# Islands-of-Cores Approach for Harnessing SMP/NUMA Architectures in Heterogeneous Stencil Computations

Lukasz Szustak[1(✉)], Roman Wyrzykowski[1], and Ondřej Jakl[2]

[1] Czestochowa University of Technology,
Dabrowskiego 69, 42-201 Czestochowa, Poland
{lszustak,roman}@icis.pcz.pl
[2] Institute of Geonics of the Czech Academy of Sciences,
Studentská 1768, 708 00 Ostrava-Poruba, Czech Republic
ondrej.jakl@ugn.cas.cz

**Abstract.** SMP/NUMA systems are powerful HPC platforms which could be applied for a wide range of real-life applications. These systems provide large capacity of shared memory, and allow using the shared-variable programming model to take advantages of shared memory for inter-process communications and synchronizations. However, as data can be physically dispersed over many nodes, the access to various data items may require significantly different times. In this paper, we face the challenge of harnessing the heterogeneous nature of SMP/NUMA communications for a complex scientific application which implements the Multidimensional Positive Definite Advection Transport Algorithm (MPDATA), consisting of a set of heterogeneous stencil computations.

When using our method of MPDATA workload distribution, which was successfully applied for small-scale shared memory systems with several CPUs and/or accelerators, significant performance losses are noticeable for larger SMP/NUMA systems, such as SGI UV 2000 server used in this work. To overcome this shortcoming, we propose a new islands-of-cores approach. It exposes a correlation between computation and communication for heterogeneous stencils, and enables an efficient management of trade-off between computation and communication costs in accordance with the features of SMP/NUMA systems. In consequence, when using the maximum configuration with 112 cores of 14 Intel Xeon E5-4627v2 3.3 GHz processors, the proposed approach accelerates the previous method more then 10 times, achieving about 390 Gflop/s, or approximately 30% of the theoretical peak performance.

## 1 Introduction

In the last years, it appears evident [7,22] that emerging computing platforms will combine multi- and manycore architectures. In particular, this trend is noticeable in an environment of large-scale computations (High Performance Computing, HPC) where supercomputers are built with massively parallel components [24], such as multicore processors and manycores accelerators. The most


V. Malyshkin (Ed.): PaCT 2017, LNCS 10421, pp. 351–364, 2017.
DOI: 10.1007/978-3-319-62932-2_34

common solutions for such systems are based on the cluster architectures, that are delivered by many vendors. More than 80% of supercomputers in the TOP500 list for November 2016 refer to these systems (http://top500.org).

However, despite the considerable popularity of clusters, other powerful computing platforms are also perceptible in HPC environments. Among them are systems based on the SMP/NUMA (symmetric multiprocessor/non-uniform memory access) architectures [4], which are usually built around high-performance networks as distributed shared memory (DSM) systems. DSM is a form of memory architecture where physically separated memories can be addressed as one logically shared address space. These systems provide extremely large capacity of shared memory, and are able to achieve high levels of memory throughput performance. At the same time, because data can be physically dispersed over many nodes, the access time for different data items may well be different which explains the term non-uniform data access. In SMP/NUMA architectures, the parallelism can be successfully expressed with the OpenMP library, or the MPI standard - a common solution for clusters. Also, the mixture of MPI and OpenMP is possible. However, is worthwhile to mention that OpenMP is capable of itself to fully utilize such systems without demanding more complex message passing operations [5,23] required by MPI.

One of leading vendors of these systems is SGI, that has been delivering SMP/NUMA architectures for more than 20 years. Its newest SMP/NUMA product series, SGI UV [11] is based on Intel multicore processors and the high-speed NUMAlink system interconnect, offering up to thousands of cores in a single system which shares large main memory capacity. An example of using the SGI UV 2000 server for accelerating a complex real-world application, MapReduce is presented in [2], where a topology-aware placement algorithm is proposed to speed up the data shuffling phase of MapReduce. The first generation of SGI UV platforms is applied in [3] to parallelize the Generalized Conjugate Residual (GCR) elliptic solver with preconditioner, using a mixture of MPI and OpenMP. In order to place properly all MPI processes and OpenMP threads on the underlying hardware, a specialized scheduler was developed to take into account the network topology. Apart from numerical applications, the SGI UV 2000 systems are also reported to be efficiently used in other areas, such as computation on graphs [25] and combinatorial optimization problems [1].

In this paper, we face the challenge of efficient utilization of SMP/NUMA systems in practice, for a rather complex scientific application. The application we study implements the Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) [13,14], which consists of a set of heterogeneous stencils. Besides the GCR solver, MPDATA is the second major part of the dynamic core of the EULAG (Eulerian/semi-Lagrangian) geophysical model [15]. It is an established numerical model developed for simulating thermo-fluid flows across a wide range of scales and physical scenario. In particular, it can be used in numerical weather prediction, simulation of urban flows, turbulences, and ocean currents [9,16,17].

