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

Group in Computational Science and HPC, DA-IICT, India.

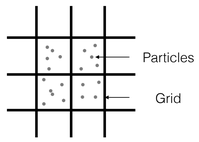
Keval Shah, Anusha Phadnis, Miral Shah and Bhaskar Chaudhury

Motivation

- The PIC-MCC model can be used for studying low temperature plasmas. This study has been subject of many experimental and theoretical works.
- Numerical modelling and simulation of this phenomenon is challenging and computationally expensive.
- Studying interactions for 1 ms of physical time requires long simulations of the order of days.
- In this research poster, the authors present strategies for optimization of the various modules of the PIC algorithm on an Intel Xeon/Xeon-Phi architecture. A novel approach that deals with 'Smart-Sorting' is introduced to enhance the overall performance of the paralleled version of the code.

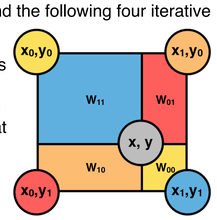
Computational Model and PIC-MCC Algorithm

- In the implementation of the model, two specific data structures are used:
  i) Particles: To store the positions and velocities of the particles and their individual traits.
  ii) Grid: To represent the grid and in turn store various parameters on the grid points like charge density, electric potential, electric field etc.



- The PIC-MCC simulation involves **A.** Initialization and the following four iterative procedures:

B. Charge deposition: The charge on each particle is used to calculate the charge density at each point in the entire grid. This is done using bilinear interpolation from each particle to its surrounding grid points, so that the appropriate proportion of charge is deposited on the grid point according to the distance of the particle from it.

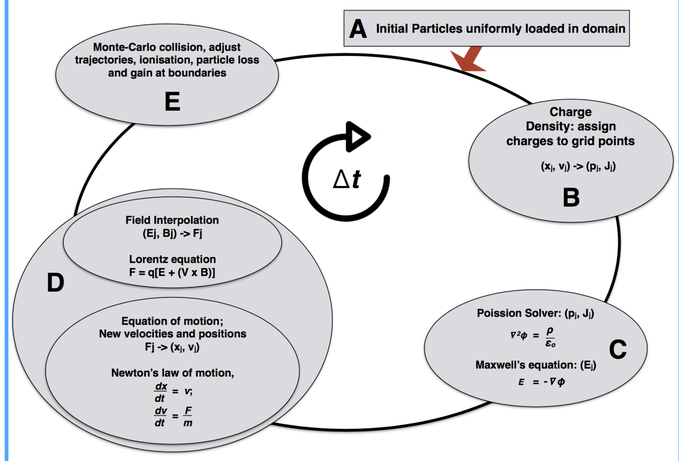


(*Figure: Bilinear Interpolation*)

C. Calculating Electric Potential and Field: The electric potential for each cell is calculated according to the positions of the particles and the charge densities of the grid points. This is done by solving the Poisson equation through Pardiso solver.

D. Particle Movement: Using the electric field values calculated in the previous step, the force on each particle is calculated and accordingly, the new positions of the particles are calculated. This is also done using bilinear interpolation, but this time the field values of the grid are interpolated onto the particle to estimate the force experienced.

E. Monte Carlo Collisions: These are applied to randomly selected particles obtained through collision probability. This method is used to select type of the collision and calculate the new particle velocities after collision.

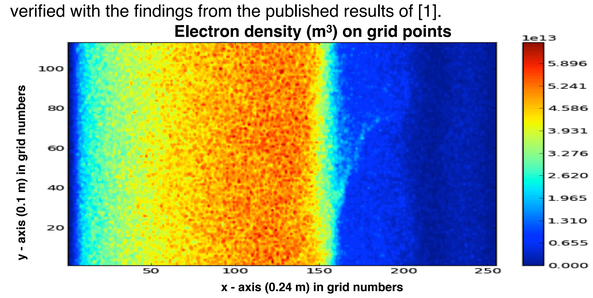


Parallelization Challenges

- Computationally intensive code: We need to iterate through each particles and grid multiple times for each iteration.
- Interpolation and non-linear memory access: particle-to-grid and grid-to-particle. There will be race condition while executing the former in parallel.
- Interactions between multiple data structures: This leads to poor cache utilization in most of modules.

Simulation Verification and Sample Outcome

- The validity of the output of all the versions of the simulation code has been verified with the findings from the published results of [1].

