Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Optimizing the Multi Level Multi Domain Particle-in-Cell code Parsek2D-MLMD

T. Ponweiser[a,*], M.E. Innocenti[b,†], G. Lapenta[b], A. Beck[c], S. Markidis[d]

[a] *Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria*
[b] *Center for mathematical Plasma Astrophysics, Department of Mathematics, K.U. Leuven, Celestijnenlaan 200B, B-3001 Leuven, Belgium*
[c] *Laboratoire Leprince-Ringuet, Ecole Polytechnique, CNRS-IN2P3, France*
[d] *KTH Royal Institute of Technology, Stockholm, Sweden*

Abstract

Parsek2D-MLMD is a semi-implicit Multi Level Multi Domain Particle-in-Cell (PIC) code for the simulation of astrophysical and space plasmas. In this Whitepaper, we report on improvements on *Parsek2D-MLMD* carried out in the course of the PRACE preparatory access project 2010PA1802. Through algorithmic enhancements – in particular the implementation of smoothing and temporal sub-stepping – as well as through performance tuning using *HPCToolkit*, the efficiency of the code has been improved significantly. For representative benchmark cases, we consistently achieved a total speedup of a factor 10 and higher.

1. Introduction

Parsek2D-MLMD is an advancement of the semi-implicit Particle-in-Cell (PIC) code Parsek2D [5] for the simulation of astrophysical and space plasmas. Scientific cases of interest for the code are collisionless plasma processes in the heliosphere, with particular attention to magnetic reconnection in the magnetosphere and kinetic instabilities. Kinetic simulations of plasmas are extremely challenging because processes develop on multiple scales, from the ion (large, slow) to the electron (small, fast) scales. In order to meet the challenge of capturing the relevant physical phenomena at different spatial scales, *Parsek2D-MLMD* employs a novel adaptive technique, the so-called Multi Level Multi Domain (MLMD) method [6], [8], which is shortly described in Figure 1.

In this project we aimed for improving the performance and efficiency of *Parsek2D-MLMD*. To quantify the targeted improvement by means of a representative example, we consider a problem setting consisting of two levels with a grid size of 1280 × 758 cells each and a total number of computational particles of about 1.5×10^9. At the time of the project start, approximately 100 thousand core-hours (4096 cores for 24 hours) were necessary to compute 5,500 simulation cycles for such a system. However, for research purposes longer simulation periods of 20 to 25 thousand cycles are of great interest. Hence a speedup of *Parsek2D-MLMD* by a factor of 4 to 5 was initially desired in order to make such simulations feasible.

* Principal PRACE expert, E-mail address: Thomas.Ponweiser@risc-software.at
† Principal investigator, E-mail address: MariaElena.Innocenti@wis.kuleuven.be

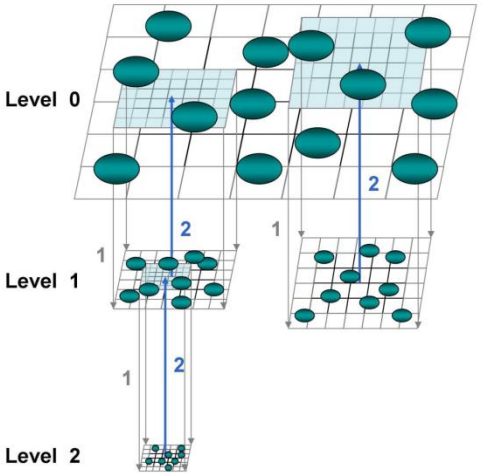

Figure 1: The Multi Level Multi Domain (MLMD) method. "The system is simulated as a cloud of self-similar, partly overlapping domains all complete in fields and particles. Particles are present at all levels, including the overlapping areas, and are sized according to the local grid size. Each level executes the same operations for fields and particles and the exchange of information for the fields is limited to the "downwards" interpolation of the boundary conditions for the refined levels from the coarser level fields (1) and the "upwards" projection of the refined fields from the refined to the coarser levels (2). Particles are regenerated at the boundaries of the refined grid with a splitting algorithm and are lost when they exit the refined areas, thus eliminating the need of coalescence operations." [6]

This ambitious goal has been achieved in two ways: First, two major algorithmic improvements which are described in Section 2 have been implemented which both enhance the computational efficiency of the code. Second, detailed profiling of the code, mainly using the performance measurement and analysis tool suite *HPCToolkit* [10], suggested several performance optimizations which have been carried out. The most important of these optimizations as well as their impact are discussed in Section 3. In Section 4, we report on the overall success of this project and compare performance and scalability of the old and new code version.

