Parallelization of the Particle-In-Cell Monte Carlo Collision (PIC-MCC) Algorithm for Plasma Simulation on Intel MIC Xeon Phi Architecture

Keval Shah
DA-IICT
Gandhinagar, India
kevalds51@gmail.com

Anusha Phadnis
DA-IICT
Gandhinagar, India
anushaphadnis@gmail.com

Miral Shah
DA-IICT
Gandhinagar, India
miral.physics@gmail.com

Bhaskar Chaudhury
DA-IICT
Gandhinagar, India
bhaskar_chaudhury@daiict.ac.in

ABSTRACT

The implementation of 2D-3v (2D in space and 3D in velocity space) PIC-MCC (Particle-In-Cell Monte Carlo Collision) method involves the computational solution of Vlasov-Poisson equations. This provides the spatial and temporal evolution of the charged-particle velocity distribution functions in plasmas under the effect of self-consistent electromagnetic fields and collisions. Stringent numerical constraints associated with the PIC code makes it computationally prohibitive on CPU.

In our work, parallelization and optimization techniques have been extended to this simulation, along with a novel approach that involves developing a 'self-aware' code that triggers sorting in order to maintain cache-coherence while reducing the total sorting time during iterations. We present the effect of important numerical parameters on speed-up. Finally, we compare the scalability and performance of the parallelization and optimization strategies on Intel® Xeon™ E5-2630, Xeon Phi™ 5110p and Xeon Phi™ 7250 relative to a serial implementation on Intel® i5.

Author Keywords

HPC (High Performance Computing), Interpolation, Monte Carlo collision (MCC), particle in cell (PIC), Self-aware sorting, plasma.

INTRODUCTION

Over the past several years, much effort has been put in to develop parallel PIC codes, particularly for plasma simulations [2], [14]. In the early stages, the development was central to GPU architectures. Stantchev et al. 2008 [12], focus their work on parallelizing the charge deposition module on a GPU and have compared the modified fast algorithm to the CPU. In the brief document by Madduri et al. 2009 [9], the authors have presented the improvements arriving due to the new method of grid decomposition and efficient synchronization against the traditional optimized MPI implementation. Madduri et al. 2012 [10], extended their previous work for the next generation of multi-core architectures. This time, the authors adapted the strategy that involves handling shared target grid. Their results were very specific to the initialization factors and the architecture, and less general. Kai et al. 2013 [6] argues that grid replication for attaining straight forward parallelism in charge deposition module is not suitable for GPU architecture due to very limited shared memory. The code developed by Decyk et al. 2014 [5] has different versions specific to the architectures they are executed on. The authors had implemented two new strategies for GPU, and suggested that highly efficient openMP implementation of the PIC code on core i7 shall work similarly on upcoming MIC architectures from Intel. In recent progress, Surmin et al. 2016 [13] have suggested potential bottlenecks and hotspots for parallelization of PIC for plasma simulation on Xeon Phi architecture.

We have developed a serial PIC-MCC code from scratch and validated it with published results from [4]. The main target is to build an efficient parallel CPU-MIC architecture based PIC-MCC code with integration of some applications of [13] and [8] et al. 2016 implemented Bucket sort for PIC simulations. The extension of [8] sorting strategy into our implementation of the code has led to a significant speed-up for plasma simulations through PIC. We have devised a self-aware code that balances trade-off between sorting time and the time added due to cache miss to obtain the best performance in execution time. Due to this addition, there is a significant increase in the overall final speed-up. Since we are on a different architecture base from [6], we have implemented what the authors had rejected on a GPU due to limitations of shared memory to optimise the charge deposition module. The document [10] does favor the application of full replication strategies, but limit it's use for high density grids only.

SC'17, 2017, Denver, Colorado USA©

COMPUTATIONAL MODEL AND PIC-MCC ALGORITHM

In our work, we have considered a 2D electrostatic PIC-MCC code. "Electrostatic" signifies that the magnetic fields produced by the particles are negligible and hence, the only magnetic field that needs to be accounted for, is the one present externally [7].

