Using Intel Xeon Phi Coprocessor to Accelerate Computations in MPDATA Algorithm

Lukasz Szustak[1(✉)], Krzysztof Rojek[1], and Pawel Gepner[2]

[1] Czestochowa University of Technology,
Dabrowskiego 69, 42-201 Czestochowa, Poland
{lszustak,krojek}@icis.pcz.pl
[2] Intel Corporation, Swindon, UK
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. 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.

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. Notably, this coprocessor executes the MPDATA algorithm 2 times faster than two Intel Xeon E5-2697 v2 CPUs, and 2.86 times faster than two Intel Xeon E5-2643 processors. Both the utilization of Intel Xeon Phi many cores and vectorization play the leading role in performance exploitation.

Keywords: EULAG model · Stencil computation · MPDATA · Intel Xeon Phi · Multi-/manycore programming · OpenMP · Adaptation

1 Introduction

The multidimensional positive definite advection transport algorithm (MPDATA) [7] is one of the two major parts of the dynamic core of the EULAG geophysical

R. Wyrzykowski et al. (Eds.): PPAM 2013, Part I, LNCS 8384, pp. 582–592, 2014.
DOI: 10.1007/978-3-642-55224-3_54,

model. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established computational model for simulating thermo-fluid flows across a wide range of scales and physical scenarios, including the numerical weather prediction (NWP).

The resent research of EULAG parallelization have been carried out using IBM Blue Gene/Q and CRAY XE6 [4]. Three-dimensional 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 %. In this study, we propose to rewrite the EULAG dynamical core and replace standard HPC systems by smaller heterogeneous clusters with accelerators such as GPU [5] and Intel Many Integrated Cores (MIC) [3].

Preliminary studies of porting anelastic numerical models to modern architectures, including hybrid CPU-GPU architectures, were carried out in works [5,10,11]. The results achieved for porting selected parts of EULAG to CPU-GPU architectures revealed potential in running scientific applications 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. The main focus is using MPDATA to modelling geophysical flows on NWP. The size of computational grid in such problems typically does not exceed 270 thousand grid points ($2048 \times 1024 \times 128$). Here, the starting point is an unoptimized parallel implementation of the MPDATA algorithm. In our work, we use OpenMP standard for multi- and many-core programming.

The content of the paper is organized as follow. In Sect. 2, architecture overview is outlined. The introduction to 3D MPDATA algorithm, including characterization of computation and communication, is presented in Sect. 3. Section 4 introduces the proposed approach to adaptation of MPDATA to Intel MIC Architecture, including block decomposition of 3D MPDATA algorithm, improving efficiency of block decomposition, and parallelization. Preliminary performance results are presented in Sect. 5, while Sect. 6 gives conclusions and future work.

2 Architecture Overview

2.1 Architecture of Intel Many Integrated Cores

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 real 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 onboard 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

A summary of key features of tested platforms is shown in Table 1. In this study, we use two platforms containing a single Intel Xeon Phi coprocessor. The first platform is equipped with two newest CPUs, based on the Ivy Bridge architecture, and the Intel Xeon Phi 3120A card. The second one includes two Sandy Bridge-EP CPUs, and the top-of-the-line Intel Xeon Phi 7120P coprocessor.

It should be noted that values of peak performance shown in Table 1 are given for the double precision arithmetic, with taking into account the usage of SIMD vectorization.

3 Introduction to MPDATA Algorithm

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. The full description of the MPDATA algorithm can be found in [6,7].

