Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture

Lukasz Szustak
Czestochowa University of Technology
lszustak@icis.pcz.pl

Krzysztof Rojek
Czestochowa University of Technology
krojek@icis.pcz.pl

Roman Wyrzykowski
Czestochowa University of Technology
roman@icis.pcz.pl

Pawel Gepner
Intel Corporation
pawel.gepner@intel.com

ABSTRACT

The multidimensional positive definite advection transport algorithm (MPDATA) belongs to the group of nonoscillatory forward-in-time algorithms, and performs a sequence of stencil computations. MPDATA is one of the major parts of the dynamic core of the EULAG geophysical model.

The Intel Xeon Phi coprocessor is the first product based on the Intel Many Integrated Core (Intel MIC) architecture. This architecture offers notable performance advantages over traditional processors, and supports practically the same traditional parallel programming model.

In this work, we outline an approach to adaptation of the 3D MPDATA algorithm to the Intel MIC architecture. This approach is based on combination of temporal and space blocking techniques, and allows us to ease memory and communication bounds and better exploit the theoretical floating point efficiency of target computing platforms. In order to utilize computing resources available in Intel Xeon Phi, the proposed approach employs two main levels of parallelism: (i) task parallelism which allows for utilization of more than 200 logical cores, and (ii) data parallelism to use efficiently 512-bit vector processing units.

An important method of improving the efficiency of the block decomposition is partitioning of available cores/threads into teams. It allows us to reduce inter-cache communication overheads. Also, this method increases opportunities for the efficient distribution of MPDATA computation onto available resources. The purpose is to provide the trade-off between two coupled criteria: load balancing and intra-cache communication.

We discuss performance results obtained on two platforms, including either two Intel Xeon E5-2643 CPUs and Intel Xeon Phi 3120A, or two Intel Xeon E5-2697 v2 CPUs and Intel Xeon Phi7120P. The top-of-the-line Intel Xeon Phi 7120P gives the best performance results for all tests.

The achieved performance results provide a basis for further research on optimizing the distribution of MPDATA computations across all the computing resources of the Intel MIC architecture, taking into consideration features of its on-board memory, cache hierarchy, computing cores, and vector units.

HiStencils 2014
First International Workshop on High-Performance Stencil Computations
January 21, 2014, Vienna, Austria
In conjunction with HiPEAC 2014.

http://www.exastencils.org/histencils/2014/

Keywords

Stencil computation, MPDATA algorithm, temporal/space blocking techniques, Intel MIC, OpenMP, task scheduler

1. INTRODUCTION

MPDATA [7] is one of the two major parts of the dynamic core of the EULAG model. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established geophysical model for simulating thermo-fluid flows across a wide range of scales and physical scenarios, including the numerical weather prediction (NWP).

The newest research of EULAG parallelization have been carried out using IBM Blue Gene/Q and CRAY XE6 [4]. 3D MPI parallelization has been used for running EULAG on these systems with tens of thousands of cores, or even with more than 100K cores. When parallelizing EULAG computation on supercomputers and CPU clusters, the efficiency is declined below 10%. We propose to rewrite the EULAG dynamical core and replace standard HPC systems by small heterogeneous clusters with accelerators such as GPU [5] and Intel MIC [3].

Preliminary studies of porting anelastic numerical models to modern architectures, including hybrid CPU-GPU architectures, were carried out in works [5, 11, 10]. The results achieved for porting selected parts of EULAG to non-traditional architectures revealed potential in running scientific applications, including anelastic numerical models, on novel hardware architectures.

In this work, we outline an approach to adaptation of the 3D MPDATA algorithm to the Intel MIC architecture. This approach is based on combination of temporal and space blocking techniques, and allows us to ease memory and communication bounds, and better exploit the theoretical floating point efficiency of target computing platforms. We show some of the optimization methods that we found effective, and demonstrate their impact on the performance of both the Intel CPU and MIC architectures. We mainly focus on the use of MPDATA in NWP, where the size of grids is limited by $n \leq 2048$, $m \leq 1024$, and $l = [64, 128]$. The starting point in these research is an unoptimized parallel implementation of the MPDATA algorithm. In our work, we use the OpenMP standard for multi-/many-core programming.

