This is the accepted version of a paper presented at *The Exascale Applications & Software Conference (EASC2016)*.

Citation for the original published paper:

A Particle-in-Cell Method for Automatic Load-Balancing with the AllScale Environment.
In:

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-187878
A Particle-in-Cell Method for Automatic Load-Balancing with the AllScale Environment

[AllScale Project]

Roman Iakymchuk
CST/PDC, CSC, KTH Royal Institute of Technology
11428 Stockholm, Sweden
riakymch@kth.se

Herbert Jordan
University of Innsbruck
Institute of Computer Science
6020 Innsbruck, Austria
herbert.jordan@uibk.ac.at

Ivy Bo Peng
CST/PDC, CSC, KTH Royal Institute of Technology
11428 Stockholm, Sweden
bopeng@kth.se

Stefano Markidis
CST/PDC, CSC, KTH Royal Institute of Technology
11428 Stockholm, Sweden
markidis@kth.se

Erwin Laure
CST/PDC, CSC, KTH Royal Institute of Technology
11428 Stockholm, Sweden
erwinl@pdc.kth.se

ABSTRACT

We present an initial design and implementation of a Particle-in-Cell (PIC) method based on the work carried out in the European Exascale AllScale project. AllScale provides a unified programming system for the effective development of highly scalable, resilient and performance-portable parallel applications for Exascale systems. The AllScale approach is based on task-based nested recursive parallelism and it provides mechanisms for automatic load-balancing in the PIC simulations. We provide the preliminary results of the AllScale-based PIC implementation and draw directions for its future development.

Keywords

AllScale Environment, Particle-In-Cell method, task-based nested recursive parallelism, prec

1. INTRODUCTION

The Particle-in-Cell (PIC) method is one of the most common and powerful numerical techniques for the simulation of fusion, astrophysical and space plasmas. For instance, PIC simulations are used to study the interaction of the Earth’s electromagnetic field with the hot plasma emanated by the sun, the so-called solar wind. High energy plasma in space can damage spacecrafts and put in danger the life of astronauts in space. For this reason, it is important to enable efficient large-scale PIC simulations that are capable to predict different phenomena in space.

In the PIC method, the electrons and protons (plasma) are represented by computational particles whose trajectories are determined by the interaction with other particles and an existing background magnetic field. The Particle-in-Cell method solves the kinetic equation of plasmas by first sampling plasma distribution functions with computational particles and then following their trajectories by solving the equation of motion for each particle. Particle positions and velocities are updated in the particle mover stage. The electromagnetic field determining the particle trajectory is calculated by computing the Maxwell’s equations on a grid: each grid cell is characterized by an electric and magnetic field. The coupling between particle and both electric and magnetic fields is provided by the so-called interpolation functions: the electric field and magnetic fields acting on the particle are calculated by extrapolating the values on the grid cells at the particle position.

Typically, parallel PIC simulations divide the simulation box in several equal-in-size domains with initially the same number of particles [2]. Each domain is assigned to a process that carries out the computation for the particles in the domain. When a particle exits the domain, it is communicated to a different domain. Because of the non-uniform configuration of the electromagnetic field in space, computational particles concentrate in relatively small spatial regions while few particles cover other spatial regions. This results in having more particles in certain simulation domains than other and consequently the work-imbalance problem: processes with fewer particles have to wait for other processes with more particle to finish their computations at the synchronization point.

Workload-imbalance is the most severe and limiting problem in large-scale PIC simulations. For instance, the parallel simulation of the magnetic reconnection (a phenomena occurring on Earth’s magnetotail) on 8,192 cores leads to processes spending 22.2% of their time waiting for all processes to become available for collective operation. Other simulation set-up show even more severe work-imbalance: two-dimensional simulations of planetary magnetospheres on 2,048 cores present the process imbalance of 61.7% [4]. It is clear that on the road to Exascale, the problem of work-imbalance needs to be tackled and solved.

