Modernization of the PIC codes for exascale plasma simulation

I. Vasileska *, P. Tomšič** and Leon Kos***
Faculty of Mechanical Engineering, University of Ljubljana, Ljubljana, Slovenia
* ivona.vasileska@lecad.fs.uni-lj.si
** pavel.tomsic@lecad.fs.uni-lj.si
*** leon.kos@lecad.fs.uni-lj.si

Abstract - Particle simulation in the field of nuclear fusion is a well-established technique which has spawned dozens of codes around the world through years (e.g. BIT1, VPIC, VSIM, OSIRIS, REMP, EPOCH, SMILEI, FBPIC, GENE, WARP, PEPC) with varying degrees of specialization for different physical areas and accessibility. By leveraging advanced algorithmic features we propose to implement key for genuine pre-exascale capability. While more expensive than conventional mesh-based particle simulation, proposed approach eliminates artificial grid-generated noise by design and offers a more direct method for treating complex simulation case. A combined library would make coupling of mesh-based and mesh-free methods a lot easier. Strategically this work is expected to benefit not only by future users, but will also provide a fully alternative approach for high-fidelity modeling.

Keywords – HPC, GPUs, PIC codes, pre-exascale computing.

I. INTRODUCTION

Numerical modeling in the field of fusion physics often demands a kinetic approach to handle extreme nonlinearities in the codes. Currently the model of choice in this work are the particle-in-cell (PIC) codes. The PIC codes are used to describe the time evolution of the distribution functions of different particle species (ions, electrons, impurities) in different kind of studies, for example: in gyrokinetic modeling for fusion devices or in laser-based particle beam and etc.

State of the art three dimensional PIC simulation involve up to 1012 particles on 106 cores. The complexity of the problem requires calculation of more than 10^6 equations with many variables. This possesses huge computational demand, as well as computing time. With computer architecture advancement, computational demand can be sufficiently handled, but time efficiency (needed CPU hours) remains a challenge. During the last decade technological advancement has made GPU calculation less time consuming than CPU calculation. For that reason it is beneficiary to make transition of computation from CPUs to GPUs.

It would clearly be advantageous to offer generic algorithmic components for kinetic plasma simulation on a single platform or within an easily portable library which are capable of exploiting modern supercomputing architectures. The general aim here is to transfer HPC best practices and software technology known to work well in algorithmically related, but physically much simpler applications, in order to enable PIC codes for Exascale-class simulations.

The paper is organized in six sections. After introduction, problem description and raising computational demand of PIC codes is briefly presented. Transition from traditional usage of CPUs towards modern day implementation of GPUs and its benefits is described. In the fourth section different techniques of transition of current CPU orientated towards GPU orientated codes is described. In the section Results and benefits the current activities and related challenges are described. In the closing section Conclusion, summary assessments and a view towards future challenges is presented.

II. PROBLEM DESCRIPTION

Even when performed on modern supercomputers, PIC simulation still has its limitations: apart from the computational expense of solving the particle and field equations for the full 3D electromagnetic system, the necessity of transferring information to and from the spatial grid makes it inherently noisy, collisional regimes are only accessible with the help of ad-hoc extensions, and some form of adaptive mesh refinement is often required to handle geometrically complex problems. Various simplifications are possible by exploiting geometrical symmetry, enabling 1D and 2D codes to be used for preliminary studies or parameter scans.

![Figure 1. History of PIC simulations of fusion edge plasmas](image)

The need for computational effort is ever growing as the PIC simulation in fusion become more complex,
shown in Figure 1. In the last 20 years the complexity of the calculations have led to increase of numerical points of factor more than 10E+6. If the phenomena of interest in PIC simulations require decreasing the time intervals, code fragmentation using implicit algorithms can be used to overcome timestep limitations of explicit codes.

III. FROM CPU TOWARDS GPU

Advancement in computer architecture design has led to an explosion in amount of data that needs to be evaluated. Processing the ever-growing data in a reasonable amount of time has been one of the key aspects of computing development.