Table 1. Specification of tested platforms [1]

Product	Code name	# of cores (threads)	SIMD vector [bits]	Freq. [GHz]	Peak DP [GFlop/s]	Cache size [MB]	Memory size [GB]	Memory band. [GB/s]
Intel Xeon E5-2697 v2	Ivy Bridge	2×12 (2×24)	256	2.7	518	2×30	64	2×51.2
Intel Xeon Phi 3120A	Knights Corner	57 (228)	512	1.1	1003	28.5	6	240
Intel Xeon E5-2643	Sandy Bridge-EP	2×4 (2×8)	256	3.3	211	2×10	64	2×51.2
Intel Xeon Phi 7120P	Knights Corner	61 (244)	512	1.238	1208	30.5	16	352

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: prior outcomes from stages are usually input data for the subsequent computations. A part of the MPDATA implementation is shown in Fig. 1. It corresponds to the linear version of MPDATA [7].

A single MPDATA time step requires 5 input and 1 output matrices. Other 16 matrices are temporary, and do not play role in the further computational steps. In the basic, unoptimized implementation of the MPDATA algorithm, every stage uses 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 heavy 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 take full advantage of the novel architecture, the adaptation of MPDATA to the Intel MIC architecture is considered. The new implementation takes into account the memory-bounded character of the algorithm.

4 Adaptation of MPDATA to Intel MIC Architecture

4.1 Block Decomposition of 3D MPDATA Algorithm

Since the MPDATA algorithm includes so many intermediate computation, one of the primary methods for reducing the intensity 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. Such treatment 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. The grid is partitioned into blocks. Every block provides computation for all the 17 stages on the assigned part of the grid. Within a single block,

```
#define fdim(a, b) ( (a>b) ? (a-b):(0.0) )
#define donor(y1, y2, a) ( fdim(a, 0.0) * (y1) - fdim(0.0, a) * (y2) )
//stage 1
for( ... ) // i - dimension
 for( ... ) // j - dimension
 for( ... ) // k - dimension
 f1[i][j][k] = donor(xIn[i-1][j][k],xIn[i][j][k],u1[i][j][k]);
//stage 2
for( ... ) // i - dimension
 for( ... ) // j - dimension
 for( ... ) // k - dimension
 f2[i][j][k] = donor(xIn[i][j-1][k],xIn[i][j][k],u2[i][j][k]);
//stage 2
for( ... ) // i - dimension
 for( ... ) // j - dimension
 for( ... ) // k - dimension
      f3[i][j][k] = donor(xIn[i][j][k-1],xIn[i][j][k],u3[i][j][k]);
//stage 4
for( ... ) // i - dimension
  for( ... ) // j - dimension
    for( ... ) // k - dimension
      x[i][j][k] = xIn[i][j][k]-(f1[i+1][j][k]-f1[i][j][k]+f2[i][j+1][k]-
                   f2[i][j][k]+f3[i][j][k+1]-f3[i][j][k])/h[i][j][k];
/*...*/
```

Fig. 1. Part of MPDATA implementation

each stage computes the adequate chunk of the corresponding matrix. Executing of a sequence of blocks determines the final outcomes 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. 2. This decomposition determines four dimensions of distribution of MPDATA calculation across computing resources: i-, j-, and k-dimensions relate to the grid partitioning, while s-dimension is associated with the order of executing MPDATA stages.

Due to data dependencies between subsequent stages additional computation and communication within each data block are required. These overheads are needed on the borders between adjacent blocks. In the proposed method, the computation associated with all the 17 stages, and executed within each block are extended by adequate halo areas. Adding halo allows to avoid data dependency 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,