In this paper, we present a PIC method formulation to
provide automatic load-balancing. This is achieved by using an environment that is developed by the AllScale project \[1\] and it is based on the nested-recursive task parallelism. The PIC method uses the divide-and-conquer strategy: the simulation is recursively divided in smaller domains and assigned to a task for the computation. Each task is processing approximately the same number of particles. This strategy is recursive and nested as each task can spawn new tasks to handle smaller spatial regions.

The paper is organized as follows. The Allscale environment and its use to develop the new PIC method are described in Section 3. Section 4 shows the preliminary results of the new PIC method to simulate particle trajectories in the Earth magnetic field. Section 5 summarizes the results and outlines future work.

2. ALLSCALE ENVIRONMENT

AllScale is an EU funded FET-HPC project \[1\] targeting the development of a unified programming model for a large variety of target architectures, ranging from laptop and desktop devices, over multi-socket shared memory servers, to large-scale clusters, and future Exascale systems. Within this section the program model central to the AllScale environment will be outlined, followed by a description on its utilization for running PIC simulations.

Among the biggest problems in utilizing large-scale systems are load balancing and latency hiding. In the past, within smaller, shared memory systems, the load balancing problem has been successfully tackled by utilizing nested recursive task-based parallelism, as for example offered by Cilk \[3\]. In this model, tasks are created and, if required, dynamically re-distributed among the available computational resources to even out load imbalance. Furthermore, each task may create during its execution additional, nested tasks covering a share of the overall workload. This leads to (a) increased parallelism within the application and (b) introduces additional load balancing opportunities. By maintaining a list of processable tasks for each individual computational resource (e.g. hardware threads or GPUs) latency introduced by one task, e.g. due to a blocking IO operation, may be mitigated by continue processing another task in the meanwhile. While not supported by Cilk, systems like HPX and the Insieme Runtime System, or OpenCL/CUDA implementations utilize this technique for effectively hiding latencies within parallel applications.

The AllScale environment is extending this programming model to the world of distributed memory and heterogeneous computation systems. Parallel algorithms are expressed utilizing nested recursive parallelism. Thus, every task in the system provides the option of being split into smaller sub-tasks if required. The actual splitting is controlled by the runtime system, as well as the distribution of the produced tasks among the available resources. As a result, the runtime system is in control the number, locality, and granularity of all the processed tasks – thus empowering it to effectively manage the workload distribution and conduct latency hiding.

Nevertheless, in every non-embarrassingly parallel application, tasks will have to address shared data. For instance, within the PIC simulation, task will have to update the state of particles within a (logical) global shared data structure. However, in general, the data structures themselves may provide a structure enabling it to be decomposed and distributed throughout multiple nodes of a cluster. For instance, an array or grid like data structure can be partitioned into sub-ranges and correspondingly distributed.

To facilitate the AllScale runtime system to migrate tasks across address space boundaries of distributed memory or CPU/GPU systems, the runtime is additionally empowered in managing the distribution of the data objects operations are performed on. Thus, the AllScale runtime system can (gradually) adjust the distribution of data throughout the system, aiding to the ability of system wide load management.

2.1 The AllScale PIC Implementation

To fully utilize the AllScale environment’s ability of conducting dynamic load balancing and latency hiding, both, the parallel algorithm and the underlying data structure have to be formulated according to the AllScale program model. Fortunately, in both cases those modifications are readily comprehensible.

The algorithmic part of the PIC simulation code is covering four steps: the projection of particle properties to the grid boundaries, the solution of force-field equations over those grid boundaries, the application of those forces on the involved particles, and finally the movement of particles in space. All four of those are essentially parallel loops iterating over all cells in the 3D-space grid – and such parallel loops can be directly mapped to a recursive formulation.

To implement a parallel loop using nested recursive parallel tasks, each tasks covers a range of elements. The initial task covers the entire loop range. If split, the task’s range is split in half, producing two new tasks. This general principle is encoded into a generic pfor operator offering the same abstract interface like a conventional loop.