2. Algorithmic improvements

In this section we shortly describe two major algorithmic improvements which have been implemented in the course of this project. Both of these enhancements allow a more efficient usage of the computational resources in the sense that now fewer communication and computational operations are required to obtain results of similar physical significance and quality. For further details see Innocenti et al. [9].

2.1. Smoothing

Smoothing is a very powerful tool in Particle-In-Cell simulations, where it is usually used to suppress numerical instabilities and in particular the Finite Grid Instability, an aliasing instability which arises when physical quantities are sampled on low-resolution grids [1]. It became recently clear that smoothing is fundamental in the MLMD system to control numerical noise on the refined grid. Smoothing has to be applied in a specific sequence with respect to MLMD operations to avoid the formation of artefacts on the coarse grid. The use of smoothing allowed to increase of the stopping criterion in the GMRES iterative field solver (from 10^{-6} to 10^{-3}) [3] and to decrease the number of inner iterations used in the particle mover (from 7 to 3) [2]. A very demanding stopping criterion for the field solver and a very high number of inner iterations for the mover were previously used in order to reduce the noise on the refined grid and to limit the development of artefacts on the coarse grid. The implementation of smoothing and the consequent use of less demanding termination criteria for the field solver and the particle mover reduced the communication requirements (the GMRES solver performs most of the communication, both Point to Point and global, in single-grid PIC codes) and, most importantly, the execution times of the code (the particle

mover is the most computationally expensive block in single grid PIC codes) – see also profiling results for single grid Implicit Moment Method PIC codes in [7].

2.2. Temporal sub-stepping

Implicit Moment Method Particle In Cell simulations are supposed to respect the constraint $\varepsilon < v_{th,e}\,\Delta t/\Delta x < 1$, where $\varepsilon \approx 0.01$ (but it can be lower if smoothing is applied), $v_{th,e}$ is the average thermal velocity for electrons, Δx is the spatial resolution and Δt is the time step. A grid over-resolved in time may develop the Finite Grid Instability [1]. Conversely, a grid under-resolved in time may incur accuracy problems [4]. When employing the MLMD method in a system with two grids (coarse and refined) using a high (spatial) Refinement Factor while having the same temporal resolution on both grids, the coarse grid is exposed to the former risk and the refined grid to the latter. In order to overcome this problem, *temporal sub-stepping* has been implemented.

In the new version of the code, a Time Ratio *TR* between the time steps in two grids can be specified as an input parameter. The refined grid executes *TR* cycles for each coarse grid cycle. In this way the number of computational operations on the coarse grid decreases. The reason for the better performance with sub-stepping lies in the reduction of the frequency of MLMD operations: C2R (Coarse to Refined) and R2C (Refined to Coarse) communication is executed only one time every *TR* cycles, rather than every cycle. This reduces communication overhead and waiting times on the refined grid. In particular, particle splitting and communication operations for the refined grid are carried out only once for each *TR* cycles, which has a positive effect on scalability and execution time. Note that the physical significance of the simulations is preserved when choosing lower temporal resolutions for the coarse grid. In fact, *TR*s up to six have been successfully used in magnetic reconnection simulations, employing a spatial Refinement Factor of *RF* = 12 [9].

3. Performance optimization

A further speedup of *Parsek2D-MLMD* has been achieved by profiling and subsequent optimization of the code. In this section, we summarize the most important applied optimizations and their impact to the runtime of *Parsek2D-MLMD*. The primary tool for gathering and analyzing performance data has been *HPCToolkit* [10]. In-depth analysis of *HPCToolkit* tracing results can be found in [9].

Note that all reported speedups in this section refer to a small test case with two grids (coarse and refined), simulated for 100 time steps (temporal sub-stepping disabled) on a total number of 256 cores. With respect to this benchmark case, we achieved a total speedup by a factor of 2.7 purely through code optimization (disregarding the algorithmic enhancements of the previous section).

3.1. Speedup of the initialization phase

One of the first and very straight-forward optimizations has been to remove unnecessary testing code involving costly MPI synchronizations during the initialization phase. By this change, we observed a speedup for the above mentioned 100 cycles test case of about 10%.

