The Journal of Supercomputing (2019) 75:7765–7777
https://doi.org/10.1007/s11227-018-2460-0

Unleashing the performance of ccNUMA multiprocessor architectures in heterogeneous stencil computations

Lukasz Szustak[1] · Kamil Halbiniak[1] · Roman Wyrzykowski[1] · Ondřej Jakl[2]

Published online: 23 June 2018

Abstract

This paper meets the challenge of harnessing the heterogeneous communication architecture of ccNUMA multiprocessors for heterogeneous stencil computations, an important example of which is the Multidimensional Positive Definite Advection Transport Algorithm (MPDATA). We propose a method for optimization of parallel implementation of heterogeneous stencil computations that is a combination of the islands-of-core strategy and (3+1)D decomposition. The method allows a flexible management of the trade-off between computation and communication costs in accordance with features of modern ccNUMA architectures. Its efficiency is demonstrated for the implementation of MPDATA on the SGI UV 2000 and UV 3000 servers, as well as for 2- and 4-socket ccNUMA platforms based on various Intel CPU architectures, including Skylake, Broadwell, and Haswell.

Keywords Stencil codes · MPDATA · Shared memory platforms · ccNUMA · SGI UV 3000 · SGI UV 2000 · Skylake · Broadwell · Haswell

The authors are grateful to: (i) Intel Technology Poland, (ii) IT4Innovations National Supercomputing Center, Technical Univ. of Ostrava, Czech Republic, and (iii) MICLAB project no. POIG.02.03.00.24-093/13 for granting access to HPC platforms. This work was supported by the National Science Centre (Poland) under Grant UMO-2017/26/D/ST6/00687..

✉ Lukasz Szustak
lszustak@icis.pcz.pl

Kamil Halbiniak
khalbiniak@icis.pcz.pl

Roman Wyrzykowski
roman@icis.pcz.pl

Ondřej Jakl
ondrej.jakl@ugn.cas.cz

[1] Czestochowa University of Technology, Dabrowskiego 69, 42-201 Czestochowa, Poland

[2] Institute of Geonics of the Czech Academy of Sciences, Studentská 1768, 708 00 Ostrava-Poruba, Czech Republic

1 Introduction

Modern ccNUMA multiprocessor systems offer increasing performance and amount of memory at the cost of considerable internal complexity [4]. They allow using the shared-variable programming model to take advantages of shared memory for inter-process communications and synchronizations. However, the memory and communication constraints may strongly limit the performance attainable on ccNUMA platforms, where physical cores of processors are connected by a heterogeneous communication structure. As data can be physically dispersed over many nodes, the access to various data items may require significantly different times. This favors accesses to the local memory as fastest. As a result, applications with a poor data locality reduce the efficiency of the memory hierarchy, causing long waiting times for access to data.

One of leading vendors of ccNUMA multiprocessor servers was SGI, that for more 20 years has been delivering SMP/NUMA (symmetric multiprocessor/non-uniform memory access) platforms built around high-performance NUMAlink networks as distributed shared memory systems. An example of advanced SGI's product series is SGI UV ("Ultra Violet"), and in particular, SGI UV 2000 [18] and UV 3000 [14] servers. Based on Intel multicore CPUs and the high-speed NUMAlink system interconnect, they offer up to thousands of cores in a single system which shares large main memory. After selling SGI to HPE, these systems are now known as HPE Integrity and HPE Superdome servers [9]. A budget alternative to such expensive systems is 2- and 4-socket ccNUMA servers consisting of two or four Intel multicore CPUs. The most powerful among them utilizes the newest Intel Xeon Scalable (Skylake) CPUs and UPI interconnect [11].

As an example of using SGI UV systems to speed up a complex real-world application, MapReduce was presented in [1], where a topology-aware placement algorithm was proposed to accelerate the data shuffling phase of MapReduce. The first generation of SGI UV platforms was applied in [3] to parallelize the Generalized Conjugate Residual (GCR) elliptic solver with preconditionner, using a mixture of MPI and OpenMP.

Aside from numerical applications, the SGI UV systems were also efficiently used in other areas, such as computation on graphs [23] and combinatorial optimization problems [2]. The results of performance comparison of 2-socket servers with various Intel Xeon processors are presented in work [11], for a set of benchmarks such as NAMD parallel molecular dynamic code, NAS Parallel Benchmarks, compression.