Thus, by utilizing the pfor operation on the structure of the 3D grid of the PIC code, task covering sub-regions of the overall 3D grid are generated and distributed throughout the system. Furthermore, operations on particles within cells are additionally parallelized by utilizing parallel loops over the corresponding lists of particles – based on the same, generic pfor operator. However, within the system, all tasks are following the nested recursive paradigm and may those also be managed as such.

Figure 1 illustrates an example of the resulting task-decomposition for the PIC simulation on the static grid. Thus, each task, which contains multiple cells, can be divided into sub-tasks. For instance, Task 4 is constantly divided into sub-tasks 4.1, 4.2, 4.3, 4.4 and the sub-task 4.3 is further divided into four sub-tasks 4.3.1, 4.3.2, 4.3.3, 4.3.4. On an example of Task 4.3.2, the right side of the figure, we also show that the work on updating particles positions and velocities within a task with one cell can be divided into sub-tasks.

Finally, the grid of cells needs to be adapted to facilitate its dynamic distribution among multiple address spaces. To this end, facilities to address sub-regions of the overall grid have to be provided to the runtime system to be able to address fractions of the overall data block as well as to manage and manipulate the data distribution. While several different formats for addressing sub-regions are conceivable, for the PIC simulation unions of axis-aligned boxes will be utilized. Thus, arbitrary sub-blocks of the grid may be moved between address spaces by the AllScale runtime system.

Since n-D grids are a common concept, the AllScale infrastructure provides a predefined, generic grid implementation
covering the corresponding implementation details similar to containers in the standard template library (STL) provided by any C++ implementation. Thus, the adaptation of the data structure narrows down to using a container type instead of a raw array within the simulation code.

3. PRELIMINARY RESULTS

We present some initial results obtained using the new PIC method for automatic load-balancing. The AllScale project was started in October 2015 and there is already an initial version of the AllScale environment. This version supports multi-socket shared memory architectures. We have implemented the particle mover component of the PIC method by following the AllScale approach. In particular, we rely upon the pfor operator. The other three key components of the PIC method are still under the development.

The new PIC method has been designed to tackle simulation with severe work-imbalance. The simulation of Earth radiation belts is a good example of such problems. The Earth dipolar magnetic field traps high-energetic particles in two spatial regions around the Earth. These regions have a doughnut shape and for this reason they are called belts. The trapping of particles in the radiation belts depends on the initial pitch angle (angle between the magnetic field and the particle velocity vector) and on the energy of the particle. Because particles are trapped in these regions, there is a high concentration of particles in the radiation belts and a small number of particles outside the radiation belts. For this reason, tasks, which are responsible for the computations on the cells inside the belts regions, have very high workload compared to the rest of the grid.

Figure 2 presents the particle trajectory in the Earth radiation belts simulation on an example of one proton. The initial particle position is located at $x = 4R_E$, $y = 0$, $z = 0$, where $R_E = 6,378km$ is the Earth radius, with the pitch angle equals to 30$^\circ$. This result demonstrates that under the strong magnetic field forces any particle (electron or proton) with its position relatively close to the radiation belts and a certain value of the pitch angle is trapped in the Earth radiation belts.

4. CONCLUSIONS AND FUTURE WORK

We have presented the initial AllScale implementation of the PIC method and provided the preliminary results of the particles trajectory in the highly load imbalanced Earth radiation belts simulation.

We foresee to provide the AllScale implementation of the entire PIC method, including the field solver and both the interpolation of fields to particles and the interpolation of particles to the grid, in the first year of the AllScale project. This would enable us to conduct the Earth radiation belts simulation on multi-socket shared memory architectures, but for small data sets. Our ultimate goal is to run simulations on large-scale pre-Exascale systems as well as to enhance significantly the PIC performance on the load imbalanced problems by eliminating entirely the enormous waiting time.

Acknowledgments

This work was partially supported by the AllScale project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 671603.

5. REFERENCES