Extending a 3D Parallel Particle-In-Cell Code For Heterogeneous Hardware

***Grischa Jacobs[1], Thomas Weiland[2] 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

*Presenting author: jacobs@gsc.tu-darmstadt.de

Abstract

An evaluation for a parallel Particle-In-Cell code leveraging heterogeneous hardware is presented. Hybrid parallelization is implemented to support optional workload offloading to 40 Intel® Xeon Phi™ coprocessors. A performance model is applied to load balance the particle data for this heterogeneous setup. Performance measurements of a benchmark show the speedups for the balanced and unbalanced cases and the execution without the coprocessor. The code computes particle-field interactions in the time domain, typically used in plasma or particle physics. A multi beam gun is chosen as a benchmark. The gun uses an electrostatic field to accelerate the particles and a magnetostatic field, generated by a current driven coil to focus the particle beam. Calculated results are compared with CST Particle Studio [11]. For solving the electrodynamic and electrostatic fields, described by the coupled MAXWELL equations, a 3D solver has been implemented, facilitating the Finite Integration Technique (FIT) [1].
Keywords: High Performance Computing, Intel Xeon Phi, Particle-In-Cell

Introduction

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. Typically used in computational accelerator and plasma 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 equations 1 and 2 lead to separate computations within this approach those are referred as computational kernels.

3D Particle-In-Cell Simulation

As moving charges describe a current in eq. (1), a cyclic dependency needs to be solved for every time step. This is shown in figure 1. To solve the fields numerically, the Finite Integration Technique (FIT) [1] is implemented. For further 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 eq. (2) the well known Boris scheme is used. Charge conservation is ensured by using an algorithm described in [5]. The following subchapters will provide a coarse overview. The basic kernels of the PIC method are: (1) Calculating the dynamic electromagnetic fields in time domain with eq. 1 (”field” kernel). (2) Gather all static and dynamic field values at particle positions (”gather fields” kernel). (3) Integrate particle trajectories for one time step (”push” kernel). (4) Calculate currents introduced by the charge movement and scatter those (”scatter current” kernel). The costs of these kernels depend on the problem setting in terms of particles per cell, particle distribution and the sizes of the computational mesh.

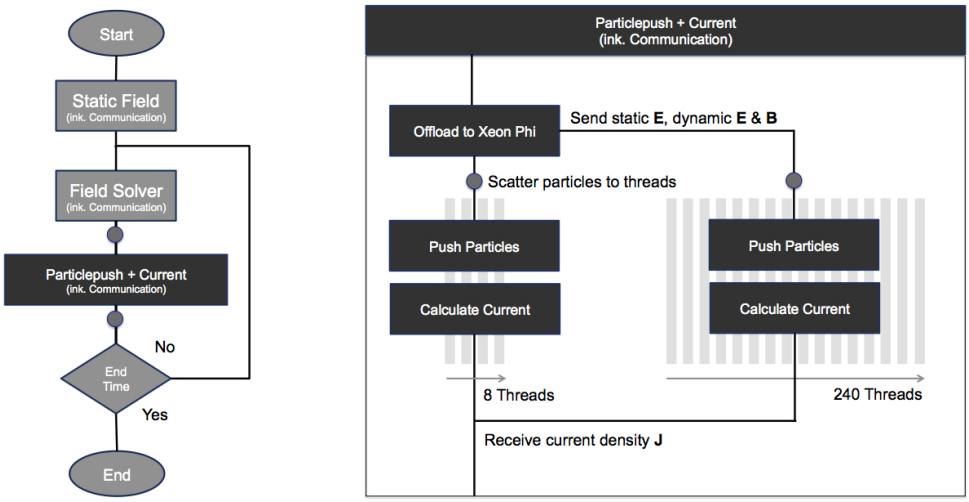

Figure 1: The left figure shows the sequential steps of the Particle-In-Cell algorithm. Each loop calculates one physical time step. The right figure explains the offloading to the Intel® Xeon Phi™ accelerator card. Only the particle trajectory and current calculations are offloaded to the card.