In our previous works [18–20], we successfully developed a new version of MPDATA, dedicated to small-scale shared memory systems with several processors and/or accelerators. In particular, we proposed a new (3+1)D decomposition of MPDATA computations that allows us to significantly reduce the main memory traffic. This implementation provides a much better usage of capabilities of novel CPUs and Intel Xeon Phi coprocessors. However, although the proposed new strategy of workload distributions gives a gain at the desired performance level not only for Intel Xeon processors, but also for the first generation of Intel Xeon Phi accelerators ([19]), significant performance losses are noticeable for larger SMP/NUMA systems, such as SGI UV 2000 server used in this work.

In this paper, to overcome this shortcoming and to improve the efficiency of the MPDATA application, we propose an islands-of-cores approach dedicated to heterogeneous stencils such as those of MPDATA. This approach reveals a correlation between computation and communication for heterogeneous stencil computations, and enables a better management of the balance between computation and communication costs in accordance with the features of SMP/NUMA systems such as the SGI UV 2000 server. The proposed approach is based on the analysis of two scenarios for the parallel execution of a set of heterogeneous stencils. While the first scenario performs less computations but requires more data transfers, the second one allows us to replace the implicit data traffic between nodes by extra computations, and overcome the non-uniform memory constraints. In consequence, when using the maximum number $P = 14$ of processors with 112 cores totally, the proposed approach accelerates the pure (3+1)D decomposition more then 10 times, achieving approximately 30% of the theoretical peak performance of the system.

To our best knowledge, there exists no investigations of the correlation between computation and communication for heterogeneous stencils computations which consist of a set of stencils with different patterns. The closest approaches were proposed in papers [6,26]. Similarly to our study, these works consider the code transformation using the overlapped tiling technique. It enables removing the synchronization and enhancing the data locality at the cost of redundant computations. However, these works take into account only the homogeneous stencil computations, with a single pattern only. Opposite to our study, these approaches are addressed to small computing platforms with one or two processors.

## 2 SMP/NUMA Architecture: SGI UV 2000 Server

The parallel computer architecture that we are interested in this paper has all its processing elements interconnected to a shared main memory. One of the most prominent manufacturers of shared-memory systems is SGI. The latest SGI UV ("UltraViolet") product line is delivered since 2009. In all the experiments described in this paper we employ a machine of the second UltraViolet generation known as UV 2 [12], launched in 2012. For a single system, its cache-coherent shared memory can be extended up to 64 TB, and accessed from up to

2048 Intel CPU cores, thanks to the high-speed NUMAlink 6 proprietary interconnect with a point-to-point bandwidth of 6.7 GB/s per direction; doubled with respect to NumaLink 5. This allows putting hundreds of NUMA nodes together to behave as a single multicore system.

The target SGI UV 2000 server was acquired by the IT4Innovations National Supercomputing Center in Ostrava [8] to support applications with extraordinary large memory requirements. It consists of one "individual rack unit" (IRU) that features 3328 GB of RAM and 112 cores in total, distributed across 14 NUMA nodes in 7 compute/memory modules called blades, connected to each other via a backplane (one blade position in this IRU enclosure is empty). Each NUMA node is based on the 8-core Intel Xeon E5-4627v2 3.3 GHz processor with roughly 236 GB RAM. IRU has ports that are brought out to external NUMAlink 6 connectors. This UV 2000 server shares some infrastructure with the Salomon supercomputer of the IT4Innovations center, in June 2015 placed #40 on the TOP500 list (http://top500.org).

## 3 Parallelization of MPDATA for Shared-Memory Model

### 3.1 Introduction to MPDATA Application

The MPDATA application implements a general approach for integrating the conservation laws of geophysical fluids on micro-to-planetary scales [10,13]. The MPDATA algorithm enables solving advection problems, and offers several options to model a wide range of complex geophysical flows. The MPDATA computations correspond to the group of iterative, forward-in-time algorithms. This application is used typically for long running simulations, such as the numerical weather prediction, that require execution of several thousand time steps for a given size of domain. Moreover, since the accuracy of computation plays a key role for MPDATA, these simulations usually are performed using the double-precision floating-point format. The application allows solving 1-, 2- or 3-dimensional problems. In this paper, we consider the last case, when the MPDATA algorithm is defined on 3D grids with $i$, $j$, and $k$ dimensions.