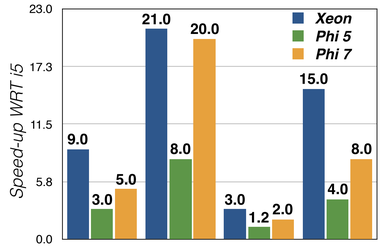


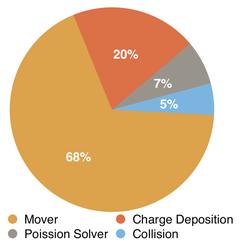
(Figure: A snapshot of electron charge density visualisation on a grid corresponding to a simulation of 0.09 ms of physical time for 4 x 10^{13} m^{-3} real particle density using 10^6 computational particles. Domain has 0.24 m x 0.1 m dimension, which is divided in 264 x 112 grid)

Serial Analysis and Optimization Strategies

- **Sequential code takes 140 seconds on Xeon E5-2630 (Haswell, tag: Xeon), 386 seconds on i5, 971 seconds on Xeon Phi 7250 (KNL, tag: Phi 7) and 3892 seconds on Xeon Phi 5110p(KNC, tag: Phi 5) to simulate 18 ns.** (grid size: 512 x 512 with 20 particles per cell).
- Module-wise analysis and profiling of the serial time measurements indicate that 88% of the total time is being spent in the Mover and CD modules.

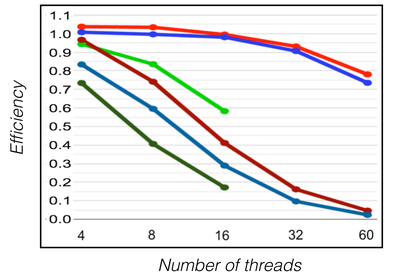
(*Figure: Module-wise Speedup is calculated WRT an equally optimized sequential code on i5*)



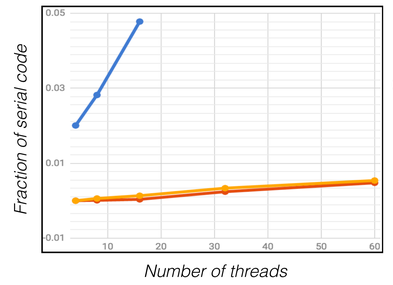


- **Leveraging from Xeon Phi:** Xeon Phi has a large L2 cache which can increase performance greatly if temporal and spatial locality is increased in a memory bound program. To extend this strategy to the PIC-MCC code, we use Bucket-sort to achieve cache coherence through sorting [2]. This shall work in O(n) time, where n is the number of particles.
- **Parallelization of CD Module:** In the naive parallelization of particle-to-grid interpolation, are frequently occurring race conditions. Hence, we use an approach where each thread is assigned a private instance of the grid which is then combined in order to obtain the global grid as suggested by [3].
- **Parallelization of Mover:** We resort to naive parallelization where each thread is assigned equal number of particles. Moreover, the particles are already stored on the main memory such that cache reusability increases. Similarly, other modules where the race condition is not prevalent we resort to load balancing with minimal critical regions if required.

Efficiency and Scalability



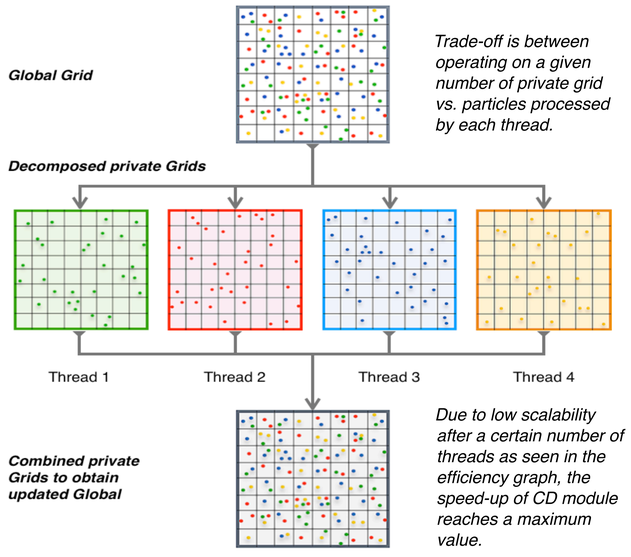
(Figure: Module-wise efficiency for 20 particles per cell on a 512 x 512 grid



(Figure: Analysis of performance of the mover module with through Karp-Flatt metric)

On MIC architecture, the efficiency of the mover module decreases with the number of threads but the corresponding Karp-Flatt metric stays stable which shows there is still scope for further optimization.

CD Parallelization Strategy



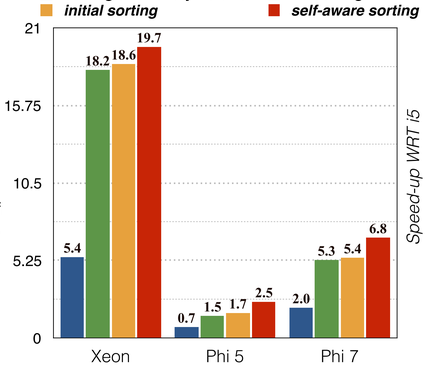
Trade-off is between operating on a given number of private grid vs. particles processed by each thread.