2. ARCHITECTURE OVERVIEW

2.1 Intel MIC architecture

The Intel MIC architecture combines many Intel CPU cores onto a single chip [2, 3]. The Intel Xeon Phi coprocessor is the first product based on this architecture. The main advantage of these accelerators is that it is built to provide a general-purpose programming environment similar to that provided for Intel CPUs. This coprocessor is capable of running applications written in industry-standard programming languages such as Fortran, C, and C++.

The Intel Xeon Phi coprocessor includes processing cores, caches, memory controllers, PCIe client logic, and a very high bandwidth, bidirectional ring interconnect [3]. Each coprocessor contains of more than 50 cores clocked at 1 GHz or more. These cores support four-way hyper-threading, which gives more than 200 logical cores. The actual number of cores depends on the generation and model of a specific coprocessor. Each core features an in-order, dual-issue x86 pipeline, 32 KB of L1 data cache, and 512 KB of L2 cache that is kept fully coherent by a global-distributed tag directory. As a result, the aggregate size of L2 caches can exceeds 25 MB. The memory controllers and the PCIe client logic provide a direct interface to the GDDR5 memory on the coprocessor and the PCIe bus, respectively. The coprocessor has over 6 GB of on-board memory (maximum 16 GB). The high-speed bidirectional ring connects together all the cores, caches, memory controllers and PCIe client logic of Intel Xeon Phi coprocessors.

An important component of each Intel Xeon Phi processing core is its vector processing unit (VPU) [2], that significantly increases the computing power. Each VPU supports a new 512-bit SIMD instruction set called Intel Initial Many-Core Instructions. The new ability to work with 512-bit vectors enables operating on 16 float or 8 double elements per iteration, instead of a single element.

The Intel Phi coprocessor is delivered in form factor of a PCI express device, and cannot be used as a stand-alone processor. Since the Intel Xeon Phi coprocessor runs Linux operating system, any user can access the coprocessor as a network node, and directly run individual applications in the native mode. These coprocessors also support heterogeneous applications wherein a part of the application is executed on the host (CPU), while another part is executed on the coprocessor (offload mode).

2.2 Target platforms

In this study, we use two platforms containing a single Intel Xeon Phi coprocessor. The first platform is equipped with two newest Intel Xeon E5-2697 v2 CPUs (totally 2×2 cores), based on the Ivy Bridge architecture, and the Intel Xeon Phi 3120A card (57 cores). The second one includes two Sandy Bridge-EP Intel Xeon E5-2643 CPUs (2×4 cores in total), and the top-of-the-line Intel Xeon Phi 7120P coprocessor (61 cores). The peak performances of these platforms are 1521 (2x259 + 1003) GFlop/s and 1419 (2x105.5 + 1208) GFlop/s. These values are given for the double precision arithmetic, with taking into account the usage of SIMD vectorization. The important feature of Intel Xeon Phi coprocessors is the high memory bandwidth. In particular, Intel Xeon Phi 7120P provides 352 GB/s of memory bandwidth, as compared with 2×51.2 GB/s for both CPU platforms.

A summary of key features of tested platforms can be found in [1].

3. OUTLINE OF MPDATA

MPDATA belongs to the group of nonoscillatory forward-in-time algorithms, and performs a sequence of stencil computations. The 3D MPDATA algorithm is based on the first-order-accurate advection equation:

$$\frac{\partial \Psi}{\partial t} = -\frac{\partial}{\partial x}(u\Psi) - \frac{\partial}{\partial y}(v\Psi) - \frac{\partial}{\partial z}(w\Psi), \qquad (1)$$

where x, y and z are space coordinates, t is time, $u, v, w = \mathit{const}$ are flow velocities, and Ψ is a nonnegative scalar field. Eqn. (1) is approximated according to the donor-cell scheme, which for the $(n+1)$-th time step $(n = 0, 1, 2, \dots)$ gives the following equation:

