Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 642705, 14 pages
http://dx.doi.org/10.1155/2015/642705

Research Article

Adaptation of MPDATA Heterogeneous Stencil Computation to Intel Xeon Phi Coprocessor

Lukasz Szustak,[1] Krzysztof Rojek,[1] Tomasz Olas,[1] Lukasz Kuczynski,[1] Kamil Halbiniak,[1] and Pawel Gepner[2]

[1] *Czestochowa University of Technology, Częstochowa, Poland*
[2] *Intel Corporation, Pipers Way, Swindon, Wiltshire SN3 1RJ, UK*

Correspondence should be addressed to Lukasz Szustak; lszustak@icis.pcz.pl

Received 28 May 2014; Accepted 2 November 2014

Academic Editor: Ewa Deelman

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. In this work, we outline an approach to adaptation of the 3D MPDATA algorithm to the Intel MIC architecture. In order to utilize available computing resources, we propose the $(3 + 1)$D decomposition of MPDATA heterogeneous stencil computations. This approach is based on combination of the loop tiling and fusion techniques. It allows us to ease memory/communication bounds and better exploit the theoretical floating point efficiency of target computing platforms. An important method of improving the efficiency of the $(3+1)$D decomposition is partitioning of available cores/threads into work teams. It permits for reducing inter-cache communication overheads. This method also increases opportunities for the efficient distribution of MPDATA computation onto available resources of the Intel MIC architecture, as well as Intel CPUs. We discuss preliminary performance results obtained on two hybrid platforms, containing two CPUs and Intel Xeon Phi. The top-of-the-line Intel Xeon Phi 7120P gives the best performance results, and executes MPDATA almost 2 times faster than two Intel Xeon E5-2697v2 CPUs.

1. Introduction

In the last years, we can observe that the computational power of processors has been rising much more faster than the memory bandwidth. As a result, modern processor architectures are very unbalanced concerning the relation of theoretical peak performance versus memory bandwidth [1]. One of the main problems of porting codes to the latest computing platforms is to take the full advantage of memory hierarchies.

Intel MIC is a novel architecture for high performance computing [2–4]. It contains a large number of cores and wide vector processing units. The Intel Xeon Phi coprocessor is the first product based on this architecture. It offers notable performance advantages over traditional processors and supports practically the same traditional parallel programming model. Although it is designed for massively parallel applications, there is still an open question of how scientific applications can utilize the computing power of Intel MIC. The primary challenge is an efficient utilization of available computing resources which correspond to 57–61 cores and powerful vectors units with 512-bit width. When tackling this challenge, the main issue is delivering on-time data required for computations, taking into account features of the Intel MIC's main memory and cache hierarchy. Firstly, the main memory bandwidth should be shared across more than 228 threads. Secondly, apart from providing an efficient placement of data in the cache hierarchy, the distributed structures of L2 cache force programmers to minimize the intercache traffic between cores.

In this work, the efficient adaptation of the multidimensional positive definite advection transport algorithm (MPDATA) to the Intel MIC architecture is investigated. As one of the most time-consuming routines, MPDATA is among the two major parts of the dynamic core of the EULAG geophysical model [5–7]. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established numerical model for simulating thermofluid flows across a wide range of scales and physical scenarios, including the numerical weather prediction (NWP). The structure of MPDATA consists of a set of heterogeneous stencils, where each stencil may depend on one or more others. Stencil computations are widely used in scientific algorithms and simulations [8–10]. In these computations, each point in a data grid is updated based on its neighbours [9] according to a fixed rule. MPDATA requires the loading of 5 input matrices and returns only one. We currently focus on the use of MPDATA in NWP, where the size of grids is limited by $n \leq 2048$, $m \leq 1024$, and $l = [64, 128]$. In our previous work [11], it has been shown that MPDATA is strongly memory-bounded.

In this paper, we show how to use some of optimization methods that we found effective, and demonstrate their impact on the performance of both Intel MIC and CPU architectures. The proposed adaptation of MPDATA to Intel MIC is based on the (3 + 1)D decomposition of MPDATA heterogeneous stencil computations, using combination of loop tiling and loop fusion techniques. It allows us to ease the memory and communication bounds and better exploit the floating point efficiency of target computing platforms. Another contribution of the paper is a method for increasing efficiency of computations by reducing intercache communications. This method is based on the partitioning of available cores/threads into independent work teams. This paper is an extended version of work presented in [1, 12]. This study not only proposes modifications in the (3 + 1)D decomposition of MPDATA, but also introduces the notion of work team partitioning.

The paper is organized as follows. Related works are outlined in Section 2, while Section 3 presents the target Intel MIC architecture. Section 4 presents an overview of the MPDATA algorithm. Sections 5 and 6 introduce the (3 + 1)D decomposition of MPDATA and the method of the partitioning of cores into independent work teams, respectively. An approach to the MPDATA parallelization based on exploiting the task and data parallelism is described in Section 7. Preliminary performance results are presented in Section 8, while Section 9 gives conclusions and future work.

2. Related Work

In our previous works [7, 13], we proposed two decompositions that provide the adaptation of MPDATA computations to CPU and GPU architectures separately. The achieved performance results showed the possibility of achieving high performance both on CPU and GPU platforms. Recently, we have developed [14] a hybrid CPU-GPU version of 2D MPDATA, to fully utilize all the available computing resources by spreading computations across the entire machine. To reveal performance constraints for the MPDATA algorithm running on hybrid architectures, we follow the simple methodology presented in [8], where the attainable performance is estimated based on the flop-per-byte ratio.

Preliminary studies of porting anelastic numerical models to modern architectures, including hybrid CPU-GPU architectures, were carried out in works [15, 16]. The results achieved for porting selected parts of EULAG to nontraditional architectures revealed a considerable potential in running scientific applications, including anelastic numerical models, on novel hardware architectures.

The newest research of EULAG parallelization on conventional HPC architectures has been carried out using IBM Blue Gene/Q and CRAY XE6 [17, 18]. The 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. However, when parallelizing EULAG computations on supercomputers and CPU clusters, the efficiency is declined below 10%.

The MPDATA algorithm is a collection of stencils kernels, which are commonly known as memory-bounded [10, 19]. Such kernels have been investigated by many authors over the years [8, 9, 14, 20–24]. The main direction of memory optimizations for stencil computations has principally focused on different decomposition strategies, like space and temporal blocking techniques [20], that attempt to exploit locality by performing operations on data blocks of a suitable size before moving on to the next block. These strategies have been used to improve the efficiency of implementing stencil codes in the context of variety of multi-/manycore architectures (see, e.g., [14, 22, 25]). The main assumption for using the temporal blocking method is that no other computations need to be performed between consecutive stencils (or stages). This assumption has been aggressively used by us in [14] to improve the efficiency of implementing 2D stencil codes on hybrid CPU-GPU platforms by removing or delaying synchronization between stages.

The Intel MIC architecture is a relatively fresh computing platform; however, the management of memory hierarchy has been the target of optimizations in the past. In particular, the performance evaluation of sparse-matrix multiplication kernels on the Intel Xeon Phi coprocessor was presented in [4]. The authors show that Xeon Phi's sparse kernels performance is very promising and even better than that of cutting-edge CPUs and GPUs. This is mostly due to the Xeon Phi's wide registers and vectorization capabilities. Additionally, they report that a relatively small size of L2 cache per core is not a problem for the coprocessor, but having 61 cores induces a significant intercache traffic overhead. In this paper, we observe a similar problem and propose how to solve it for MPDATA heterogeneous stencil computations.

