU for all time steps, enabling asynchronous offloading with overlapping computation and I/O. At simulation end, arrays are released, PinM is freed via `omp_free`, and the allocator destroyed with `omp_destroy_allocator`, ensuring clean memory management.

```

1 // Host pointer already mapped into its device pointer
2 #pragma omp target data use_device_ptr(np_GPU, x_GPU, y_GPU, vx_GPU, vy_GPU)
3 {
4     for (isp=0; isp<nsip; isp++) {
5         ...
6         #pragma omp target teams distribute parallel for
7         // Do not map; pointer already device-resident
8         is_device_ptr(poff, nstep, maxL, np_GPU, x_GPU, vx_GPU)
9         depend(inout: x_GPU[:lenA])
10        depend(in: poff[:lenA], nstep[:lenA], maxL[:lenA], vx_GPU[:lenA], np_GPU[:lenA])
11        firstprivate(cio) private(i, j, idx, idx0, cij)
12        thread_limit(512) num_teams(391) nowait
13        for (j = 0; j < nc; j++) {
14            idx0 = poff[isp] + j*maxL[isp];
15            cij = cio + j;
16            #pragma omp simd
17            for (i=0; i < np_GPU[cij]; i++) {
18                idx = idx0 + i;
19                x_GPU[idx] += nstep[isp]*vx_GPU[idx];
20            }
21        }
22    }
23    } else {
24        ...
25    }
26 } // end of #pragma omp target data region
27 // Wait for all asynchronous OpenMP tasks (GPU kernels)
28 #pragma omp taskwait
29
```

Listing 1.3: Simplified C code snippet illustrating OpenMP Target Tasks with "nowait" and "depend" clauses, and enabling GPU runtime interoperability using "use_device_ptr" and "is_device_ptr" clauses for direct device-pointer access.

Enable GPU Runtime Interoperability. Following the transition from UM to PinM to optimize data transfers, hybrid BIT1 further enables GPU runtime interoperability via direct device-pointer access using OpenMP `use_device_ptr` and `is_device_ptr` clauses. In Listing 1.3, the `#pragma omp target data use_device_ptr (x_GPU, y_GPU, vx_GPU, vy_GPU)` region informs the runtime that the host pointers are already mapped to device memory, allowing GPU kernels to access device-resident data directly without additional mapping or allocation. Within this data region, GPU kernels are launched with `#pragma omp target teams distribute parallel for is_device_ptr (...)` so that arrays such as `poff`, `nstep`, `maxL`, `np_GPU`, `x_GPU`, `y_GPU`, `vx_GPU`, and `vy_GPU` are used as device pointers. The GPU kernels are issued asynchronously using `nowait` and are ordered through explicit `depend (in/out/inout)` clauses, with a final `#pragma omp taskwait` ensuring that all OpenMP tasks (GPU kernels) are completed.

4.2 Use Case & Experimental Environment

We focus on evaluating the performance of hybrid BIT1 on two test cases that differ in problem size and BIT1 functionality. We consider two cases: *i*) a relatively straightforward run simulating neutral particle ionization due to interactions with electrons, and *ii*) the formation of a high-density sheath in front of

so-called *divertor* plates in future magnetic confinement fusion devices, such as the ITER and DEMO fusion devices.

More precisely, the two cases are as follows:

- **Neutral Particle Ionization Simulation.** An unbounded, unmagnetized plasma of electrons, D^+ ions, and D neutrals is considered. Neutral concentration decreases via $\partial n/\partial t = nn_e R$, with n , n_e , and R the neutral density, plasma density, and ionization rate coefficient. We use a 1D geometry with 100K cells, three species, and initial 100 particles per cell per species (total 30M). Simulations run for 200K time steps unless stated otherwise. An important point of this test is that it does not use the Field solver and smoother phases. Baseline simulation: one node (Dardel), 128 MPI processes [20,24].
- **High-Density Sheath Simulation.** A double-bounded, magnetized plasma layer between two walls is considered, initially filled with electrons and D^+ ions. Plasma is absorbed at the walls, recycling D^+ ions into D neutrals and forming a sheath. The 1D geometry has 3M cells, three species, and initial 200 particles per cell per charged species (≈ 1.2 B total). Simulations run for 100K time steps unless stated otherwise. Baseline simulation: five nodes (Dardel), 640 MPI processes, exceeding single-node memory [20,24].