Field Solver

The field solver facilitates FIT. The FI discretization scheme is related to the well known Yee scheme and based on a dual grid-doublet $\{G, \widetilde{G}\}$, which decomposes the computation domain into two sets of dual cells. Integral quantities $\widehat{\mathbf{q}}$, $\widehat{\mathbf{e}}$ and $\widehat{\widehat{\mathbf{b}}}$ are defined on the grid G, corresponding to the total charge in the cell volumes, to the electric voltage along the cell edges and to the magnetic induction flux on the cell facets, respectively. Electric voltage $\widehat{\mathbf{e}}$ is defined by

$$\int_{L_v(i,j,k)} \vec{\mathbf{E}}(\vec{\mathbf{r}}, t) \cdot \vec{\mathbf{e}}_v \, dv \;=\; \widehat{\mathbf{e}}_v(i,j,k). \qquad v \in \{x, y, z\} \tag{3}$$

The integral quantities $\widehat{\widehat{\mathbf{j}}}$, $\widehat{\widehat{\mathbf{d}}}$ and $\widehat{\mathbf{h}}$ are the vectors of charge current, electric displacement flux and magnetic voltage defined on the facets and edges of the dual grid $\widetilde{G}$. Fig. 3 illustrates the allocation of the electric voltage in the case of rectangular dual grids G and $\widetilde{G}$. Using these integral quantities, Maxwell's equations in discrete form, the so-called Maxwell-Grid-Equations are obtained:

$$\mathbf{C}\,\widehat{\mathbf{e}} \;=\; -\frac{d}{dt}\widehat{\widehat{\mathbf{b}}}, \tag{4a}$$

$$\widetilde{\mathbf{C}}\,\widehat{\mathbf{h}} \;=\; \widehat{\widehat{\mathbf{j}}} \;+\; \frac{d}{dt}\widehat{\widehat{\mathbf{b}}}, \tag{4b}$$

$$\widetilde{\mathbf{S}}\,\widehat{\widehat{\mathbf{d}}} \;=\; \mathbf{q}. \tag{4c}$$

The support matrix operators $\{C, S\}$ and $\{\widetilde{C}, \widetilde{S}\}$ defined on G and $\widetilde{G}$ are discrete mappings of the differential "curl" and "div". The operators $C, S, \widetilde{C}$ and S fulfill the identities $SC = C\widetilde{S}^T = 0$ and $\widetilde{S}\widetilde{C} = \widetilde{C}S^T$. This corresponds to the continuum relations $div\ curl = 0$ and $curl\ grad = 0$. The discretization approximation enters FIT through the constitutive material equations

$$\widehat{\widehat{\mathbf{d}}} \;=\; \mathbf{M}_{\epsilon}\,\widehat{\mathbf{e}} \quad , \quad \widehat{\mathbf{h}} \;=\; \mathbf{M}_{\mu^{-1}}\,\widehat{\widehat{\mathbf{b}}} \quad \text{and} \quad \widehat{\widehat{\mathbf{j}}} \;=\; \mathbf{M}_{\sigma}\,\widehat{\mathbf{e}}. \tag{5}$$

Particle Solver

The particle solver models groups of particles "macroparticles" using a ballistic approach by solving eq. (2). Solving it requires a three step process: 1. interpolating $\mathbf{E}$ and $\mathbf{B}$ fields in the center of the macroparticle within one cell by choosing an interpolation with at least order one. 2. Solving eq. (2) with the a method proposed by Boris [5] using the interpolated field values of step (1). Note that solving this equation is not trivial as of the term $\vec{v} \times \vec{B}$. 3. Calculating current densities induces by the particle movement, with the equations 6 for the 2D case as shown in figure 2. Bunemann et. al. [2] describe how to solve this with rigorous charge conservation.

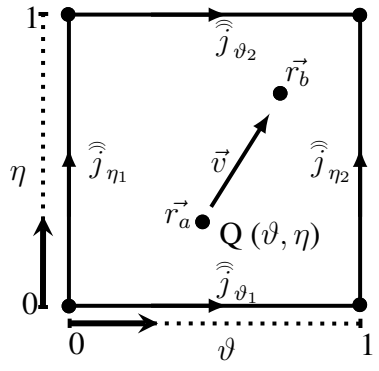

Figure 2: Macroparticle moving in a 2D cell. As the particle is moving it creates currents on the edges of the cell.