In our previous work [18], we have taken up the challenge of harnessing the heterogeneous nature of communications in SGI SMP/NUMA platforms for the MPDATA application (Multidimensional Positive Definite Advection Transport Algorithm), which consists of a set of heterogeneous stencil kernels. The islands-of-cores approach (strategy) was proposed for heterogeneous stencils, based on replacement of implicit data transfers between caches of adjacent processors at the cost of extra computations. The proposed strategy allowed us to manage the balance between computations and communications in accordance with features of SMP/NUMA systems such as SGI UV 2000.

In this paper, we adapt this strategy to develop a method for optimization of parallel implementation of heterogeneous stencil computations on ccNUMA multiprocessor architectures. This method combines the island-of-core strategy with the (3+1)D hier-

archical decomposition proposed previously in [19,21]. The efficiency of the method is evaluated for the implementation of MPDATA on the SGI UV 2000 and UV 3000 servers, as well as 2- and 4-socket ccNUMA platforms based on various Intel CPU microarchitectures, including Skylake, Broadwell, and Haswell.

To our best knowledge, there exists no investigation 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 [7,24], and particularly in work [8] devoted to generating and optimizing stencil programs automatically. 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 also to our study, these approaches are addressed to small computing platforms with one or two processors.

2 Overview of ccNUMA multiprocessor systems

The ccNUMA multiprocessor systems that we are explored in this paper share a unique logical address space, which is mapped on a physical memory that is distributed among the processors [6]. It is the hardware layer that makes all CPUs of a multiprocessor system see the main memory in a consistent way, including the cache hierarchy. Every computing unit can read and write a data item simply using load and store operations, so the processes communication is through the shared memory. With the ccNUMA architecture, although the whole main memory is shared via an interconnecting subsystem (point-to-point links, bus, crossbar, etc.), the access to different portions of distributed shared memory may require significantly different times. Generally, an access to local memory blocks is quicker than access to remote ones. However, due to automatic data movements and coherent replications in caches, the cost of communication is invisible for programmers.

In the last period, 2- and 4-socket platforms with Intel CPUs dominate the segment of relatively inexpensive multiprocessor servers. They are built around a high-speed point-to-point interconnect. Previously, it was QPI (QuickPath Interconnect), which currently has been replaced by UPI (Ultra Path Interface) that provides the coherent interconnect for systems based on Intel Xeon Scalable (Skylake) CPUs [11]. Equipped with 2 or 3 UPI links connecting to other CPUs, they are able to scale a single platform up to 8 sockets. Apart from increasing the data rate, UPI allows also improving the power efficiency.

At the same time, the third generation of the SGI UV product line launched in 2014, and specifically the UV 3000 model [13,14], enables expanding the hardware configuration up to 256 sockets (4096 cores) with 64TB of cache-coherent shared memory, for a single system. This is possible thanks to the NUMAlink proprietary interconnect with an enhanced hypercube topology.

Table 1 summarizes parameters of servers used in our benchmarks. The considered platforms provide the theoretical peak performance from about 0.5 to about 6 Tflops/s for double-precision floating-point format. These values are calculated for

Table 1 Specification of multiprocessor platforms (https://ark.intel.com)

Computing platform		CPU archit./ Interconnect	# CPUs (Cores)	Freq. (GHz) (SIMD Freq.)	SIMD	Peak (Tflop/s)
A	4× Intel Xeon	Skylake	4	2.5	AVX-512	3.04
	Platinum 8180	UPI	(4 × 28)	(1.7)		
B	4× Intel Xeon	Broadwell	4	2.2	AVX2	1.38
	E7-8890 v4	QPI	(4 × 24)	(1.8)		
C	2× Intel Xeon	Haswell	2	2.6	AVX2	0.49
	E5-2697 v3	QPI	(2 × 14)	(2.2)		
E	SGI UV 3000					
	32× Intel Xeon	Haswell	32	2.6	AVX2	5.88
	E5-4627 v3	NUMAlink 6+	(32 × 10)	(2.3)		
E	SGI UV 2000					
	14× Intel Xeon	Ivy Bridge	14	3.3	AVX	1.47
	E5-4627 v2	NUMAlink 6	(14 × 8)			

SIMD operations of multiplication type (`Vec Mul`), considering the total number of cores, and base SIMD frequency [5,10].

3 Introduction to parallelization of MPDATA application on shared memory systems

The MPDATA application implements a general approach for integrating the conservation laws of geophysical fluids on micro-to-planetary scales [15]. The MPDATA algorithm enables solving advection problems, and offers several options to model a wide range of complex geophysical (and even stellar [16]) 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. In this paper, we consider using the algorithm to solve problems defined on 3D grids.