Every MPDATA time step performs the same computations, which consist of the set of 17 stages [19,20]. The MPDATA stages represent the heterogeneous stencils codes which update grid elements according to different patterns. All the stages are dependent on each other: outcomes of prior stages are usually input data for the subsequent computations. A single MPDATA time step loads five 3D input arrays from the main memory, and saves one output 3D array that is necessary for the next steps. In the original version of code, a lot of intermediate results (3D arrays) are also transferred to/from the main memory. In consequence, a significant data traffic to the main memory is generated, which mostly limits the attainable performance on novel architectures.

### 3.2 (3+1)D Decomposition

In our previous works [18–20], we proposed a new strategy of workload distribution for the MPDATA application. This strategy contributes to ease the

memory and communication bounds, and to better exploit computation resources of shared-memory systems including CPUs and the first generation of Intel Xeon Phi accelerators. The main challenge of these works was to minimize data transfers between the main memory and the cache hierarchy. To improve the overall performance, we reorganized computation inside each time step of MPDATA.

The main aim of the new computational flow for the MPDATA application is to eliminate accesses to the main memory associated with all the intermediate computations. This idea implies that all the intermediate outcomes of computations have to be kept in cache only - without transferring them to the main memory. As a result, for each MPDATA time step, the main memory traffic will be generated only by transfers required by input/output data (arrays). To reach this goal, we proposed the (3+1)D decomposition of MPDATA computation [19,20] that is based on a combination of loop fusion and loop tiling optimization techniques.

The implementation of the (3+1)D decomposition requires to partition the MPDATA domain (grid) onto a set of sub-domains of size that enables to kept all the necessary intermediate data in the cache memory. The consecutive sub-domains are processed sequentially, one by one, while every sub-domain is processed in parallel by available computing resources. Every sub-domain is responsible for computing all the MPDATA stages that perform computations on chunks (or blocks) of the corresponding arrays, and returns an adequate part of the output array.

The proposed (3+1)D decomposition allows us to significantly reduce the main memory traffic, where the real profit depends on the size of domains, as well as computational characteristic of a given computing platform. For example, using a single Intel Xeon CPU E5-2660v2 processor, the volume of the main memory traffic is reduced from 133 GB to 30 GB, and computations are accelerated about 2.8 times for domains of the size $256 \times 256 \times 64$, and 50 time steps. In this research, the `likwid-perfctr` tool [21] is used for the performance analysis of developed codes. However, although the proposed (3+1)D decomposition gives a gain at the desired performance level not only for Intel Xeon processors, but also for the first generation of Intel Xeon Phi accelerators (see [19]), significant performance losses are noticeable for large shared-memory architectures, such as SMP/NUMA systems.

Table 1 presents the comparison of execution times of MPDATA obtained for the SGI UV 2000 server introduced in Sect. 2, for different versions of code. The performance results are generated for the various number of processors, benchmarking both versions: original and after (3+1)D decomposition. It should be noted that in order to get the optimal performance for this server, it is necessary for each thread of execution to allocate memory *closest* to a core on which it is executed. This is achieved by initializing memory using the technique known [22] as the first-touch policy with parallel initialization.

The obtained performance results reveals the performance limitations for the proposed (3+1)D decomposition. Particularly, the performance gain at the

**Table 1.** Execution times of 50 MPDATA time steps and grid of size $1024 \times 512 \times 64$ obtained for the original parallel version of code and after the (3+1)D decomposition, using the SGI UV 2000 server

| #CPUs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Original | 30.4 | 44.5 | 58.2 | 61.5 | 64.3 | 70.1 | 71.6 | 73.7 | 75.4 | 77.6 | 78.4 | 78.2 | 80.6 | 82.2 |
| Original[a] | 30.4 | 15.4 | 10.5 | 7.9 | 6.6 | 5.6 | 5.0 | 4.3 | 4.0 | 3.6 | 3.3 | 3.1 | 3.0 | 2.8 |
| (3+1)D[a] | 9.0 | 8.2 | 7.4 | 8.0 | 7.1 | 7.2 | 7.3 | 7.7 | 9.1 | 9.5 | 10.2 | 10.1 | 10.3 | 10.4 |

[a]The first-touch policy with parallel initialization is used

desired level is achieved only for a single processor (3.37× faster than the original version), while significant performance losses take place for the MPDATA executions with a higher number of processors. It should be also underlined here that the original version returns even better execution times than the (3+1)D decomposition, for all the benchmarks with the number of processors greater than 4. To overcome this shortcoming and to improve the efficiency of the MPDATA application, we propose the islands-of-cores approach, dedicated to heterogeneous stencils such as those of MPDATA.

## 4 Islands-of-Cores Approach for MPDATA

### 4.1 Trade-off Between Computation and Communication for Heterogeneous Stencils

To eliminate the revealed performance losses, the analysis of computational flow for heterogeneous stencils has to be considered. Figure 1(a) presents an example of forward-in-time computations with a set of heterogeneous stencils, when every time step consists of three stages. Here each stage corresponds to execution of an 1D stencil. Figure 1(b) and (c) show two scenarios for parallelization of this example using two processors.

