Particle-in-Cell algorithms on DEEP: The iPiC3D case study

Anna Jakobs[1], Anke Zitz[1], Norbert Eicker[1], Giovanni Lapenta[2]

[1]Forschungszentrum Jülich
Jülich Supercomputing Centre
D-52425 Jülich
a.jakobs@fz-juelich.de
a.zitz@fz-juelich.de
n.eicker@fz-juelich.de
[2]Katholieke Universiteit Leuven
BE-3001 Heverlee
giovanni.lapenta@wis.kuleuven.be

Abstract: The DEEP (Dynamical Exascale Entry Platform) project aims to provide a first implementation of a novel architecture for heterogeneous high-performance computing. This architecture consists of a standard HPC Cluster and – tightly coupled – a cluster of many-core processors called Booster. This concept offers application developers the opportunity to run different parts of their program on the best fitting part of the machine striving for an optimal overall performance. In order to take advantage of this architecture applications require some adaption. To provide optimal support to the application developers the DEEP concept includes a high-level programming model that helps to separate a given program to the Cluster and Booster parts of the DEEP System. This paper presents the adaption work required for a Particle-in-Cell space weather application developed by KULeuven (Katholieke Universiteit Leuven) done in the course of the DEEP project. It discusses all crucial steps of the work starting with a scalability analysis of the different parts of the program, their performance projections for the Cluster and the Booster leading to the separation decisions for the application and finally the actual implementation work. In addition to that some performance results are presented.

1 Introduction

Even though today's supercomputer systems reach multi-Petaflop compute power (examples are: Tianhe-2 at Guangzhou, Titan at Oak Ridge National Lab, IBM Sequoia at LLNL, or JSC's JUQUEEN) the HPC community prepares for the next step, i.e. having Exascale systems (10^{18} floating-point operations per second) by the end of the decade. The DEEP (Dynamical Exascale Entry Platform) project[1] aims to develop a novel, Exascale-enabling supercomputing platform, the Cluster-Booster architecture. On this platform application developers can map their code onto two diverse parts of the system supporting the different demands of scalability of their programs in an optimized way. Those two parts are: (1) a Cluster of multi-core Xeon processors interconnected by an InfiniBand network with a fat-tree topology; and (2) a second cluster of self-hosted Xeon Phi many-core processors called Booster. The latter utilizes the EXTOLL network [2] supporting a 3D torus topology. A comprehensive software environment including low-level communication libraries, programming environments and run-time systems complete the system.

As an assessment of this novel supercomputer architecture six scientific pilot applications were chosen within DEEP. In the course of the project these applications are ported to the platform acting as the yard-stick to evaluate the software environment and to act as benchmarks of the overall architecture. They have been selected with regard to their high scientific, industrial and social relevance and the urgent need for Exascale compute power in their research fields. Furthermore, the existence of highly scalable parts in their code that shall profit from the Booster was crucial. This paper will focus on the iPiC3D application from the Katholieke Universiteit Leuven. iPiC3D is a Particle-in-Cell space weather simulation predicting conditions of the magnetized plasma that permeates the space between Sun and Earth as well as the whole solar system.

The actual steps performed in the first three years of the DEEP project are explained in detail in the course of this paper. As a first step this includes a detailed analysis of the application with different performance tools. Next a plan to separate the code in a Booster and a Cluster part based on this analysis was created. It serves as a basis to finally implement this separation. To help the application developers to perform the separation the OmpSs runtime[3] – developed by BSC (Barcelona Supercomputing Center) and extended in the DEEP project – was used.

The paper is organized as follows: As a first step we motivate the division of applications by the description of the Cluster-Booster architecture in section 2. Next we give a short overview of the iPiC3D application including the analysis results from the beginning of the DEEP project in section 3. After a brief summary of Xeon Phi optimization strategies in section 4 we focus on the actual work done for adapting the code to the DEEP system and the resulting problems in section 5. Finally we present some performance results in section 6. The last section gives a short conclusion and an outlook on the next steps.