In this work, we simulate hybrid BIT1 on four HPC systems (Table 3) using key parameters in Table 4.

Characteristic	Dardel CPU/GPU	MareNostrum5 (MN5) GPP/ACC	LUMI-C / LUMI-G	Frontier (OLCF-5)
HPC System	HPE Cray EX Supercomputer	Pre-Exascale EuroHPC Supercomputer	Pre-Exascale EuroHPC Supercomputer	HPE Cray EX Exascale supercomputer
Processor	2 × AMD EPYC Zen2 (64 cores)	2 × Intel Sapphire Rapids 8480+ (56 cores)	2 × AMD EPYC 7763 (64 cores)	1 × AMD EPYC (64 cores)
CPU Nodes	1,278	6,480	2,048	–
CPU + GPU Nodes	62	1,120	2,978	9,856
GPUs per Node	4 × AMD MI250X (w/ 2 × GCDs)	4 × Nvidia H100	4 × AMD MI250X (w/ 2 × GCDs)	4 × AMD MI250X (w/ 2 × GCDs)
Memory per Node	256 GB – 2 TB	128 GB HBM – 1 TB	256 GB – 1 TB	512 GB DDR4
Network	Slingshot 200 GB/s	NDR200 InfiniBand	Slingshot-11	Slingshot up to 800 Gb/s
Storage	679 PB	248 PB Online / 402 PB Archive	117 PB	679 PB
Location	KTH Royal Institute of Technology	Barcelona Supercomputing Center	CSC – IT Center for Science	Oak Ridge Leadership Computing Facility

Table 3: Key characteristics of the HPC systems used in performance tests.

Parameter	Description	Minimal I/O & Diagnostics	Heavy I/O & Diagnostics
slow	Sets the plasma profiles and distribution function diagnostics (default=0).	0	0
datfile	Enables time-averaged diagnostics of plasma and particle distributions.	0	1000
dmpstep	Defines when the simulation state is written for restart or checkpointing.	0	5000
mvflag	Specifies a diagnostic snapshot averaged over mvflag time steps.	0	1000
mvStep	Sets the interval between successive diagnostic outputs.	0 (< Last_step)	5000
Last_step	Specifies the final time step at which the simulation terminates.	2000	10000
origdmp	Sets the format; 1=File (Serial) I/O & 2=openPMD (Parallel) BP4/SST.	1/2	1/2

Table 4: Main input parameters for simulation output and control, with key flags for Minimal and Heavy I/O & Diagnostics. [21,23]

5 Performance Results

In this work, we evaluate a portable, multi-GPU hybrid MPI+OpenMP BIT1 using persistent GPU memory, contiguous 1D layouts, runtime interoperability, asynchronous execution, OpenMP offloading, and integrated openPMD and ADIOS2 on Nvidia and AMD GPU architectures.

5.1 Profiling & Understanding the Impact of openPMD on BIT1

We begin by utilizing `gprof`, an open-source profiling tool, to analyze execution time and identify the most frequently used functions across MPI processes. The consolidated `gprof` report provides a detailed performance analysis of the Original BIT1 with and without openPMD and the ADIOS2 backends.

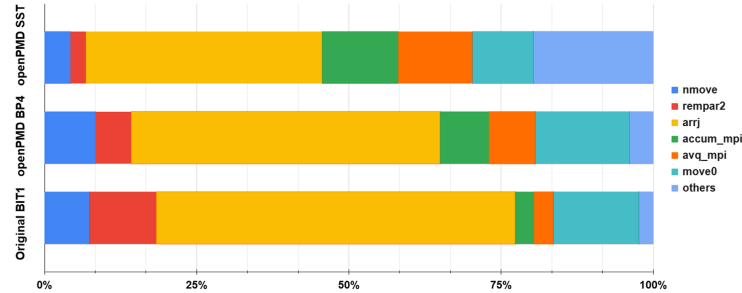


Fig. 2: Ionization case function percentage breakdown (using `gprof`) on Dardel, showing where most of the execution time is spent for Original BIT1, openPMD BP4, and openPMD SST simulations [20,23,25]. The `arrj` sorting function (yellow) dominates but drops from 75.5% (Original BIT1) to 65.5% (BP4) and 35.5% (SST).