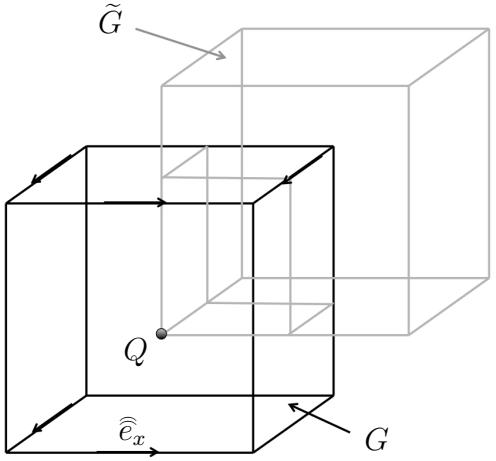

Figure 3: Illustrates the allocation of fluxes and voltages in the case of rectangular dual grids G and $\widetilde{G}$.

The currents in figure 2 for the 2D case are calculated by:

$$\hat{\tilde{j}}_{\vartheta_1} = Q \cdot \frac{\vartheta_2 - \vartheta_1}{\triangle t} \cdot \left(1 - \frac{\eta_1 + \eta_2}{2}\right) \qquad \hat{\tilde{j}}_{\vartheta_2} = Q \cdot \frac{\vartheta_2 - \vartheta_1}{\triangle t} \cdot \frac{\eta_1 + \eta_2}{2} \tag{6a}$$

$$\hat{\tilde{j}}_{\eta_1} = Q \cdot \frac{\eta_2 - \eta_1}{\triangle t} \cdot \left(1 - \frac{\vartheta_1 + \vartheta_2}{2}\right) \qquad \hat{\tilde{j}}_{\eta_2} = Q \cdot \frac{\eta_2 - \eta_1}{\triangle t} \cdot \frac{\vartheta_1 + \vartheta_2}{2} \tag{6b}$$

Integration in the Time-Domain

The time domain equivalent to the FI Method is the well known FDTD scheme of leapfrog integration. Applied to the time dependent equations (4) this procedure is restricted by the Courant–Friedrichs–Lewy stability criterion on the time step length

$$\triangle t \leq \triangle t_{maxCFL}. \tag{7}$$

For the time integration an explicit forward time difference scheme is used. The corresponding update relation is

$$\hat{\mathbf{h}}^{(m+1)} = \hat{\mathbf{h}}^{(m)} - \triangle t\, \mathbf{M}_{\mu^{-1}}\, \tilde{\mathbf{C}}\, \hat{\mathbf{e}}^{(m+\frac{1}{2})} + O(\triangle t^2)\,, \tag{8a}$$

$$\hat{\mathbf{e}}^{(m+\frac{3}{2})} = \hat{\mathbf{e}}^{(m+\frac{1}{2})} + \triangle t\, \mathbf{M}_{\epsilon}{}^{-1} \left(\tilde{\mathbf{C}}\, \hat{\mathbf{h}}^{(m+1)} - \hat{\tilde{\mathbf{j}}}^{(m+1)}\right) + O(\triangle t^2). \tag{8b}$$

Integrating the particle trajectories is straight forward. Replacing the differential operator in eq. (2) with a central differential quotient will lead to a numeric representation. Again a leapfrog scheme for the integration of $\vec{r}$ and $\vec{u}$ is used.

$$\mathbf{u}^{n+1/2} = \mathbf{u}^{n-1/2} + \triangle t \cdot \frac{Q}{m_0 \cdot c} \cdot (\mathbf{E}^n + \mathbf{v}^n \times \mathbf{B}^n)\,, \tag{9a}$$

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \triangle t \cdot \frac{c}{\gamma^{n+1/2}} \cdot \mathbf{u}^{n+1/2} \tag{9b}$$