Every MPDATA time step corresponds to the same computations, which consist of a set of 17 kernels [20,21]. The MPDATA kernels represent the heterogeneous stencils codes which update grid elements according to different patterns. The kernels depend on each others: Outcomes of prior stages are usually input data for subsequent computations. In the original version of MPDATA code, the consecutive kernels are processed sequentially, one by one, where each kernel is processed in a parallel way using the OpenMP standard. Every MPDATA kernel reads a required set of arrays from the main memory and writes results to the main memory after computation. The consequence is significant data traffic to the main memory, that mainly limits the efficient usage of novel multi-/manycore architectures.

To alleviate the memory-bound nature of MPDATA, we proposed [19–21] a new strategy of workload distribution for heterogeneous stencils computations. The main aim was to better exploit the cache hierarchy by moving the bulk of data traffic from the main memory to the cache hierarchy. As a result, a new version of MPDATA was successfully developed for small-scale shared memory systems. In particular, we developed the (3+1)D decomposition of MPDATA, that is based on a combination of two loop optimization techniques: loop tiling and loop fusion. The proposed decomposition allows us to significantly reduce the main memory traffic, where the real profit depends on the size of domains, as well as characteristics of a given computing platform.

The proposed strategy allows a significant performance gain not only for 1- and 2-socket platforms, but also for Intel Xeon Phi accelerators [12,20]. However, it does not permit avoiding significant performance losses for larger multiprocessor architectures [18]. The (3+1)D decomposition improves both cache reusing and the data locality, but results in an intensive intra- and inter-cache communication between cores and processors. In particular, if a single processors is used, the data traffic is restricted to its cache hierarchy only. But if more processors cooperate to execute MPDATA, the required data are also implicitly transferred between caches of neighbor processors.

4 Islands-of-cores strategy

In order to overcome the described shortcoming and improve the efficiency of the MPDATA application, we proposed the islands-of-cores strategy [18] dedicated to heterogeneous stencils such as those of MPDATA. This strategy exposes the correlation between computation and communication for heterogeneous stencils, enabling a better management of the trade-off between computation and communication costs in accordance with features of various shared memory systems. The idea of islands-of-cores strategy is presented in Fig. 1. It presents an example of 1D stencil computations with three kernels (Fig. 1a), and two general scenarios of parallelizing these computations on two interconnected processors (Fig. 1b, c).

The first scenario (Fig. 1b) reveals an implicit inter-processor communication because of data dependencies. In particular, the output element $C[d]$ computed by CPU_B within the third kernel depends on the element $B[c]$ that is computed by CPU_A as a result of kernel 2. However, $B[c]$ depends on the element $A[d]$ which is returned by CPU_B in the first kernel. In consequence, implicit transfers of two elements take place between the processors CPU_A and CPU_B of a shared memory platform. Furthermore, three synchronization points are required for ensuring the correctness of parallel computations. In general, the data transfers between cores/processors are implemented through the cache hierarchy. As a result, the required data will be implicitly transferred between caches of neighbor processors through, e.q., QPI or UPI links [10].

The second scenario (Fig. 1c) allows avoiding transfers of data between processors at the cost of extra computations. Instead of transferring elements $A[d]$ and $B[c]$ computed by the first and second kernels, let both processors compute these elements once more. In consequence, CPU_A computes one extra element $A[d]$ of the first

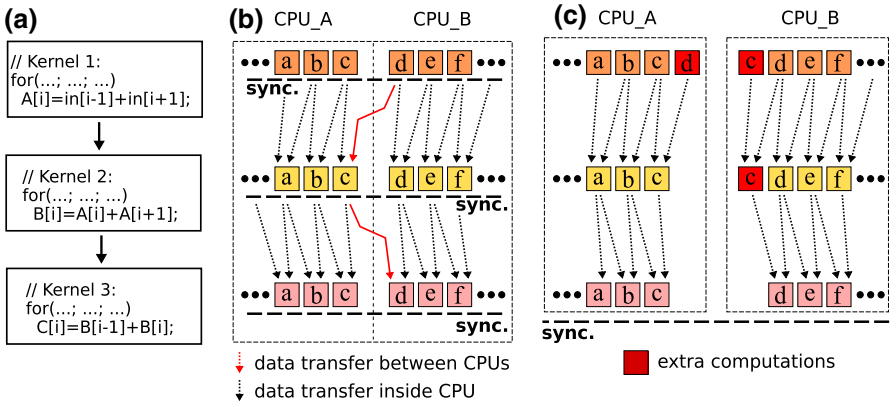

Fig. 1 Idea of Islands-of-cores strategy [18]: **a** example of stencil computations with three kernels; **b** parallelization with implicit data transfers between processors; **c** avoiding data transfers and synchronization, at the cost of extra computations

kernel, and CPU_B delivers the additional element $B[c]$ within kernel 2. However, the additional element $B[c]$ depends on the element $A[c]$ from kernel 1, which is computed by CPU_A in the first scenario. To avoid the transfer of element $A[c]$ between processor, let both of them compute this element twice. As a result, similar to independent islands, both processors perform computations independently of each other, at the cost of computing some amount of extra elements.

