A Parallel Particle-In-Cell Code For Heterogeneous Hardware

Grischa Jacobs[1], Thomas Weiland[2], *Fellow, IEEE* and Christian Bischof[3]

[1]Graduate School of Computational Engineering, TU Darmstadt, Darmstadt 64293, Germany
[2]Computational Electromagnetics Laboratory (TEMF) , TU Darmstadt, Darmstadt 64293, Germany
[3]Scientific Computing, TU Darmstadt, Darmstadt 64293, Germany

An evaluation for a parallel Particle-In-Cell code leveraging heterogeneous hardware is presented. Two parallelization strategies are investigated on Intel® Xeon Phi™ coprocessors. Hybrid parallelization is implemented to support optional workload offloading to coprocessors. A performance model is applied to load balance heterogeneous setups. Performance measurements of a benchmark show the quality of the proposed load balancing for both parallelization strategies.

Index Terms—**High performance computing, Space charge, Parallel algorithms, Partitioning algorithms, Computational efficiency**

I. Introduction

Typically used in computational accelerator physics, particle-in-cell (PIC) simulations calculate the movement of free charges in electromagnetic fields. Solving those physics requires a solution of the coupled Maxwell equations

$$\begin{aligned} \nabla\times \vec{E} &= -\frac{\partial \vec{B}}{\partial t}, & \nabla\cdot \vec{B} &= 0, \\ \nabla\times \vec{H} &= \frac{\partial \vec{D}}{\partial t} + \vec{J}, & \nabla\cdot \vec{D} &= \rho, \end{aligned} \tag{1}$$

and the relativistic Newton-Lorentz equation

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} &= \frac{q}{m_0 c}\left(\vec{E} + \vec{v}\times\vec{B}\right), & \frac{\partial \vec{r}}{\partial t} &= \vec{v}, \\ \vec{u} &= \gamma\frac{\vec{v}}{c}, \end{aligned} \tag{2}$$

where $\vec{u}$ is the normalized momentum and $q, m_0, \vec{r}, \vec{v}$ represent charge, rest mass, position and velocity of particles. As moving charges describe a current in eq. (1), a cyclic dependency needs to be solved for every time step. To solve the fields numerically, the Finite Integration Technique (FIT) is implemented. For more information about FIT the reader is referred to [1] for the general theory and to [4] for a setting with PIC. For the time integration of the fields a leap-frog scheme is chosen. For the integration of the eq. (2) the well known Boris scheme is used. Charge conservation is ensured by using an algorithm described in [2]. As equations 1 and 2 lead to separate computations within this approach those are referred as computational kernels.

A. Heterogenous Computing

Modern HPC systems provide diverse processor architectures, making efficient parallel computing a difficult task. Keeping the physical limitations with high clock speed rates and energy consumptions of processors in mind, the attractiveness of modern multicore processors becomes obvious. To leverage their benefits, hybrid parallelization strategies become necessary. As the variety of heterogeneous computing systems will increase in the future, this motivates investigations for realistic performance and scalability models to explore potentials for code optimizations and load balancing strategies.

II. Parallel Particle-In-Cell

To minimize the overall runtime, a suitable parallelization strategy needs to be chosen. Such a strategy may be influenced by application specific properties, e.g. different particle distributions or geometry resolutions and by hardware specific properties such as vectorization in cpu's, multicore systems and coprocessors. In this work Intel® Xeon Phi™ coprocessors are evaluated, for PIC parallelization strategies. The existing parallel PIC code facilitates distributed and shared memory parallelization using MPI and OpenMP. To decrease communication costs, non-blocking communication is implemented with MPI. Two strategies for PIC parallelization from [5], [6] and [7] are investigated. 1.) A strategy where the whole computational domain is decomposed by the number of computing nodes available. Every node calculates the DOF's for the fields and the trajectories for the particles, that are moving within the domain assigned to the node. Hence only uniform particle distributions, where every node calculates on equal number of particles, benefit from this strategy. 2.) A strategy where only the field DOF's are distributed to the nodes, whereas the particle calculations are equaly distributed independent from their position. This guarantees an equal workload for every node, with the drawback of additional communication costs. This strategy is characterized by a satisfying weak scaling behaviour, but may not be the fastet solution.

A. Performance Expectations and Load Balancing

From a performance bottleneck perspective computational kernels can be classified as memory bounded and cpu bounded. Measurements show that solving eq. (1) leads to memory (bandwidth) bounded calculations, whereas the particle solver for eq. (2) tends to be cpu bounded. Using one Intel® Xeon Phi™ coprocessor 7120P with 16GB main memory, 60 effective cores each with four hardware threads, 1.238 GHz clock speed and a peak memory bandwidth of 352 GB/s, the field kernel

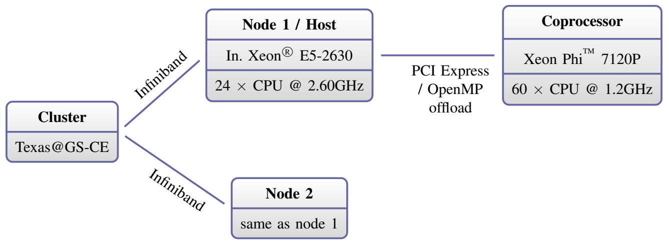

Fig. 1. Simplified setup of the Texas Cluster at the Graduate School of Computational Engineering. The Intel® Xeon Phi™card is connected to the host node over PCI Express. Load balance is measured for two nodes where only one node holds a coprocessor.