The first scenario (Fig. 1b) reveals an implicit data traffic between processors in a shared-memory system because of data dependencies. This data traffic takes place on borders of sub-domains distributed between processors. For example, the output element $C[d]$ computed by *CPU_B* within the 3rd stage depends on the element $B[c]$ that is computed by *CPU_A* as a result of stage 2. However, $B[c]$ depends on the element $A[d]$ which is returned by *CPU_B* in the 1st stage. In consequence, the implicit transfers of two elements take place between the processors *CPU_A* and *CPU_B* of a shared-memory system. Furthermore, three synchronization points have to be added in order to ensure the correctness of parallel computations.

The second scenario (Fig. 1c) demonstrates how to avoid exchanging data between processors at the cost of extra computations. To compute the output element $C[d]$, the processor *CPU_B* has to provide the element $B[c]$ computed in the 2nd stage by *CPU_A* in the first scenario. Instead of transferring this element, let *CPU_B* compute the required element $B[c]$ once more. However, this element depends on the element $A[c]$ from the Stage 1, which is returned by *CPU_A* in the

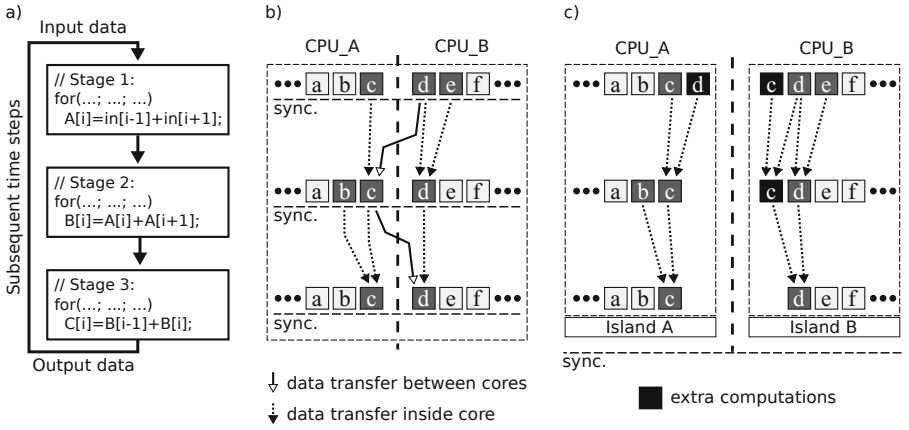


**Fig. 1.** Idea of Islands-of-cores approach: (a) computations corresponding to three exemplary heterogeneous stencils; (b) parallelization with transfers of data between CPUs; (c) parallelization without transfers of data and synchronization points between CPUs, at the cost of extra computations

first scenario. Again, let both processors compute the element $A[c]$ twice, rather than transferring it from *CPU_A* to *CPU_B*. In consequence, *CPU_B* computes two elements more, independently of *CPU_A*. This strategy can be also applied for *CPU_B* that requires the element $A[d]$ computed by *CPU_A* in the first scenario. As a result, something like independent islands, both processors are enabled to perform computations independently of each other within every time step, at the total cost of computing three extra elements.

As shown in Fig. 1, both scenarios enable performing parallel computations. The first scenario performs less computations but requires more data traffic, while the second one allows us to replace the implicit data traffic between processors by replicating some computations. In fact, both solutions should be considered, but the key point is how they fit to the architecture of a computing system. It is expected that the second scenario will be able to get a higher performance in the case of powerful computing resources with relatively less efficient interconnects. On the contrary, the first scenario is more suitable for systems with more efficient networks that connect less powerful computing resources.

Taking into account the architecture of the SGI UV 2000 server, the second scenario seems to fit perfectly to processors connected each other by NUMAlink. At the same time, the first scenario should be well suited inside each processor, where a more efficient, internal memory hierarchy is used to implement the data traffic between available cores.

### 4.2 Implementation: From Islands-of-Cores to Work-Teams

The (3+1)D decomposition moves the data traffic from the main memory to the cache hierarchy. In consequence, a lot of intra- and inter-cache communications

between cores/processors is generated. This approach corresponds to the first scenario (Fig. 1b). When using a single processor, this traffic is restricted to the cache hierarchy of this CPU. However, in the case of the whole server, the required data are implicitly transferred between caches of neighbor processors through the NUMAlink interconnect [12]. As Table 1 shows, it is particularly significant when more than two processors cooperate to execute the application.

To face this issue, we propose to adopt the islands-of-cores approach to the MPDATA application, which has a significantly more complex computing structure than the example shown in Fig. 1. In this work, we focus on the MPDATA algorithm defined on a 3D grid, where every time step consists of 17 stages with heterogeneous stencils that depend on each other in all three dimensions.