Eq. (9a) cannot be solved explicit as of $\vec{v}$. Changes in the velocity $\vec{v}$ will also effect the normalized momentum $\vec{u}$. For this reason eq. (9a) is split into three steps as suggested by Boris [5]. First the momentum gets calculated over half a time step by

$$\mathbf{u}_- = \mathbf{u}^n + \frac{\triangle t}{2} \cdot \frac{Q}{m_0 \cdot c} \cdot \mathbf{E}^n, \tag{10}$$

second a rotation is calculated according to the LORENTZ force over a full time step

$$\mathbf{u}^* = \mathbf{u}_- + \mathbf{u}_- \times \mathbf{T}, \tag{11a}$$

$$\mathbf{u}_+ = \mathbf{u}_- + \mathbf{u}^* \times \frac{2 \cdot \mathbf{T}}{1 + |\mathbf{T}|^2}, \tag{11b}$$

$$\mathbf{T} = \triangle t \cdot \frac{Q \cdot \mathbf{B}^n}{2 \cdot m_0 \cdot \sqrt{1 + |\mathbf{u}_-|^2}}. \tag{11c}$$

Finally the momentum gets calculated over the lasting half time step

$$\mathbf{u}^{n+1} = \mathbf{u}_+ + \frac{\triangle t}{2} \cdot \frac{Q}{m_0 \cdot c} \cdot \mathbf{E}^n. \tag{12}$$

Parallel Particle-In-Cell

In this work Intel® Xeon Phi™ coprocessors are evaluated for the parallelization of the PIC method. The existing parallel PIC code facilitates distributed and shared memory parallelization using MPI and OpenMP. The code is build on top oh the PETSC framework [10] supporting sophisticated MPI data structures to be used. The Intel® Xeon Phi™ card has been chosen instead of a GPU card, as the the existing OpenMP code may be easy to offload.

Strategies for Parallelization

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. Two strategies for distributed memory PIC parallelization have been investigated in the context of accelerator physics (e.g. beam simulation) [6], [7], [8] and [9]:
1.) 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 an equal number of particles, benefit from this strategy.
2.) Only the field DOF's are spatially distributed to the nodes, whereas the particle calculations are equally 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 fastest solution.

Shared Memory Parallelization / Offloading to Coprocessors

Using one Intel® Xeon Phi™ coprocessor with 60 effective cores, each with four hardware threads, the "field" kernel can make use of the (theoretical) high memory bandwidth and the "push" and "current" kernels can leverage the highly concurrent SIMD nature of the particle calculations using up to 240 hardware threads available on the card. The coprocessor can be used in two different modes. A "native mode" where the executable gets compiled to run on the coprocessor as a standalone MPI process and a "offload mode" that enables offloading selected kernels that benefit from the multicore architecture. Due to the memory limitation of 8 GB main memory for the smaller Intel® Xeon Phi™ 5110P card and the fact that not every computational kernel can benefit from the shared memory scalability of the coprocessor (e.g field solver), the "native mode" is not evaluated in this work. In the benchmark used for evaluation, the computations of the particle solver takes up to 80% of the overall time to compute one physical time step, suggesting that particle integrations and current density calculations are offloaded to the Xeon Phi™ coprocessor, whereas the field computations and all MPI communications are exclusively performed on the host. As the communication to the coprocessor over PCIe is one bottleneck, this work evaluates only the second parallelization strategy mentioned above making benefit of the fact that particle data will stay on the coprocessor across all time step calculations, thus PCIe traffic is reduced. In order to calculate on the coprocessor in parallel with the host, an asynchronous offload with OpenMP 4.0 LEO is implemented. This way only one thread of the host executes the offloading procedure to the coprocessor, whereas the remaining $n-1$ threads of the host are facilitated to compute the particle movement and current calculations.

Performance

In this work performance is defined as "time-to-solution". From a performance bottleneck perspective computational kernels can be classified as memory bounded and CPU bounded.

Evaluating the code showed, that both the ”field” kernel and the ”push” kernel tend to be to memory (bandwidth) bounded.