2 Cluster-Booster architecture

On the way to an Exascale supercomputer the DEEP project pursues the strategy of a new architecture consisting of a Cluster part and a Booster part. This architecture represents an alternative approach to organize heterogeneity in high-performance computing. As sketched in figure 1 the Cluster nodes utilize Intel Xeon multi-core processors and utilize an InfiniBand fabric for communication. In contrast to that, the Booster is based on Xeon Phi's many-core processors interconnected by an EXTOLL fabric developed at University of Heidelberg. A so called Booster Interface bridges between the two different interconnects of Cluster and Booster and allows for a most efficient communication between the two parts of the system.

The general idea of the Cluster-Booster concept is based on the observation that complex

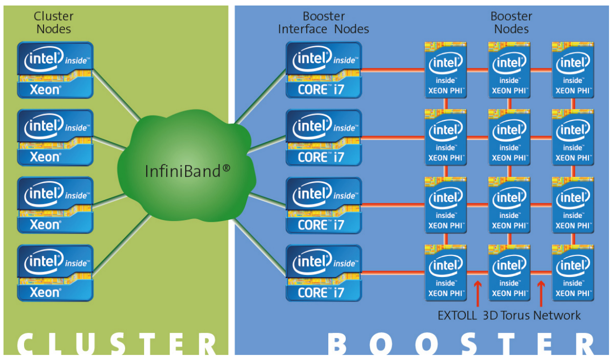


Figure 1: DEEP architecture scheme

applications in HPC often have code-parts with differing ability to scale on parallel machines. In order to utilize the DEEP system in a most efficient way, the less scalable code parts and the I/O operations are run on the Cluster while highly scalable parts are offloaded from the main application to the Booster. The DEEP system, in contrast to today's GPU-based heterogeneous systems, allows for communication between the processes offloaded to the Booster Nodes. This gives the application developers the chance to offload more complex kernels; code parts with intensive collective communications should be executed on the Cluster instead of the Booster anyway.

In order to support application developers to port their applications to this unconventional heterogeneous architecture the DEEP project develops a rich software infrastructure. While its basic mechanisms rely on MPI and its `MPI_Comm_spawn` functionality to start additional processes within the system, special emphasis was taken to relief the application programmer from having to re-organize the code manually. For this OmpSs, an OpenMP based data-flow programming model with directives to support asynchronous parallelism and heterogeneity, is extended allowing the user to just annotate the code leaving the main work to the Mercurium source-to-source compiler and the Nanos++ runtime system.

Of course, separating applications into two parts and distributing them to the Cluster and Booster parts of the DEEP system might introduce new bottle-necks if the parts have to exchange data. Therefore, it is crucial to split the application in a way that the amount of data to be exchanged is minimized. In addition to that a highly optimized Cluster-Booster protocol was introduced into the DEEP software stack reducing the overhead of the necessary bridging between the two fabrics as much as possible.

Since this paper will not explain neither the overall concept nor the hardware or software architecture in more detail, the interested reader is referred to [4] for more information.

3 iPiC3D application

The iPiC3D application of KULeuven is a massively parallel code to simulate the evolution of a magnetized plasma traveling from the Sun to the Earth. This topic is highly relevant in order to forecast space weather related events which may lead to severe problems like damage to spacecraft electronics, GPS signal scintillation or even disturbance of wide-area power grids. There are different approaches to model the plasma evolution; the iPiC3D application implements a kinetic approach wherein both ions and electrons are simulated as particles. iPiC3D is written in C++ and was parallelized using MPI before the beginning of the DEEP project. At the current state of work also OpenMP is used allowing for an hybrid parallelization. An application run consists of multiple time steps. For each of them particles are moved under the effect of the electric and magnetic fields defined on a discrete mesh in physical space.

Particle information is deposited on this spatial grid through interpolation procedures in the form of moments, i.e. densities, currents and pressures defined on the mesh, which act as sources for the equation solved for obtaining the fields at the next time step. More details on the actual algorithm are provided in [5].

The main part of iPiC3D is to calculate the evolution of the electric and magnetic fields and the positions and velocities of the computational particles in time. The particle information is averaged and collected as moments; these also have their part in evolving the electric field. During the analysis phase at the beginning of the DEEP project figure 2 was created, representing the logical structure of the application.