Some results on porting stencil computation on Intel Xeon Phi were presented in [26], where the regular 7-point 3D stencil kernel was investigated. After observing that its performance is bounded by memory access, the cache blocking is used by dividing the grid into multiple blocks of size $B_i \times B_j \times B_k$. As compared with the naive parallelization, this approach allows the authors to increase the performance by 64% or 23%, depending on the grid size. The memory behavior of stencil codes related to their performance on Xeon Phi was the primary focus of paper [27], where different types of regular stencils were studied. In contrast to these researches, where only regular (or homogeneous) stencils codes were ported to the Intel MIC architecture, a much more complex case of heterogeneous stencils is considered in this paper.

3. Architecture Overview

3.1. Intel MIC Architecture. The Intel MIC architecture combines many Intel CPU cores onto a single chip [28]. 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 [28, 29]. Each coprocessor contains more than 50 cores clocked at 1 GHz or more. These cores support four-way hyperthreading, which gives more than 200 logical cores. The actual number of cores (from 57 to 61) depends on the generation and model of a specific coprocessor. Each core features an in-order, dual-issue ×86 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 exceed 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) [28], that significantly increases the computing power. Each VPU supports a new 512-bit SIMD instruction set called Intel Initial ManyCore Instructions. The new ability to work with 512-bit vectors enables the fact of operating on 16 float or 8 double elements, instead of a single one.

The Intel Phi coprocessor is delivered in form factor of a PCI express device and can not be used as a stand-alone processor. Since the Intel Xeon Phi coprocessor runs the 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).

3.2. Target Platforms. In this study, we use two platforms containing a single Intel Xeon Phi coprocessor. The first platform is equipped with the two newest Intel Xeon E5-2697 v2 CPUs (total of 2 × 12 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 E5-2665 CPUs (2 × 8 cores in total) and the top-of-the-line Intel Xeon Phi 7120P coprocessor (61 cores). The peak performances of these platforms are 1521.6 (2 × 259.2 + 1003.2) GFlop/s and 1515.5 (2 × 153.6 + 1208.3) GFlop/s, respectively. These values are given for the double precision arithmetic, 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 is shown in Table 1.

Table 1: Specification of tested platforms [36].

	First platform		Second platform	
Product	Intel Xeon E5-2697 v2	Intel Xeon Phi 3120A	Intel Xeon E5-2665	Intel Xeon Phi 7120P
Code name	Ivy Bridge	Knights Corner	Sandy Bridge-EP	Knights Corner
Number of cores	2 × 12	57	2 × 8	61
Number of threads	2 × 24	228	2 × 16	244
SIMD length [bits]	256	512	256	512
Freq. [GHz]	2.7	1.1	2.4	1.238
Peak [GFlop/s]	518	1003	307	1208
LLC* size [MB]	2 × 30	28.5	2 × 20	30.5
Memory size [GB]	64	6	64	16
Memory bandwidth [GB/s]	2 × 51.2	240	2 × 51.2	352

*Last level cache (LLC) refers to L3 cache for CPU, and to L2 cache for Intel MIC.

4. Outline of MPDATA

MPDATA belongs to the group of nonoscillatory forward-in-time algorithms and performs a sequence of stencil computations [5, 30]. The 3D MPDATA algorithm is based on the first-order-accurate advection equation:

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

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

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

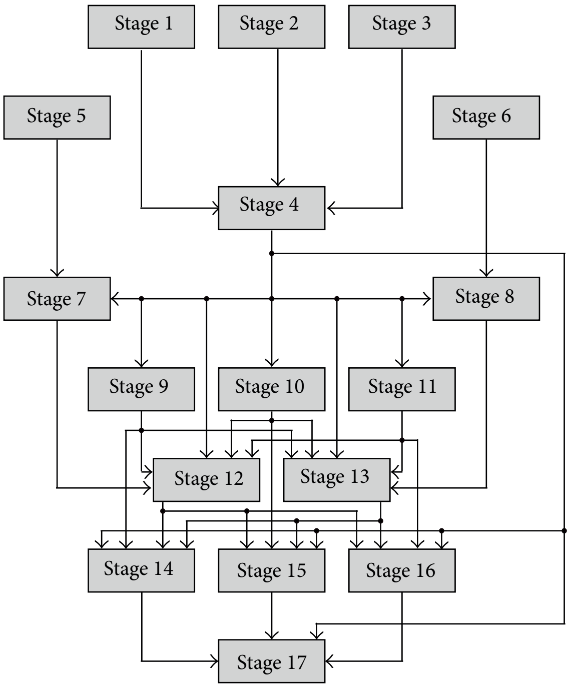


Figure 1: Data dependency graph for MPDATA.

$$
\begin{aligned}
&- \left[F\left(\Psi^n_{i,j,k}, \Psi^n_{i,j,k+1}, W_{i,j,k+1/2}\right) \right. \\
&\qquad \left. -F\left(\Psi^n_{i,j,k-1}, \Psi^n_{i,j,k}, W_{i,j,k-1/2}\right)\right], \\
&U \equiv \frac{u\delta t}{\delta x}, \qquad [U]^+ \equiv 0.5\left(U + |U|\right), \\
&[U]^- \equiv 0.5\left(U - |U|\right).
\end{aligned}
\tag{2}
$$

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 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 [5, 11].

As a part of the EULAG model, the MPDATA algorithm is interleaved with other important computation (e.g., elliptic solver) in each time step [31]. It limits the possibility to apply any optimization beyond a single time step and, as a consequence, the adaptation is performed inside each time step.

The whole MPDATA computations in each time step are decomposed into a set of 17 heterogeneous stencils, called further stages. Each stage is responsible for calculating elements of a certain matrix, based on the corresponding stencil. The stages depend on each other, where outcomes from prior stages are usually input data for the subsequent computations (Figure 1).

A single MPDATA time step requires 5 input and one output matrices; the other 16 matrices are intermediate ones. In the basic, unoptimized implementation of the MPDATA algorithm (Algorithm 1), every stage reads a required set of matrices from the main memory and writes results to the main memory after computation. In consequence, a significant traffic to the main memory is generated. Moreover, compute units (cores/threads, and vector units) have to wait for data transfers from the main memory to the cache hierarchy.

For example, each loop iteration in the first stage reads one value of $u1$- and two values of x In-matrices and then performs two multiples, two subtractions, and the max and min operations, and finally writes one value to the xOut-matrix. For double precision floating point element, this gives 32 bytes of transferred data. Thus, the operational intensity is 6 flops/32 bytes = 0.187 (flop/byte), assuming that caches exploit no reuse. As most modern HPC platforms, processors can sustain at about several times as much instruction throughput as DRAM data traffic [10, 19], so it is crucial to reduce the main memory data transfers. In order to better utilize features of novel accelerators, the adaptation of MPDATA computations to the Intel MIC architecture is considered in this work, taking into account the memory-bounded character of the algorithm.