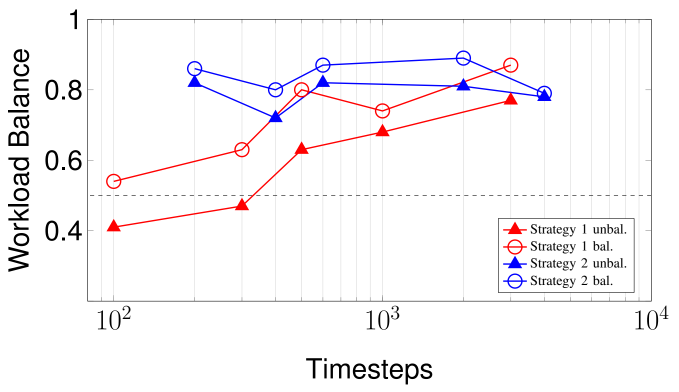

Fig. 2. Measurements of two parallelization strategies executed with the setup shown in fig.1. Balancing workload with eq.3 improved the execution of strategy 1. by up to 18%. Workload balance of strategy 2 is improved by 12%.

(eq. 1) can make use of the high memory bandwidth and the particle kernel can leverage the highly concurrent SIMD nature of the particle solver, using up to 240 hardware threads available on the card. As the computations of the particle solver take up to 80% of one time step, particle integrations and current density calculations are offloaded to the Xeon Phi™ coprocessor. By counting all floating point operations $\#op_j$ and the number of network bytes exchanged $\#x_j$ form kernel j and taking the communication bandwidth b^k for node k into account, the performance is estimated with

$$l_k \leq \frac{\sum_{j=0}^{Kernel} \#op_j}{\sum_{j=0}^{Kernel} \frac{\#op_j}{l_j^k} + \sum_{j=0}^{Kernel} \frac{\#x_j}{b^k}}. \tag{3}$$

Using roofline models [3] for all hardware architectures, the performance l_j^k of all kernels can be estimated by evaluating the operational intensity for each kernel. The operational intensity is defined as the ratio of the number of floating point operations ($\#op$), to the number of transferred bytes from the last level cache to the main memory. The performance may be measured in Flops or e.g. particle integrations per second. Using this performance model, a static load balancing algorithm using recursive orthogonal besectioning is applied, improving the computational balance in the setup shown in fig. (1). The setup consists of two nodes where only one is supported with a coprocessor. Using our performance model both nodes should get the same execution time per time step.

III. Results

The performance model proposed is being investigated by simulating electron bunches with free movement in a tube with a constant electric field and perfect electric conducting boundaries as a benchmark problem. Figure 2 shows the workload balance for each time step for the load balanced and unbalanced case, evaluated for both parallelization strategies. Workload balance is defined as the timespan between the first MPI process being finished, waiting for the last process. It is normalized in such a way, that a value 1 refers to a perfect balanced execution and value 0.5 to the case where the slowest MPI process needs twice the execution time of the fastest MPI process, until he is finished. Strategy 1 improves imbalance over time as the particle bunches are getting equally distributed in the computational domain. In this strategy the process with the most particles at initialization was supported with the coprocessor. The imbalance of strategy 2 is mainly caused by the coprocessor.

IV. Conclusion

Two parallelization strategies for parallel Particle-In-Cell codes have been evaluated concerning the load balance for heterogeneous hardwarein a setup shown in fig. (1), including the Intel® Xeon Phi™coprocessor. The imbalance for parallelization strategy 1 and 2 could be reduced by up to 18% and 12%, respectively. Further work will focus on data distribution an parallel efficiency achieved with the coprocessor.

Acknowledgment

The work of Grischa Jacobs is supported by the ’Excellence Initiative’ of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universitt Darmstadt.

References

[1] Thomas Weiland, *A Discretization Method for the Solution of Maxwell’s Equations for Six-Component Fields*, Electronics and Communication, Vol.31, p116-121, 1977.
[2] John Villasenor and Oscar Buneman, *Rigorous Charge Conservation for Local Electromagnetic Field Solvers*, Computer Physics Communications, 69:306-316, 1992.
[3] Samuel Williams, Andrew Waterman, and David Patterson, *Roofline: an insightful visual performance model for multicore architectures*, Commun. ACM 52, 65-76, 2009.
[4] U. Becker, T. Weiland, *Particle-in-Cell Simulations within the FI-Method*, Surveys on Mathematics in Industry, Vol.8, No.3-4, pp.233-242, 1999.
[5] F. Wolfheimer, E. Gjonaj, T. Weiland: *Parallel Particle-In-Cell (PIC) Codes.* Proceedings of the 9th International Computational Accelerator Physics Conference (ICAP 2006), 2006.
[6] F. Wolfheimer, E. Gjonaj, T. Weiland: *A Parallel 3D Particle In Cell (PIC) with Dynamic Load Balancing.* Nuclear Instruments and Methods in Physics Research (NIM), Vol. 558, pp. 202-204, 2006
[7] E. A. Carmona, L. J. Chandler, *On parallel PIC versatility and the structure of parallel PIC approaches*, Concurrency: Practice and Experience, Vol.9(12), pp.1377-1405, 1997.