Performance Modeling

In some cases it might be inefficient to offload computations to a coprocessor, as the time for sending and receiving data from the coprocessor makes the speedup for calculation neglectable. Therefore a performance model, on the basis of a model proposed by Kredel et.al. [3], is introduced predicting performance achieved by the PIC code. Further performance prediction creates space for robust load balance strategies. By counting all floating point operations $\#op_j$ as well as the number of network bytes exchanged $\#x_j$ by kernel j, taking the communication bandwidth b^k into account and measuring floating point operation per second l_j^k for each node k the performance is estimated by

$$t_k \leq \sum_{j=0}^{Kernel} \frac{\#op_j}{l_j^k} + \sum_{j=0}^{Kernel} \frac{\#x_j}{b^k}. \tag{13}$$

The parameters $\#op_j$ and $\#x_j$ describe the software performance where as b^k and l_j^k are hardware representatives. Software parameters can be derived from the code by hand or with measurements by sweeping all parameters (e.g. mesh and particle size). As it is intended to model a system with one host and one Intel® Xeon Phi™ coprocessor the effective bandwidth for the communication between those needs to be measured as shown in figure 7. For large data sizes ($>$ 30MB) the bandwidth for sending and receiving may differ by up to one dimension. As the offloaded kernels are running in parallel on the host and on the coprocessor, the performance is estimated by

$$t_k \leq \max \left[\frac{w_1 op_1}{l_1^1}, \frac{w_2 * op_1}{l_1^2} + \frac{w_2 * x_{send}}{b_{send}^2} + \frac{w_2 * x_{recv}}{b_{recv}^2} \right] \tag{14}$$

where the *max* operator describes the parallel execution, as the slower system will degrade the performance. The sum of the performance of the ”push” and ”current” kernels, executed by the host, is denoted by $l_{1,2}^1$ and the operation count by $op_{1,2}$. For the coprocessor those are denoted by $l_{1,2}^2$ and $op_{1,2}$, respectively. As the bandwidth for sending and receiving data from the coprocessor differs, two terms modeling the data transfer are added. As it is intended to perform load balancing, two scalar weights are added, w_1 and w_2. The sum of both weights must be equal to one.

Optimization for Accelerator

In order to get the code running efficiently on the coprocessor some optimizations are carried out. When calculating current density values by the ”current scatter” kernel after the movement of the particles, those values are stored in a hash map where the key is the face index in the computational mesh, of the face the particle has crossed. This is done by every thread for all particles those threads are responsible for and merged in the end. Reading and writing to 240 hash maps on the coprocessor, each controlled by one thread, is decreasing the performance in an order of one dimension compared to the performance of the host with 16 threads. Using the concurrent unordered map provided by Intel Thread Building Blocks library [13] the performance is improved to be competitive with the host’s performance. The data of particles and fields transmitted from the host to the coprocessor and current data transmitted from the coprocessor to the host is communicated over PCIe 2, having a peak bandwidth of 6 GB/s as shown in figure 7. To achieve high communication speeds the environment variable MIC_USE_2M_BUFFERS of the coprocessor is set to 2 MB.

Architectural Testbed

The Lichtenberg cluster [12] located at Technische Universität Darmstadt, has 647 computing nodes available for various applications and provides an accelerator section that with 24 nodes, configured to be used with 48 Intel® Xeon Phi™ coprocessors (two cards each node). Every host node has two sockets with one Intel® Xeon® Processor E5-2670 having 8 cores, hyperthreading disabled and 32 GByte main memory. Each core runs on 2.6 GHz. Nodes in this section are connected with 1x FDR-10 InfiniBand. Two nodes provide Intel® Xeon Phi™ 7120P coprocessors whereas the remaining 22 nodes provide Intel® Xeon Phi™ 5110P coprocessors. The Intel® Xeon Phi™ coprocessor 5110P has 8GB main memory, 59 effective cores each with four hardware threads, 1.05 GHz clock speed and a theoretical peak memory bandwidth of 320 GB/s. This system is used to evaluate the speedup achieved by PIC code when incorporating Intel® Xeon Phi™ coprocessors.