5. (3 + 1)D Decomposition of MPDATA Heterogeneous Stencil Computations

5.1. Basic Concepts of Adaptation of MPDATA to Intel Xeon Phi Coprocessor. The main aim of this work is to develop an optimal parallel version of MPDATA heterogeneous stencil computations that allows us to take the maximum advantage of Intel Xeon Phi coprocessor, as well as modern CPUs. First of all, the memory-bounded character of the MPDATA algorithm must be alleviated in order to reveal the potential of the available computing resources. To work around this limitation, the $(3+1)$D decomposition of MPDATA heterogeneous stencil computation is proposed. This decomposition is based on a block decomposition using mixture of loop tiling and loop fusion techniques. The prime assumption here is to reduce a saturation of the main memory traffic.

This goal can be achieved by taking advantage of cache memory reuse by shifting the data transfers from the main memory to the cache hierarchy. After such modification, the available computing resources are able to execute MPDATA computations more efficiently. Hence, the $(3+1)$D decomposition of MPDATA computation is considered firstly, while utilization of cores and vector units is taken into account subsequently. Further optimizations include partitioning of threads into work teams to improve the efficiency of intra-cache communication and synchronization, as well as the MPDATA block parallelization based on the task and data parallelisms.

5.2. Block Decomposition Using a Mixture of Loop Tiling and Loop Fusion Techniques. Since 3D MPDATA algorithm includes so many intermediate computations, one of the primary methods for reducing the memory traffic within each time step is to avoid data transfers associated with these computations. It also allows us to improve the cache reuse and

```
//stage 1
for i-dim
  for j-dim
    for k-dim
      f1[i][j][k] = xIn[i][j][k] * max(0.0, u1[i][j][k])
                  - xIn[i-1][j][k] * (-min(0.0, u1[i][j][k]));
//stage 2
for i-dim
  for j-dim
    for k-dim
      f2[i][j][k] = xIn[i][j][k] * max(0.0, u2[i][j][k])
                  - xIn[i][j-1][k] * (-min(0.0, u2[i][j][k]));
//stage 3
for i-dim
  for j-dim
    for k-dim
      f3[i][j][k] = xIn[i][j][k] * max(0.0, u3[i][j][k])
                  - xIn[i][j][k-1] * (-min(0.0, u3[i][j][k]));
//stage 4
for i-dim
  for j-dim
    for k-dim
      xOut[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];
/*···*/
```

Algorithm 1: Part of MPDATA implementation.

operational intensity ratio. The main requirement is to keep all the intermediate results in the cache memory only. When the intermediate results will be held in the cache hierarchy, the memory traffic will be generated only to transfer the required input and output data for each MPDATA time step. This aim can be achieved by using a mixture of two well-known loop optimization techniques [32, 33]: loop tiling and loop fusions. Both techniques are most often used to maximize the operational intensity ratio, reduce the loop overheads, increase the instruction parallelism, and improve the cache locality [33, 34].

The idea of block decomposition using the mixture of loop tiling and loop fusion techniques is shown in Algorithm 2. The proposed block decomposition is based on the loop tiling technique, then adding the loop fusion to the tiling (block) management level, and finally reusing the loop fusion, but on the intratile level.

The starting point of the proposed block decomposition is applying the loop tiling technique for the original version of the MPDATA code. This process is applied for each stage separately, where the loop's iteration space of every stage is partitioned into smaller chunks or blocks. Naturally, the code does not require heavy modifications. Moreover, this technique is also commonly used by compilers to make the execution of certain types of loops more efficient [32]. Algorithm 2(b) presents an example of such modifications of the MPDATA code.

Enabling the loop tiling for all the stages separately does not give the desired performance gain. This is mainly due to the fact that each stage is still characterized by a relatively small arithmetic intensity ratio, and the main memory traffic associated with intermediate computations is not reduced. However, this step is necessary for the further optimization steps based on the loop fusion technique.

The loop fusion optimization assumes merging the selected loops, in order to reduce the loop overheads, increase the instruction parallelism, improve the data locality, and even reduce data transfers [32–34]. Taking into account all the 17 MPDATA stages after applying the loop tiling (each stage corresponds now to 6 loops, see Algorithm 2(b)), the loop fusion optimization can be successfully applied for loops associated with tilling management that correspond to indices *nBlockOff*, *mBlcokOff*, and *kBlockOff*. In consequence, all these loops now are merged (Algorithm 2(c)).

As a result of using the combination of these two optimization techniques, the MPDATA grid is partitioned into some MPDATA blocks, where subsequent blocks are processed one by one. Every block includes all the stages that perform MPDATA computations on chunks of the corresponding matrices. Furthermore, each computational block is processed in parallel by the available computing units. The primary aim here is the possibility to avoid main memory data transfers associated with all intermediate computations. This advantage can be achieved when relevant data, which are required for computing every MPDATA block, are kept in the cache hierarchy. Hence, the size $nB \times mB \times lB$ of each block has to be suited to the cache size. While the intermediate

```
(a)
for i-dim
    for j-dim
        for k-dim
            S1 = ...
/ ∗ ··· ∗ /
for i-dim
    for j-dim
        for k-dim
        S3 = ...
/ ∗ ··· ∗ /
for i-dim
    for j-dim
        for k-dim
            S14 = ...
/ ∗ ··· ∗ /
for i-dim
    for j-dim
        for k-dim
            S17 = ...
(b)
for nBlockOff tiles
    for mBlockOff tiles
        for lBlockOff tiles {
            for i-dim
                for j-dim
                    for k-dim
                        S1 = ...
        }
/ ∗ ··· ∗ /
for nBlockOff tiles
    for mBlockOff tiles
        for lBlockOff tiles {
            for i-dim
                for j-dim
                    for k-dim
                        S3 = ...
        }
/ ∗ ··· ∗ /
for nBlockOff tiles
    for mBlockOff tiles
        for lBlockOff tiles {
            for i-dim
                for j-dim
                    for k-dim
                        S17 = ...
        }
(c)
for nBlockOff tiles
    for mBlockOff tiles
        for lBlockOff tiles {
            for i-dim
                for j-dim
                    for k-dim
                        S1 = ...
            / ∗ ··· ∗ /
            for i-dim
                for j-dim
                    for k-dim
                        S3 = ...
            / ∗ ··· ∗ /
```

Algorithm 2: Continued.

```
      for $i$-dim
        for $j$-dim
          for $k$-dim
            S14 = . . .
      / ∗ ⋯ ∗ /
      for $i$-dim
        for $j$-dim
          for $k$-dim
            S17 = . . .
    }
(d)
for $n$BlockOff tiles
  for $m$BlockOff tiles
    for $l$BlockOff tiles {
      for $i$-dim
        for $j$-dim
          for $k$-dim {
            S1 = . . .
            S2 = . . .
            S3 = . . .
          }
        / ∗ ⋯ ∗ /
        for $i$-dim
          for $j$-dim
            for $k$-dim {
              S14 = . . .
              S15 = . . .
              S16 = . . .
            }
        / ∗ ⋯ ∗ /
        for $i$-dim
          for $j$-dim
            for $k$-dim {
              S17 = . . .
            }
    }
```

Algorithm 2: Idea of block decomposition: (a) basic scheme of MPDATA code, codes after applying loop tiling (b), loop fusion on tiles management level (c), and loop fusion on intratile level (d), where S1, S2, . . ., and S17 are exemplary MPDATA stages.

results are stored in cache, the main memory traffic is only generated for transfers of required input and output data.

