Integrating Research into High Performance Computing Education via Particle-in-Cell Simulations

Libin Varghese, Ayushi Sharma and Bhaskar Chaudhury
Group in Computational Science and HPC, DA-IICT, Dhirubhai Ambani University, Gandhinagar, India
Email: bhaskar_chaudhury@dau.ac.in

***Abstract*—Integrating active research themes into HPC education provides an effective way to engage students with real-world scientific challenges while deepening their understanding of performance engineering. This paper presents a recent initiative that connects an advanced undergraduate HPC course with ongoing research in computational plasma physics through a research driven laboratory assignment based on the Particle-in-Cell algorithm. Building on the *CS301: High Performance Computing* course at DA-IICT, DAU and the *Let's HPC* pedagogical framework, a customized assignment was designed where students parallelized and optimized a computationally intensive PIC charge-deposition (scatter point-to-mesh interpolation) kernel. A dedicated webpage provided PIC background with visualizations, simulation animations, baseline C code, benchmark datasets, and literature resources. As part of the evaluation, the performance of the parallel codes submitted by the students was assessed across three computing systems. The overall evaluation revealed three performance based learning tiers - basic, intermediate, and advanced involving basic OpenMP parallelization to advanced optimizations, highlighting how progressively layered optimization strategies correlate with conceptual maturity. This new activity in the course improved student engagement, learning outcomes, research motivation, and understanding of scalability and performance trade offs. Overall, the approach aligns with Bloom's higher cognitive levels (Analyze, Evaluate, Create) by blending research with hands-on problem solving in HPC.**

***Index Terms*—Parallel Computing, HPC Education, Computational Science, PIC Algorithm.**

I. INTRODUCTION

Teaching High Performance Computing (HPC) effectively at the undergraduate level and preparing students for modern scientific computing requires bridging the gap between theoretical understanding of parallel algorithms and practical system level performance challenges in real research problems [1], [2]. Conventional HPC assignments relies on standard textbook problems such as matrix operations, sorting, stencil kernels to teach parallel constructs but rarely convey how HPC accelerates discovery in physics or engineering, or the scientific and engineering relevance of computational performance. At our university, the *CS301-HPC* course has been designed to cultivate "thinking in parallel" and "system's perspective" through a balance of lectures, hands-on labs, and performance analysis assignments using *Let's HPC* pedagogical framework [3]. This course is designed for advanced undergraduate students. Prerequisites include a understanding of data structures and algorithms, proficiency in C programming, shell scripting and familiarity with systems programming.The course covers parallel algorithm design, shared-memory and distributed-memory parallelism (OpenMP and MPI), performance metrics, and system level profiling. Continuous evaluation emphasizes practical programming, benchmarking, and analysis on both lab workstations and a local HPC cluster. PhD students in our research group lead the development and optimization of parallel scientific algorithms, and generally they are assigned as the primary teaching associate (TA) in the HPC course. Over several course iterations, we observed that while students mastered programming syntax and parallel patterns, their understanding of scalability bottlenecks and data movement often remained superficial. We therefore sought to (1) integrate domain-specific computational problems that demand multi-level reasoning, from algorithm to architecture, and (2) make students aware of how HPC accelerates real scientific discovery. Building on this foundation, we have recently extended the course with a research integrated final lab assignment (as mini-research project) that exposes students to the research in plasma simulation through a simplified yet realistic scientific kernel from the Particle-in-Cell (PIC) algorithm. The motivation stems from our research group's long standing work on performance modeling and hybrid parallelization of plasma simulation codes [4]–[8].