To sum up, the first scenario performs less computations but requires more data traffic, while the second one minimizes the data traffic between processors by replicating some computations. In general, both solutions have to be considered, but the key point is how they fit to the architecture of a given computing platform. The second scenario seems to fit perfectly to reduce inter-processor communications between caches of processors in multiprocessor architectures. On the contrary, the first scenario is well suited to be implemented inside processors as they provide a more efficient interconnect between cores.

5 Optimization of MPDATA using the Island-of-core strategy

In this section, we propose a method that allows us to customize the island-of-core strategy to the MPDATA application, which has a much more complex structure [20] than the example shown in Fig. 1. We use the second scenario to reduce communications between caches of neighbor processors inside every MPDATA time step. To achieve this aim, the abstraction of islands-of-cores is applied across P processors of a given multiprocessor system, where a processor is identified with an island (or work team) of cores.

Assuming that each work team consists of the same number of cores, the MPDATA domain is evenly decomposed into sub-domains (Fig. 2a), so their number corresponds to the number of processors. The MPDATA sub-domains are then processed in parallel

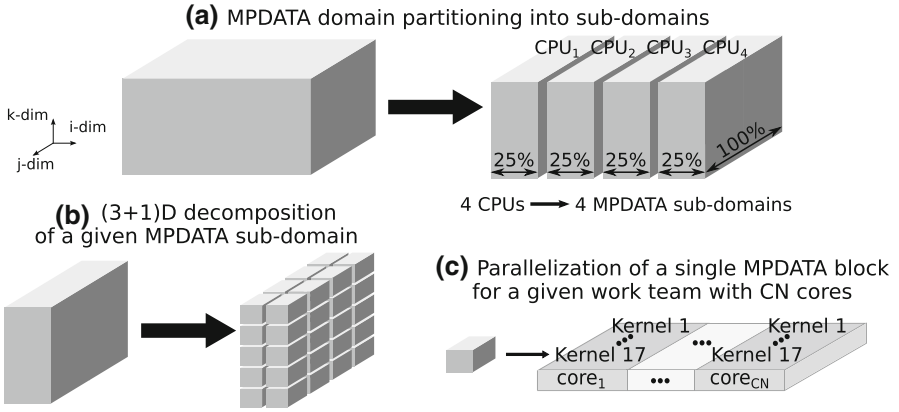

Fig. 2 Hierarchical domain decomposition of MPDATA: **a** domain partitioning into sub-domains following the islands-of-cores strategy, **b** sub-domain decomposition into blocks of size adjusted to capacity of the cache memory, and **c** parallel execution of MPDATA kernels within a single block by a given work team

by work teams, since every processor is able to execute computations within each MPDATA time step independently of other processors, at the cost of extra computation.

Afterward, following the (3+1)D decomposition, each sub-domain is further partitioned into a set of blocks with size that enables keeping all the necessary data in the cache memory (Fig. 2b). The successive blocks are processed sequentially, one by one, where a block is processed in parallel by a work team of cores (Fig. 2c). Each work team executes computations for all the MPDATA kernels on corresponding chunks of arrays, including computing extra elements instead of transferring them from other processors.

For a given multiprocessor, all work teams will perform the following activities in each time step (see Fig. 3):

1. All the work teams share input data, utilizing the first-touch policy with parallel initialization.
2. At the cost of extra computations, each team independently executes the set of blocks within its sub-domain, following the (3+1)D decomposition.
3. After completing the whole time step, each team returns outcomes to the main memory. Additionally, all the teams synchronize their operations, in order to ensure the correctness of input data for the next time step.

The islands-of-cores strategy alleviates also the cost of synchronization between processors. As shown in Fig. 3, we distinguish two levels of synchronization: between processors, and between cores within each processor. In particular, after finishing independent computations within a given time step, all the processors have to synchronize their activities, in order to ensure the correctness of input data for the next time step. This level of synchronization is successfully implemented by using the traditional barrier approach to synchronize all threads.