However, the heterogeneous nature of the MPDATA stages makes it difficult to implement the proposed block decomposition. It is due to the stencil structure of dependencies, as well as the dependencies between stages, where outputs of the stages are usually input data for the next one. Since each MPDATA block provides computations for all the stages, some extra calculations are required for each block. As a negative effect, each block also requires more input data, so some extra transfers to the main memory are required.

Extra calculation and communication take place on the borders between adjacent blocks. Thus, blocks have to be extended by adequate halo areas. Opposite to the well-known ghost cell expansion strategy [10, 35], which is commonly used across homogeneous stencils corresponding to successive time steps, the ghost halo expansion in the proposed (3 + 1)D decomposition deals with heterogeneous stencils corresponding to 17 MPDATA stages within a single time step. The sizes of halo areas are determined in three dimensions: i, j, and k, according to the data dependencies between subsequent MPDATA stages placed along the fourth dimension s. Hence, each of 5 input, 16 temporary, and one output matrices is partitioned into chunks of size $nB \times mB \times lB$, which are further expanded by adequate halo areas of sizes iL and iR, jL and jR, and kL and kR, respectively. Table 2 presents sizes of ghost halo areas for the MPDATA algorithm.

The proposed approach allows us to reduce the main memory traffic at the cost of extra computations associated with halo areas in chunks of intermediate matrices. Another advantage of this approach is the possibility of reducing the main memory consumption because all the intermediate results are stored only in the cache memory. In the case of coprocessors, it plays an important role because the size of the on-board main memory is fixed and it is significantly smaller than that in the case of traditional CPU solutions. The requirement of expanding halo areas is one of the major difficulties when applying the proposed approach, taking into

Table 2: Sizes of ghost halo areas for MPDATA.

	iL	iR	jL	jR	kL	kR
Input matrices						
$u1$	2	3	2	2	2	2
$u2$	2	2	2	3	2	2
$u3$	2	2	2	2	2	3
h	2	2	2	2	2	2
$x\text{In}$	3	3	3	3	3	3
Intermediate matrices						
S1	2	3	2	2	2	2
S2	2	2	2	3	2	2
S3	2	2	2	2	2	3
S4	2	2	2	2	2	2
S5	1	2	1	1	1	1
S6	1	1	1	2	1	1
S7	1	1	1	1	1	2
S8	1	1	1	1	1	1
S9	1	1	1	1	1	1
S10	1	1	1	1	1	1
S11	1	1	1	1	1	1
S12	1	1	1	1	1	1
S13	1	1	1	1	1	1
S14	0	1	0	0	0	0
S15	0	0	0	1	0	0
S16	0	0	0	0	0	1
Output matrix (xOut)						
S17	0	0	0	0	0	0

account data dependencies between MPDATA stages and the heterogeneous nature of MPDATA stencils.

Although some memory transfers are now eliminated, the transfers associated with the input and output data still have a negative impact on the overall system performance. In particular, it is noticeable for the first stages that they are strongly dependent on the input data. This constraint can be alleviated by the reuse of the loop fusion inside each MPDATA block. It is applicable because some stages depend on the same MPDATA input data (see Algorithm 1). Hence, compressing (merging) the appropriately chosen stages (loops) into packages of stages leads to the reduction of the main memory traffic for the input data. As a result, all the stages included within every block are compressed into 6 packages of stages, according to MPDATA data dependencies. This strategy also allows us to reduce the loop overheads and improves the data cache locality. The main disadvantage is the uniform range of loops for the MPDATA stages contained in each package. Thus, the loops ranges of some stages have to be increased and, as a consequence, unnecessary calculations will be performed. Algorithm 2(d) uncovers an effect of applying the loop fusion technique inside MPDATA blocks.

Summarizing all of the above, the MPDATA grid is partitioned according to i-, j-, and k-dimensions into several blocks. Every block is responsible for computing all the 17 stages placed along the fourth dimension s, taking into account an appropriate ghost expansion. This aim is achieved by a mixture of the loop tiling and loop fusion techniques (Algorithm 2(c)). Within a single block, each stage provides computations for an adequate extended chunk of the corresponding matrix. Subsequently, the selected stages are compressed into packages according to the loop fusion optimization applied inside every block (Algorithm 2(d)). Therefore, subsequent blocks are processed one by one, and each computational block is processed in parallel by available computing units. The execution of a sequence of blocks determines the final output result for a single MPDATA time step.

The proposed (3 + 1)D decomposition refers to the four directions of distribution of the MPDATA computation, where i-, j-, and k-dimensions are related to the grid partitioning, while the fourth s-dimension is associated with a particular order of executing MPDATA stages (packages). This decomposition corresponds to the partitioning of the MPDATA computational domain which is illustrated in Figure 2.

5.3. Improving Efficiency of Decomposition. Although the block decomposition of MPDATA allows for the reduction of the memory traffic, it still does not guarantee a satisfying utilization of target platforms. It is mainly due to the cost of extra computations and communications, which have impact on the performance degradation. In particular, the three groups of extra computation and communication can be selected according to the 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 $l \in [64, 128]$.

(2) The overheads associated with j-dimension are avoided by leaving partial results in the cache memory. It becomes possible when extra computations are repeated by adjacent blocks. In this case, some results of intermediate computation have to reside in cache for executing the next block. Such an approach corresponds to a temporal and space blocking techniques [10, 24]. This rule requires us to develop a flexible management of computations 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 Figure 3.

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

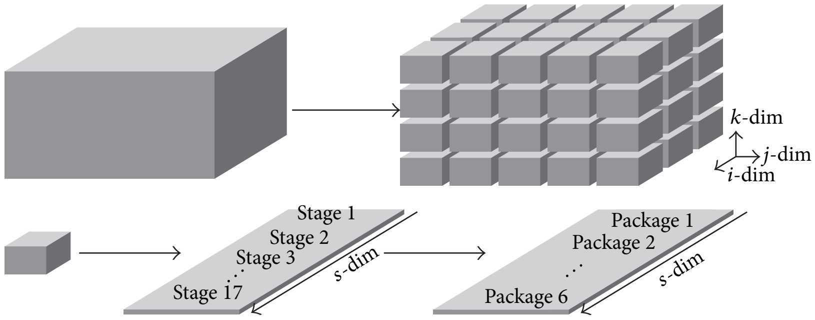


Figure 2: Idea of MPDATA domain decomposition.

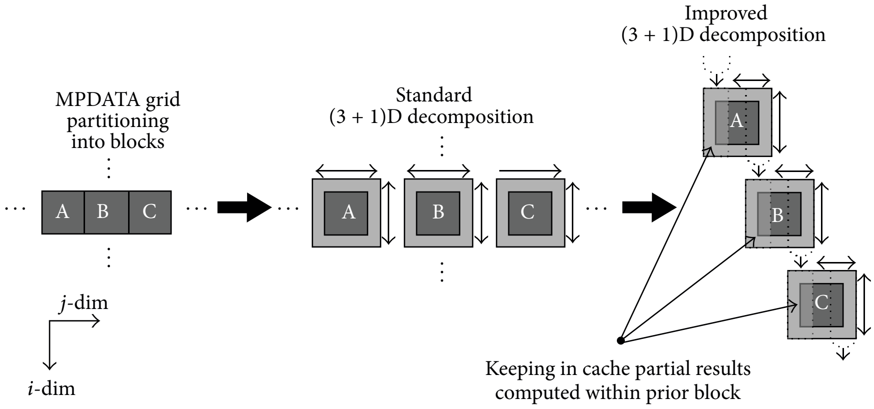

MPDATA grid partitioning into blocks
A
B
C
j-dim
i-dim
Standard (3 + 1)D decomposition
A
B
C
Improved (3 + 1)D decomposition
A
B
C
Keeping in cache partial results computed within prior block

Figure 3: Idea of leaving partial results in cache memory.