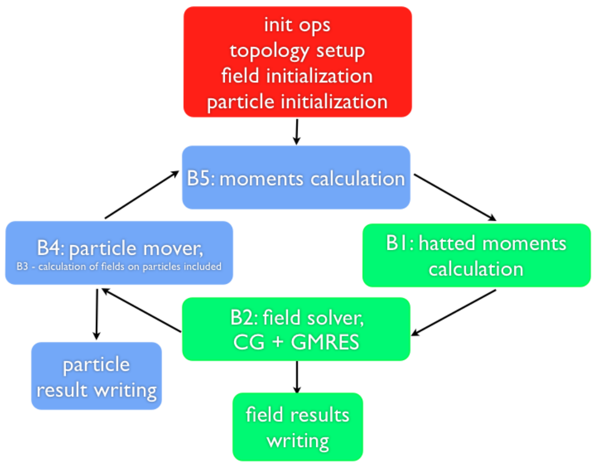


Figure 2: Logical structure of iPiC3D

The different parts can be shortly explained as follows:

B5: moments calculation: density, currents and pressures are calculated starting from particles.

B1: hatted moments calculations: the hatted moments, i.e the effective moments of interaction with the electric and magnetic fields, are calculated from the moments.

B2: field solver: the electric and magnetic fields are calculated for the new time step.

B4: particle mover: particles are moved under the influence of the newly calculated fields.

The green boxes of figure 2 are related to the fields and therefor the grid, the blue ones refer to the particles.

When describing the different phases we will concentrate on the moments calculation, the field solver and the particle mover, as they are the most time consuming or communication intensive parts of the application. This was shown by Scalasca profiling runs performed on the JUDGE cluster and the BlueGene/Q system JUQUEEN at Jülich Supercomputing Centre; the results are presented in figure 3.

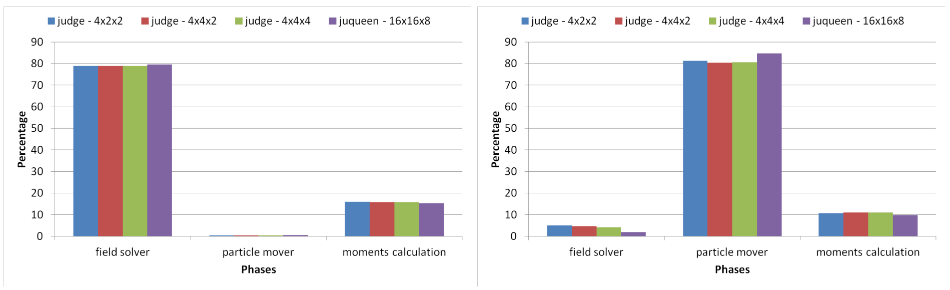


Figure 3: Percentage of execution time (left) and communication time (right) of the relevant phases for the JUDGE tests with $4 \times 2 \times 2$, $4 \times 4 \times 2$ and $4 \times 4 \times 4$ cores (small test case) and the JUQUEEN test with $16 \times 16 \times 8$ cores (big test case)

During the moments calculation, the particle information is gathered and accumulated in grid moments. After the collection the total moments of quantities like density or pressure are calculated. The nearest neighbors then exchange ghost node information using `MPI_Sendrecv_replace`. This phase is the second most relevant phase with regard to percentage of execution time, communication calls and bytes exchanged. Especially the collective communication of the ghost node information should be noted here.

In the field solver phase a Poisson correction of the electric field is performed and both the electric and the magnetic field are updated. This phase is not very time consuming, but is the most important one related to communication. As in the moment calculation the ghost node information is exchanged using point to point communication. Additional collective communication is required during each iteration of the solver in order to determine the stopping criterion.

Finally, the particle mover updates the particle positions and velocities. Since the particles are moved independently, this phase is highly scalable. After the movement all particles that are now located on a part of the grid hosted by a different processor have to be identified and exchanged; this is done with a series of point to point communications between nearest neighbors. In addition to that, the number of particles to be moved is assessed grid-wide through an `MPI_Allreduce`. These are the only few collective communications not done in the solver phase. This phase is essentially compute intensive.