$$\begin{aligned} \Psi^{*}_{i,j,k} = \Psi^{n}_{i,j,k} \quad - \quad & [F(\Psi^{n}_{i,j,k}, \Psi^{n}_{i+1,j,k}, U_{i+1/2,j,k}) - \\ & F(\Psi^{n}_{i-1,j,k}, \Psi^{n}_{i,j,k}, U_{i-1/2,j,k})] - \\ & [F(\Psi^{n}_{i,j,k}, \Psi^{n}_{i,j+1,k}, V_{i,j+1/2,k}) - \\ & F(\Psi^{n}_{i,j-1,k}, \Psi^{n}_{i,j,k}, V_{i,j-1/2,k})] - \\ & [F(\Psi^{n}_{i,j,k}, \Psi^{n}_{i,j,k+1}, W_{i,j,k+1/2}) - \\ & F(\Psi^{n}_{i,j,k-1}, \Psi^{n}_{i,j,k}, W_{i,j,k-1/2})]. \end{aligned} \qquad (2)$$

$$U \equiv \frac{u\delta t}{\delta x}; \; [U]^{+} \equiv 0,5(U + |U|); \; [U]^{-} \equiv 0,5(U - |U|). \quad (3)$$

The same definition is true for the local Courant numbers V and W.

The first-order-accurate advection equation is approximated to the second order in δx, δy and δt, through defining the advection-diffusion equation. Such a transformation is widely described in the literature. For the full description of the most important aspects of the second-order-accurate formulation of MPDATA, the reader is referred to [6, 7].

The whole MPDATA computation in each time step are decomposed into a set of 17 stencil sweeps, called further stages. Each stage is responsible for calculating elements of a certain matrix, based on the corresponding stencil. The stages dependent on each other. Results of stages are usually input data for the next one.

A single MPDATA time step requires 5 input and 1 output matrices, other 16 matrices are temporary ones. In the basic, unoptimized implementation of the MPDATA algorithm, every stage reads a required set of matrices from the main memory, and writes results to the main memory after computation. This scheme is repeated for all the stages.

In consequence, a significant traffic to the main memory is generated. Moreover, compute units (cores/threads, and VPUs) have to wait for data transfers from the main memory to the cache hierarchy. In order to better utilize features of novel accelerators, the adaptation of MPDATA computation to the Intel MIC architecture is considered in this work, taking into account the memory-bounded character of the algorithm.

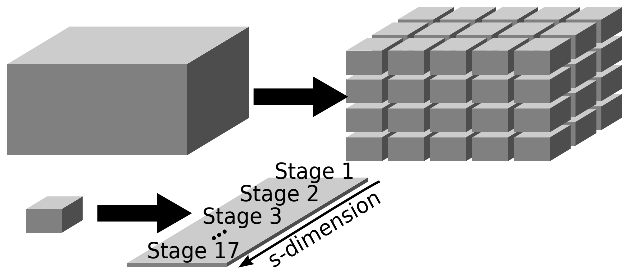

Figure 1: Idea of block decomposition of MPDATA computation

4. BLOCK DECOMPOSITION OF MPDATA

4.1 Basic idea

Since the 3D MPDATA algorithm includes so many intermediate computation, one of the primary methods for reducing the saturation of memory traffic is to avoid data transfers associated with these computation. For this aim, all the intermediate results must be kept in the cache memory, which increases the cache reusing. The memory traffic is generated only to transfer the required input and output data. Such an approach is commonly called the temporal blocking [8, 9].

In order to implement this approach efficiently, the loop tiling technique is applied. As a result, the grid is partitioned into blocks. Every block is responsible for computing all the 17 stages within the part of grid assigned to it. Within a single block, each stage provides computation for the adequate chunk of the corresponding matrix. Executing of a sequence of blocks determines the final output result for a single MPDATA time step.