6. Partitioning of Cores/Threads into Independent Work Teams

Another method of improving the efficiency of the proposed (3 + 1)D decomposition is the partitioning of available cores/threads into independent work teams. The (3 + 1)D decomposition moves the data transfers from the main memory to the cache hierarchy. In consequence, a lot of inter- and intracache communications are generated because of dependencies between stages. Particularly, it is noticeable when more than 200 Intel MIC's threads cooperate, using L2 caches distributed across cores.

To alleviate this overhead, all the cores/threads are partitioned into independent work teams. This aim is achieved by performing some extra computations within every work team. The MPDATA grid is distributed into P pieces of size $nP \times mP \times lP$ which are extended by adequate halo areas (see Table 2). All computations within each piece are processed by a single work team of threads, according to the proposed (3 + 1)D decomposition. Hence, each piece is decomposed into some MPDATA blocks, where subsequent blocks are processed one by one, and each computational block is processed in parallel by the corresponding work team. Figure 4 illustrates the idea of partitioning of Intel MIC's processing cores into independent work teams.

The proposed method allows us to reduce inter- and intracache communication overheads due to communication between neighbor threads, as well as their synchronization. This method also increases opportunities for the efficient load distribution of MPDATA computations onto available computing resources. The work teams execute computations in parallel and independently of each other, within each time step. These advantages are achieved at the cost of some extra computations performed by teams.

In general, pieces of the grid corresponding to different teams are characterized by various sizes. The numbers of cores/threads assigned to different teams are varied, as well. Figure 4 also shows an example of partitioning 60 Intel MIC's processing cores into 8 teams, and distributing the MPDATA grid into P pieces. To provide load balancing, we distinguish 4 teams with 8 cores each, and 4 teams with 7 cores each. Moreover, pieces of the MPDATA grid corresponding to these teams have different sizes along i-dimension. At this point of our study, the estimation of $nP_i \times mP_i \times lP_i$ size of the ith piece, and the number of cores/threads per each team is based on empirical tests. Furthermore, it still requires some performance modeling. In particular, the autotuning technique [14] is a promising direction for estimating the best configuration of required parameters.

7. MPDATA Parallelization

7.1. Task and Data Parallelism. In order to utilize the computing resources available in the Intel Xeon Phi coprocessor, the proposed approach employs two main levels of parallelism:

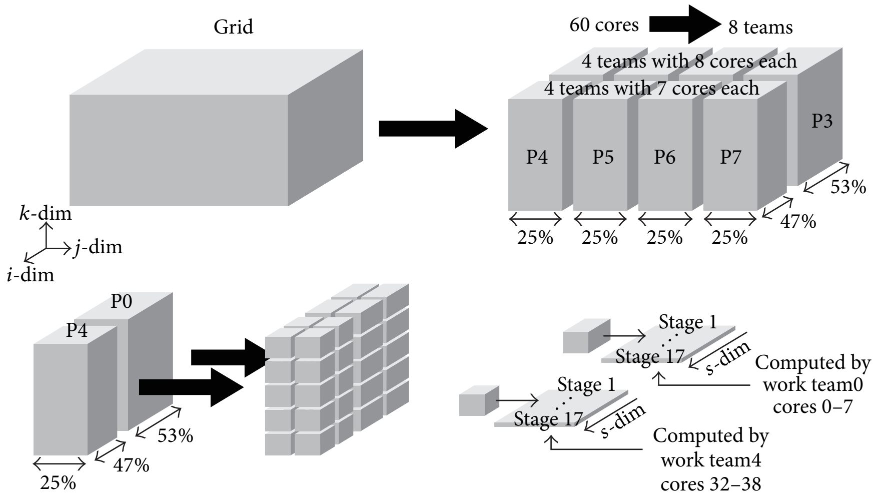


Figure 4: Partitioning of Intel MIC's processing cores into work teams when performing MPDATA computations.

(i) task parallelism which allows for utilizing more than 200 logical cores;

(ii) data parallelism to use efficiently 512-bit vector processing units.

The MPDATA grid is distributed into a set of pieces. All computations within each piece are performed by the corresponding work team, according to the proposed (3 + 1)D decomposition (see Figure 4). Hence, every piece is partitioned into some MPDATA blocks, where subsequent blocks are processed one by one, and each block of size $nB \times mB \times lB$ is processed in parallel by the work team.

Every block is further decomposed into subblocks of size $nB^* \times mB^* \times lB^*$, where each subblock is processed by a certain thread of the work team. A sequence of all the MPDATA stages is executed within every subblock, taking into account the adequate halo areas. Due to the data dependencies of MPDATA, appropriate synchronizations of all the threads within the work team are required between stages. All computations within every subblock correspond to a task. As a result, all the work teams provide parallel computations, and tasks are processed in parallel by the threads of every work team.

Another level of parallelization is SIMDification applied within each task (thread). The scalar implementation of MPDATA is converted to a vector version, which processes one operation on multiple pairs of operands at once. So, the scalar instructions are replaced by the Intel SIMD instructions. The vectorization is performed within k-dimension, where the value of size lB^* is adjusted to the vector size.

7.2. Distribution of Calculations within Work Team. An appropriate distribution of calculations 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 intracache 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.

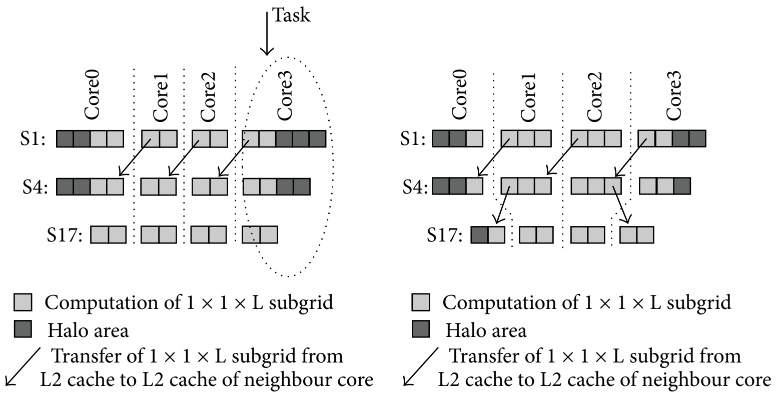
Figure 5 illustrates an example of two scenarios of distributing MPDATA calculations within a given team of cores for the block of size $1 \times 8 \times l$. In this example, a single team corresponds to 4 cores (one thread per core is assumed). The first scenario (Figure 5(a)) features less amount of intracache communications between tasks (threads) than the second one. However, the load imbalance within the team of cores is noticeable in this scenario. The second one provides a better load balance across available resources assigned to a team, but it requires more intracache communications.

Because of the intracache communications between tasks, the overall system performance depends on a suitable method of pinning the task to available cores. 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 intracache bandwidth, as well as reduces cache misses, and the latency of access to the cache memory.