In recent years Graphics Processing Units (GPUs) have been developed rapidly and can easily outperform Central Processing Units (CPUs) in processing large-scale data parallel workloads, but are considered weak in processing serialized tasks and communication with other devices. Current trends lead towards the CPU-GPU heterogeneous computing era [1]. For the purpose of high-performance computing and embedded computing GPUs have been widely adopted thanks to their massively-parallel architecture [2], high memory throughput and energy efficiency.

![Figure 2. Top HPC systems CPU versus GPU performance growth](image)

The CPU is composed of a few cores with lots of cache memory, in contrast to a GPU that is composed of hundreds of cores that can handle thousands of threads simultaneously. The GPU makes the parallel computing more easier. This is the reason why nowadays the PIC codes, which are massively parallel, are being rewritten and are used with GPU [4]. As we compare Fig. 1 and Fig.2 we can see that during the years, as the complexity of the PIC codes has increased, also the GPUs computing performance has been on the rise. Hardware vendors have supported the move of a wider range of applications to a GPU with constantly adding new features to the accelerator so that they no longer need to work under strict close control of a CPU device. Nowadays GPUs can assume more responsibilities that would be typically carried out by the CPUs. Therefore there is a need of adopting current computer HPC codes for GPUs.

The EU has recognized the need for HPC to develop top-of-the-range supercomputers for processing big data. EuroHPC is a Joint Undertaking of the EC and 29 European countries with an aim to fund world-class integrated European HPC and data infrastructure and support a highly competitive and innovative HPC and Big Data ecosystem.

IV. METHODS

The main approach in this work is to utilize and exploit techniques which are known to work successfully in related particle-kinetic codes, adapting and transferring them. Nevertheless, we anticipate that considerable innovation will still be necessary to address the moving goal-posts of modern HPC architecture. Interestingly, some of the key components of the PIC can be characterized as:

- Generic algorithmic components of particle codes as library (field solvers, domain decomposition, particle pusher);
- Application of new developments in I/O, task-based parallelism, vectorization and memory management, paying due attention to available prototype hardware and likely adoption in future Exascale systems;
- Benchmarking against competitor PIC codes such as OSIRIS, SMILEI, PSC, PConGPU, in approach, scalability and flexibility.

For each of the target codes, there are specific priorities to be addressed to overcome recognized bottlenecks.

For a better fit load-balancing, vectorization, tasks and I/O, the data-structures have to be evaluated and refactored. In general this comprises field and particle structures, possibly grouped for locality. Here it is important to keep localized, regular (streamed) access to particle/field data in mind in order to use GPUs/MICs effectively.

Consider a new domain-decomposition in form of an over-decomposition to enable better load-balancing over MPI ranks by using space-filling curves. This has been demonstrated previously in the PSC code [5]. As additional benefit, removing the link between the number of domains and available cores might help with approaches to fault tolerance (failing nodes).

Implementation of a task-based programming model to enable more asynchronous execution of code, improving scalability by reducing synchronization points between nodes, possibly reducing the effect of load-imbalance between nodes. This also benefits load-balancing within nodes (hybrid parallelisation) when tasks on different data can overlap.

The main challenge is on reorganization of the PIC code structure to allow more asynchronous communication in order to reduce the time consuming simulations. The much higher scaling up to 100k cores can be achieved with task-based parallelism, for which a number of options are available. This means that the main part of the code needs to be rewritten, for example as in a recent implementation of the cosmology code SWIFT [6]. Several task-based implementations exist, and differ mainly in how tasks and dependencies are created/specified, for example:
• Task spawning: any function can spawn a task, i.e. call a function that will be executed as a task. Dependencies are implicitly given by the order in which tasks are spawned, e.g. Cilk [7], OpenMP 4.0 [8].

• Dependency deduction: tasks and the data they operate on are specified explicitly, dependencies are deduced from the data and the order in which the tasks are created, e.g. QUARK, OmpSs [9], StarPU [10].

• Explicit task graph construction: tasks and dependencies are explicitly specified by the user, e.g. Intel TBB [11], QuickSched [12].