As previously reported by Williams et al. [20,23,25], Fig. 2 shows the most time-consuming functions in the "BIT1 openPMD SST" simulation compared with the "BIT1 openPMD BP4" and "Original BIT1" simulations. In the "Original BIT1", `arrj` (a sorting function) dominates at 75.5%, decreasing to 65.5% in BP4 and 35.5% in SST, highlighting improved data handling. The particle mover `move0` drops from 18% to 9.2%, and the particle removal routine `rempar2` from 14% to 2%. The neutral particle mover `nmove` increases in BP4 but falls to 3.9% in SST. The profiling functions `avq_mpi` (profile computation) and `accum_mpi` (smoothed profile computation) increase in BP4 and SST, reflecting enhanced MPI communication, while the `others` functions grows in SST, indicating redistributed computation. Overall, the "BIT1 openPMD SST" simulation spends the least amount of time in the most time-consuming functions, reducing the cost of `arrj` and `move0` compared with the "BIT1 openPMD BP4" and "Original BIT1" simulations, and improving parallel efficiency and overall performance.

5.2 Porting & Accelerating Hybrid BIT1 with OpenMP on 4 GPUs

Next, we extend BIT1 by combining MPI with OpenMP in a task-based hybrid version to mitigate load imbalance and optimize resource utilization, and then port it to MN5 ACC (GPU partition) using one node with four MPI ranks and four GPUs (one GPU per rank) to analyze the impact of a contiguous 1D data layout, OpenMP target parallelism, and multi-GPU asynchronous execution.

As seen in Fig. 3, the original BIT1 version, using 4 MPI + 4 GPUs (Mover) with a 3D data layout and UM, executed with a total simulation time of ≈ 1919 s,

with the `arrj` function dominating at ≈ 1242 s and the `mover` function taking ≈ 568 s. Transitioning to a 1D UM layout improves memory access, reducing the total time to ≈ 830 s and lowering the `mover` and `arrj` times to ≈ 292 s and ≈ 430 s, respectively. Introducing OpenMP thread parallelism to the `arrj` function further decreases its execution time to ≈ 94 s, bringing the total simulation time to ≈ 397 s. Using PinM for the `mover` reduces data transfer overhead, decreasing the total time to ≈ 269 s, and combining PinM with OpenMP `arrj` parallelism further reduces it to ≈ 215 s. Finally, the fully optimized version with persistent GPU memory, asynchronous `mover` execution, and OpenMP `arrj` parallelism reduces the `mover` time to near zero (≈ 0.06 s) and `arrj` to ≈ 11.5 s, resulting in a total simulation time of around ≈ 113 s, corresponding to a $17.04\times$ speed up over the original BIT1 (3D data layout + UM) simulation.

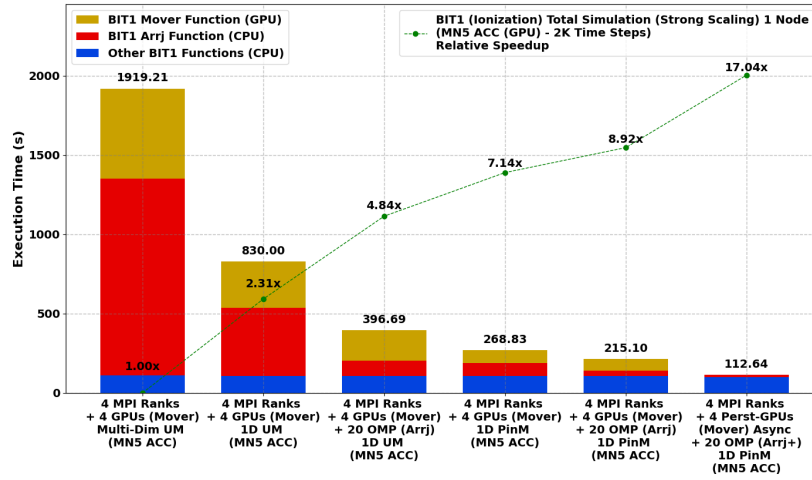


Fig. 3: Hybrid BIT1 (Ionization Case) Total Simulation (Development Progression) strong scaling on 1 Node (4 MPI ranks & 4 GPUs) on MN5 ACC for 2K times steps.