8. Preliminary Performance Results

In this section, we present preliminary performance results obtained for the double precision 3D MPDATA algorithm on the platforms introduced in Section 3. 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, sizes of pieces, size of block, and distribution of computation within teams, are chosen in an empirical way, individually for each platform.

To take advantage of Intel MIC computing resources, the OpenMP standard is used for multi-/manycore programming. Moreover, to perform computations according to the proposed adaptation, a dedicated mechanism for the management of MPDATA parallel computations is developed. Firstly, the synchronization of threads required after each time step


Task
Core0
Core1
Core2
Core3
S1:
S4:
S17:
Computation of 1 × 1 × L subgrid
Halo area
Transfer of 1 × 1 × L subgrid from L2 cache to L2 cache of neighbour core
Core0
Core1
Core2
Core3
S1:
S4:
S17:
Computation of 1 × 1 × L subgrid
Halo area
Transfer of 1 × 1 × L subgrid from L2 cache to L2 cache of neighbour core

(a)

(b)

Figure 5: Example of distribution of calculations within a team of cores: (a) first scenario decreases load balancing for reducing intracache communications; (b) second scenario improves load balancing at the cost of increasing intracache communications.

is performed using the OpenMP barrier construct. Secondly, a dedicated synchronization mechanism is implemented to ensure synchronization among threads of each work team, providing the correct execution of MPDATA stages inside each time step. This mechanism is based on using the OpenMP atomic directive. Furthermore, to utilize vector units available inside each Intel MIC core, scalar instructions are replaced by appropriate Intel SIMD intrinsics.

Currently, only the first four stages of MPDATA are implemented and tested. It should be noted that these stages require to transfer all 5 input and one output matrices and, as a consequence, the overall system performance for the proposed adaptation is still limited by the main memory traffic. According to the proposed (3 + 1)D decomposition of MPDATA, the other stages will perform computations on the data kept in cache. The final performance gain for the proposed adaptation will be revealed when the computations for all the MPDATA stages will be programmed.

Figure 6 presents the normalized execution time of the 3D MPDATA algorithm, for 500 time steps and the grid of size 1024 × 512 × 64.

Figure 6(a) shows the performance gain for the improved version of (3 + 1)D decomposition. The proposed method of reducing extra computations allows us to speed up the MPDATA block version from 2 to 5 times, depending on the platform used and size of the grid.

The advantages of applying the loop fusion technique on the tile management level are shown in Figure 6(b). This technique permits the increase of the performance by about 5–10%.

The impact of block size on the overall performance is illustrated in Figure 6(c). In general, the larger the block size the higher the performance. However, the limiting factor is the cache size.

According to Figure 6(d), among the four tested configurations, the best results are obtained for the configuration containing 10 teams, with 24 threads per each team. These configurations are highly distinguished with respect to the load balancing of MPDATA computations and intracache communications. Therefore, significant performance differences are observed in these tests.

The advantages of using vectorization are observed for all the platforms. In particular, for Intel Xeon Phi 7120P, it allows us to accelerate computations more than 3 times, using all the available cores/threads (Figure 6(e)).

Figure 6(f) shows the performance obtained for the different numbers of threads per core, using Intel Xeon Phi 7120P. The best efficiency of computation is achieved when running 4 threads per core.

The performance comparison of all the platforms is shown in Figure 7. For each platform, we use all the available cores with the vectorization enabled. As expected, the best performance result is obtained using Intel Xeon Phi 7120P. This coprocessor executes the MPDATA algorithm almost 2 times faster than two Intel Xeon E5-2697 v2 CPUs, containing 24 cores totally. Both models of the Intel Xeon Phi coprocessor give similar performance results.

Additionally, a basic parallel version of the MPDATA algorithm has been implemented and compared to the proposed adaptation. This version is based on applying the OpenMP API and using a set of compiler directives. For Intel Xeon Phi 7120P, the proposed adaptation allows us to accelerate computations more than 4 times in comparison to the basic version, while for the Intel CPUs this adaptation increases the performance about 3 times. It should be noted that the efficiency of our adaptation scheme will increase when all the MPDATA stages will be included into the code. At the same time, the efficiency of the basic parallel version will be at a similar level as for the four MPDATA stages.

9. 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 efficient 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 prospective

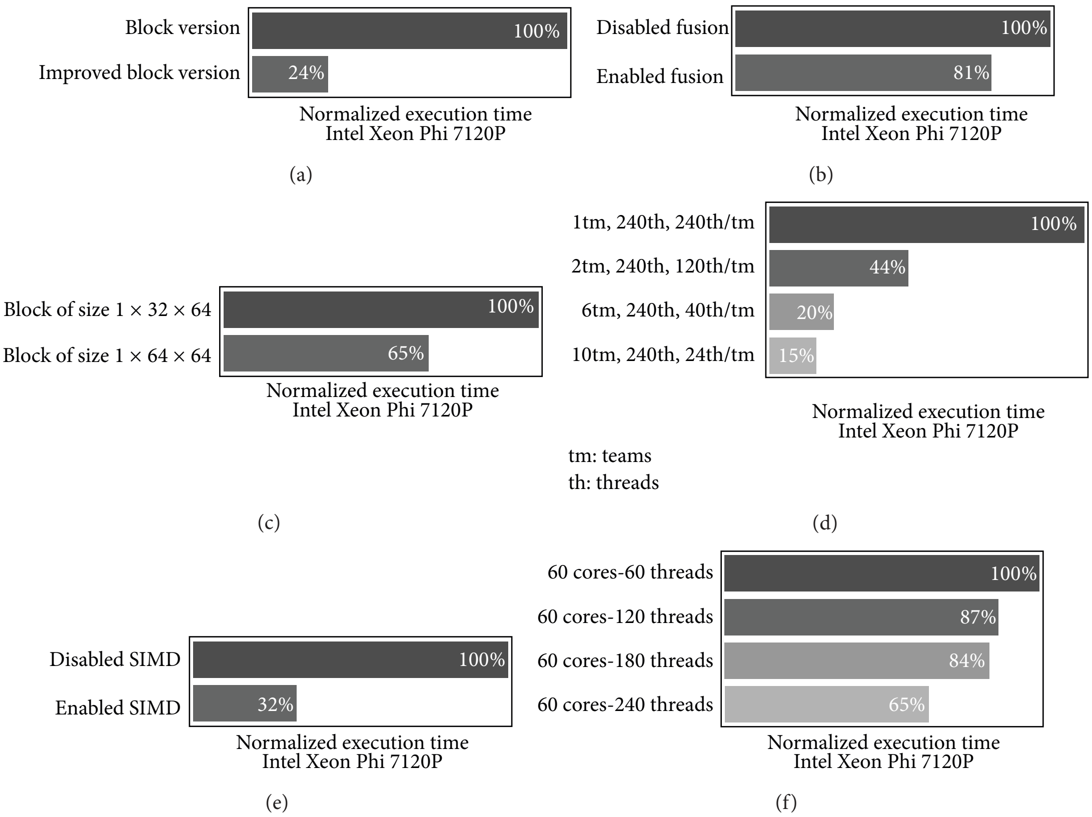


Figure 6: Preliminary performance results: (a) performance gain for improved version of (3 + 1)D decomposition; (b) advantages of applying the loop fusion on tile management level; (c) performance for different block sizes; (d) performance for different configurations of teams; (e) advantages of using vectorization; (f) performance for different numbers of threads per core.