II. THE PIC ALGORITHM

The Particle-In-Cell (PIC) method, introduced in the 1950s, remains a fundamental and one of the most widely used algorithms in computational plasma physics because of its ability to capture kinetic effects by coupling particle motion with self-consistent field evolution [9]. Another variant is the PIC-MCC (Monte Carlo Collisions) technique, a kinetic model that combines particle dynamics with collision modeling for accurate Low Temperature Plasma (LTP) simulations [10], [11]. Despite its immense importance in nuclear fusion research and LTP applications, computationally intensive PIC simulations involving spatio-temporal evolution of millions to billions of particles, and in particular its *charge deposition* (particle-to-grid interpolation) step poses severe challenges for parallelization due to race conditions, memory contention, and irregular data access patterns. These challenges have driven decades of HPC research in algorithmic design and parallel programming on both CPU and GPU architectures

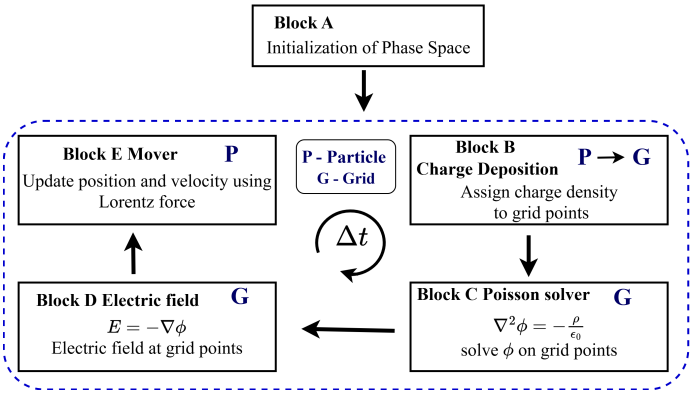

Fig. 1. Flowchart of typical iterative PIC code

to accelerate device-scale 2D and 3D PIC simulations [12]–[14]. Recent benchmarking efforts by seven research groups worldwide have focused on the development, validation, and acceleration of PIC simulations, revealing that such simulations can require tens of thousands of CPU hours and state of the art supercomputing facilities [15].

There are two primary data structures in a standard PIC code. The first is the particle data structure, which stores information on the Lagrangian grid, including the particle position (x, y, z), velocity components (v_x, v_y, v_z), and particle type (electron, ion, etc.). The second is the grid data structure, which contains the electric field, magnetic field, potential, and charge density calculated on the Eulerian grid, and mediates the collective interaction among particles. As shown in Figure 1, the process begins with a phase-space initialization (Block A), followed by particle-to-grid mapping, in which the particle charge is assigned to the four nearest grid points via interpolation (Block B) for charge density calculation. Next, the potential and electric field are calculated on the grid (Blocks C and D). Particle motion is then updated via the Lorentz force (Block E). Since particles are at continuous positions, it is necessary to back-interpolate the grid fields to the particle positions before computing the force. The particle (P) and grid (G) operations are coupled and iterated over discrete time steps Δt to evolve the plasma self-consistently.

The charge deposition (CD) module (Block B) Figure 1 remains the primary bottleneck in most parallel PIC codes, since data dependencies limit effective parallelization. In the conventional approach, private grids associated with charge density are generated for each processing core, as described in [5]. This method does not scale efficiently with an increasing number of cores. A notable contribution addressing this challenge is reported by Stanchev et al. [14]. Nevertheless, over the past decade, only limited progress has been made in further improving its performance. Given this historical and computational significance, PIC *charge deposition* (CD) offers a unique gateway for introducing undergraduate students to real-world HPC problems such as how algorithmic correctness, memory locality, synchronization, and scalability must be balanced in practice. Specifically, students are tasked with implementing a bilinear scatter-point-to-mesh interpolation algorithm which has wide applicability across domains such as computer graphics, meteorology and oceanography, data assimilation, medical imaging etc. which helps students appreciate the cross-disciplinary impact of parallel algorithms.

III. PIC Based Assignment - Charge Deposition

For simplicity, the charge deposition scheme in a full PIC pipeline is broken into a very simple problem, namely, scattered-point-to-mesh interpolation. In this method, each scattered point distributes its value to the four nearest grid points according to distance-based weights. Although simple in principle, this operation is challenging to scale in parallel. Conventional approaches require one private mesh per thread, which increases memory complexity, thereby limiting scalability. The nature of the problem is illustrated in Fig. 2 (a), which shows a 2D domain with scattered data points distributed irregularly across the region. The task is to transfer these scattered values to a regular mesh so that the underlying field can be represented continuously. The structured mesh with N_x and N_y number of grid points in each direction is defined as