The decision on how to divide the application into a Cluster and a Booster part was made on the basis of the following aspects:

- There are two different kind of phases, grid related (fields and moments) and particle related ones. They operate mostly on different data, so they should be kept on one part of the DEEP system each.
- The phases which have the most intensive collective communication should be kept on the Cluster, as these are implemented more efficiently on this part of the system. In the case of iPiC3D these are the grid related ones, i.e. moment and field calculations.
- The particle related phases can be vectorized more easily and have better scalability, as particles are processed independently; therefore, they fit better on the Booster part.

4 Xeon Phi optimization

For a most beneficial use of the DEEP Booster the application parts that will be launched there shall be optimized for the Xeon Phi processor. Different code changes were done to achieve this goal. They will be briefly explained within this section.

The most important step in the course of the optimization process is the introduction of OpenMP thread parallelization of the loops that process particles. Tests unveiled that using one MPI process for each hardware thread – 240 in total on a 60 core MIC – would introduce a significant communication overhead. On the other hand, with only 60 MPI processes of 4 OpenMP threads each the performance was about 2 to 4 times better.

A second approach was to introduce an improved localization of field data by using an array of `structs` (AoS) instead of a `struct` of arrays (SoA). Typically an SoA is preferred over an AoS when vectorizing code. Although, in this application using an AoS has considerable advantages like a faster sorting of particles (needed to eliminate random access) or a superior cache performance. As transpositions are rather cheap in this case it was decided to use an AoS for the basic particle representation and convert it to an SoA in blocks when it is beneficial for vectorization.

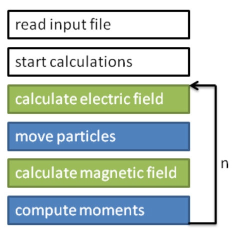
The next optimization step would be the vectorization of particle processing; for that step a sorting of the particle is needed, which has not yet been implemented for the code.

5 Offload adaption work

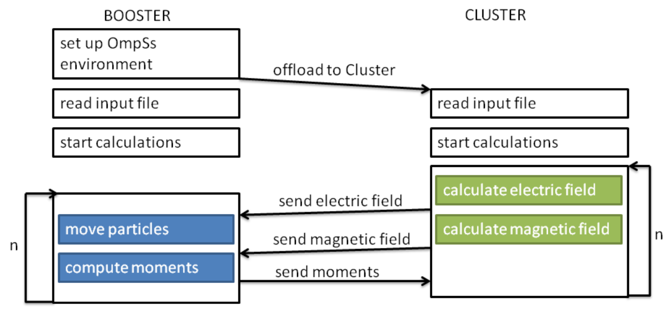
The application was logically divided into a part to run on the Booster and one to be offloaded to the Cluster[1] as can be seen in figure 2. This section explains the necessary steps performed to achieve a successful division.

5.1 Code division

The changes in the application workflow with the offload can be shown best along the flow charts in figure 4.



(a)



(b)

Figure 4: Workflow without (a) and with offload (b)

The original version of the application contains a single `run` function to start the whole simulation. In the offload version two corresponding functions are called, `run_Cluster` and `run_Booster`. For a clearer logical division the different calculation steps were divided between host and offload part as sketched in figure 4b and separated into the two functions. The necessary communication of the moments, the particles and the magnetic field between Cluster and Booster were added after the corresponding calculations. For the sending and receiving of data between host and offload the proper MPI communicators have to be used. In the offload part the parent communicator can be fetched via `MPI_Comm_get_parent`. This function returns an inter-communicator to the processes on the host part. For sending to or receiving from the offload on the host part the inter-communicator is used that was created and returned by the call of `deep_booster_alloc` (see section 5.2). To have access to this communicator in the relevant classes it is given as a parameter to the `run_Booster` function and from there forwarded to the communication functions.

[1]In fact, this is contrary to the basic concept described in section 2. Nevertheless, DEEP's software stack is flexible enough to support the reverse offloading used in iPiC3D, too.