Based on the conclusion formulated in the end of the previous subsection, the abstraction of islands-of-cores is applied across $P$ processors of an SMP/NUMA platform. In consequence, the MPDATA domain is partitioned into $P$ parts that are mapped onto $P$ islands. Following the islands-of-cores approach, each processor is now an island of cores, and these islands perform the following phases:

1. All islands share all input data for each MPDATA time step, utilizing the first-touch policy with parallel initialization.
2. Every island processes the part of MPDATA assigned to it according to the (3+1)D decomposition.
3. Each island performs independent computations within every time step, at the cost of extra computations (see Fig. 1).
4. All islands return common outcomes to the main memory, after each time step.
5. All islands synchronize their works after each time step, in order to ensure correctness of input data for subsequent time steps.

Since every island consists of the same number of cores, the MPDATA domain is decomposed into equals parts, where the number of parts is equal to the number of processors used in computations. Each part is further partitioned into the set of sub-domains according to the proposed (3+1)D decomposition. While different sub-domains are executed sequentially, each of sub-domain is processed in parallel by utilizing a work team of cores which belong to every island. Each work team of cores performs computations corresponding to all the 17 stages of MPDATA, including computing extra elements instead of transferring them from other teams. As a result, every work team is able to perform computations for each MPDATA time step independently of other teams.

To adapt the proposed islands-of-cores approach to the MPDATA application, an efficient method of mapping parts of MPDATA onto processors has to be developed. It is expected to obtain too large communication overheads when the MPDATA domain is partitioned in all three dimensions. The reason is that data layouts of all the MPDATA arrays allow performing required transfers of the continuous areas of memory only in the first and second dimensions. As a result, only 1D and 2D variants of partitioning the MPDATA domain should be taken into account. When evaluating the proposed approach, the 1D partitioning

is considered as a starting point in this paper, while investigating more complex 2D variants will be among the main goals of our future works.

As data transfers take place only between neighbour parts of the MPDATA domain, to reduce the communication paths through the network topology, all the neighbour parts should be assigned to the adjacent processors that are closely connected each other within the interconnect. It can be achieved by controlling the OpenMP Thread Affinity interface that allows us to bind threads to physical processing units.

The total amount of extra elements which have to be computed redundantly depends on the problem size, number of islands, and shape of partitioning, as well as data dependencies between all the MPDATA stages. Table 2 presents an example how the total number of extra elements increases with the number of work teams, in comparison with the original version. We compare results for two variants of mapping the MPDATA domain onto 1D grids of processors - across either the first (A) or second (B) dimension of the MPDATA grid. It can be concluded that the first variant gives fewer extra elements, for any number of islands.

**Table 2.** The total amount of extra elements in percentage in comparison with the original version, obtained for mapping the MPDATA grid onto 1D grids of processors using variants A and B, for different number of islands and the domain of size $1024 \times 512 \times 64$

| # of islands | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Variant A [%] | 0.00 | 0.25 | 0.49 | 0.74 | 0.99 | 1.24 | 1.48 | 1.73 | 1.98 | 2.22 | 2.47 | 2.72 | 2.96 | 3.21 |
| Variant B [%] | 0.00 | 0.49 | 0.99 | 1.48 | 1.98 | 2.47 | 2.96 | 3.46 | 3.95 | 4.45 | 4.94 | 5.43 | 5.93 | 6.42 |

## 5 Performance Results

This section outlines the performance results obtained for the new implementation of MPDATA developed using the approach described in the previous sections. The new strategies proposed for the workload distribution and data parallelism require also to develop a proprietary scheduler with the affinity-aware placement of threads/cores. To achieve this goal, the OpenMP application programming interface is used only for creating threads and controlling their affinity policy, while all parallel computations are managed by our scheduler which supports the proposed approach.

All the benchmarks are compiled using the Intel compiler icpc v.17.0.1 with the compilation flags: `-O3 -xavx -fp-model precise -fp-model source`, and executed using the SGI UV 2000 server equipped with 14 CPUs. All performance results are obtained for the double-precision floating-point format, the grid of size $1024 \times 512 \times 64$, and 50 times steps. Such a relatively small number of time steps is sufficient to provide the performance evaluation because of homogeneity of all time steps.

In this work, we test the 1D mapping of parts of the MPDATA domain onto a grid of processors. Two variants of experiments are performed, which correspond to distributing the MPDATA domain across either its first or second dimension. Only the results for the first variant are presented in the rest of the paper as it gives better results for all the benchmarks. This is a consequence of a smaller number of extra elements provided by this variant (Table 2).