$$G = \{(X_i, Y_j) \mid X_i = i\Delta x,\ Y_j = j\Delta y\},$$

where $\Delta x = X_{\max}/N_x$ and $\Delta y = Y_{\max}/N_y$ describe the grid spacing. A scattered point (x, y, f) lying inside a grid cell distributes its value f to the four surrounding grid points using bilinear weights,

$$w_{i,j} = (1 - dx)(1 - dy), \tag{1}$$
$$w_{i+1,j} = dx(1 - dy), \tag{2}$$
$$w_{i,j+1} = (1 - dx)dy, \tag{3}$$
$$w_{i+1,j+1} = dxdy, \tag{4}$$

with

$$dx = \frac{x - X_i}{\Delta x}, \qquad dy = \frac{y - Y_j}{\Delta y}.$$

The effect of these weights is shown in Fig. 2 (b), which demonstrates how a single scattered point distributes its contribution to the four corner nodes of the enclosing mesh cell. This step presents significant challenges due to race conditions from dependencies, as multiple particles write information to the same grid points. The interpolated values at the grid points are updated accordingly,

$$F(X_i, Y_j) \mathrel{+}= w_{i,j} f, \tag{5}$$
$$F(X_{i+1}, Y_j) \mathrel{+}= w_{i+1,j} f, \tag{6}$$
$$F(X_i, Y_{j+1}) \mathrel{+}= w_{i,j+1} f, \tag{7}$$
$$F(X_{i+1}, Y_{j+1}) \mathrel{+}= w_{i+1,j+1} f. \tag{8}$$

A. Lab Assignment Implementation

As part of the assignment, students were provided with a baseline C implementation (serial and parallel) of a bilinear scatter point-to-mesh interpolation kernel along with instruction manual, abstracted from the charge deposition routine of a 2D-PIC code. Input-output datasets for accuracy validation

and reproducibility were also provided Github. To support the exercise, a publicly accessible webpage GICS-EduCD was developed, introducing the PIC Charge Deposition problem statement, illustrating its physical relevance through animations of plasma evolution, and providing links to research works in PIC and parallelization of PIC codes. The instruction-manual contained the following important steps for implementation - Read Input, Initialize Structured Grid, Perform Interpolation multiple times, Check Accuracy, Perform Theoretical Analysis, Compare Execution Times, Perform Speedup Analysis and Explain Results. They conducted scalability experiments on both, laboratory systems and a four-node HPC cluster. The task required students to parallelize the code using OpenMP (and optionally MPI), apply memory aware optimizations taught during the lecture sessions of the course such as blocking, privatization, cache tuning, and benchmark multiple test cases to analyze speedup and efficiency trends. Each group, consisting of two students, documented their implementation and performance analysis in a structured report.

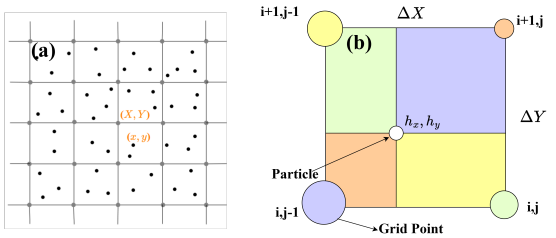

Fig. 2. Domain with scattered points distributed irregularly across the region (adapted from assignment).

B. Hardware and Benchmarking Platforms

Three computational platforms were utilised for the implementation and evaluation of the assignment. A personal laboratory workstation (Intel Core i5-9500, 6 cores, 32 GB RAM) was made available to students for development and debugging, while a four node shared HPC cluster (dual Intel Xeon E5-2640 v3, 8 cores per socket, 64 GB RAM per node) was reserved for performance evaluation and reporting. The third system was explicitly used by the TAs for evaluating the submitted codes as part of the assignment submission. The objective was to perform systematic performance measurements, uniform evaluation and evaluate the scalability of the implemented codes on a large-scale 88-core evaluation system, which was not available to students

C. Evaluation

The code submitted by the students was evaluated using several test cases. These included a small mesh that can be stored in the last level cache (LLC) along with its points, a small mesh with a large number of points that cannot fit into the LLC, a large mesh with a small number of points, and a large mesh with a large number of points. Finally, a very large mesh was evaluated. The following details provide the mesh size and the number of points: 1) Nx=250, Ny=100, points=0.9 million. 2) Nx=250, Ny=100, points=5 million. 3) Nx=500, Ny=200, points=3.6 million. 4) Nx=500, Ny=200, points=20 million. 5) Nx=1000, Ny=400, points=14 million.

