First experience with particle-in-cell plasma physics code on ARM-based HPC systems

2015 J. Phys.: Conf. Ser. 640 012064

(http://iopscience.iop.org/1742-6596/640/1/012064)

Journal of Physics: Conference Series **640** (2015) 012064 doi:10.1088/1742-6596/640/1/012064

First experience with particle-in-cell plasma physics code on ARM-based HPC systems

Xavier Sáez[1], Alejandro Soba[2], Edilberto Sánchez[3], Mervi Mantsinen[1,4], Sergi Mateo[1], José M. Cela[1] and Francisco Castejón[3]

[1] Barcelona Supercomputing Center (BSC-CNS), Barcelona, Spain
[2] Centro de Simulación Computacional para Aplicaciones Tecnológicas (CONICET), Argentina
[3] Laboratorio Nacional de Fusión (CIEMAT), Madrid, Spain
[4] Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain

E-mail: xavier.saez@bsc.es

Abstract. In this work, we will explore the feasibility of porting a Particle-in-cell code (EUTERPE) to an ARM multi-core platform from the Mont-Blanc project. The used prototype is based on a system-on-chip Samsung Exynos 5 with an integrated GPU. It is the first prototype that could be used for *High-Performance Computing* (HPC), since it supports double precision and parallel programming languages.

1. Introduction

During the last two decades, supercomputers have grown rapidly in performance to provide scientists the required computing power, at the cost of a similar growth in power consumption. The most used metric for evaluating supercomputers performance has been the speed of running benchmark programs (as Linpack [1]).

However, nowadays the computer's performance is limited by power consumption and power density. Energy has become one of the most expensive resources and, as a consequence, the energy expenses in a HPC center can easily exceed the cost of infrastructure after a few years in operation. Therefore, energy efficiency is already a primary concern for the design of any computer system and will define the maximum achievable performance. This change of the point of view is reflected in the increasing popularity of the Green500 list [2], in which supercomputers are ranked in terms of their power efficiency.

Looking to future, new developed platforms to build a sustainable exaflop supercomputer will have to be based on the power efficiency. The *Mont-Blanc project* [3] has the aim to design computer architectures capable of delivering an Exascale performance using 15 to 30 time less energy than present architectures. The reduction of energy consumption will be achieved by developing a HPC prototype based on the energy-efficient technology originally designed for mobile and embedded systems.

Particle-in-cell (PIC) is one of the most used methods in plasma physics simulations [4]. The quality of results achieved by this method relies on tracking a very large number of particles. Therefore, PIC codes require intensive computation and need to be adapted to new computing platforms constantly. *EUTERPE* is a parallel gyrokinetic PIC code for global linear and non-linear simulations of fusion plasma instabilities in three-dimensional geometries [5], specifically

in tokamaks and stellarators [6, 7]. It has been written to target traditional HPC clusters using Message Passing Interface (MPI).

In this paper we study the portability of EUTERPE to the ARM-based prototype within the Mont-Blanc project. The rest of the paper is organized as follows. In section 2, we describe the PIC method basics and the EUTERPE code. In section 3, we describe the Mont-Blanc project and the architecture of the ARM-based prototype [8]. Next we explain the first experiences on porting EUTERPE to an ARM platform. Finally some conclusions are given.

2. Particle-in-cell methods

PIC methods are used to model physical systems whose behavior varies over different ranges of spatial scales. Macroscopically the dynamics is described by a system of partial differential equations (continuous model), while microscopically is modeled by a set of discrete particles. Explicitly, a PIC method follows the individual particles (or fluid elements) in a continuous phase space, whereas moments of the distribution (such as densities and currents) are computed concurrently on stationary mesh points.

PIC methods are one of the most popular approaches in plasma physics to simulate the interaction of independent charged particles with each other and with electromagnetic fields [9].

A *full kinetic description* using the PIC method is implemented by replacing the distribution function f_s by a number of *macroparticles*, which represents a cloud of particles. The charges and densities of macroparticles are accumulated by interpolation on the spatial mesh and then the field equations are solved on the mesh. Finally, the forces acting on macroparticles are obtained by the interpolation of the fields at the macroparticles positions [10].