The main requirement for this approach is to keep in the cache hierarchy all the data required for MPDATA computation within each block. Therefore, the size $nB \times mB \times lB$ of each block has to be selected in an appropriate way. The idea of block decomposition of the MPDATA algorithm is shown in Fig. 1. This decomposition determines four dimensions of distribution of MPDATA calculation across computing resources: i-, j-, and k-dimensions are related to the grid partitioning, while s-dimension is associated with the order of executing MPDATA stages.

Computing each MPDATA block requires extra calculation and communication for every stage because of data dependencies between stages. Extra calculations and communication have to take place on the borders between adjacent blocks, in order to ensure the correct results of the whole algorithm. Therefore, blocks are extended by adequate halo areas. As a result, blocks are independent of each other, and there are no communication between blocks within a single MPDATA time step.

The sizes of halo areas are determined in all the four dimensions (i, j, k and s), according to data dependencies between MPDATA stages. Thus, each of 5 input, one output, and 16 temporary matrices, is partitioned into chunks of size $nB \times mB \times lB$, which further is expanded by adequate halo areas with sizes iL, iR, and jL, jR, as well as kL, kR.

This approach allows us to avoid memory transfers for intermediate computation at the cost of extra computation associated with halo areas in chunks of temporary matrices, as well as extra communication between the main and cache memories, corresponding to halo areas in chunks of the input matrices. Another advantage of this approach is reducing the main memory consumption because all the intermediate results are stored in the cache memory only. In the case of coprocessors, it plays an important role because the size of on-board main memory is fixed, and significantly smaller than for traditional CPU solutions.

The requirement of expanding halo areas is one of the major difficulties when applying the proposed approach, taking into account data dependencies between MPDATA stages. It requires to develop a dedicated task scheduling for the MPDATA block decomposition.

4.2 Improving efficiency of block decomposition

Although the block decomposition of MPDATA allows for reducing the memory traffic, it still does not guarantee a satisfying utilization of target platforms. The main difficulty here is associated with extra computation and communication, which have impact on the performance degradation. In particular, there are three groups of extra computation and communication, corresponding to i-, j-, and k-dimensions. Some of them can be reduced or even avoided by applying the following rules:

1. The additional calculation and communication in k-dimension can be avoided if $lB = l$, and the size $nB \times mB \times lB$ of block is small enough to save in cache all the required data. This rule is especially useful when the value of l is relatively small, as it is in the case of NWP, where l is in range $[64, 128]$.

2. The overheads associated with j-dimension is avoided by leaving partial results in the cache memory. It becomes possible when extra computation are repeated by adjacent blocks. In this case, some results of intermediate computation have to reside in cache for executing the next block. This rule requires to develop a flexible management of computation for all the stages, as well as an adequate mapping of partial results onto the cache space. In consequence, all the chunks are still expanded by their halo areas, but only some portions of these chunks are computed within the current block. It means that this approach does not increase the cache consumption. The idea of improving the efficiency of block decomposition is shown in Fig. 2.

3. In order to reduce additional calculations in i-dimension, the size nB should be as large as possible to save in the cache hierarchy all the data required to compute a single block.

5. MPDATA PARALLELIZATION

5.1 Partitioning of cores/threads into independent teams

Another method of improving the efficiency of the proposed block decomposition is partitioning of available cores/threads into teams. Each team corresponds to a piece of the MPDATA grid, and executes calculation according to the block decomposition strategy. As a results, the MPDATA

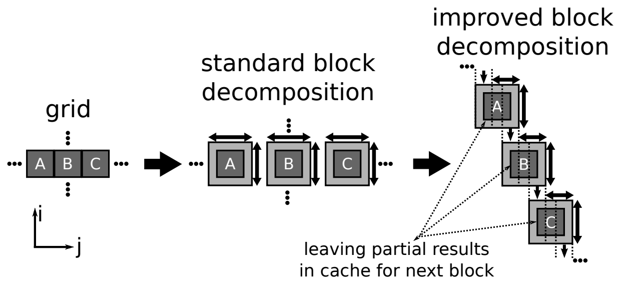
improved block decomposition
standard block decomposition
grid
A B C
A B C
A B C
i
j
leaving partial results in cache for next block