The second level of synchronization is required to synchronize cores of each processor within time steps. In fact, each MPDATA block requires five synchronization points to ensure the correctness of results. The amount of synchronization points depends on

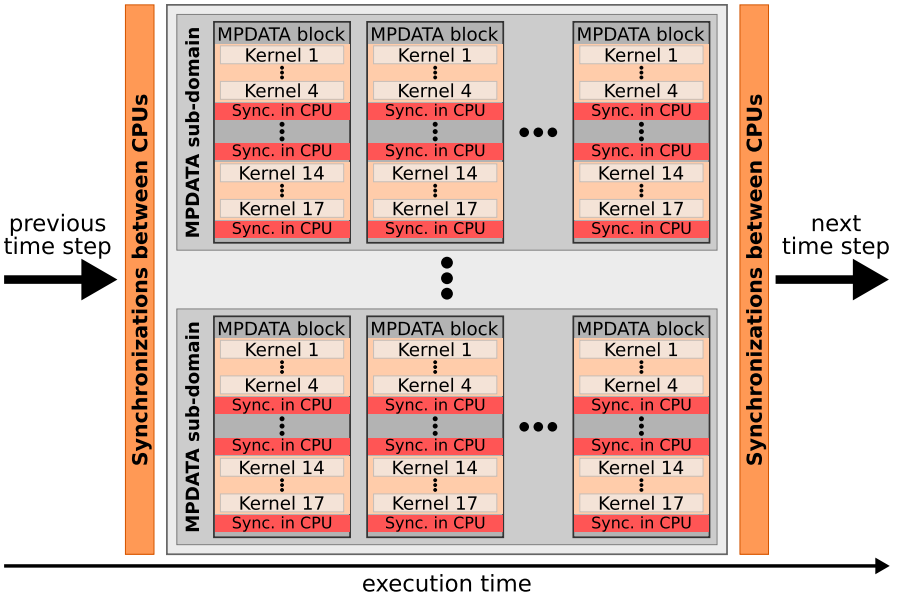

Fig. 3 Executing a single MPDATA time step

Table 2 Percentage of extra elements for the MPDATA domain of size $1024 \times 512 \times 64$

# of islands	1	2	4	8	14	16	32
% of extra elements	0.00	0.25	0.74	1.73	3.21	3.71	7.66

the number of MPDATA blocks. In consequence, a huge number of synchronization points is expected for the second level (see Fig. 3). Its efficiency becomes critical for the overall performance. For this reason, in our previous work [17] we proposed a novel strategy for the data-flow synchronization in shared memory systems. The main idea of this strategy is to synchronize only interdependent cores assigned to a given work team, instead of using the barrier approach to synchronize all the threads.

The overall efficiency of the proposed method depends on a way how the MPDATA domain is partitioned. In practice, only 1D variants of partitioning are considered. The reason for avoiding 2D and 3D variants is that data layouts of all the MPDATA arrays allow transfers of contiguous areas of memory along the first dimension only. In consequence, too high communication overheads are expected when the MPDATA domain is partitioned along two or three dimensions. Finally, the MPDATA domain of size $m \times n \times l$ is evenly decomposed into sub-domains of size $\frac{m}{P} \times n \times l$. The way of partitioning together with the structure of data dependencies between the MPDATA kernels impacts the total amount of extra elements that have to be computed redundantly. Table 2 illustrates how the total number of extra elements increases with the number of work teams.

As data transfers take place only between borders of neighbor MPDATA sub-domains, the adjacent sub-domains are mapped onto processors that are physically

Table 3 Execution times achieved for the basic version, pure (3+1)D decomposition, and the new implementation, using 2- and 4-socket ccNUMA platforms, as well as partial S^* and overall S speedups obtained against the (3+1)D decomposition and basic version

Computing platform		Execution time (s)				Speedups	
		Basic	(3+1)D	2 teams	4 teams	S^*	S
A	4× Intel Xeon Platinum 8180	385.3	225.2	140.3	37.1	6.07	10.38
B	4× Intel Xeon E7-8890 v4	518.3	216.6	167.7	69.3	3.12	7.48
C	2× Intel Xeon E5-2697 v3	1123.4	320.4	193.0	–	1.66	5.82

closely connected with each other, in order to reduce the communication paths. This is achieved by selecting the correct policy for the OpenMP thread affinity interface, that allows binding threads to cores. At the same time, the complexity of the proposed hierarchical decomposition makes it impossible to implement efficiently the multithreading parallelization using OpenMP constructs such as `#pragma omp for`. Instead, we develop a proprietary scheduler responsible for the management of workload distribution and data parallelism. For each OpenMP thread, this scheduler explicitly defines the scope of work. Furthermore, for optimizing the performance of shared memory platforms, it is of vital importance to allocate memory *closest* to a physical core on which a given thread is executed. This is achieved with the use of the technique known [22] as the first-touch policy with parallel initialization.