After an initialization phase, it is possible to summarize a *PIC algorithm* with three steps repeated at each time step [11]:

- **pull**: particle properties are interpolated to neighboring points in the computational mesh.
- **solve**: moment equations are solved on the mesh.
- **push**: momentum of each particle is calculated by interpolation on the mesh. The particles are repositioned under the influence of the momentum and the particle properties are updated.

In general, many PIC codes designed to simulate various aspects of the plasma behavior [12, 13] were not originally developed for new supercomputers based on multi-core architectures and basically only exploit the *task level parallelism*. The inclusion of new parallel programming techniques (as hybridization) allows to make the best of new multiprocessor supercomputers under development. There are several codes written in this way [14, 15], but in general they are simplified versions of production codes used by scientific groups in the simulation of real plasma physics environments. Nevertheless, this trend is reinforcing to provide more power computing to these applications.

2.1. EUTERPE code

In the EUTERPE code, the distribution function f_s of each kinetic species (s) is discretized using particles and a control variables scheme (δf) is used to reduce noise. The electrons are assumed to respond adiabatically and only electrostatic perturbations are taken into account. The evolution of the distribution function of each kinetic species is given by the gyrokinetic equation:

$$\frac{\partial f_s}{\partial t} + \frac{\mathrm{dv}_\parallel}{\mathrm{dt}} \frac{\partial f_s}{\partial v_\parallel} + \frac{\mathrm{d}\mathbf{R}}{\mathrm{dt}} \frac{\partial f_s}{\partial \mathbf{R}} = C_s, \tag{1}$$

where C_s represents collisions with all species. The evolution in time of $v_{\|}$ and $\mathbf{R}$ is integrated from equations depending of the fields through the Poisson-Ampère equations [16].

Initially, EUTERPE was parallelized applying domain decomposition and domain cloning [17]. These strategies only allow to parallelize at task level using MPI. For that reason, we developed a hybrid version of the code to take advantage of all the levels of parallelism that a multi-core architecture can offer [18], using OpenMP in the most time-consuming routines and developing a hybrid solver (mixing MPI and OpenMP) for the quasi-neutrality equation.

The OpenMP was introduced to parallelize the movement of the particles inside a domain (*push phase*). It is a suitable approach because the computation of the movement of any particle is independent from the rest of particles, so several threads can read simultaneously the electric field on the nearest grid points to a given particle without conflicts (figure 1a). OpenMP was also introduced in the computation of the charge density on the grid points (*pull phase*). This time there is a conflict because several threads can update the same grid point at the same time (figure 1b). Depending on the number of cores and the available amount of memory this issue is solved by atomic arithmetic operations or a mesh copy per thread.

The hybrid solver developed was an implementation of the Block Jacobi Preconditioning of the Conjugate Gradient method [19], where the preconditioner is a set of diagonal blocks.

3. The Mont-Blanc project

Transitions in the HPC world are not casual facts. The highest-volume commodity market (desktop computers) tends to drive lower-volume HPC market, because the design of an HPC chip has to be amortized selling the maximum number of units to minimize its final price.

For example, figure 2 shows how systems based on special-purpose HPC systems were replaced (from 1990 to 2000) by RISC microprocessors in the Top500 list [20]. This transition took place due to the cheapness of the RISC microprocessors although they were slower.

Currently, we observe a similar trend: low-power microprocessors (used in mobile devices) are improving their performance and are including features needed to run HPC applications (as an on-chip floating-point unit). It is reasonable to consider that the same market forces that replaced vector processors by RISC processors will be able to replace present options with mobile processors [3].

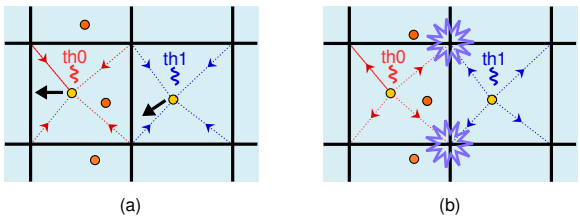

Figure 1. Two threads (*th0* and *th1*) that work with neighbor particles in the push phase (a) and the pull phase (b).