Figure 2: Idea of leaving partial results in cache memory

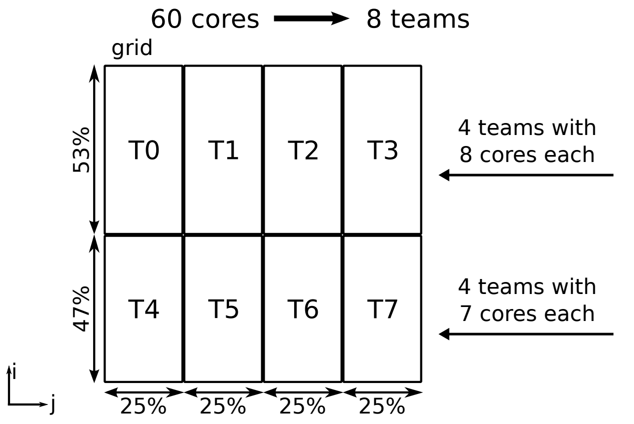

Figure 3: Partitioning of Intel MIC processing cores into teams when performing MPDATA computation

grid is distributed into pieces, and then each piece is decomposed into MPDATA blocks. Computation executed within teams are independent within each time step.

The proposed method allows us to reduce inter-cache communication overheads due to communication between neighbour threads, as well as their synchronization. What is also important, this method increases opportunities for the efficient load distribution of MPDATA computation onto available resources. These advantages are achieved at the cost of some extra computation performed by teams.

In general, pieces of the grid corresponding to different teams are characterized by various sizes. Numbers of cores/threads assigned to teams are different, as well. Fig. 3 shows an example of partitioning 60 Intel MIC processing cores into 8 teams, and distribution of the MPDATA grid onto teams. To provide load balancing, we distinguish 4 teams with 8 cores each, and 4 teams 7 cores each. Moreover, pieces of the MPDATA grid corresponding to these teams have different sizes along i-dimension.

5.2 Task and data parallelisms

In order to utilize computing resources available in the Intel Xeon Phi coprocessor, the proposed approach employs two main levels of parallelism:

- task parallelism which allows for utilization of more than 200 logical cores;
- data parallelism to use efficiently 512-bit vector processing units.

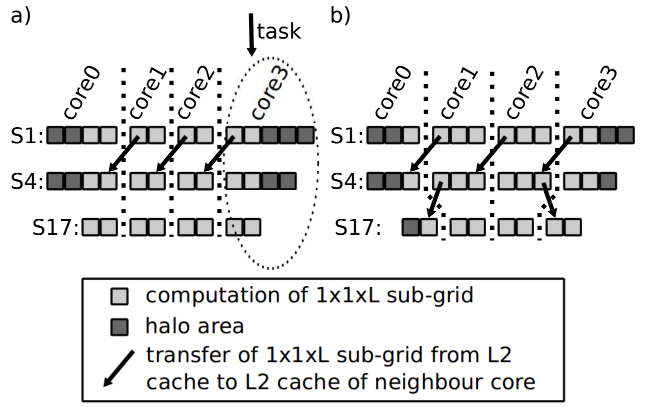
a)
b)
task
core0
core1
core2
core3
S1:
S4:
S17:
computation of 1x1xL sub-grid
halo area
transfer of 1x1xL sub-grid from L2 cache to L2 cache of neighbour core

Figure 4: Example of distribution of calculation within a team of cores: (i) first scenario sacrifices load balancing for reducing intra-cache communication; (ii) second scenario improves load balancing at the cost of increasing intra-cache communication

All computation executed within every MPDATA block of size $nB \times mB \times lB$ are distributed across available threads in each team. Each block is partitioned into sub-blocks of size $nB^* \times mB^* \times lB$, where partitioning takes place along i and j dimensions. Within an MPDATA block, tasks are assigned to sub-blocks, where each sub-block is computed by a corresponding thread.