Table 3 and Fig. 2 present the execution times achieved for the proposed islands-of-cores approach in comparison with the original version and the pure (3+1)D decomposition. Also, we show the partial $S_{pr}$ and overall $S_{ov}$ speedups which define the performance gains of the proposed approach against the pure (3+1)D decomposition and original version, respectively.

The main conclusion is that the proposed islands-of-cores approach, which combines the (3+1)D decomposition and the second scenario of parallelizing stencil computations, allows us to improve radically the efficiency of the MPDATA computations in comparison with the pure (3+1)D decomposition. As expected, despite the extra computations (Table 2), MPDATA is now executed faster for all values of $P$. What should be underlined here, the usage of the islands-of-core approach together with the (3+1)D decomposition permits preserving the high efficiency of such a decomposition.

**Table 3.** Execution times for the original version, pure (3+1)D decomposition, and the proposed islands-of-cores approach, as well as partial $S_{pr}$ and overall $S_{ov}$ speedups of the proposed approach

| #CPUs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Execution times* | | | | | | | | | | | | | | |
| Original | 30.40 | 15.40 | 10.50 | 7.87 | 6.55 | 5.61 | 4.95 | 4.27 | 4.01 | 3.58 | 3.31 | 3.14 | 2.95 | 2.81 |
| (3+1)D | 9.00 | 8.20 | 7.38 | 7.98 | 7.06 | 7.22 | 7.26 | 7.69 | 9.11 | 9.48 | 10.20 | 10.10 | 10.30 | 10.40 |
| Islands of cores | 9.00 | 5.62 | 4.17 | 2.93 | 2.34 | 1.97 | 1.72 | 1.49 | 1.36 | 1.25 | 1.12 | 1.06 | 1.05 | 1.01 |
| *Speedups* | | | | | | | | | | | | | | |
| $S_{pr}$ | 1.00 | 1.46 | 1.77 | 2.72 | 3.02 | 3.66 | 4.22 | 5.16 | 6.70 | 7.58 | 9.11 | 9.53 | 9.81 | 10.30 |
| $S_{ov}$ | 3.38 | 2.74 | 2.52 | 2.69 | 2.80 | 2.85 | 2.88 | 2.87 | 2.95 | 2.86 | 2.96 | 2.96 | 2.81 | 2.78 |

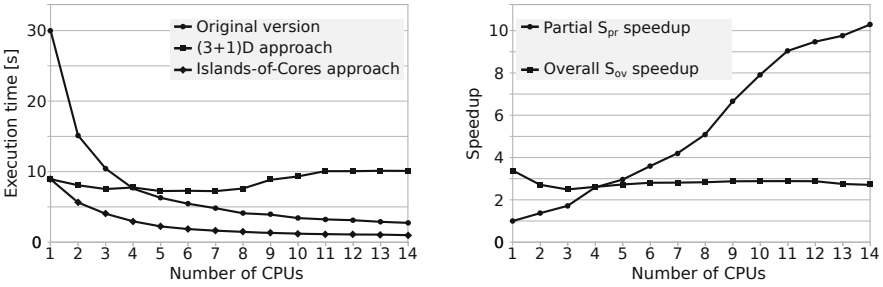


**Fig. 2.** Performance results for the different number $P$ of processors: (a) comparison of the execution time for the different versions of MPDATA; (b) partial and overall speedups of the islands-of-cores approach

In fact, only for configurations with one 8-core processor and two processors (16 cores totally), the pure (3+1)D decomposition is able to shorten radically the execution time in comparison with the original version, while already for $P = 4$ this decomposition gives a worse performance. For larger values of $P$, it is the original version that overpowers the pure (3+1)D decomposition. The disadvantage of utilizing only the (3+1)D decomposition increases with the growing number of processors, achieving the ratio of about 3.7 for $P = 14$ (112 cores totally). The reason for such a disappointing behavior of the pure (3+1)D decomposition is large overheads of data transfers between NUMA nodes (processors) when data should be extracted from the deep memory hierarchy of each node before performing transfers, while in the original version these data are simply located in the main memory.

On the contrary, the acceleration of the proposed island-of-core approach against the original version is kept on a similar level, independently of the number of processors, with $S_{ov} = 2.74$ and $S_{ov} = 2.78$ for $P = 2$ and $P = 14$, respectively. As shown in Fig. 2, the performance gain of the combined approach against the pure (3+1)D decomposition increases with the growing number of processors that together perform computations. Finally, when using the maximum number $P = 14$ of processors, the proposed approach accelerates the (3+1)D decomposition more then 10 times.