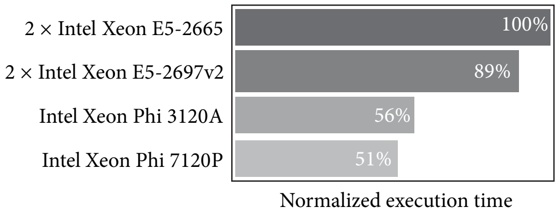


Figure 7: Comparison of Intel Xeon CPU and Intel Xeon Phi (best configurations with SIMD).

way to improve the efficiency of using this model in practical simulations.

The main challenge of the proposed parallelization is to take advantage of multi-/manycore, vectorization, and cache reusing. For this aim, we propose the block version of the 3D MPDATA algorithm, based on combination of loop tiling and loop fusion 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 the partitioning of available cores/threads into teams. Each team corresponds to a piece of the MPDATA grid and executes calculations according to the block decomposition strategy. It allows us to reduce intercache communication overheads due to communications between neighbour threads/cores, and their synchronizations. This method also increases opportunities for the efficient load distribution of MPDATA computations on available resources.

An appropriate distribution of calculations within teams 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 intracache communication. Aiming at improving the load balancing within a team, the possibility of an 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 and executes the MPDATA algorithm almost 2 times faster than two Intel Xeon E5-2697 v2 CPUs, containing 24 cores totally. The proposed adaptation allows us to utilize both Intel Xeon Phi and Intel Xeon CPU computing resources. For Intel Xeon Phi 7120P, the proposed adaptation executes MPDATA computation 4 times faster than the basic version, while for the Intel CPU this adaptation increases the performance about 3 times. The multi-/manycore and vectorization features of the Intel MIC and Intel CPU play the leading role in the performance exploitation. The other important feature is a suitable selection of the block size, number of teams, number of threads per core, and 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 the MPDATA calculation 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 us to develop a flexible data and task scheduler, supported by adequate performance models. Another direction of future work is adaptation of MPDATA to heterogeneous clusters with Intel MICs, with a further development and optimization of our code.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

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

References

[1] L. Szustak, K. Rojek, and P. Gepner, “Using intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm,” in *Parallel Processing and Applied Mathematics*, vol. 8385 of *Lecture Notes in Computer Science*, pp. 582–592, Springer, Berlin, Germany, 2014.

[2] J. Dongarra, M. Gates, A. Haidar et al., “Portable HPC programming on Intel many-integrated-core hardware with MAGMA port to Xeon Phi,” in *Parallel Processing and Applied Mathematics*, vol. 8384 of *Lecture Notes in Computer Science*, pp. 571–581, Springer, Heidelberg, Germany, 2014.

[3] S. P. Pissis, C. Goll, P. Pavlidis, and A. Stamatakis, “Accelerating string matching on MIC architecture for motif extraction,” in *Parallel Processing and Applied Mathematics*, vol. 8386 of *Lecture Notes in Computer Science*, pp. 258–267, Springer, Berlin, Germany, 2014.

[4] E. Saule, K. Kaya, and Ü. V. Çatalyürek, “Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi,” in *Parallel Processing and Applied Mathematics*, vol. 8385 of *Lecture Notes in Computer Science*, pp. 559–570, Springer, Berlin, Germany, 2014.

[5] P. K. Smolarkiewicz, “Multidimensional positive definite advection transport algorithm: an overview,” *International Journal for Numerical Methods in Fluids*, vol. 50, no. 10, pp. 1123–1144, 2006.

[6] M. Ciznicki, P. Kopta, M. Kulczewski, K. Kurowski, and P. Gepner, “Elliptic solver performance evaluation on modern hardware architectures,” in *Parallel Processing and Applied Mathematics*, vol. 8384 of *Lecture Notes in Computer Science*, pp. 155–165, Springer, Berlin, Germany, 2014.

[7] K. Rojek and L. Szustak, “Parallelization of EULAG model on multicore architectures with GPU accelerators,” in *Parallel Processing and Applied Mathematics*, vol. 7204 of *Lecture Notes in Computer Science*, pp. 391–400, Springer, Berlin, Germany, 2012.

[8] G. Hager and M. Wittmann, *Introduction to High Performance Computing for Science and Engineers*, CRC Press, New York, NY, USA, 2011.

[9] S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick, “Impact of modern memory subsystems on cache optimizations for stencil computations,” in *Proceedings of the Workshop on Memory System Performance (MSP ’05)*, pp. 36–43, Chicago, Ill, USA, June 2005.

[10] 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 Processing Letters*, vol. 20, no. 4, pp. 359–376, 2010.

[11] K. Rojek, L. Szustak, and R. Wyrzykowski, “Performance analysis for stencil-based 3D MPDATA algorithm on GPU architecture,” in *Parallel Processing and Applied Mathematics*, vol. 8384 of *Lecture Notes in Computer Science*, pp. 145–154, Springer, Berlin, Germany, 2014.

[12] L. Szustak, K. Rojek, R. Wyrzykowski, and P. Gepner, “Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture,” in *Proceedings of the 1st International Workshop on High-Performance Stencil Computations (HiStencils '14)*, pp. 51–56, 2014.

[13] R. Wyrzykowski, K. Rojek, and L. Szustak, “Using blue gene/P and GPUs to accelerate computations in the EULAG model,” in *Large-Scale Scientific Computing*, vol. 7116 of *Lecture Notes in Computer Science*, pp. 670–677, 2012.

[14] R. Wyrzykowski, L. Szustak, and K. Rojek, “Parallelization of 2D MPDATA EULAG algorithm on hybrid architectures with GPU accelerators,” *Parallel Computing. Systems & Applications*, vol. 40, no. 8, pp. 425–447, 2014.

[15] K. Kurowski, M. Kulczewski, and M. Dobski, “Parallel and GPU based strategies for selected CFD and climate modeling models,” *Environmental Science and Engineering*, vol. 3, pp. 735–747, 2011.

[16] R. Wyrzykowski, L. Szustak, K. Rojek, and A. Tomas, “Towards efficient decomposition and parallelization of MPDATA on hybrid CPU-GPU cluster,” in *Large-Scale Scientific Computing*, vol. 8353 of *Lecture Notes in Computer Science*, pp. 457–464, 2014.

[17] Z. P. Piotrowski, A. A. Wyszogrodzki, and P. K. Smolarkiewicz, “Towards petascale simulation of atmospheric circulations with soundproof equations,” *Acta Geophysica*, vol. 59, no. 6, pp. 1294–1311, 2011.

[18] D. K. Wójcik,, M. Kurowski, B. Rosa, and M. Z. Ziemiański, “A study on parallel performance of the EULAG F90/95 code,” in *Parallel Processing and Applied Mathematics*, vol. 7204 of *Lecture Notes in Computer Science*, pp. 419–428, Springer, Berlin, Germany, 2012.

[19] R. Wyrzykowski, K. Rojek, and L. Szustak, “Model-driven adaptation of double-precision matrix multiplication to the cell processor architecture,” *Parallel Computing*, vol. 38, no. 4-5, pp. 260–276, 2012.