3.2. Improvements for particle communication

A much more notable improvement, speeding up the code by a factor of approximately 2.1, has been achieved by optimizing the operations related to particle communication.

First of all, it turned out that the buffer arrays hosting particle information for data exchange had been oversized. The choice of a more adequate buffer size had two positive effects: First, the memory consumption of the code has been drastically reduced and second, buffer reset operations (which in the original implementation had been necessary once every simulation time step) could be carried out significantly faster, resulting in a speedup of 35%.

Additionally, using a slightly more sophisticated method for collecting the particle information to be exchanged, costly buffer reset operations could entirely be removed. Moreover, the actual communication of particle

information has been improved by switching from fixed-sized messages (containing unneeded padding data) to variable-length messages (whose actual length is queried via *MPI_Get_count*). In this way we achieved a further performance improvement of about 57%.

3.3. Optimization of the particle mover

Also the particle mover has been optimized very successfully, resulting in an additional speedup for the above benchmark case of approximately 20%. This improvement has been achieved by changing the 3-dimensional array structure hosting the field and particle moment information. Previously, 3-dimensional arrays have been allocated in a top-down approach, resulting in a tree-like structure of many nested one-dimensional arrays. Now, we are using a bottom-up approach, where we allocate just one contiguous block of memory for the data elements of each 3-dimensional array. Besides speeding up the allocation phase by a factor of 12, the new data structure allows for a more direct and cache-friendly data access. Previously, for the interpolation every single data element has been accessed through virtual getter methods. We now bypass these getter methods as well as tree-traversal and access the underlying one-dimensional data element array directly. In this way, we were able to speed up the weight calculation and interpolation operations carried out by the mover by a factor of 2.5.

4. Results

In this section we compare the performance and scalability of the old and new code version and discuss in detail the improvements achieved in the course of this project.

4.1. Strong scaling

The test cases used here are magnetic reconnection problems simulated with 2 grids and a Refinement Factor between the grids of $RF = 12$. The coarse grid domain is $80di \times 80di$ wide, with di being the ion skin depth; 1024x1024 cells are used per grid level and 100 particles per species and cell are loaded. 4 particle species are used. This relative low number of particles per species and cell is used to avoid memory problems when the number of cores is decreased.

For assessing the codes performance in terms of strong scaling, we adhere to a definition also used for the PRACE application benchmark suite: *A code is considered to scale to X Tflop/s if its performance on a machine partition with a peak of X Tflops is at least 1.7 times the performance on a partition of the same machine of half the size (i.e. X/2 Tflop/s peak)*. [11]

We carried out a strong scaling experiment for 512, 2048 and 8192 processes (doubling each time the number of processes in x- and y- direction of the 2D grid). Taking the above factor of 1.7 as lower bound for the expected speedup when the computational resources are doubled, we thus should observe a speedup of at least $1.7^2 \approx 2.9$, when resources are quadrupled. As can be seen in Table 1, the new version of the code in this sense exhibits reasonable scalability up to 2048 processes. Note that the poor speedup between 2048 and 8192 processes is explained by the fact that for 8192 processes the problem size per processor is already too small (strong scaling experiments with larger problem sizes have not been carried out in this project due to resource limitations). Finally we can see that performance and scalability is considerably improved by applying temporal sub-stepping (in this experiment, a Time Ratio of $TR = 6$ is applied).

	v1, NS	v1, TR6
Cores	Time [s] / Speedup	Time [s] / Speedup
512	512.0 / -	401.3 / -
2048	207.5 / 2.47	113.4 / 3.54
8192	189.5 / 1.09	86.2 / 1.32

Table 1: Strong scalability experiment for the new code version (v1), without (NS) and with (TR6) temporal sub-stepping. The speedups between 512 and 2048 processes are satisfactory. The usage of temporal sub-stepping has a positive effect on performance and scalability.

4.2. Weak scaling

The test cases used here are magnetic reconnection problems simulated with 2 grid levels and a Refinement Factor between the grids of $RF = 12$; 16x16 cells are used per core, with 196 particles per species per cell and 4 particle species. The number of cells and the domain size are scaled with the number of cores used.