Table 4 presents the sustained performance (in Gflop/s) for the islands-of-cores approach, as well as the utilization rate in comparison with the theoretical peak performance of the server. As shown in this table, approximately 30% of the theoretical peak is achieved when using less than 12 processors, while it decreases up to the level of 26% for larger values of $P$. In this benchmark, the maximum sustained performance of about 390 Gflop/s is obtained for $P = 14$, which corresponds to about 77% of the linear scaling. For smaller values of $P$, the parallel efficiency decreases from 96.6% for $P = 4$ to 80.7% for $P = 12$.

**Table 4.** Sustained performance [Gflop/s] obtained for the islands-of-cores approach when using the SGI UV 2000 server, as well as utilization rate [%], and parallel efficiency expressed as percentage of linear scaling

| Number of processors | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 |
| Theoretical performance | | | | | | | | | | | | |
| 105.6 | 211.2 | 316.8 | 422.4 | 528.0 | 633.6 | 739.2 | 844.8 | 950.4 | 1056.0 | 1161.6 | 1267.2 | 1478.4 |
| Sustained performance | | | | | | | | | | | | |
| 42.7 | 68.5 | 92.5 | 131.9 | 165.5 | 197.0 | 226.1 | 261.4 | 287.0 | 325.9 | 349.8 | 370.3 | 390.1 |
| Utilization rate [%] | | | | | | | | | | | | |
| 40.4 | 32.4 | 29.2 | 31.2 | 31.3 | 31.1 | 30.5 | 30.9 | 30.2 | 30.8 | 30.1 | 29.2 | 26.3 |
| Parallel efficiency: % of linear scaling | | | | | | | | | | | | |
| 100.0 | 98.7 | 96.5 | 96.6 | 92.8 | 90.3 | 87.7 | 89.0 | 84.2 | 84.9 | 83.5 | 80.7 | 77.3 |

## 6 Conclusions and Future Work

Accelerating memory access by arranging data and computations in an appropriate way is vital for achieving the high application performance on modern computing architectures. Applications with a poor data locality reduce the effectiveness of the memory hierarchy, causing long stall times waiting for data accesses. A purposeful management of data locality plays the primary role for enabling applications to run on different architectures efficiently. The above statement refers in particular to SMP/NUMA systems, which are characterized by heterogeneous network structures. In consequence, since data can be physically dispersed over many nodes, the access to various data items may require significantly different times. This favours accesses to the local memory as fastest.

The present paper faces this challenge for heterogeneous stencil computations, where MPDATA is an important example of such scientific codes. For this purpose, the new islands-of-cores approach is proposed aiming at increasing the efficiency of stencil computations on SMP/NUMA platforms, by improving the data locality. This approach exposes a correlation between computation and communication for heterogeneous stencils, enabling a better management of the trade-off between computation and communication costs in accordance with the features of SMP/NUMA systems, such as the SGI UV 2000 server used in this work. To overcome the non-uniform memory access constraints, the proposed approach combines the previously developed (3+1)D decomposition and the scenario of parallelizing stencil computations when the implicit data traffic between nodes is replaced by extra computations. As a result, the resulting parallel code scales well with increasing the number of processors, and radically better than both the original version and pure (3+1)D decomposition.

In particular, for the MPDATA grid of size $1024 \times 512 \times 64$, approximately 30% of the theoretical peak is achieved when using less than 12 processors, while it decreases up to the level of 26% for large configurations. In this benchmark, the maximum sustained performance of about 390 Gflop/s is obtained for the maximum configuration with 112 cores of 14 Intel Xeon E5-4627v2 3.3 GHz processors. It corresponds to about 77% of the linear scaling. For smaller values of $P$, the parallel efficiency decreases from 96.6% for $P = 4$ to 80.7% for $P = 12$.

The achieved results justify further research on improving the efficiency of heterogeneous stencil computations on modern architectures. In particular, the proposed islands-of-cores approach can be applied to optimize computations within every multicore CPU (or manycore accelerator). At the opposite edge of the scale, we plan to study the usage of MPI for extending the scalability of our approach for much large system configurations. This requires to build performance models and methods for modeling and management of the correlation between computation and communication costs, to study its impact on the sustained performance. The optimal trade-off between computations and communications inside and between processors should be determined on this basis.

**Acknowledgments.** This work was supported by the National Science Centre (Poland) under grant UMO-2015/17/D/ST6/04059, as well as partially supported by the Ministry of Education, Youth and Sports of Czech Republic from the project "IT4Innovations National Supercomputing Center LM2015070", and by EU under the COST Program Action IC1305 "Network for Sustainable Ultrascale Computing (NESUS)" and its Czech supporting project LD15105 "Ultrascale Computing in Geosciences".