5.3 Hybrid BIT1 (Minimal I/O & Diagnostics) up to 800 GPUs

Moving to the high-density sheath (production-like case), we evaluate the portable, multi-GPU hybrid MPI+OpenMP asynchronous version of BIT1 in both strong and weak scaling tests under minimal I/O and diagnostics. This is performed using a 2K time step sheath simulation up to 100 nodes (up to 800 GPUs), with the parameters listed in Table 4, representing a near compute-only workload with a final checkpoint at the end of the simulation.

As seen in Fig. 4, the original BIT1 version on MN5 GPP saturates under strong scaling beyond 20 nodes, with total execution times between ≈ 211 s and ≈ 247 s at 40–100 nodes, while the optimized CPU version shows only modest improvements and follows a similar trend. In contrast, the hybrid BIT1 versions exhibit substantially better scaling. On MN5 ACC, the total time de-

creases from ≈ 283.99 s at 5 nodes to ≈ 29.02 s at 100 nodes, corresponding to a speed up of $9.73\times$. On LUMI-G, execution time decreases from ≈ 175.21 s to ≈ 15.81 s ($11.08\times$ speed up). On Frontier, the fully optimized GPU version reaches ≈ 12.96 s ($12.80\times$ speed up). Enabling openPMD with ADIOS2 introduces only marginal overhead, with the BP4 and SST backends completing in ≈ 11.85 s ($13.64\times$ speed up) and SST in ≈ 11.71 s ($13.81\times$ speed up) respectively.

For weak scaling, where the problem size increases proportionally with the number of nodes, the hybrid GPU versions maintain nearly constant execution times from 5 to 100 nodes, whereas both CPU versions show a steady increase. On Frontier, execution time remains almost flat, increasing from ≈ 165.84 s at 5 nodes to ≈ 171.06 s at 100 nodes, resulting in a parallel efficiency (PE) of 97.0%. With openPMD BP4, execution increases from ≈ 161.45 s to ≈ 164.84 s (97.9% PE), and with SST, from ≈ 161.36 s to ≈ 164.75 s (98.0% PE), demonstrating that the GPU versions achieve substantial acceleration while maintaining high weak scaling efficiency under minimal I/O and diagnostics.

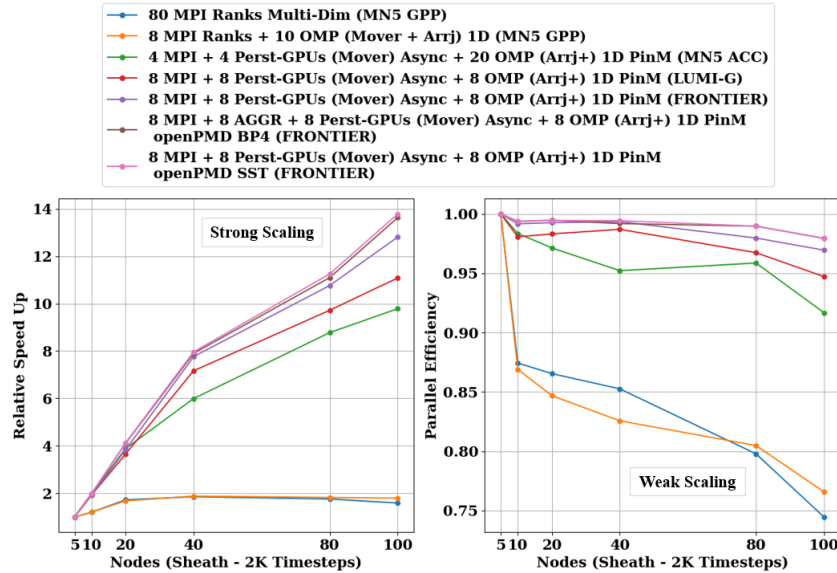


Fig. 4: Hybrid BIT1 (Sheath) Total Simulation (Relative) Speed Up (left) and PE (Right) - Strong and Weak Scaling up to 100 Nodes (up to 800 GPUs) on MN5 ACC, LUMI-G and Frontier for 2K times steps.

5.4 Hybrid BIT1 (Heavy I/O & Diagnostics) up to 16,000 GPUs

We evaluate the exascale readiness of the portable, multi-GPU hybrid MPI + OpenMP version of BIT1 in both strong and weak scaling tests under heavy I/O and diagnostics by performing a 10K time-step sheath simulation on Frontier up to 2000 nodes (up to 16,000 GPUs) with the parameters listed in Table 4.