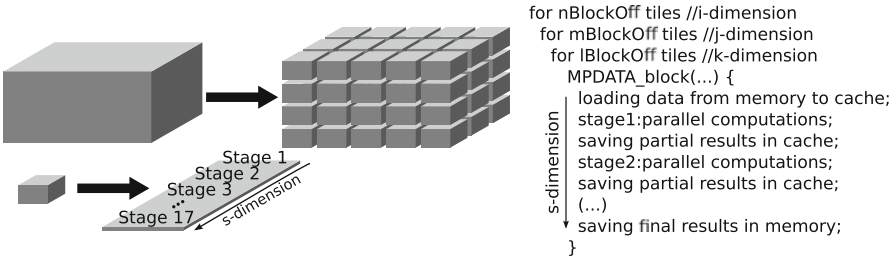

Fig. 2. Idea of block decomposition of MPDATA computation

Table 2. Sizes of halo areas for MPDATA algorithm

Halo areas	Matrices																				
	Input					Temporary															Output
	u1	u2	u3	h	x	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
iL	2	2	2	2	3	2	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0
iR	3	2	2	2	3	3	2	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0
jL	2	2	2	2	3	2	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0
jR	2	3	2	2	3	2	3	2	2	1	2	1	1	1	1	1	1	1	0	1	0	0
kL	2	2	2	2	3	2	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0
kR	2	2	3	2	3	2	2	3	2	1	1	2	1	1	1	1	1	1	0	0	1	0

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. Table 2 presents the sizes of halo areas in i-, j-, and k-dimensions for chunks of all the matrices.

This approach allows us to avoid data 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 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 used platforms. The main difficulty here results from 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 computation 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 (Table 2), 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. 3.
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.

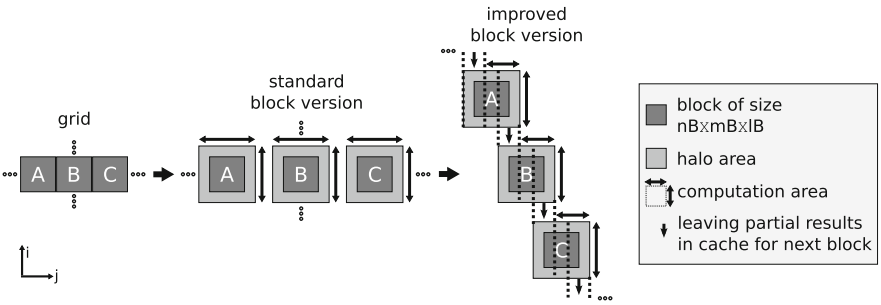

Fig. 3. Idea of leaving partial results in cache memory

4.3 Parallelization

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.

Different MPDATA blocks are processed sequentially, following the order proposed for the CPU block decomposition in the previous subsection (Fig. 3). For a fixed MPDATA block, a sequence of stages is executed, taking into account the adequate sizes of halo areas. All computation executed within every stage are distributed across available threads. Assigned chunk to each stage is partitioned into sub-chunk of size $nB^* \times mB^* \times lB$, where partitioning takes place along i and j dimensions. As a result, a task is assigned to each thread, as a part of MPDATA block. Due to the data dependencies of MPDATA, appropriate synchronizations between MPDATA stages are necessary.

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.

Because of intra-cache communication between tasks, the overall system performance depends strongly on a chosen task placement onto available 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. This increases the sustained intra-cache bandwidth, as well as reduces cache misses, and the latency of access to the cache memory.

5 Preliminary Performance Results

In this section we present preliminary performance results obtained for the double precision 3D MPDATA algorithm on the platforms introduced in Sect. 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 for our approach is 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 are implemented and tested. These four stages correspond to the linear version of the MPDATA algorithm. 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.

Figure 4 presents the normalized execution time of the 3D MPDATA algorithm, for 500 time steps and the grid of size $1022 \times 512 \times 63$. The achieved performance results correspond to the following setups: (a) comparison of the block and improved block versions; (b) advantages of using vectorization; (c) performance for different numbers of threads per core; (d) comparison of Intel Xeon CPU and Intel Xeon Phi (best configurations with SIMD).

Figure 4a presents 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 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. 4b).