Results

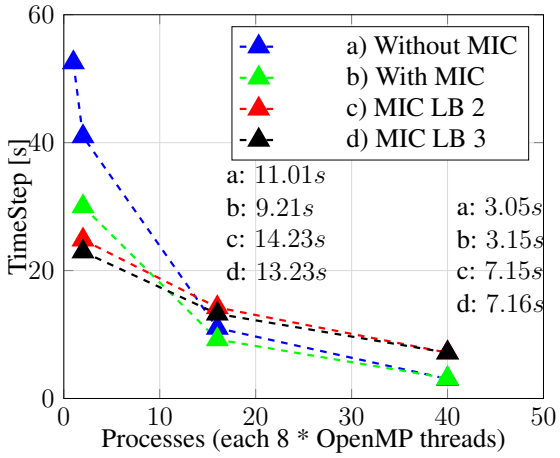

Figure 4: Execution times to compute one physical time step. The overall problem size is fixed (10^6 DOF and 10^7 particle), whereas the number of MPI processes is increased. Each computing node executes two MPI processes, as each MPI process has one Intel® Xeon Phi™ card. The blue line shows measured times without the support of coprocessors. Green line shows measured times with support of the coprocessors. Red and black line show execution times with particle load balancing, using the ratios of 1:2 and 1:3 between the host and the coprocessor.

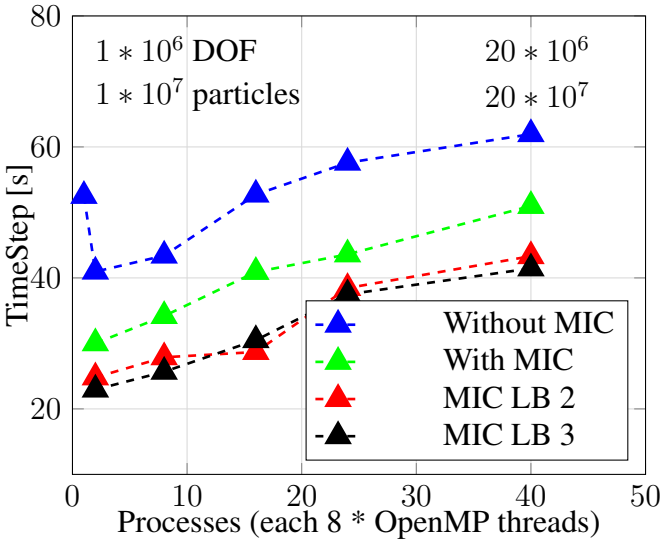

Figure 5: Execution times to compute one time step. The problem size per MPI process is constant (10^6 DOF and 10^7 particle), whereas the number of MPI processes is increased. Each computing node executes two MPI processes. The blue line shows measured times without the support of coprocessors. Green line shows measured times with support of the coprocessors. Red and black line show execution times with particle load balancing, using the ratios of 1:2 and 1:3 between the host and the coprocessor.

As a benchmark problem, a multi beam particle source of a particle accelerator is chosen. The benchmark is provided by CST [11]. It simulates multiple electron beams with free movement in a constant electric and magnetic field and perfect electric conducting boundaries. The magnetic field is generated by a current driven coil to focus the beam. The problem size is designed

to meet the main memory limitations of one computing node in the accelerator supported section of the Lichtenberg cluster, calculating 10 million DOF for the mesh and 100 million particles on one node. Scaling up to 20 nodes a problem with 200 million DOF and 2 billion particles is solved.

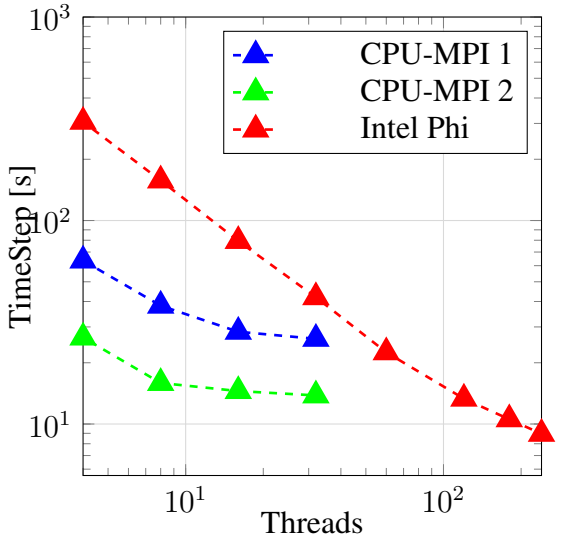

Figure 6: Execution time for "push" and "current" kernels calculating $5 * 10^6$ particles with a varying number of threads. Red line shows the execution times of the offloaded kernels on Intel® Xeon Phi™ 5110P. A speedup of 2-3 against the hosts execution is measured. Blue line shows the execution on the host with one MPI process and varying threads. Green line shows the execution on the host with two MPI processes and varying threads.