Due to low scalability after a certain number of threads as seen in the efficiency graph, the speed-up of CD module reaches a maximum value.

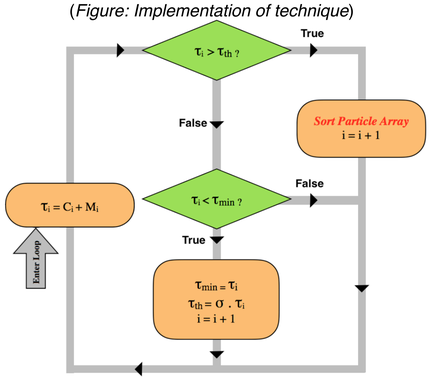
Self-Aware Sort Trigger

- We face a trade-off between the sorting time against the additional execution time due to reduced cache-hits. To achieve overall performance we present a novel mechanism that **triggers sorting when performance is degrading due to cache-miss**.
- Despite initial sorting, the arrangement of particles on the main memory will not favour cache coherence after a few iterations. This is because particles are constantly changing positions, and in turn their cells.
- For a given iteration i, τ_i (total iteration time) is the sum of C (charge deposition time) and M (mover time). τ_{th} is the threshold time, τ_{min} is the benchmark time and σ is a constant that determined by the clock speed of a single core of the CPU.

(*Figure: To compare the sorting strategies we have calculated the Speed-up of the parallel implementation WRT non-sorted serial code on i5 processor*)



(*Figure: Implementation of technique*)

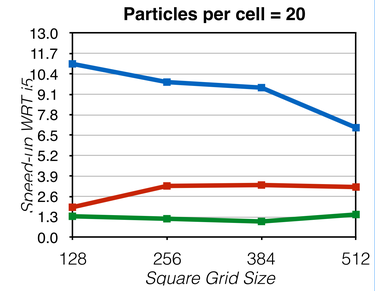


- Xeon Phi has a more primitive prefetching, slower clock frequency, in-order processing, a smaller pipeline, one instruction per two cycles etc. as compared to the multi-core Xeon or i5. Therefore, the key to high performance is doing the best optimization in all the three aspects: vectorization, parallelization, and memory utilization.

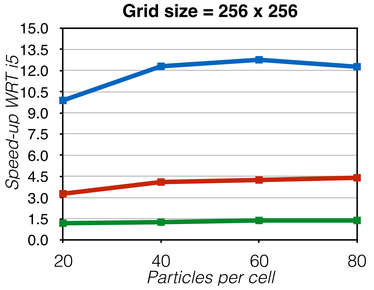
Observations by varying parameters

Xeon *Phi 5* *Phi 7*

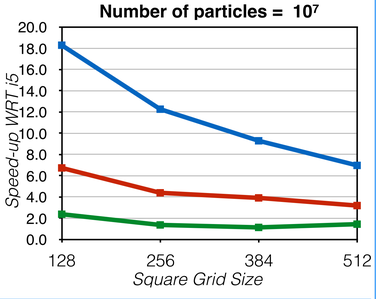
(Figure: Varying the grid size with while keeping the particle number per cell constant



(Figure: Varying the particle density in a cell for a fixed grid size



(Figure: Varying the grid size and particle density for a given fixed number of particles



Conclusion

- We have proposed and investigated in detail an optimized parallel implementation of the PIC-MCC implemented plasma simulation for Intel Xeon Phi and Xeon architecture. The parallelization is efficient on the MIC architecture. However, the lack of scope of vectorization of PIC algorithm and the huge difference in scalar serial code execution time on multi-core and many-core architecture becomes a performance limiting factor for Xeon Phi. For a complete simulation, We achieved a speed-up 19.7x on Xeon, 2.5 on Phi 5 and 6.8 on Phi 7 WRT the serial execution on i5.

Selected References

(1) J. P. Boeuf, B. Chaudhury, and L. Garrigues, "Physics of a magnetic filter for negative ion sources. I. Collision transport across the filter in an ideal, 1D filter," *Physics of Plasmas*, vol. 19, no. 11, 2012.

(2) 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.

(3) K. Madduri, J. Su, S.Williams, L. Oliker, S. Ethier and K. Yelick, "Optimization of parallel particle-to-grid interpolation on leading multicore platforms," IEEE Transactions on Parallel and distributed systems, vol. 23, no. 10, 2012.