This represents a heavier diagnostic workload with frequent time dependent output and checkpointing, which heavily stresses computation, communication, I/O, in-situ analysis and visualization, and most importantly, the file system.

As seen in Fig. 5 and strong scaling tests, the original hybrid BIT1 GPU (serial I/O) version achieves a speed up of $2.99\times$ when scaling from 50 to 2,000 nodes. In contrast, the openPMD GPU (parallel I/O) implementation with the ADIOS2 BP4 backend reaches a speed up of $4.90\times$, while the SST backend further improves this to $5.25\times$ speed up. This demonstrates that both openPMD backends improves scalability under heavy I/O, with SST providing the highest scaling efficiency for large GPU runs.

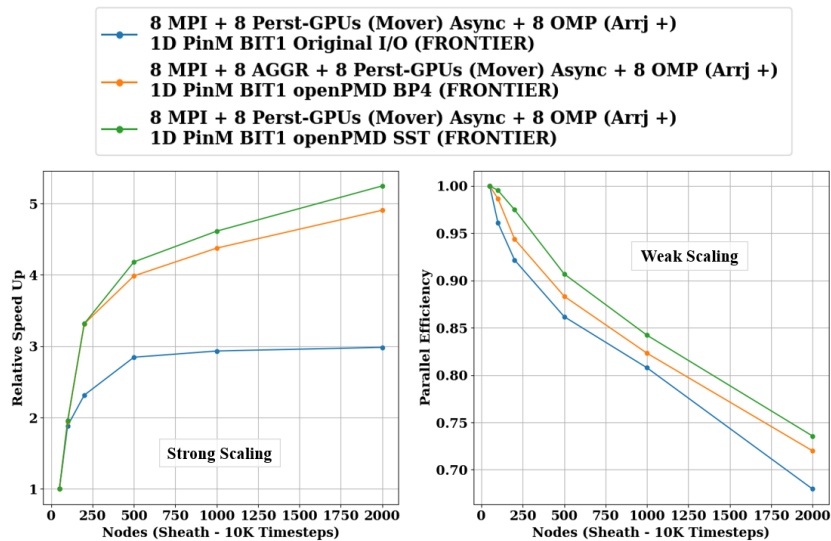


Fig. 5: Hybrid BIT1 (Sheath) Total Simulation (Relative) Speed Up (left) and PE (Right) - Strong and Weak Scaling up to 2000 Nodes (up to 16,000 GPUs) on Frontier for 10K times steps.

For weak scaling, the corresponding PE at 2,000 nodes is 67.9% for the original hybrid BIT1 GPU version, while the openPMD GPU version with BP4 sustains 72.0% PE, and the openPMD GPU version with SST further improves this to 73.6% PE. Despite the extremely heavy diagnostic and I/O workload, the openPMD GPU implementations with ADIOS2 backends (BP4 and SST) consistently maintain higher PE than the original hybrid BIT1 GPU implementation, indicating more robust weak scaling behavior and improved resilience to I/O and diagnostics pressure at scale.

6 Discussion & Conclusion

This work presents a portable, multi-GPU hybrid MPI+OpenMP implementation of BIT1 for large-scale PIC MC simulations on pre-exascale and exascale

supercomputing architectures. By combining persistent device-resident memory, a contiguous 1D data layout, PinM, runtime interoperability and asynchronous multi-GPU execution using OpenMP target tasks with explicit dependencies, BIT1 effectively reduces data movement and synchronization overheads while improving utilization of multiple accelerators.

On Frontier, hybrid BIT1 demonstrates good scalability under both minimal and heavy I/O and diagnostics. With minimal I/O and diagnostics up to 100 nodes (up to 800 GPUs), the fully optimized GPU version reaches ≈ 12.96 s ($12.80\times$ speed up), while openPMD BP4 and SST achieve $13.64\times$ and $13.81\times$ speed up, respectively, with weak scaling remaining nearly flat (≈ 165.84 – ≈ 171.06 s, 97%–98% PE). Under heavy I/O and diagnostics from 50 to 2000 nodes (up to 16,000 GPUs), strong scaling speed up reaches $5.07\times$ (BP4) and $5.25\times$ (SST), while weak scaling achieves 72.0%–73.6% PE, sustaining intensive I/O and frequent diagnostics, demonstrating the effectiveness of combining asynchronous multi-GPU execution with standardized data management at scale.