In order to prepare for the technical and scientific challenges in Exascale computing, we propose to improve the scalability and adaptability of the PIC prototype codes. This work consists of two main tasks:

A. refactoring of the codes to make them more accessible and adaptable,

B. implementing task-based parallelisation to achieve performance portability and scalability.

A. REFACTORING

Code refactoring is a process of restructuring an existing code, a process which requires braking the complex system into smaller parts, which are computational easier to handle. Thus the refactoring also improves the code structure clarity. The benefits of refactoring are maintainability (it is easier to fix bugs because the source code is clearer), structural improvement (enables faster handling of the code) and speeding up development or/and analyzing.

During refactoring the codes some risks might occur. Inside the code bugs could be introduced due to incorrect utilization and using inadequate mathematical techniques.

The PIC codes are complex structured because they consist of a lot of subcodes, which describe the particles motions, fields (electric and magnetic), particles collisions and cross sections. With their connection, the PIC codes need a lot of CPU time consuming for simulations. For that reason with correct refactoring and transferring from CPU to GPU will speed up the simulations and will decrease the time consuming.

The refactoring of the PIC codes demands different types of restructuring, thus increasing the pool of new code parts for transversal activities and assures its reusability.

Refactoring of the existing codes to have a clearer separation of concern is the key to achieve the adaptability and portability needed to fully utilize extreme-scale HPC systems. Code modernization and modularization will allow for more accessible and adaptable codes with a clear separation between different levels of the code. This separation will make it easier to use external libraries and to fine-tune computational kernels to different computer architectures in order to achieve maximal performance.

In light of the current trend of increasing diversity of architectures within a single high-end HPC system (CPUs with large vector registers, GPUs connected via PCIe or NVLINK, FPGAs, vector processors as accelerators etc.), it is clear that a modular design with clear separations is needed more than ever. In order to cater for specific programming models, special treatment of data or other architecture-dependent tweaks that are required to fully utilize the hardware, it is crucial to either limit the code changes needed to a small part of the code or to offload the burden to an external library.

B. TASK-BASED PARALLELISATION

Task-based parallelization is a process running many different tasks at the same time of the same data. This form of parallelization uses multiple processors in parallel computing environments. One of the most important benefits using task-based parallelization is to increase the potential computing power, which the main key in CPU to GPU transfer. The risks that can occur using the parallelization are the lack of flexibility, scaling, requiring a large amount of arrays to be effective and difficulties regarding getting a good performance.

This task offers a methodological approach towards parallel computing, both in terms of algorithms and computing tasks and also in terms of diverse hardware environments.

By splitting the computation into reasonably small tasks and by defining the dependencies among the tasks, it is possible to efficiently utilize the full computing power of a single computational node, even in a heterogeneous architecture that may contain e.g. GPUs or other accelerators. Task-based parallelisation in combination with message-passing communication (MPI) for multi-node parallelisation is well suited to achieve good performance and scalability on extreme-scale computing resources containing a diversity of architectures.

For the exploration phase of this parallelization scheme, we propose to implement a simpler but (from the numerical point of view) similar grid-based code with StarPU using task-based parallelism. The main purpose of this approach is to reduce the development risk and time of the implementation of task-based parallelism in the target codes, by anticipating upcoming pitfalls, mistakes and problems. All insights will be valuable for finally introducing the task-based parallelism into the target codes. The preparation phase will develop ideas, workflows, algorithms that might be used in all of the three main codes. It will also serve as a testbed for new ideas concerning the task-based parallelism.

Because of the structure complexity of PIC codes, it is not trivial to do task-based parallelisation. For that reason in this work we will use our own prototype code, which includes the same scheme and algorithm as other PIC codes. First step is refactoring the code, following by task-based parallelisation.