6 Performance results

This section presents the performance results obtained for the new implementation of MPDATA developed using the method proposed in this work. All benchmarks are obtained for the double-precision floating-point format on the domain of size $1024 \times 512 \times 64$, assuming 5000 times steps. The benchmarks are executed for the platforms outlined in Table 1. In the tests, the Intel icpc compiler is used (v.18.0.1 for 2- and 4-socket platforms, and v.17.0.1 for SGI systems) with the optimization flag -O3 and properly chosen compiler arguments that support the full use of SIMD hardware [5]. In order to guarantee the reliability of benchmark results, the measurements of execution time are repeated several times, and the median value of measurements is used finally.

At first, the efficiency of the proposed method is evaluated for 2- and 4-socket ccNUMA platforms based on various Intel CPU architectures, including Skylake, Broadwell, and Haswell. The performance results are shown in Table 3, which presents the execution time achieved for different numbers of work teams. This time is compared with the execution time obtained for the original (basic) version, and for the pure (3+1)D decomposition, so without partitioning the MPDATA workload across work teams. For this aim, the partial S^* and overall S speedups are calculated to show the performance gains of the proposed method against the pure (3+1)D decomposition and basic version, respectively.

Table 4 Parallel efficiency achieved for different numbers of processors in 2- and 4-socket platforms, expressed as percentage of linear scaling

Computing platform		Number of utilized CPUs			
		1 (%)	2 (%)	3 (%)	4 (%)
A	Intel Xeon Platinum 8180	100	97.91	94.06	91.84
B	Intel Xeon E7-8890 v4	100	97.39	94.50	90.51
C	Intel Xeon E5-2697 v3	100	99.01	–	–

The main conclusion from Table 3 is that the proposed method allows us to improve radically the efficiency of the MPDATA application in comparison with the other two versions. As expected, despite extra computations, the new MPDATA code is executed much faster for all the tested platforms. The highest performance gain is achieved for the 4-socket server with Skylake CPUs (platform A), where the new code yields the partial speedup of about 6 times and overall speedup of about 10 times against the pure (3+1)D decomposition and original version, respectively. But even for the 2-socket server, the new implementation allows accelerating the MPDATA application considerably, that follows from the speedup of 1.66× against the pure (3+1)D decomposition.

The islands-of-cores strategy permits us to speed up the computations even when using less work teams than the available processors. However, the performance gain is considerably reduced in this case. For example, for the platform A, the execution time is decreasing from about 225 s, for the pure (3+1)D decomposition, to about 140 s when using 2 work teams mapped onto 4 processors. But partitioning the workload across 4 work teams permits further decrease to only 37.1 s. In addition, Table 4 presents the parallel efficiency expressed as percentage of linear scaling. The tested 4-socket platforms demonstrate relatively small drops in efficiency with increasing the number of processors. A point worth noting is 99% of linear scaling achieved on the 2-socket platform.

Then, the performance and scalability of the proposed method are evaluated for the SGI UV 3000 and UV 2000 servers. The obtained results are collected in Table 5, which shows the execution time, and sustained performance Q (in Gflop/s), as well as the utilization rate $R = Q/Q_{\max}$, where $Q_{\max}$ is the theoretical peak performance of a given server. Finally, we present the parallel efficiency E_p expressed as percentage of linear scaling, that is achieved for different values of the number p of utilized CPUs. For SGI UV 2000, the partial speedup S_p^* over the (3+1)D decomposition is included as well.