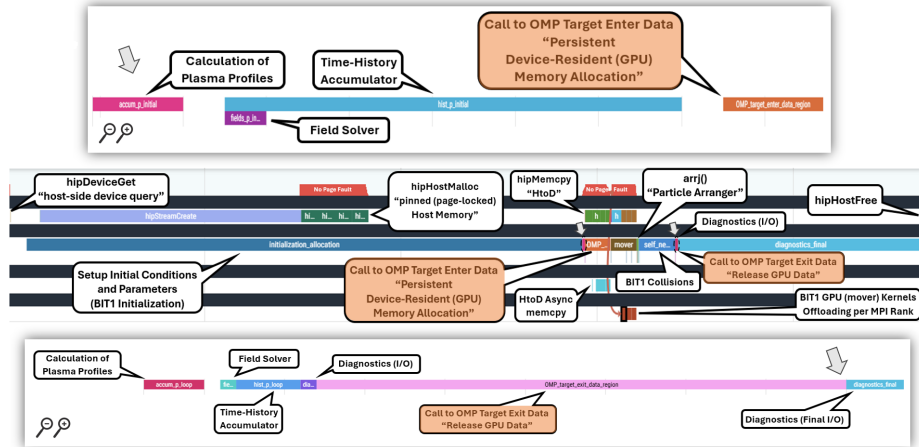


Fig. 6: A single time step Hybrid BIT1 AMD MI250X GPU activity trace showing HSA and HIP activity, ROCTX regions, asynchronous data copies, and mover kernel execution, obtained using `rocprof` and visualized with Perfetto on Dardel GPU, with corresponding confirmation traces on both LUMI-G and Frontier.

Future research will extend hybrid BIT1 to Intel GPU platforms at exascale, targeting *Aurora*, and Europe’s first exascale system, *JUPITER Booster*, to assess portability across Nvidia, AMD and Intel GPUs, and to identify any remaining architecture specific limitations. Deeper performance analysis will be carried out using specialized profiling and tracing tools, including *Nvidia Nsight Systems* (Nvidia GPUs), the open-source portable profiling and tracing toolkit *Tuning and Analysis Utilities* (TAU), and AMD GPU profiling and tracing tools, such as AMD ROC-Profiler (`rocprof`). For instance, we can use `rocprof` to collect correlated traces of Heterogeneous System Architecture

(HSA) and Heterogeneous-computing Interface for Portability (HIP) activity, including ROCm Tools Extension (ROCTX) regions (`roctxRangePush()` and `roctxRangePop()`), asynchronous data transfers, and GPU kernel execution, all captured in JSON format for use with Perfetto, enabling immediate visualization of key hybrid BIT1 trace activity (see Fig. 6). Finally, the integration of HPC workflows with AI-based methods will be investigated to further enhance real-time analysis, adaptive control, and performance optimization beyond pre-exascale and exascale supercomputing capabilities.

Acknowledgments. Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Sweden, Finland, Germany, Greece, France, Slovenia, Spain, and Czech Republic under grant agreement No 101093261 (Plasma-PEPSC). The computations/data handling were/was enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Advanced Scientific Computing Research programs in the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

References

1. Chaudhury, B., et al.: Hybrid Parallelization of Particle in Cell Monte Carlo Collision (PIC-MCC) Algorithm for Simulation of Low Temperature Plasmas. In: Workshop on Software Challenges to Exascale Computing, pp. 32–53. Springer (2018)
2. Choi, J., et al.: Comparing Unified, Pinned, and Host/Device Memory Allocations for Memory-Intensive Workloads on Tegra SoC. *Concurrency and Computation: Practice and Experience* **33**(4), e6018 (2021)
3. Huebl, A., et al.: openPMD: A meta data standard for particle and mesh based data (2015). <https://doi.org/10.5281/zenodo.591699>, available at: <https://www.openPMD.org>, <https://github.com/openPMD>
4. Huebl, A., et al.: openPMD-api: C++ & Python API for Scientific I/O with openPMD (06 2018). <https://doi.org/10.14278/rodare.27>, available at: <https://github.com/openPMD/openPMD-api>
5. IPP-CAS: Bit1 OpenMP Tasks Particle Mover Parallelization. (2025), available at: https://repo.tok.ipp.cas.cz/tskhakaya/bit1/-/blob/feature/CPU-OpenMP/BIT1_c8/mover.c (updated: 2025-12-12)
6. Krishnaamy, E., et al.: OpenMP Offloading on AMD and NVIDIA GPUs: Programmability and Performance Analysis. In: Proceedings of the 2025 9th International Conference on High Performance Compilation, Computing and Communications. pp. 44–56 (2025)
7. Mehta, N., et al.: Evaluating Performance Portability of OpenMP for Snap on Nvidia, Intel, and AMD GPUs using the Roofline Methodology. In: International Workshop on Accelerator Programming Using Directives. pp. 3–24. Springer (2020)
8. Milojevic, D., Faraboschi, P., Dube, N., Roweth, D.: Future of HPC: Diversifying Heterogeneity. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 276–281. IEEE (2021)
9. Mishra, A., et al.: Benchmarking and Evaluating Unified Memory for OpenMP GPU offloading. In: Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC. pp. 1–10 (2017)