[20] R. Cruz, M. Araya-Polo, and J. Cela, "Introducing the semi-stencil algorithm," in *Parallel Processing and Applied Mathematics*, vol. 6067 of *Lecture Notes in Computer Science*, pp. 496–506, Springer, Berlin, Germany, 2010.

[21] K. Datta, S. Kamill, S. Williams, L. Oliker, J. Shalf, and K. Yelick, "Optimization and performance modeling of stencil computations on modern microprocessors," *SIAM Review*, vol. 51, no. 1, pp. 129–159, 2009.

[22] K. Datta, M. Murphy, V. Volkov et al., "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '08)*, pp. 1–12, IEEE, Austin, Tex, USA, November 2008.

[23] G. Rivera and C.-W. Tseng, "Tiling optimizations for 3D scientific computations," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '00)*, p. 32, Dallas, Tex, USA, November 2000.

[24] J. Treibig, G. Wellein, and G. Hager, "Efficient multicore-aware parallelization strategies for iterative stencil computations," *Journal of Computational Science*, vol. 2, no. 2, pp. 130–137, 2011.

[25] D. Unat, X. Cai, and S. B. Baden, "Mint: realizing CUDA performance in 3D stencil methods with annotated C," in *Proceedings of the 25th ACM International Conference on Supercomputing (ICS '11)*, pp. 214–224, June 2011.

[26] J. Fang, A. L. Varbanescu, and H. Sips, *Benchmarking Intel Xeon Phi to Guide Kernel Design*, Delft University of Technology Parallel and Distributed Systems Report Series, PDS-2013-005, 2013.

[27] J. Peraza, A. Tiwari, M. Laurenzano, L. Carrington, W. A. Ward, and R. Campbell, "Understanding the performance of stencil computations on Intel's Xeon Phi," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '13)*, IEEE, September 2013.

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

[29] 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, Sunnyvale, Calif, USA, 2013.

[30] P. Smolarkiewicz and W. W. Grabowski, "The multidimensional positive definite advection transport algorithm: nonoscillatory option," *Journal of Computational Physics*, vol. 86, no. 2, pp. 355–375, 1990.

[31] P. K. Smolarkiewicz and L. G. Margolin, "MPDATA: a finite-difference solver for geophysical flows," *Journal of Computational Physics*, vol. 140, no. 2, pp. 459–480, 1998.

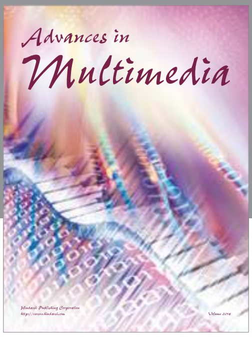
[32] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-perfomance computing," *ACM Computing Surveys*, vol. 26, no. 4, pp. 345–420, 1994.

[33] K. Kennedy, "Fast greedy weighted fusion," *International Journal of Parallel Programming*, vol. 29, no. 5, pp. 463–491, 2001.

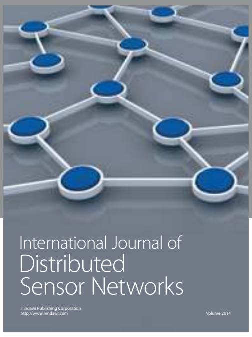
[34] G. Gao, R. Olsen, V. Sarkar, and R. Thekkath, "Collective loop fusion for array contraction," in *Languages and Compilers for Parallel Computing*, vol. 757 of *Lecture Notes in Computer Science*, pp. 281–295, 1993.

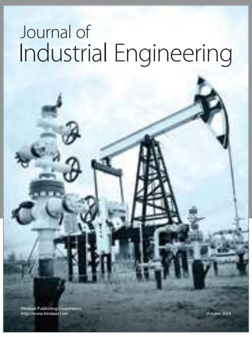
[35] C. Ding and Y. He, "A ghost cell expansion method for reducing communications in solving PDE problems," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '01)*, p. 50, ACM Press, Denver, Colo, USA, November 2001.

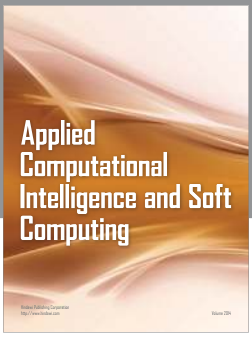
[36] Intel Architectures Comparison, http://ark.intel.com/.

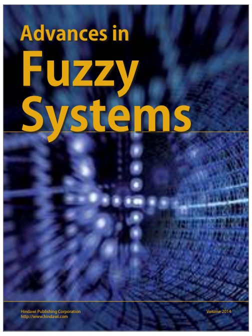

Advances in
Multimedia

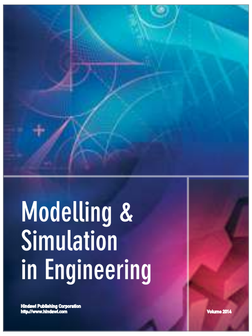
The Scientific
World Journal

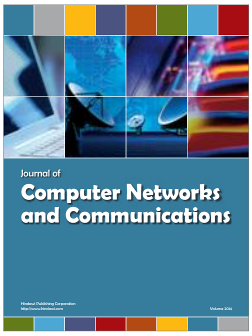

International Journal of
Distributed
Sensor Networks

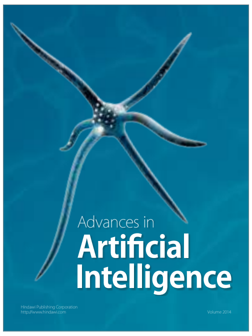

Journal of
Industrial Engineering

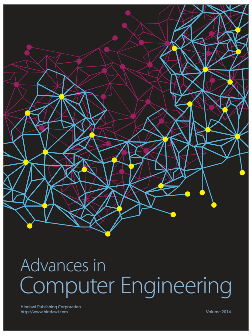

Applied
Computational
Intelligence and Soft
Computing


Advances in
Fuzzy
Systems

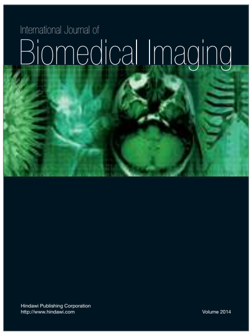

Modelling &
Simulation
in Engineering

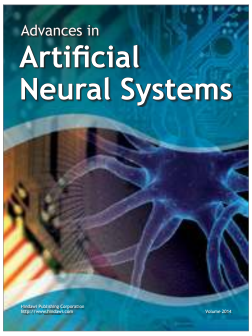

Journal of
Computer Networks
and Communications

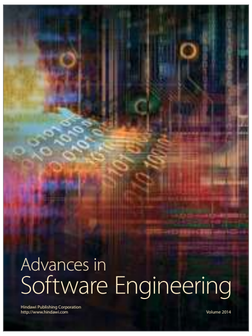

Advances in
Artificial
Intelligence


Advances in
Computer Engineering

International Journal of
Computer Games
Technology

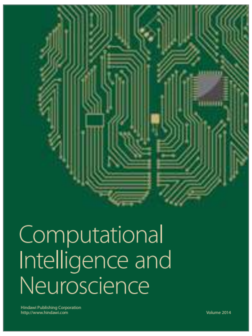

International Journal of
Biomedical Imaging


Advances in
Artificial
Neural Systems


Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer
Interaction


Computational
Intelligence and
Neuroscience

International Journal of
Reconfigurable
Computing

Journal of
Electrical and Computer
Engineering