Two data structures are used: the first one *particle* data structure being used for storing the positions and velocities of the particles in the phase space on a Lagrangian grid; and the *grid* data structure being used as a Euler grid to store the electric fields, magnetic fields and charge densities on grid points. This simulation is carried out by the following iterative processes of the modules:

- Charge deposition: Calculates the charge density on the grid points due to the charged particles using bilinear interpolation. The particles deposit charge on the four grid points surrounding them, in proportion to their distance from those points.
- Poisson solver: Calculates the electric field on the grid points using a sparse matrix of charge density values, by solving the Poisson equation through PARDISO library [11] to obtain the potential at those points.
- Particle mover: Uses the electric field and magnetic field values to calculate the force being exerted on the particles by the four grid points surrounding them, again by using bi-linear interpolation. The new positions and velocities of the particles are calculated using this force.
- Monte Carlo collision: Picks random particles using collision probability. The new velocities of particles due to collisions are calculated [3].

This simulation is computationally costly. For practical use, the size of the grid and the number of particles is very large, which leads to intensive memory usage. Both the data structures interact with each other in every iteration, which means that optimizing the modules keeping just one data structure in mind is not possible. The nonlinear memory access for four grid points surrounding one particle, during bilinear interpolation, adds to the computational complexity. [1]

PARALLELIZATION AND OPTIMIZATION STRATEGIES

The strategies are specific to the module they are applied to. Firstly, we implemented sorting of particles (using a modified version of bucket sort) according to their positions in the grid prior to carrying out these four operations mentioned earlier. The particles in the same cell would have the same four corner grid point, working as an advantage due to increased cache re-usability during bilinear interpolation. Next, to parallelize charge deposition, we divided particles among threads and created private grids for each thread, to prevent race conditions and enable load balancing. However, this strategy is only scalable when we increase the particle numbers so that the particles assigned to each thread remains above a given threshold in order to compensate for overhead of launching threads and producing private grids. The Particle Mover module and the MCC module do not lead to any race conditions and can be parallelized using efficient load balancing and cache optimization. Since the particles change their cell frequently, we need to sort more than once. However, sorting too often will lead to serial code executing on a many-core architecture. Hence, in our novel self-aware implementation the code maintains a threshold for a single iteration time. If my next iteration exceeds this threshold, sorting is triggered and cache coherence is restored.

OBSERVATIONS AND RESULTS

We have been successful in obtaining good efficiency for the mover module in the case of MIC machines. However, the serial implementation of the code on Intel Xeon is 3x faster than that on Intel i5, whereas the same execution takes 3x more time on the Xeon Phi 7250 with respect to Intel i5. Xeon Phi has a more primitive prefetching, slower clock frequency, in- order processing, a smaller pipeline, one instruction per two cycles etc. as compared to the multi-core Xeon or i5. Due to this gap, the speed-up relative to i5 is comparable for both varying architectures. Therefore, the key to high performance is doing the best optimization in all the three aspects: vectorization, parallelization, and memory utilization.

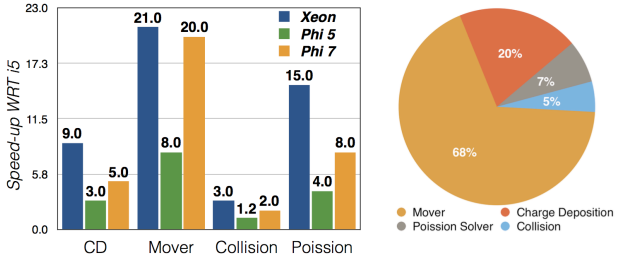

Figure 1. Module-wise Speedup calculated with respect to an equally optimized sequential code on i5.

In the charge deposition module, due to the overhead for generating private grids and irregular load balancing we observe the speed-up to saturate around 8-16 thread. Extending the sorting approach with the 'self-aware' implementation proved to be advantageous in reducing the entire run time. The speed-up on MIC architecture jumped up by at least one-forth of previous value.