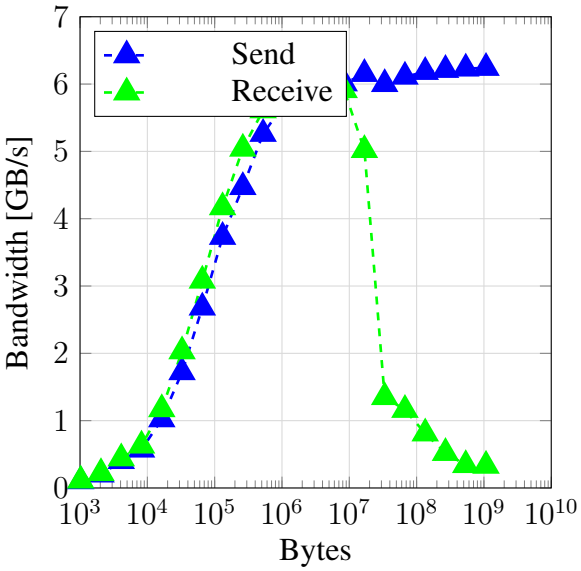

Figure 7: Showing measured bandwidth values when communicating with an Intel® Xeon Phi™ 5110P coprocessor. The blue line shows performance for received data, whereas the green line shows the bandwidth measured when data was sent. Both measurements can be fittet with spline functions.

Speedup and Execution Time

Three studies are carried out to evaluate the speedup achieved with the coprocessors. A study evaluating strong scalability shown in figure 4, where the problem size is constant and the number of parallel units is increased, a weak scalability study, where the problem size increases linearly with the number of parallel units, shown in figure 5 and a study measuring the shared memory scalability shown in figure 6. All measurements shown are executed with two MPI processes per node, each running 8 threads in parallel. This way each MPI process makes use of one Intel® Xeon Phi™ coprocessor, as one node holds two cards. Also the performance for one MPI process running on one node with 16 threads is measured, but only for the setup where no coprocessors were used. Each figure shows time values in seconds measured when executing one physical time step. This physical time step is calculated by the sequential execution of each computational kernel in parallel by all MPI processes. The values are mean values of ten physical time steps measured. Four setups are configured and shown in both figures 4 and 5. The blue line ("w/o MIC") shows measurements were the code is executed without a coprocessor, whereas the green line ("w MIC") incorporates the coprocessors. The red and black

lines are setups, where the number of particles calculated by the host and the ones calculated by the coprocessor are load balanced by the ratio 1:2 and 1:3, respectively. From figure 4 one can infer that using two coprocessors reduces the time to compute one physical time step by up to 56% compared to a host only execution with one MPI process, and a reduction of 23% is achieved compared to a host only execution with two MPI processes. One can also derive that the support of the coprocessor gets insignificant as the number of particles per MPI process gets to small, as measured with 16 and 40 MPI processes in figure 4. Having a constant problem size on each node, as shown in figure 5, the benefit of the coprocessors becomes noticeable, as a mean runtime reduction of 39% for the load balanced setup is measured. Figure 6 shows the scalability of the ”push” and ”current” kernels, when increasing the number of threads, having the problem size kept fixed. The blue line plots a setup where the code is executed without the coprocessor using one MPI process and a varying number of threads, whereas the green line shows measurements for a setup with two MPI processes executed in parallel on one node. The red line plots the time the coprocessor (Intel® Xeon Phi™ 5110P) needs to execute both kernels. Calculating the same number of particles on both accelerator cards, the performance difference between the smaller and the bigger accelerator card is negligible. In figure 6 it is also shown that calculating the same number of particles with the same number of threads on the host and on the coprocessor, the host system exceeds the coprocessors. This may be caused by the different core cache architectures, bigger caches sizes and the existence of L3 caches on the host. As the coprocessor can scale up to 240 threads (the Intel® Xeon Phi™ 5110P only to 236 threads) a speedup between 2 and 3 can be measured compared to the host with 16 threads running. The speedup of the host saturates at 8 threads.