5.2 OmpSs

As mentioned before iPiC3D was parallelized in a hybrid way using MPI and OpenMP. Utilizing the OpenMP offload was no option in the case of the DEEP project for two reasons: On the one hand this technology was not yet introduced in the OpenMP standard until almost two years in the project. On the other hand the OpenMP offload assumes a local target and does not allow for MPI-communication between offloaded processes. The latter is crucial when the offloaded tasks are computationally heavy as in the DEEP examples. Although OmpSs and OpenMP are similar in many aspects OmpSs does not support all OpenMP pragmas. All pragmas aside from `#pragma omp task` and `#pragma omp for` were commented in the OmpSs version of the application.

For using OmpSs the application has to be compiled with the Mercurium compiler developed by BSC[6]. Some minor changes in the code were necessary for that but will not be discussed further in this paper.

The next step was to integrate the call of the OmpSs function `deep_booster_alloc` and the offload pragma into the code. We are implementing a reverse offload here where we start the application on the Booster part and offload to the Cluster. This is due to the fact, that logically the application is build around the particle calculation, offloading the fields calculations fits better into the overall concept of the code (see figure 5).

```
MPI_Comm clustercomm;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

deep_booster_alloc(MPI_COMM_WORLD, 1, size, &clustercomm);

//create offload task
#pragma omp task device(mpi) onto(clustercomm,rank) \
     in([...]) copy_deps
{
   solver.run_Cluster();
}

solver.run_Booster(clustercomm);
```

Figure 5: Integration of OmpSs in main part of the code

In the original code, the input parameters were read from the input file, the necessary objects were created and the `run` function was called to start the calculations. It was planned to give the objects essential for the calculations to the offload as input parameters, but this was not feasible due to OmpSs not supporting the offload of C++ objects. A serialization and deserialization of all necessary objects seemed to be a solution, but with some testing this approach turned out to be way to complicated, as too many objects were needed.

Therefore it was decided to recreate the whole environment in the offload part. For this approach, only `argc` and `argv` are crucial, as `argv` contains the name of the input file, which is sufficient for establishing the whole setup in the offload part. `argv` is now serialized and given as an input parameter to the offload pragma. Generally speaking, the whole calculation is set up and started both on the host and in the offload part.

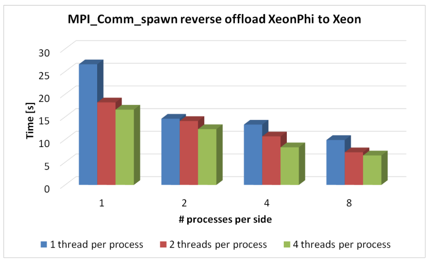
5.3 MPI_Comm_spawn

It has to be taken into account that the OmpSs runtime and the Mercurium compiler are still under development as part of the DEEP project; during the months of work shown in this paper they were constantly enhanced but issues like a significantly decrease of application performance or compiling problems appeared from time to time that needed investigation and hindered us to get detailed and meaningful performance results. To avoid these issues for the moment and to have a possibility for later comparison (mostly to see if using OmpSs creates overhead) yet another version of the application was implemented. This variant uses `MPI_Comm_spawn` explicitly for offloading. In contrast to that OmpSs uses this function, too; nevertheless the actual calls are hidden from the application developers. It required only minor changes in the code to create this version; mainly the OmpSs offload pragma was replaced by the call of `MPI_Comm_spawn`. Additionally the application was compiled directly with the Intel compiler without performing a source to source compilation with Mercurium beforehand. The main benchmarks were performed with this version of iPiC3D to analyze the general behavior of the application when offloading from Xeon Phi to Xeon. Lately, some tests revealed that with the newest versions of OmpSs the issues of a decrease of performance were solved and the overhead compared to the `MPI_Comm_spawn` has mostly vanished; future experiments will give specific numbers.

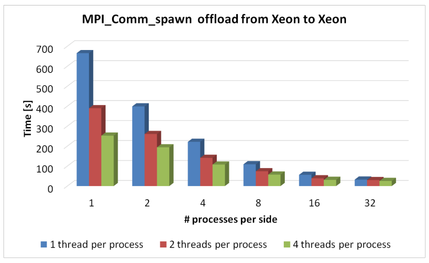
6 First results

During the adaption work the application was mainly tested on a KNC system installed at JSC with two compute nodes. This was done due to lack of the actual Booster hardware. The node that was used consists of two Xeon ES-2670 processors with 8 cores clocked at 2.6 GHz and four Xeon Phis 7120 co-processors with 61 cores each running at 1.23 GHz. Each Xeon Phi is equipped with 16 GB of memory. As it was decided to use a reverse offload model, the application was started on the Xeon Phis and offloaded to the Xeons. In addition to that tests were performed on the DEEP Cluster, offloading from Xeon to Xeon (Intel Xeon E5-2680 at 2.70 GHz).