CONCLUSION

We have proposed and investigated in detail an optimized parallel implementation of the PIC-MCC algorithm for plasma simulation on Intel Xeon-Phi and Xeon architecture. Several strategies for cache optimization and sorting have been implemented which improves the performance of the parallel code. Our novel self-aware sorting strategy significantly improves the performance of PIC code. For a complete simulation, we achieved a speed- up of 19.7x on Intel Xeon E5-2630 (16-cores), 2.5 on Intel Xeon Phi 5110p and 6.8 on Intel Xeon Phi 7250 compared to a serial execution on Intel i5 processor. Unlike Xeon multi-core architecture, observed speedup is below expectation in case of Xeon-Phi MIC processors due to lack of scope of vectorization of PIC algorithm and other limiting factors pertaining to serial executions on Xeon-Phi.

REFERENCES

[1] M.F. Adams, S. Ethier and N. Wichmann, “Perfomance of particle in cell methods on highly concurrent computational architectures,” Journal of Physics: Conference Series, vol. 78, p. 012001, 2007.

[2] C.K. Birdsall and A.B. Langdon, “Plasma Physics via Computer simulations,” 1991

[3] C.K. Birdsall and L. Fellow, “Particle-in-Cell charged-particle simulations, Plus Monte Carlo collision with neutral atom, PIC-MCC,” IEEE Transactions on Plasma Science, vol. 19, no. 2, pp 65-85, 1991

[4] J.P. Boeuf, B. Chaudhury and L. Garrigues, “Physics of a magnetic filter for negative ion sources. I. Collisional transport across the filter in an ideal, 1D filter,” Physics of Plasmas, vol. 19, no. 11, 2012

[5] Viktor K. Decyk and Tajendra V. Singh, “Particle-in-cell algorithms for emerging computer architectures,” Computer Physics Communications, Volume 185, Pages 708-719, 2014.

[6] Kai Germaschewski, William Fox, Stephen Abbott, Narges Ahmadi, Kristofor Maynard, Liang Wang, Hartmut Ruhl and Amitava Bhattacharjee, “The Plasma Simulation Code: A modern particle-in-cell code with load-balancing and GPU support,” arXiv preprint arXiv:1310.7866, 2013.

[7] F. Iza, S.H. Lee and J.K. Lee, “Computer modeling of low-temperature plasmas”, vol. 661, 2007

[8] A. Jocksch, F. Hariri, T.M. Tran, S. Brunner, C. Gheller and L. Villard, “A bucket sort algorithm for the particle-in-cell method on manycore architectures,” Lecture Notes in Computer Science, Springer series, vol. 9573, p. 43-52, 2016.

[9] Kamesh Madduri, Samuel Williams, Stéphane Ethier, Leonid Oliker, John Shalf, Erich Strohmaier and Katherine Yelick, “Memory-Efficient Optimization of Gyrokinetic Particle-to-Grid Interpolation for Multicore Processors,” Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Article No. 48 , 2009.

[10] Kamesh Madduri, Jimmy Su, Samuel Williams, Leonid Oliker, Stéphane Ethier and Katherine Yelick, “Optimization of parallel particle-to-grid interpolation on leading multicore platforms,” IEEE Transactions on Parallel and distributed systems, vol. 23, no. 10, October 2012.

[11] O. Schenk and K. Gartner, “Solving unsymmetrical sparse systems of linear equations using PARDISO,” Future Generation Computer Systems, vol. 20, no. 3,pp. 475-487, 2004.

[12] George Stantchev, William Dorland and Nail Gumerov, “Fast parallel particle-to-grid interpolation for plasma PIC simulations on the GPU,” Journal of Parallel and Distributed Computing Volume 68, Issue 10, Pages 1339-1349, 2008.

[13] I.A. Surmin, S.I. Bastrakov, E.S. Efimenko, A.A. Gonoskov, A.V. Korzhimanova and I.B. Meyer, “Particle-in-Cell Laser-Plasma Simulation on Xeon Phi Coprocessors,” Computer Physics Communications, Volume 202, Pages 204-210, 2016.

[14] D. Tskhakaya, K. Matyash, R. Schneider, and F. Taccogna, “The particle-in-cell method,” Contributions to Plasma Physics, vol. 47, no. 8-9, pp. 563-594, 2007.

[15] Hoefller Torsten and Belli Roberto, “Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results,” in Proceedings of the International Conference for High performance Computing, Networking, Storage and Analysis, p. 73, 2015.