V. RESULTS AND BENEFITS

Every PIC code has the same basic algorithm, as presented in Fig. 3. The PIC code simulates the motion of each plasma particle (described with the particle pusher)
and calculates all macro-quantities (like density, current density etc.) from the position and velocity of each of these particles. The macro-force acting on the particles is calculated from the field equations i.e. set of Maxwell’s equations (presented by block Field solver). If the PIC code solves the whole set of Maxwell’s equations it is called electromagnetic solver.

\[ \nabla \times B = \frac{\partial E}{\partial t} = 0 \]  

(1)

In equation the \( E = -\nabla \phi \) present self and applied electric fields \( E \) directed along coordinate \( x \). There are no variations in \( y \) or \( z \) directions. The plane parallel problem consists of single (electron) species. Input file is provided in presented form:

\begin{align*}
NT &= 200 \quad \text{# of time steps} \\
PPC &= 100 \quad \text{# of particles per cell} \\
CPP &= 64 \quad \text{# of cell per process} \\
DTFACTOR &= 0.001 \quad \text{#defines a fraction of } dt \text{ vs. time steps from plasma frequency; } \\
\text{#must be positive} \\
NPROC &= 4 \quad \text{# of processors} \\
LHSV &= 25000 \quad \text{#applied voltage on left-hand side; RHS is grounded;}
\end{align*}

Transferring the current run code from CPU to GPU enables computing on a GPU node cluster. This is achieved through building the StarPU. First refactoring of the code was done, during which every solver was divided into more simple parts using extract method. After that parallelization of the code was done.

The output of the simple test PIC is presented in the form of physical particle values, available in the format of four .dat files, containing data:

- **e.dat** (electric field), file contains dependences of electrical field and coordinate,
- **phi.dat** (potential), file contains information on how the potential is changed at the coordinate,
- **vxx.dat** (phase space), file contains the values of the velocity,
- **nt.dat**; record of time evolution of number of particles per processor

Graphical illustration of a result of a test simulation is presented in Fig. 5. The presented results from the transfer from CPU to GPU show adequacy in the approach as the output values are aligned with original CPU calculations. What are the advantages of such approach on a larger scale is yet to be researched, as it is needed to further
adapt the procedure and codes to be used in cases with higher code complexity. The benefits of this work are:

1. Refactoring of the codes to achieve a clear separation between different levels in the algorithms will not only improve the portability to new architectures but also make it easier to use external libraries when and where they provide optimized solutions. The portability of the simple PIC prototype code enables its universality and therefore is not dependent from the computer architecture. By testing this prototype code, which is a simple one-dimensional electrostatic code with only one species (electrons), it will be easier to refactor the other complex PIC codes, because the workflow scheme is similar. This will also make it easier to follow the latest developments in algorithms and will encourage code reuse in other projects in the community. By making the codes more accessible and maintainable, the benefits from this effort will be far-reaching and long-lasting.

2. Task-based parallelization is expected to directly improve the performance and scalability of the codes allowing for new science. In addition, the restructuring of the codes needed to accommodate for task-based parallelism will be beneficial for the adaptability to new architectures in heterogeneous environments. Using an external runtime task scheduler should also offer an even greater level of independence from the underlying hardware.

The whole transfer from CPU to GPU for every PIC codes is a complex procedure. For that reasons we have split the work in different independent parts that are solved separately. In this work we solve just the beginning part, where we check the compatibility of this transfer using the prototype simple PIC code. For future work we will continuing for solving the other problems to get whole CPU to GPU transfer for other complex PIC codes.

VI. CONCLUSION
Transfer of existing computer codes, written for numerical calculation on CPUs towards modern day usage on GPU orientated HPCs is a challenging task. In the work our goal was to test an approach on PIC code for particle simulation in the field of nuclear fusion.

We used the StarPU to transfer the simple PIC code from calculating on CPUs into GPUs. The results obtained show that the method used is adequate. It presents a base for further work. The main issue was compiling the correct StarPU, used for our simple code transfer.

The discussed approach for code adaptation from traditional CPU to GPU calculation is promising and further scaling of the idea for more complex codes seems to be a way forward for the future.

ACKNOWLEDGMENT
This work was supported in part by the grants P2-0073 and P2-0256 of the Slovenian Research Agency (ARRS). This work has been carried out within the framework of the EU’s Horizon 2020 Research and Innovation Programme (2014-2020) under grant agreement 823767 (PRACE-6IP WP8). The views and opinions expressed herein do not necessarily reflect those of the European Commission.

REFERENCES