Two different test cases were simulated. The smaller one contains 460,800 particles and was used for performance runs on the smaller KNC system. The larger test case of about 93 million particles was performed on the DEEP Cluster, taking advantage of the bigger system. In both cases 30 cycles were simulated.



(a)



(b)

Figure 6: Execution time on the KNC system with small test case (a) and on the DEEP Cluster with larger test case (b)

Figure 6a shows the performance results generated on the KNC system with the small test case, in figure 6b the execution times from the DEEP Cluster simulating the larger test case are shown.

Especially for the larger test case the application scales quite well with an increasing number of MPI processes. For the smaller example, the gain of the faster calculation is likely too small to hide the communication for more than 2 processes. A larger test case with more intense calculations should lead to a better scalability but couldn't be tested on the KNC system due to memory restrictions. One should to take into account that these performance numbers were created shortly after integrating the offload in iPiC3D; therefore they should be seen as first evaluation of the application behavior.

7 Conclusion and outlook

At the current point of the project, both the OmpSs and the `MPI_Comm_spawn` reverse offload are fully integrated in the iPiC3D application. The code was separated into different functions for the Cluster and the Booster part. Starting on the Booster, the parts of the code which operate on the moments or fields are offloaded to the Cluster. The minimal required communication between Cluster and Booster to exchange the fields and moments was set up.

Additional tests were performed on a bigger system, showing promising results. The next step will be to run a large set of benchmarks on the whole DEEP system as soon as possible.

As illustrated in this paper, working on a highly experimental project can lead to unforeseen challenges that might influence the outcome of the whole project. E.g. due to the late availability of hardware the applications could not be tested on the full DEEP system in time. Instead it had to be settled for a smaller test system for Xeon Phi performance evaluations and the DEEP Cluster for larger tests on Xeon cards; but these systems are able to give a prospect on what to expect from the final system and were used extensively for optimizing the applications for the two different kinds of architectures.

With hindsight it probably would have been more efficient to implement the offload in iPiC3D directly with `MPI_Comm_spawn` instead of trying OmpSs first, as the concept of OmpSs to offload multiple tasks with input and output dependencies does not fit the structure of the application in an optimal way. Nevertheless, it revealed some issues in the OmpSs runtime which could be solved this way, leading to more efficient and better performing versions in the course of the last months.

8 Acknowledgments

The authors would like to thank the people and partners involved in the DEEP consortium for their ongoing engagement and strong commitment towards the project, which led to several of the results described in this paper.

The research leading to these results has been conducted in the frame of the DEEP project, which receives funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n° 287530.

This article is dedicated to the memory of Alec Johnson who passed away in an accident right before Christmas 2014. He contributed significant parts of this work. He will remain in the memories of the authors and all collaborators of the DEEP project.

References

[1] `http://www.deep-project.eu`

[2] M. Nüssle, B. Geib, H. Fröning, and U. Brüning, "An FPGA-based custom high performance interconnection network", In "Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs", 2009

[3] `http://pm.bsc.es/ompss`

[4] Norbert Eicker, Thomas Lippert, Thomas Moschny, and Estela Suarez, "The DEEP project - Pursuing cluster-computing in the many-core era", 42nd International Conference on Parallel Processing (ICPP), p. 885 - 892, 2013

[5] G. Lapenta, J. U. Brackbill und P. Ricci, "Kinetic approach to microscopic-macroscopic coupling in space and laboratory plasmas", Physics of Plasmas, Vol. 13, Nr. 5, pp. 055904-055912, 2006.

[6] `http://pm.bsc.es/mcxx`

[7] S. Markidis, G. Lapenta und R. Uddin, "Multi-scale simulations of plasma with iPiC3D", Mathematics and Computers in Simulation, Vol. 80, Nr. 7, p. 1509 - 1519, 2010.