Another level of parallelization is vectorization applied within each thread, so the resulting SIMDification is performed within k-dimension. In consequence, the value of size lB has to be adjusted to the vector size.

At the same time, for a fixed MPDATA block, a sequence of stages is executed, taking into account the adequate sizes of halo areas. Due to the data dependencies of MPDATA, appropriate synchronizations between MPDATA stages are necessary. Finally, within each team its MPDATA blocks are processed sequentially, following the order proposed for the block decomposition in Section 4. Naturally, teams perform all computation in parallel.

5.3 Distribution of calculation within team of cores

An appropriate distribution of calculation within team of cores is crucial for optimizing the overall system performance. The purpose is to provide the trade-off between two coupled criteria: load balancing and intra-cache communication. In fact, aiming at improving the load balancing within a team, we have to take into account the possibility of undesirable effect of increasing the communication between cores/threads, implemented through the cache hierarchy.

Fig. 4 illustrates an example of two scenarios of distributing MPDATA calculation within a given team of cores for the block of size $1 \times 8 \times l$. In this example, a single team corresponds to 4 cores (one thread per core is assumed). The first scenario (Fig. 4a) features less amount of intra-cache communication between tasks (threads) than the second one. However, the load imbalance within the team of cores is noticeable in this scenario. The second scenario provides a better load balance across available resources assigned to team, but it requires more intra-cache communication.

Because of intra-cache communication between tasks, the overall system performance depends strongly on a chosen task placement onto available cores (threads). Therefore, the physical core affinity plays a significant role in optimizing the system performance. In this work, the affinity is adjusted manually, to force communication between tasks placed onto the closest adjacent cores, as much as possible. This increases the sustained intra-cache bandwidth, as well as reduces cache misses, and the latency of access to the cache memory.

6. PERFORMANCE RESULTS

In this section, we present preliminary performance results obtained for the double precision 3D MPDATA algorithm on the platforms introduced in Section 2. In all the tests, we use the icc compiler as a part of Intel Parallel Studio 2013, with the same optimization flags. The best configurations, including number of teams, size of block, and distribution of computation within team, are chosen in an empirical way, individually for each platform. Moreover, we use Intel Xeon Phi in the native mode.

Currently, only the first four stages of MPDATA are implemented and tested. These four stages correspond to the linear version of MPDATA. Since all the input matrices are required to provide the correctness of calculation, the overall performance for this part of MPDATA is strongly limited by the memory traffic between the main memory and cache memory.

Fig. 5 presents the normalized execution time of the 3D MPDATA algorithm, for 500 time steps and the grid of size $1022 \times 512 \times 63$.

Fig. 5a shows a performance gain for the improved version of block decomposition. The proposed method of reducing extra computation allows us to speedup MPDATA block version from 2 to 4 times, depending on the platform used and size of the grid.

The impact of block size on the overall performance is illustrated in Fig. 5b. In general, the larger block size the higher performance is achieved. However, the limiting factor is the cache size.

According to Fig. 5c, among five tested configurations the best results are obtained for the configuration containing 14 teams, with 16 threads per each team. Rather surprisingly this configuration uses only 56 cores. These configurations are highly distinguished with respect to load balancing of MPDATA computation and intra-cache communication. In consequence, significant performance differences are observed in these tests.

The advantages of using vectorization is observed for all the platforms. In particular, for Intel Xeon Phi 7120P, it allows us to accelerate computation more than 3 times using all the available threads/cores (Fig. 5d).

Fig. 5e shows the performance obtained for different numbers of threads per core, using Intel Xeon Phi 7120P. The best efficiency of computation is achieved when running 4 threads per each core.

The performance comparison of all the platforms is shown in Fig. 5f. For each platform, we use all the available cores with vectorization enabled. As expected, the best performance result is obtained using Intel Xeon Phi 7120P. This coprocessor executes the MPTADA algorithm 2 times faster than two Intel Xeon E5-2697 v2 CPU, totally containing 24 cores. The both models of the Intel Xeon Phi coprocessor give similar performance results.