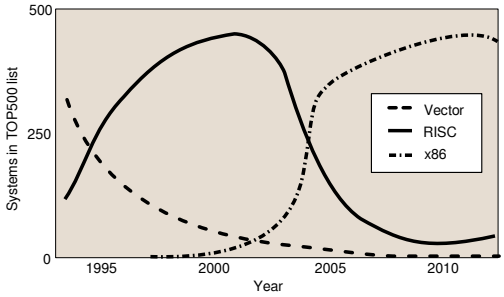

Figure 2. Number of systems in Top500.

Mont-Blanc is an European Exascale computing approach to develop a full energy-efficient HPC prototype. This project is coordinated by Barcelona Supercomputing Center (BSC) since October 2011. The aim is to develop a prototype using low-power commercially available embedded technology to exploit the large volume of these platforms and their high accessibility.

3.1. The ARM-based prototype

This work has been developed using a prototype named *Arndale*. It is based on a system-on-chip (SoC) Samsung Exynos 5 which contains an ARM Cortex-A15 dual core (at 1.7GHz) and an

ARM Mali T604 GPU.

ARM (Acorn RISC Machines) is a company which designs a RISC architecture. This kind of architectures reduces costs, heat and power use, which are desirable traits for embedded systems (smartphones, tablets, ...). Moreover, its simple design provides efficient multi-core CPUs with large number of cores at low cost and improved energy efficiency for HPC. Although ARM processors do not provide a sufficient level of performance for HPC yet, it is worthwhile to explore their potential considering ARM has a promising roadmap ahead.

The Samsung Exynos 5 with integrated GPU accelerator is the first embedded SoC that has the potential for HPC, since it supports 64-bit floating point arithmetic and provides support for parallel programming languages (such as OpenCL 1.1).

4. Work done on ARM-based platform

In order to obtain the best possible results on the prototype, all the available resources have to be used by the application: the dual core CPU (using OpenMP version developed previously) and the GPU (using a new OpenCL version).

OpenCL (Open Computing Language) [21] defines an application programming interface (API) that provides a homogeneous view of all computational resources through the following abstractions: the platform model, the memory model, the execution model and the programming model. It also support data-based and task-based parallel programming models. The execution model divides the computing system into a *host* (CPU) and a set of *compute devices* (GPU in our case). The host sends portions of the application (*kernels*) to the compute devices which execute many instances of them (*work-items*). The work-items are grouped by *work-group* and the work-items inside a group share resources such as memory.

We note here that the Mali GPU (in the prototype) has two special features which affect the common OpenCL programming. Firstly, the local and private memory are physically global memory. So, moving data from global to local or private memory typically does not improve performance. Secondly, all GPU threads have individual program counters. This means that all the threads are independent and can diverge without any performance impact.

In order to exploit the accelerator (Mali GPU) of the prototype, the selected routines have to be written in OpenCL. This mission involves some new tasks to make: the translation of the code from FORTRAN to C, the distribution of work between work-items and the creation and initialization of the OpenCL components (context, kernels, command queues and memory objects).

As a result of the previous work [18], we know which sections in the code are the most compute intensive: pushing the particles (*push*) and depositing their charge (*pull*). In this work our first aim is to minimize the necessary changes to these routines in OpenCL as compared with the previous version in order to increase the productivity.

In the *push* part, the work distribution is simple: one work-item is assigned to each particle. Since each work-item works on a different particle, all work-items write in different memory locations (as figure 1a shows but replacing threads with work-items). Therefore, the adaptation of the routine was reduced to a straightforward translation.

The *pull* part was more challenging to implement, since different particles can contribute to the charge density on the same mesh point. As we maintain the same work distribution than in push part, several threads can update the same memory location (as figure 1b shows but replacing threads with work-items). To avoid these memory conflicts, the two previously mentioned solutions used for the OpenMP implementation do not work properly due to the high number of threads on a GPU: atomic operations become inefficient because of lock contention that serializes the execution, and copies of the mesh require too much memory.