The evaluation of this work was carried out on the basis of three criteria, correctness, novelty, and performance. Correctness was treated as the primary requirement, and full marks were awarded only if the code produced accurate results for the dataset, which was not shared with the students beforehand. Novelty was considered in terms of how effectively HPC concepts were incorporated into the solution. Marks were given when students introduced meaningful innovations such as cache-aware optimizations, changes to memory layouts, or other modifications to the baseline data structures, thereby demonstrating creativity beyond the provided code. Performance was assessed by executing all codes on an 88-core HPC machine, progressively varying the number of cores and recording execution times for five different test cases. For each submission, the best runtime at a given core count was documented, and this was compared against the benchmark implementation that underwent the same evaluation. Marks for performance were given only if the student's code outperformed the benchmark timing.

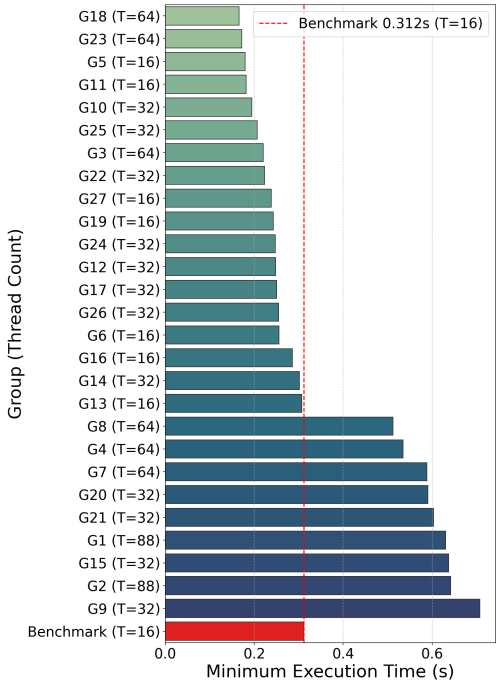

Fig. 3. The performance analysis of parallel codes submitted by different student groups, conducted by course TAs on a common benchmark computing platform.

D. Observations and futre scope

After reviewing all 26 reports, it is clear that students applied HPC concepts with varying levels of depth. As represented in Figure 3, around one third of the groups (9 groups) relied mainly on a single optimisation concept, such as adding OpenMP parallel loops or simply using thread private buffers,

TABLE I
HPC CONCEPTS ADOPTED BY STUDENT GROUPS

Category / Keywords	Groups
Basic (OpenMP, private buffers)	9 groups
Intermediate (OpenMP + cache/memory)	12 groups
Advanced (3+ strategies: SIMD, NUMA, scheduling, memory locality)	6 groups
Special technique Particle binning / blocking	13, 17, 26
Special technique Advanced scheduling	6, 17, 23
Special technique Domain decomposition	13, 23, 26

to achieve moderate speedup. About 12 groups went further, combining two core HPC strategies, for example, parallelization with OpenMP plus cache aware design (blocking/tiling) or memory aware thread-local accumulation and these groups consistently reported higher efficiency and better scaling on the HPC cluster. Finally, 6 groups demonstrated a stronger grasp of HPC ideas by integrating three or more optimization concepts, including SIMD vectorization, NUMA-aware execution, scheduling comparisons, alongside OpenMP parallelization and memory locality improvements. The top-performing submissions matched or exceeded provided benchmarks. These reports not only showed the best speedups but also reflected a deeper understanding of scalability limits, Amdahl's Law, and memory-bound behavior. Overall, the distribution suggests that while all students could apply at least one HPC concept to gain speedup, only a smaller fraction systematically layered multiple HPC strategies to maximize performance.