The factor 1.7 which according to [11] can be considered as a lower bound for the speedup to expect when going from N to $2N$ processors in a strong scaling experiment, corresponds to a maximal scaling overhead (the ratio between theoretically optimal and actual speedup) of $2/1.7 \approx 1.18$. This number can as well be used to assess code performance in terms of weak scalability. In a weak scaling experiment, we therefore consider a code to scale from N to $2N$ processes, if the scaling overhead (the relative increase in execution time) is lower than 1.18 and to scale from N to $4N$ processes, if the overhead is lower than $1.18^2 \approx 1.38$.

We were measuring weak scalability using 32, 128, 512 and 2048 processes (doubling each time the number of processes in x- and y- direction of the 2D-grids), and comparing the old and new code version. As shown in Table 2, results of comparable accuracy and quality can be obtained approximately 10 to 20 times faster than before, depending on temporal sub-stepping being used or not. Exploiting temporal sub-stepping, we can demonstrate good scalability up to 2048 processes (the scaling overhead is not greater than 1.38).

Cores	v0, NS Time [s] / Overhead	v1, NS Time [s] / Overhead	v1, TR6 Time [s] / Overhead
32	442.2 / -	40.4 / -	32.2 / -
128	524.6 / 1.19	51.4 / 1.27	38.9 / 1.21
512	846.1 / 1.61	82.5 / 1.61	51.0 / 1.31
2048	1378.9 / 1.63	134.3 / 1.63	70.4 / 1.38

Table 2: Weak scaling experiment for the old (v0) and new (v1) code version without (NS) and with (TR6) temporal sub-stepping. Smoothing and code optimizations improved the performance by a factor of 10 (compare first two columns). Performance and scalability is further improved by using temporal sub-stepping.

The achieved speedup can as well be read off from Figure 2. In the logarithmic time-scale, the lines corresponding to the old and new code version (without temporal sub-stepping) are almost parallel and differ by a factor of approximately 10. The additional speedup and scalability improvement introduced by temporal sub-stepping can as well be seen.

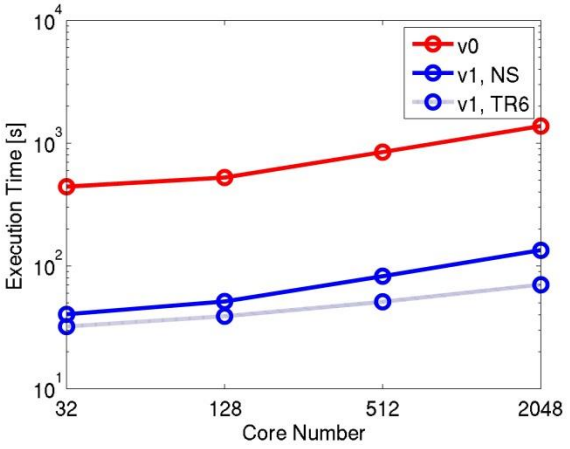

Figure 2: "Weak scaling tests for the old code version (v0), the new code version without sub-cycling (v1, NS) and the new code version with sub-stepping and Time Ratio between the grid TR=6 (v1, TR6). A logarithmic scale is used in the y-axis. Notice the code speed up between the old and the new version of the code. Sub-stepping improves scalability, which was however not the primary aim of the project." [9]

4.3. Efficiency

To conclude this section, Figure 3 highlights the benefits of the MLMD method in terms of computational efficiency compared to the original non-adaptive method, which applies the highest spatial resolution to the whole computational domain. It is obvious that the MLMD method saves a huge amount of computational resources.

It is worth noting that although optimization of the MLMD operations has been the main focus, also the full resolution simulations benefitted from the optimizations carried out in this project. It can be seen in the logarithmic time scale that the computation times for the old and new code versions consistently differ by a factor of almost 10 (temporal sub-stepping has not been applied here).

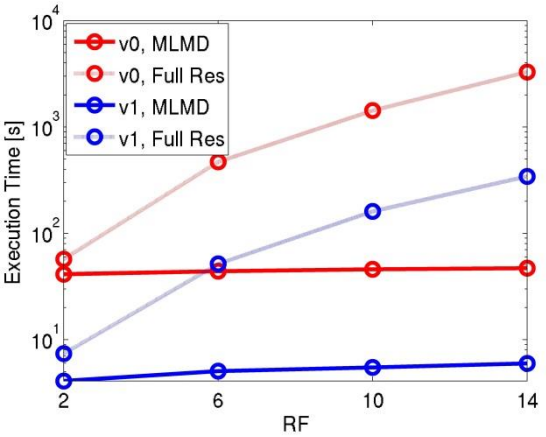

Figure 3: "Execution time in seconds for two-level MLMD simulations (straight line) with the old (v0) and new (v1) version of the code. The execution times of one-level simulations with resolution equal to the refined grid resolution are plotted in dashed lines. The y-axis is in log scale: notice the code speed up between v0 and v1." [9]