According to Intel, the pinning of threads onto the cores can have a major performance impact. Table 1 shows measurements for various thread affinity setups. Scatter, balanced and compact affinity settings are evaluated when 59, 118, 177 and 236 threads are used. Balanced and scattered thread distributions lead to similar execution times, whereas compact distribution tends to be slower. Using 236 threads all affinity settings show similar results.

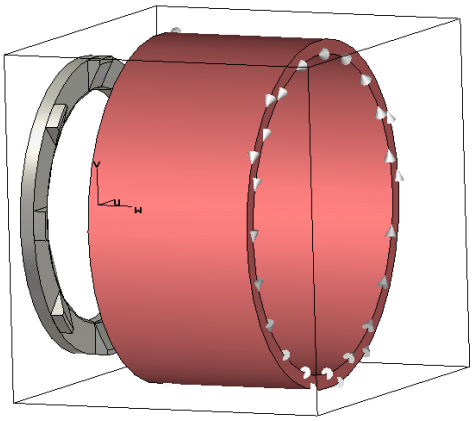

Figure 8: CAD model of the multibeam particle source created in CST Particle Studio [11]. The red structure is a current driven coil generating a magnetostatic field. The ring structure has 8 particle sources.

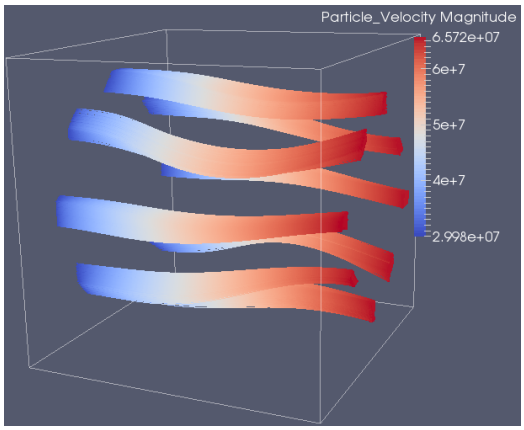

Figure 9: Particle-In-Cell multi beam benchmark simulating electron beams with free movement in a constant magnetic field and perfect electric conducting boundaries. The benchmark is calculating 10 million DOF for the mesh and 100 million particles on one node.

Table 1: Thread affinity on xeon Phi 5110P

KMPAFFINITY	59	118	177	236
granularity=fine, scatter	22.74s	12.75s	9.85s	8.29 s
granularity=fine, balanced	22.82s	12.81s	9.89s	8.16s
granularity=fine, compact	29.40s	15.06s	10.39s	8.16s

Conclusion

Performance measurements are presented evaluating "time-to-solution" with 40 Intel® Xeon Phi™coprocessors incorporated executing a parallel Particle-In-Cell code. It is shown that the on-node performance is improved by 56% for realistic problem sizes, when controlling the balance of data that is computed on the host and on the coprocessor with a load balancing strategy. Therefore an analytical performance model is used and evaluated for the host and 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 Universität 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] Heinz Kredel, Sabine Richling, Jan Philipp Kruse, Erich Strohmaier, Hans-Günther Kruse, A simple concept for the performance analysis of cluster-computing, *Supercomputing*, 165-180, 2013.

[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] Boris, J.P., Relativistic plasma simulation-optimization of a hybrid code, Proceeding of Fourth Conference on Numerical Simulations of Plasmas, November 1970

[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.

[8] Ji Qiang, Xiaoye Li, Particle-field decomposition and domain decomposition in parallel particle-in-cell beam dynamics simulation, *Computer Physics Communications*, Vol. 181, Issue 12, 2010.

[9] A. C. Elster, Parallelization issues and particle-in-cell codes, PhD Book, 1994.

[10] S. Balay et. al., Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, *Modern Software Tools in Scientific Computing*, Pages 163-202, 1997.

[11] "CST Software", https://www.cst.com/

[12] ”Lichtenberg cluster”, http://www.hhlr.tu-darmstadt.de/hhlr/index.en.jsp

[13] ”Intel Thread Building Block”, https://www.threadingbuildingblocks.org/