As follows from Table 5, the maximum sustained performance of about 1.75 Tflop/s is obtained for the SGI UV 3000 server with the use of all the available processors ($p = 32$), that corresponds to the utilization rate R of about 30%. In these benchmarks, approximately 42% of the theoretical peak performance is achieved for a single CPU of each server. The utilization rate R decreases finally to about 30 and 26% for, respectively, 32 CPUs of SGI UV 3000 and 14 CPUs of SGI UV 2000. The SGI UV 3000 server allows achieving a relatively better parallel efficiency, that decreases from

Table 5 Execution time achieved for the islands-of-cores strategy on the SGI UV 3000 and SGI UV 2000 servers, as well as sustained performance (Gflop/s), utilization rate (%), and parallel efficiency (%)

SGI UV 3000	Number p of utilized CPUs	1	4	8	16	32
	Execution time (s)	497	127	69	37	24
	Sustained performance Q (Gflop/s)	78.8	311.7	578.9	1096.4	1754.6
	Utilization rate R (%)	42.85	42.35	39.33	37.24	29.80
	Parallel efficiency E_p: % of linear scaling	100	98.26	90.57	84.49	65.66
SGI UV 2000	Number p of utilized CPUs	1	2	4	8	14
	Execution time (s)	900	562	293	149	101
	Sustained performance Q (Gflop/s)	42.7	68.5	131.9	264.4	390.1
	Utilization rate R (%)	40.44	32.43	31.23	31.30	26.39
	Parallel efficiency E_p: % of linear scaling	100	80.07	76.79	75.50	63.65
	Partial speedup S_p^*	1.0	1.46	2.72	5.16	10.30

For SGI UV 2000, the speedup S_p^* over the (3+1)D decomposition is shown as well

Table 6 Performance comparison of the new implementation of MPDATA on all platforms

Computing platform		CPUs	Cores	Time (s)	Q (Gflop/s)	R (%)
A	4× Intel Xeon Platinum 8180	4	112	37.1	1076.9	35.35
B	4× Intel Xeon E7-8890 v4	4	96	69.3	569.3	41.18
C	2× Intel Xeon E5-2697 v3	2	24	193.0	205.8	41.76
D	32× Intel Xeon E5-4627 v3	32	320	24.0	1754.6	29.80
E	14× Intel Xeon E5-4627 v2	14	112	101.1	390.1	26.39

about 98% for $p = 4$ to about 66% for $p = 32$. For the SGI UV 2000 server, we obtain about 64% of linear scaling on 14 processors. Finally, the analysis of the speedup S_p^* achieved on SGI UV 2000 against the pure (3+1)D decomposition, permits us to conclude that the performance gain of the proposed method increases with the growing number of available processor.

The performance comparison of the new implementation of MPDATA for all the considered platforms is provided in Table 6. It shows the execution time and sustained performance Q, as well as the utilization rate R. The most powerful platform is the SGI UV 3000 server that consists of 32 Intel Xeon Haswell CPUs, and allows us to achieve the sustained performance of about 1.75 Tflop/s (29.8% of the peak). At the same time, the 4-socket platform with the newest Intel Xeon Scalable (Skylake) CPUs is able to execute the new MPDATA code with the sustained performance of around 1.08 Tflop/s (35.35% of the peak).

7 Conclusions

This paper meets the challenge of harnessing the heterogeneous communication architecture of ccNUMA multiprocessors for heterogeneous stencil computations, an

important example of which is the MPDATA application. We propose the method for optimization of parallel implementation of heterogeneous stencil computations that is a combination of the islands-of-core strategy and (3+1)D decomposition. The method allows a better management of the trade-off between computation and communication costs in accordance with features of different ccNUMA architectures.

The proposed method is an efficient and flexible solution which allows us to provide the performance portability across various ccNUMA architectures, including 2- and 4-socket platforms with Intel CPUs, as well as the SGI UV 2000 and SGI UV 3000 servers. The resulting parallel code scales well with increasing the number of processors, and despite extra computations execute computation radically faster then both the original version and the code based on the (3+1)D decomposition only.

It is shown that the tested 4-socket platforms demonstrate relatively small drops in parallel efficiency with increasing the number of processors. For 2, 3, and 4 processors, the parallel efficiency decreases relatively slightly, and achieves, respectively, about 97, 94, and 91% of linear scaling. The point worth noting is 99% of linear scaling achieved on the 2-socket platform. For SGI UV 3000, we achieve about 66% of linear scaling using all 32 processors, while for SGI UV 2000 the parallel efficiency decreases to about 64% of linear scaling when utilizing 14 processors.

The highest sustained performance of 1.75 Tflop/s (about 30% of the peak performance) is obtained for the SGI UV 3000 server using 32 Intel Xeon Haswell CPUs (320 cores in total). At the same time, the 4-socket platform with the newest Intel Xeon Scalable (Skylake) CPUs executes the new MPDATA code on 112 cores with the sustained performance of 1.08 Tflop/s (35% of the peak).