5. Conclusion and Outlook

In this report, we summarized the improvements for the Multi Level Multi Domain Particle-in-Cell code *Parsek2D-MLMD*, achieved during the PRACE preparatory access project 2010PA1802. The initial goal, a speedup of the code by a factor of 4 to 5, has been more than reached. In fact, speedups of a factor 10 and higher for different representative benchmark cases could be achieved thanks to the implementation of smoothing and temporal sub-stepping as well as through further performance tuning.

Still, there is room left for further improvements of the code. In the current implementation, the computational workload is distributed in a way such that each grid of the MLMD system is handled by a dedicated subset of MPI processes. Since the computational operations on the different grids cannot be perfectly interleaved, waiting times due to inter-grid communications are problematic. This issue has now become even more pressing, as temporal sub-stepping additionally results in a high load imbalance: With temporal sub-stepping, the computational workload of coarse grid processes is significantly lower than for the refined grid processes.

As outlined in [8], the solution to this problem lies in a redistribution of all grids to all MPI processes: The different levels of the MLMD system would be treated sequentially, but each level would benefit from a better parallelized execution using all available processes. It is expected that such an approach would further improve the overall efficiency and scalability of *Parsek2D-MLMD*.

The results of this work, in particular the most current version of *Parsek2D-MLMD*, are available online at *https://github.com/KulMari/Parsek2D_MLMD*. For further questions do not hesitate to contact Maria Elena Innocenti (*MariaElena.Innocenti@wis.kuleuven.be*).

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763. The work is achieved using the PRACE Research Infrastructure resource CURIE based in France at TGCC.

References

[1] C. K. Birdsall, A. B. Langdon, "Plasma physics via computer simulation", Taylor & Francis, 2004.

[2] G. Lapenta, J. Brackbill, P. Ricci, "Kinetic approach to microscopic-macroscopic coupling in space and laboratory plasmas", Physics of plasmas 13 (2006) 055904.

[3] Y. Saad, "Iterative methods for sparse linear systems", Society for Industrial and Applied Mathematics, 2003.

[4] H. Vu, J. Brackbill, "Celest1d: an implicit, fully kinetic model for low-frequency, electromagnetic plasma simulation", Computer physics communications 69 (1992) 253–276.

[5] S. Markidis, E. Camporeale, D. Burgess, Rizwan-Uddin, G. Lapenta, "Parsek2D: An Implicit Parallel Particle-in-Cell Code", in: N. V. Pogorelov, E. Audit, P. Colella, & G. P. Zank (Ed.), Astronomical Society of the Pacifc Conference Series, volume 406 of *Astronomical Society of the Pacifc Conference Series*, pp. 237–+.

[6] M.E. Innocenti, G. Lapenta, S. Markidis, A. Beck, A. Vapirev, "A multilevel multi domain method for particle in cell plasma simulations", Journal of Computational Physics 238 (2013) 115 – 140.

[7] Mallon, D. A., Eicker, N., Innocenti, M. E., Lapenta, G., Lippert, T., Suarez, E., 2012. "On the scalability of the clusters-booster concept: a critical assessment of the deep architecture." In: Proceedings of the Future HPC Systems: the Challenges of Power-Constrained Performance. ACM, p. 3.

[8] A. Beck, M. Innocenti, G. Lapenta, S. Markidis, "Multi-level multi-domain algorithm implementation for two-dimensional multiscale particle in cell simulations", Journal of Computational Physics vol. 271 (2014), 430 – 443.

[9] M.E. Innocenti, A. Beck, T. Ponweiser, S. Markidis, G. Lapenta, "Introduction of temporal sub-stepping in the Multi Level Multi Domain semi-implicit Particle In Cell code Parsek2D-MLMD", in preparation

[10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, "HPCToolkit: Tools for performance analysis of optimized parallel programs", Concurrency and Computation: Practice and Experience 22 (2010) 685–701.

[11] M. Bull, "Unified European Applications Benchmark Suite", PRACE-2IP Public Deliverable D7.4