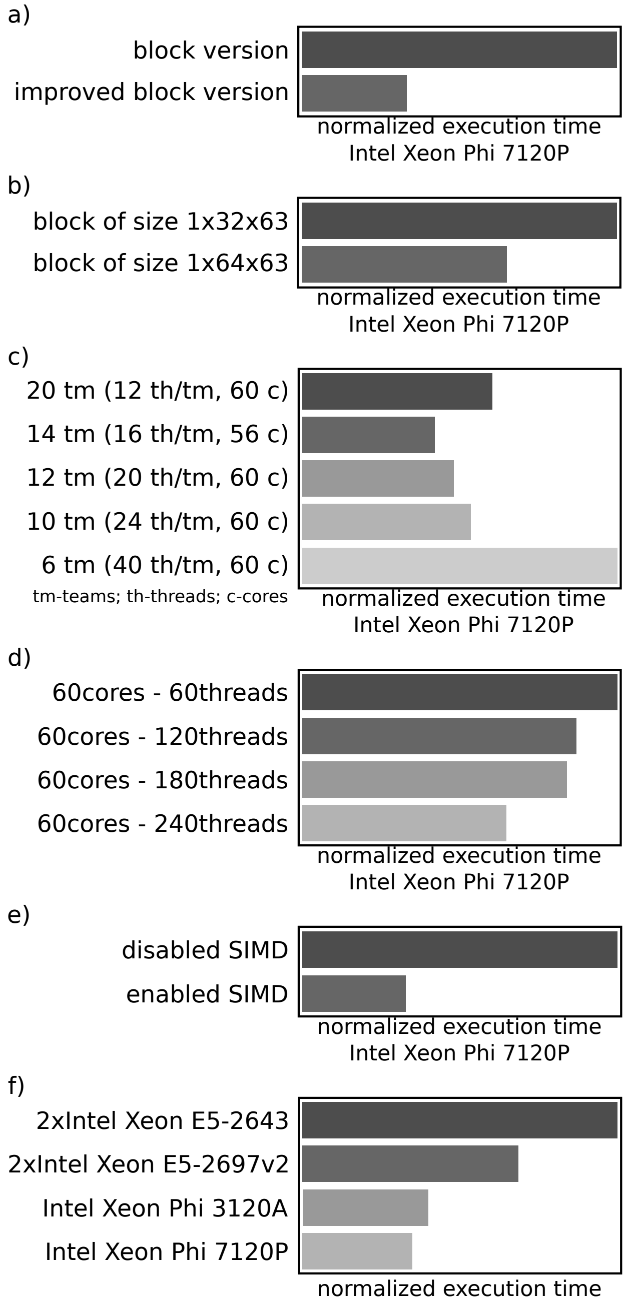

Figure 5: Preliminary performance results: (a) comparison of block and improved block versions; (b) performance for different block sizes (c) performance for different configurations of teams; (d) advantages of using vectorization; (e) performance for different numbers of threads per core; (f) comparison of Intel Xeon CPU and Intel Xeon Phi (best configurations with SIMD)

7. CONCLUSIONS

Using the Intel Xeon Phi coprocessor to accelerate computations in the 3D MPDATA algorithm is a promising direction for developing the parallel implementation of the EULAG model. Rewriting the EULAG code, and replacing conventional HPC systems with heterogeneous clusters using accelerators such as Intel MIC is a perspective way to improve the efficiency of using this model in practical simulations.

The main challenge of the proposed parallelization is to take advantage of many- and multi-core, vectorization, and cache reusing. For this aim, we propose the block version of the 3D MPDATA algorithm, based on combination of temporal and space blocking techniques. Such an approach gives us the possibility to ease memory bounds by increasing the efficient cache reusing, and reducing the memory traffic associated with intermediate computations. Furthermore, the proposed method of reducing extra computation allows us to accelerate the MPDATA block version up to 4 times, depending on the platform used and size of the grid.