References

1. Cao X, Panchputre K, Du D (2016) Accelerating data shuffling in MapReduce framework with a scale-up NUMA computing architecture. Simul Ser 48(4):122–129
2. Castro M, Francesquini E, Nguélé T.M, Méhaut J.F (2013) Analysis of computing and energy performance of multicore, NUMA, and manycore platforms for an irregular application. In: Proceedings of 3rd Workshop on Irregular Applications: Architectures and Algorithms. ACM
3. Ciznicki M, Kulczewski M, Kopta P, Kurowski K (2015) Methods to load balance a GCR pressure solver using a stencil framework on multi-and many-core architectures. Sci Program 2015:24
4. Culler D, Pal Singh J, Gupta A (1999) Parallel computer architecture: a hardware/software approach. Morgan Kaufmann Publishers Inc., Burlington
5. Eltablawy A, Vladimirov A (2015) Capabilities of Intel AVX-512 in Intel Xeon scalable processors (Skylake). Colfax International, Sunnyvale
6. Ferretti M (2017) Advanced computer architecture. Shared memory mutiprocessor. http://www-5.unipv.it/mferretti/cdol/aca/Charts/07-multiprocessors-MF.pdf. Accessed Mar 2018
7. Guo J, Bikshandi G, Fraguela BB, Padua D (2009) Writing productive stencil codes with overlapped tiling. Concurr Comput Pract Exp 21(1):25–39

8. Hagedorn B, Stoltzfus L, Steuwer M, Gorlatch S, Dubach C (2018) High performance stencil code generation with LIFT. In: Proceedings of 2018 IEEE/ACM International Symposium Code Generation and Optimization (CGO'18)
9. HPE Servers and Server Systems (2018) https://www.hpe.com/us/en/servers.html. Accessed Mar 2018
10. Intel 64 and IA-32 Architectures Optimization Reference Manual (2017) https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf. Accessed Mar 2018
11. Intel Xeon Platinum 8176 Scalable Processor Review (2018) https://www.tomshardware.com/reviews/intel-xeon-platinum-8176-scalable-cpu,5120.html. Accessed Mar 2018
12. Lastovetsky A, Szustak L, Wyrzykowski R (2017) Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. IEEE Trans Parallel Distrib Syst 28(3):787–797
13. SGI UV 3000, UV 30 (2016) https://www.risc.jku.at/projects/mach2/4555.pdf. Accessed Mar 2018
14. SGI UV 3000 Sets New Throughput Records (2016) https://www.hpcwire.com/2016/03/25/sgi-posts-new-spec-cpu2006-results/. Accessed Mar 2018
15. Smolarkiewicz P (2006) Multidimensional Positive Definite Advection Transport Algorithm: an overview. Int J Numer Meth Fluids 50(10):1123–1144
16. Strugarek A, Beaudoin P, Brun AS, Charbonneau P, Mathis S, Smolarkiewicz PK (2016) Modeling turbulent stellar convection zones: sub-grid scales effects. Adv Space Res 58(8):1538–1553
17. Szustak L (2018) Strategy for data-flow synchronizations in stencil parallel computations on multi-/manycore systems. J Supercomput 74(4):1534–1546
18. Szustak L, Jakl O, Wyrzykowski R (2017) Islands-of-cores approach for harnessing SMP/NUMA architectures in heterogeneous stencil computations. In: PaCT 2017, vol 10421. Lecture Notes in Computer Science, pp 351–364
19. Szustak L, Rojek K, Gepner P (2014) Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: PPAM 2013, vol 8384. Lecture Notes in Computer Science, pp 582–592
20. Szustak L, Rojek K, Olas T, Kuczynski L, Halbiniak K, Gepner P (2015) Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci Program 2015:10
21. Szustak L, Rojek K, Wyrzykowski R, Gepner P (2014) Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture. In: Proceedings of 1st International Workshop on High-Performance Stencil Computations—HiStencils 2014. In conjunction with HiPEAC 2014, pp 51–56
22. Unat D et al (eds) (2014) Programming abstractions for data locality. http://web.eecs.umich.edu/~akamil/papers/padal14report.pdf. Accessed Mar 2018
23. Yasui Y, Fujisawa K, Goh E.L, Baron J, Sugiura A, Uchiyama T (2016) NUMA-aware scalable graph traversal on SGI UV systems. In: Proceedings of ACM Workshop on High Performance Graph Processing. ACM, pp 19–26
24. Zhou X, Giacalone J.P, Garzarán M.J, Kuhn R, Ni Y, Padua D (2012) Hierarchical overlapped tiling. In: Proceedings of 10th International Symposium on Code Generation and Optimization. ACM, pp 207–218