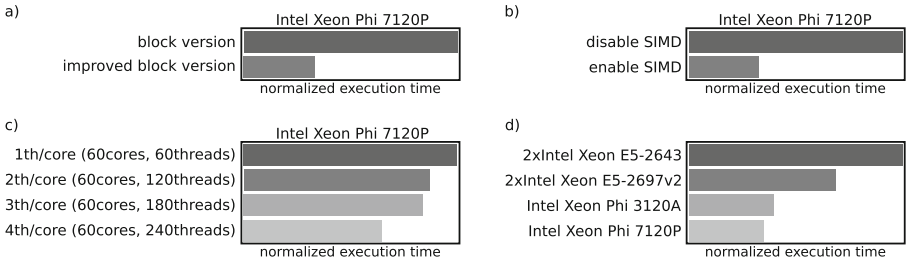

Fig. 4. Preliminary performance results: (a) comparison of block and improved block versions; (b) advantages of using vectorization; (c) performance for different numbers of threads per core; (d) comparison of Intel Xeon CPU and Intel Xeon Phi (best configurations with SIMD)

Figure. 4c 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. 4d. 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.

6 Conclusions and Future Work

Using the Intel Xeon Phi coprocessor to accelerate computations in the 3D MPDATA algorithm is a promising direction for developing the parallel implementation of this algorithm. 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.

In all the performed tests, the Intel Xeon Phi 7120P coprocessor gives the best performance results. This coprocessor executes the MPTADA algorithm 2 times faster than two Intel Xeon E5-2697 v2 CPUs, totally containing 24 cores, and 2.86 times faster than two Intel Xeon E5-2643. Both the manycore

and vectorization features of the Intel MIC architecture play the leading role in the performance exploitation. The other important features are the 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.

At this point of our research, only the first four stages of the MPDATA algorithm are implemented, and tested. They correspond to the linear part of MPDATA. The performance achieved for this part of MPDATA is still limited by memory traffic, mostly because all the input data of the whole MPDATA algorithm are required to provide the correctness of computation for the linear part. As a result, the tested part of MPDATA does not extract the full potential of applying this coprocessor to implement MPDATA computation. Moreover, since the remaining part is unleashed from the memory-cache communication, it gives the opportunity for increasing the efficiency of computation. Implementing and optimizing this part of MPDATA will be studied in future works.

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.

Acknowledgments. This work was supported in part by the Polish National Science Centre under grant no. UMO-2011/03/B/ST6/03500.

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

References

1. Intel Architectures Comparison. http://ark.intel.com/pl/compare/75799,75797,64587,75283
2. Intel: Intel Xeon Phi Coprocessor System Software Developers Guide. Intel Corporation (2013)
3. Colfax International: 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. Piotrowski, Z., Wyszogrodzki, A., Smolarkiewicz, P.: Towards petascale simulation of atmospheric circulations with soundproof equations. Acta Geophys. **59**, 1294–1311 (2011)
5. Rojek, K., Szustak, L.: Parallelization of EULAG model on multicore architectures with GPU accelerators. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 391–400. Springer, Heidelberg (2012)

6. Rojek, K., Szustak, L., Wyrzykowski, R.: Performance analysis for stencil-based 3D MPDATA algorithm on GPU architecture. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part I. LNCS, vol. 8384, pp. 145–154. Springer, Heidelberg (2014)
7. Smolarkiewicz, P.: Multidimensional positive definite advection transport algorithm: an overview. Int. J. Numer. Meth. Fluids **50**, 1123–1144 (2006)
8. Treibig, J., Wellein, G., Hager, G.: Efficient multicore-aware parallelization strategies for iterative stencil computations. J. Comput. Sci. **2**, 130–137 (2011)
9. Wittmann, M., Hager, G., Treibig, J., Wellein, G.: Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters. Parallel Process. Lett. **20**(4), 359–376 (2010)
10. Wyrzykowski, R., Rojek, K., Szustak, L.: Model-driven adaptation of double-precision matrix multiplication to the cell processor architecture. Parallel Comput. **38**, 260–276 (2012)
11. Wyrzykowski, R., Rojek, K., Szustak, L.: Using blue gene/P and GPUs to accelerate computations in the EULAG model. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2011. LNCS, vol. 7116, pp. 670–677. Springer, Heidelberg (2012)