## References

1. Cao, X., et al.: Accelerating data shuffling in MapReduce framework with a scale-up NUMA computing architecture. In: Proceedings of the 24th High Performance Computing Symposium, HPC 2016. International Society for Computer Simulation (2016)
2. Castro, M., Francesquini, E., Nguélé, T.M., Méhaut, J.F.: Analysis of computing and energy performance of multicore, NUMA, and manycore platforms for an irregular application. In: Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms. ACM (2013)
3. Ciznicki, M., Kulczewski, M., Kopta, P., Kurowski, K.: Methods to load balance a GCR pressure solver using a stencil framework on multi-and many-core architectures. Sci. Program. (2015)
4. Culler, D., Pal Singh, J., Gupta, A.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers Inc., San Francisco (1999)
5. Czarnul, P.: Benchmarking performance of a hybrid Xeon/Xeon Phi system for parallel computation of similarity measures between large vectors. Int. J. Parallel Program. 1–17 (2017)
6. Guo, J., Bikshandi, G., Fraguela, B.B., Padua, D.: Writing productive stencil codes with overlapped tiling. Concurr. Comput. Pract. Exp. **21**(1), 25–39 (2009)
7. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multi-core chips via simple machine models. Concurr. Comput. Pract. Exp. **28**(22), 189–210 (2016)
8. National Supercomputing Center IT4Innovations (2017). http://www.it4i.cz
9. Kumar, S., Bhattacharyya, R., Joshi, B., Smolarkiewicz, P.: On the role of repetitive magnetic reconnections in evolution of magnetic flux ropes in solar corona. Astrophys. J. **830**(2), 80 (2016)
10. Lastovetsky, A., Szustak, L., Wyrzykowski, R.: Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. IEEE Trans. Parallel Distrib. Syst. **28**(3), 787–797 (2017)
11. SGI Products: Servers SGI UV (2015). https://www.sgi.com/products/servers/uv/
12. SGI UV 2000 System User Guide. Document Number 007–5832-002 (2013)
13. Smolarkiewicz, P.: Multidimensional positive definite advection transport algorithm: an overview. Int. J. Numer. Methods Fluids **50**(10), 1123–1144 (2006)
14. Smolarkiewicz, P., Margolin, L.: MPDATA: a finite-difference solver for geophysical flows. J. Comput. Phys. **140**(2), 459–480 (1998)
15. Smolarkiewicz, P.K., Charbonneau, P.: EULAG, a computational model for multiscale flows: an MHD extension. J. Comput. Phys. **236**, 608–623 (2013)
16. Smolarkiewicz, P.K., Szmelter, J., Xiao, F.: Simulation of all-scale atmospheric dynamics on unstructured meshes. J. Comput. Phys. **322**(C), 267–287 (2016)

17. Strugarek, A., Beaudoin, P., Brun, A., Charbonneau, P., Mathis, S., Smolarkiewicz, P.: Modeling turbulent stellar convection zones: sub-grid scales effects. Adv. Space Res. **58**(8), 1538–1553 (2016)
18. Szustak, L., Rojek, K., Gepner, P.: Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013. LNCS, vol. 8384, pp. 582–592. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55224-3_54
19. Szustak, L., Rojek, K., Olas, T., Kuczynski, L., Halbiniak, K., Gepner, P.: Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci. Program. (2015). doi:10.1155/2015/642705
20. Szustak, L., Rojek, K., Wyrzykowski, R., Gepner, P.: Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture. In: Proceedings of the 1st International Workshop on High-Performance Stencil Computations, HiStencils 2014, pp. 51–56 (2014)
21. Treibig, J., Hager, G., Wellein, G.: LIKWID: a lightweight performance-oriented tool suite for x86 multicore environments. In: Proceedings of the First International Workshop on Parallel Software Tools and Tool Infrastructures, PSTI 2010, San Diego, CA (2010)
22. Unat, D., et al.: Programming abstractions for data locality. (2014). http://web.eecs.umich.edu/akamil/papers/padal14report.pdf
23. Utrera, G., Gil, M., Martorell, X.: In search of the best MPI-OpenMP distribution for optimum Intel-MIC cluster performance. In: 2015 International Conference on High Performance Computing and Simulation (HPCS), pp. 429–435. IEEE (2015)
24. Xue, W., et al.: Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on Tianhe-2. IEEE Trans. Comput. **64**(8), 2382–2393 (2015)
25. Yasui, Y., Fujisawa, K., Goh, E.L., Baron, J., Sugiura, A., Uchiyama, T.: NUMA-aware scalable graph traversal on SGI UV systems. In: Proceedings of the ACM Workshop on High Performance Graph Processing, pp. 19–26. ACM (2016)
26. Zhou, X., Giacalone, J.P., Garzarán, M.J., Kuhn, R.H., Ni, Y., Padua, D.: Hierarchical overlapped tiling. In: Proceedings of the Tenth International Symposium on Code Generation and Optimization, pp. 207–218. ACM (2012)