Another method of improving the efficiency of the proposed block decomposition is partitioning of available cores/ threads into teams. Each team corresponds to a piece of the MPDATA grid, and executes calculation according to the block decomposition strategy. It allows us to reduce intercache communication overheads due to communication between neighbour threads, and their synchronization. Also, this method increases opportunities for the efficient load distribution of MPDATA computation on available resources.

An appropriate distribution of calculation within team of cores is crucial for optimizing the overall system performance. The purpose is to provide the trade-off between two coupled criteria: load balancing and intra-cache communication. Aiming at improving the load balancing within a team, the possibility of undesirable effect of increasing the communication between cores/threads has to be taken into account.

In all the performed tests, the Intel Xeon Phi 7120P coprocessor gives the best performance results. Both the many-core and vectorization features of the Intel MIC architecture play the leading role in the performance exploitation. The other important features are the block size, number of teams, number of threads per core, as well as an adequate thread placement onto physical cores. All these features have a significant impact on the sustained performance.

The achieved performance results provide the basis for further research on optimizing the distribution of MPDATA computation across all the computing resources of the Intel MIC architecture, taking into consideration features of its on-board memory, cache hierarchy, computing cores, and vector units. Additionally, the proposed approach requires to develop a flexible data and task scheduler, supported by adequate performance models. Another direction of future work is adaptation to heterogeneous clusters with Intel MICs, with a further development and optimization of code.

8. ACKNOWLEDGMENTS

This work was supported by the Polish National Science Centre under grant no. UMO-2011/03/B/ST6/03500, and by the Polish Ministry of Science and Education under grant no. BS/MN-1-112-304/2013/P.

We gratefully acknowledge the help and support provided by Jamie Wilcox from Intel EMEA Technical Marketing HPC Lab.

9. REFERENCES

[1] *Intel Architectures Comparison.* http://ark.intel.com/compare/75799,75797,64587,75283.

[2] *Intel Xeon Phi Coprocessor System Software Developers Guide.* Intel Corporation, 2013.

[3] *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Handbook on the Development and Optimization of Parallel Applications for Intel Xeon Processors and Intel Xeon Phi Coprocessors.* Colfax International, 2013.

[4] Z. Piotrowski, A. Wyszogrodzki, and P. Smolarkiewicz. Towards Petascale Simulation of Atmospheric Circulations with Soundproof Equations. *Acta Geophysica*, 59:1294–1311, 2011.

[5] K. Rojek and L. Szustak. Parallelization of EULAG Model on Multicore Architectures with GPU Accelerators. *Lect. Notes in Comp. Sci.*, 7204:391–400, 2012.

[6] K. Rojek, L. Szustak, and R. Wyrzykowski. Performance analysis for stencil-based 3D MPDATA algorithm on GPU architecture. *Proc. PPAM 2013, Lect. Notes in Comp. Sci.*, in print:11, 2013.

[7] P. Smolarkiewicz. Multidimensional Positive Definite Advection Transport Algorithm: An Overview. *Int. J. Numer. Meth. Fluids*, 50:1123–1144, 2006.

[8] J. Treibig, G. Wellein, and G. Hager. Efficient multicore-aware parallelization strategies for iterative stencil computations. *Journal of Computational Science*, 2:130–137, 2011.

[9] M. Wittmann, G. Hager, J. Treibig, and G. Wellein. Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters. *Parallel Process. Lett.*, 20 (4):359–376, 2010.

[10] R. Wyrzykowski, K. Rojek, and L. Szustak. Model-Driven Adaptation of Double-Precision Matrix Multiplication to the Cell Processor Architecture. *Parallel Computing*, 38:260–276, 2012.

[11] R. Wyrzykowski, K. Rojek, and L. Szustak. Using Blue Gene/P and GPUs to Accelerate Computations in the EULAG Model. *Lect. Notes in Comp. Sci.*, 7116:662–670, 2012.