The strategy to solve this issue is inspired by the way in which OpenCL distributes data processing. A mesh copy is created per work-group, so only the work-items in the same work-

group accumulate contributions on the same mesh copy. As a consequence, the lock contention is far minor and the reduction of memory conflicts makes the use of atomic operations feasible.

Nevertheless, a new step is required to reduce all these copies to the final mesh. The final global reduction is minor and a single kernel is enough to perform it since the number of mesh copies is limited by the number of work-groups. This time each work-item will collect the result of the same point in all the copies of the mesh.

The next step was to take advantage of all resources, so a convenient decision was to distribute the particles between CPU and GPU. Since the two devices shares the memory, all the particles are distributed between them without any additional data transfer.

Figure 3 shows the results of the ARM hybrid version (OpenMP + OpenCL). The test contains 1 million particles. We can see that as more particles are sent to GPU, the execution time of kernels decreases. Being the optimal configuration when the particles sent to the GPU are 40% in the *push* kernel and 60% in the *pull* kernel. At these minimum points, the combination of the CPU and GPU devices yields a modest but significant improvement over the CPU time only: about 30% in the *push* kernel and near 50% in the *pull* kernel.

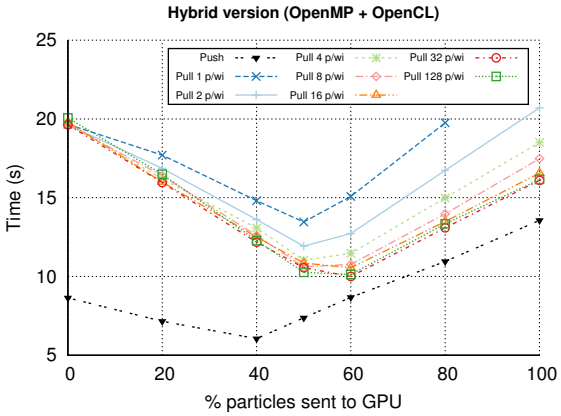

Figure 3. Execution time of the *push* and *pull* kernels for different combinations of particles sent to GPU and particles per work-item (*p/wi*).

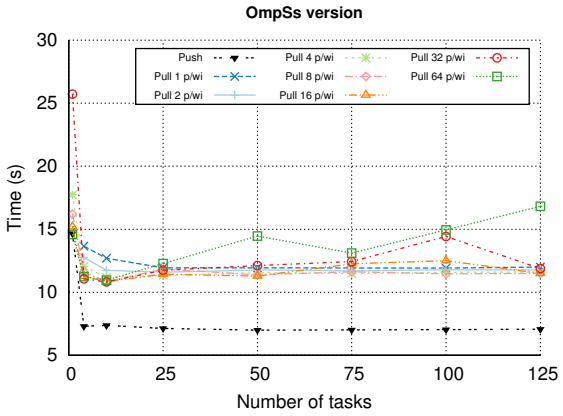

Figure 4. Execution time of the *push* and *pull* kernels for different combinations of number of tasks and particles per work-item (*p/wi*).

In order to port an application to a certain platform, besides the possibility to reach the maximum performance of it, the easiness of programming it is also important. The main drawback of OpenCL is the low programmability because is a low-level programming language. Therefore, it can be very time consuming to develop a code (meaning a low productivity).

To address this shortcoming, we ported the code to *OmpSs* [22] that is a task-based programming model developed at Barcelona Supercomputing Center (BSC). It provides an abstraction to the user which reduces programmer effort and unifies the SMP, heterogeneous and cluster programming in one model. The programmer annotates sections of the code to parallelize (*tasks*) with special directives. Over these tasks, the user can define *data dependencies* among them and specify the device where these tasks can be executed. The runtime system will analyze this information and automatically will schedule the execution of the tasks and perform the data transfers.

Figure 4 shows the results of the ARM OmpSs version. We can see that the performance is very stable and few tasks are enough to get the best performance for both kernels. Moreover, OmpSs offers a reduction in programming complexity, since the number of OmpSs directives included is far lower than the OpenCL calls included in the hybrid version (table 1).