Among the 26 submissions, a smaller set of groups (13, 17, 26) experimented with particle binning and blocking techniques, effectively reordering data to improve cache locality and reduce scattered writes, which is a non-trivial restructuring of the algorithm. Some groups (6, 17, 23) also modified workload distribution and scheduling, testing dynamic chunking, guided scheduling, and even blocked reductions across grid tiles to reduce cache conflicts and improve balance across threads. Finally, advanced teams (13, 23, 26) extended their scope to SIMD vectorization and domain decomposition. Summary of these observation are provided in Table I. Following this assignment, students were given an additional task focused on multi-node HPC parallelization using OpenMP-MPI.

IV. Conclusion and Success Story

By embedding the ongoing research of the course instructor and his doctoral students (serving as TA) in computational plasma physics into the lab assignment design of a HPC course, this initiative created an authentic, research-driven learning experience within an undergraduate HPC course. Students experienced how to connect algorithmic theory and design with real computational physics workflows, deepening their conceptual and system level understanding through performance profiling and architecture dependent optimization. The inclusion of animations and reproducible datasets on the webpage enhanced engagement, inspiring curiosity and independent exploration. Selected groups extended their work as semester-long research projects such as SYCL based heterogeneous PIC implementations (submitted to the HiPC 2025 SRS) and ML aided accelerated plasma modeling. The assignment's design principles i.e. clarity, reproducibility, and open resources, offer a transferable framework for embedding authentic research challenges into HPC education, promoting creativity, reproducibility, and interdisciplinary thinking. Building on this experience and success, future efforts will expand toward heterogeneous programming, public repository updates, and educational research to systematically assess learning outcomes.

In summary, we observed, embedding research problems into undergraduate HPC education transforms students from coders into computational thinkers, fostering scientific curiosity and system level competence. The PIC-based final lab at DA-IICT, DAU illustrates how realistic, reproducible workloads can enrich teaching by simultaneously conveying parallel programming, performance analysis, and research methodology. Coupled with open educational resources, reproducible benchmarks, and engaging web content, such initiatives create authentic learning experiences that bridge the classroom teaching and the scientific research.

V. Acknowledgment

The authors would like to thank the Anusandhan National Research Foundation (ANRF) for the support for PIC based computational plasma physics research.

References

[1] Heroux M A, Raghavan P and Simon H D 2006 *Parallel processing for scientific computing* (SIAM)
[2] Kamil S, Oliker L, Pinar A and Shalf J 2010 *Parallel and Distributed Systems, IEEE Transactions on* **21** 188 – 202
[3] Chaudhury B, Varma A, Keswani Y, Bhatnagar Y and Parikh S 2018 *Journal of Parallel and Distributed Computing* **118** 213–232
[4] Shah H *et al.* 2017 A novel implementation of 2d3v particle-in-cell (pic) algorithm for kepler gpu architecture *2017 IEEE 24th International Conference on High Performance Computing (HiPC)* (IEEE) pp 378–387
[5] Chaudhury B *et al.* 2018 Hybrid parallelization of particle in cell monte carlo collision (pic-mcc) algorithm for simulation of low temperature plasmas *Workshop on Software Challenges to Exascale Computing* (Springer) pp 32–53
[6] Chaudhury B, Gupta A, Shah H and Bhadani S 2018 *Computer Physics Communications* **229** 20–35
[7] Delwadia K, Bhatt D, Purohit S and Chaudhury B 2022 *Concurrency and Computation: Practice and Experience* **34** e7217
[8] Varghese L, Chaudhury B, Shah M and Bandyopadhyay M 2025 *arXiv preprint arXiv:2506.21524*
[9] Birdsall C K 2002 *IEEE Transactions on plasma science* **19** 65–85
[10] Vahedi V and Surendra M 1995 *Computer Physics Communications* **87** 179–198
[11] Shah M, Chaudhury B and Bandyopadhyay M 2023 *Scientific Reports* **13** 20044
[12] Miller K *et al.* 2021 *Computer Physics Communications* **259** 107633
[13] Deluzet F, Fubiani G, Garrigues L, Guillet C and Narski J 2023 *Journal of Computational Physics* **480** 112022
[14] Stantchev G, Dorland W and Gumerov N 2008 *Journal of Parallel and Distributed Computing* **68** 1339–1349
[15] Charoy T *et al.* 2019 *Plasma Sources Science and Technology* **28** 105010