10. Neth, B., et al.: Beyond Explicit Transfers: Shared and Managed Memory in OpenMP. In: International Workshop on OpenMP. pp. 183–194. Springer (2021)
11. Noaje, G., et al.: MultiGPU computing using MPI or OpenMP. In: Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing. pp. 347–354. IEEE (2010)
12. Sewall, J., et al.: A modern memory management system for OpenMP. In: 2016 Third Workshop on Accelerator Programming Using Directives (WACCPD). pp. 25–35. IEEE (2016)
13. Tian, S., et al.: Experience Report: Writing a Portable GPU Runtime with OpenMP 5.1. In: International Workshop on OpenMP. pp. 159–169. Springer (2021)
14. Tramm, J., et al.: Toward Portable GPU Acceleration of the OpenMC Monte Carlo Particle Transport Code. In: International Conference on Physics of Reactors (PHYSOR 2022). Pittsburgh, USA (2022)
15. Tskhakaya, D., et al.: Optimization of PIC Codes by Improved Memory Management. *Journal of Computational Physics* **225**(1), 829–839 (2007)
16. Tskhakaya, D., et al.: The Particle-in-Cell Method. *Contributions to Plasma Physics* **47**(8-9), 563–594 (2007)
17. Tskhakaya, D., et al.: PIC/MC Code BIT1 for Plasma Simulations on HPC. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. pp. 476–481. IEEE (2010)
18. Vasileska, I., et al.: Modernization of the PIC codes for exascale plasma simulation. In: 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO). pp. 209–213. IEEE (2020)
19. Verboncoeur, J., et al.: Simultaneous Potential and Circuit Solution for 1D Bounded Plasma Particle Simulation Codes. *Journal of Computational Physics* **104**(2), 321–328 (1993)
20. Williams, J., et al.: Leveraging HPC Profiling and Tracing Tools to Understand the Performance of Particle-in-Cell Monte Carlo Simulations. In: European Conference on Parallel Processing. pp. 123–134. Springer (2023)
21. Williams, J., et al.: Enabling High-Throughput Parallel I/O in Particle-in-Cell Monte Carlo Simulations with OpenPMD and Darshan I/O Monitoring. In: 2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops). pp. 86–95. IEEE (2024)
22. Williams, J., et al.: Optimizing BIT1, a Particle-in-Cell Monte Carlo Code, with OpenMP/OpenACC and GPU Acceleration. In: International Conference on Computational Science. pp. 316–330. Springer (2024)
23. Williams, J., et al.: Understanding the Impact of OpenPMD on BIT1, a Particle-in-Cell Monte Carlo Code, Through Instrumentation, Monitoring, and In-Situ Analysis. In: European Conference on Parallel Processing. pp. 214–226. Springer (2024)
24. Williams, J., et al.: Accelerating Particle-in-Cell Monte Carlo Simulations with MPI, OpenMP/OpenACC and Asynchronous Multi-GPU Programming. *Journal of Computational Science* p. 102590 (2025)
25. Williams, J., et al.: Integrating High Performance In-Memory Data Streaming and In-Situ Visualization in Hybrid MPI+ OpenMP PIC MC Simulations Towards Exascale. *The International Journal of High Performance Computing Applications* (2026)
26. Wu, B., et al.: Complexity Analysis and Algorithm Design for Reorganizing Data to Minimize Non-Coalesced Memory Accesses on GPU. *ACM SIGPLAN Notices* **48**(8), 57–68 (2013)