Table 1. Comparison between the hybrid version (OpenMP+OpenCL) and the OmpSs version.

Kernel	Performance - best time (s)		Programmability - Productivity	
	Hybrid	OmpsSs	OpenCL API calls	OmpSs Directives
Push	6.02	6.92	161	12
Pull	10.31	10.83	167	18

5. Conclusions

This work confirmed the feasibility of porting a PIC code to an ARM-based platform. We have developed a hand-tuned hybrid version (OpenMP + OpenCL) and an OmpSs version of the most time-consuming kernels. Although OmpSs version is a bit slower than the hybrid version, it is a simpler version and its productivity has improved considerably. We can say that OmpSs simplifies the porting of codes to this new platform.

As future work we plan to code several state-of-art techniques to improve access to the data (in this case, particle data) in the GPU. In particular, we will explore if the benefits of sorting makes up for the extra sorting cost. Further, we will extend the implementation to several nodes using the parallelization with MPI on a new prototype.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under the Mont-Blanc Project (http://www.montblanc-project.eu), grant agreement nº 288777. This work was supported in part by grant ENE2012-30832, Ministerio de Economía y Competitividad and carried out within the framework of the EUROfusion Consortium, receiving funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

References

[1] Dongarra J, Luszczek P *et al.* 2003 *Concurrency and Computation: Practice and Experience* **15** 803–820
[2] The green 500: Ranking the world's most energy-efficient supercomputers `http://www.green500.org`
[3] Rajovic N *et al.* 2013 *Proc. of the Int. Conf. on HPC, Networking, Storage and Analysis* (ACM)
[4] Birdsall C K *et al.* 1991 *Plasma Phys. via Computer Simulation* 1st ed Plasma Phys. Series (Stanford Press)
[5] Jost G, TM T *et al.* 1999 *26th EPS Conference on Controlled Fusion and Plasma Physics*
[6] Kornilov V, Kleiber R, Hatzky R, Villard L and Jost G 2004 *Physics of Plasmas (1994-present)* **11** 3196–3202
[7] Sánchez E, Kleiber R, Hatzky R *et al.* 2013 *Plasma Physics and Controlled Fusion* **55** 014015
[8] ARM Ltd Cortex-a15 processor `http://www.arm.com/products/processors/cortex-a/cortex-a15.php`
[9] Dawson J M 1983 *Rev. Mod. Phys.* **55**(2) 403–447
[10] Pritchett P 2003 *Space Plasma Simu.* (*Lecture Notes in Phys.* vol 615) (Springer Berlin Heidelberg) pp 1–24
[11] Akarsu E *et al. Proceedings of the 1996 ACM/IEEE Conf. on Supercomputing* (IEEE Computer Society)
[12] Jolliet S, Bottino A, Angelino P, Hatzky R *et al.* 2007 *Computer Physics Communications* **177** 409 – 425
[13] Heikkinen J, Janhunen S, Kiviniemi T and Ogando F 2008 *Journal of Computational Physics* **227** 5582
[14] Burau H, Widera R *et al.* 2010 *Plasma Science, IEEE Transactions on* **38** 2831–2839
[15] Ragan-Kelley M 2010 Cs267 report particle simulation on a gpu with pycuda Tech. rep.
[16] Hahm T S 1988 *Physics of Fluids (1958-1988)* **31** 2670–2673
[17] Hatzky R 2006 *Parallel Computing* **32** 325 – 330
[18] Sáez X, Soba A, Sánchez E *et al.* 2011 *Proc. of the 19th Euromicro Conf. on Parallel, Distributed and Network-based Processing (PDP)* (IEEE) pp 385–389
[19] Saad Y 2003 *Iterative Methods for Sparse Linear Systems* (Society for Industrial and Applied Mathematics)
[20] TOP500org Top500 the lists. top500 supercomputer sites `http://www.top500.org/`
[21] OpenCL: The open standard for parallel programming of heterogeneous systems `https://www.khronos.org`
[22] Duran A, Ayguadé E, Badia R M, Labarta J, *et al.* 2011 *Parallel Processing Letters* **21** 173–193