J Supercomput (2017) 73:664–675
DOI 10.1007/s11227-016-1774-z

Performance modeling of 3D MPDATA simulations on GPU cluster

Krzysztof Rojek[1] · **Roman Wyrzykowski**[1]

Published online: 13 June 2016

Abstract The goal of this study is to parallelize the multidimensional positive definite advection transport algorithm (MPDATA) across a computational cluster equipped with GPUs. Our approach permits us to provide an extensive overlapping GPU computations and data transfers, both between computational nodes, as well as between the GPU accelerator and CPU host within a node. For this aim, we decompose a computational domain into two unequal parts which correspond to either data dependent or data independent parts. Then, data transfers can be performed simultaneously with computations corresponding to the second part. Our approach allows for achieving 16.372 Tflop/s using 136 GPUs. To estimate the scalability of the proposed approach, a performance model dedicated to MPDATA simulations is developed. We focus on the analysis of computation and communication execution times, as well as the influence of overlapping data transfers and GPU computations, with regard to the number of nodes.

Keywords EULAG · MPDATA · Stencils · GPU cluster · MPI · Performance model

This work was supported by the National Science Centre, Poland under Grant No. UMO-2015/17/D/ST6/04059, and by the grant from the Swiss National Supercomputing Centre (CSCS) under project ID d25.

✉ Krzysztof Rojek
krojek@icis.pcz.pl

Roman Wyrzykowski
roman@icis.pcz.pl

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

1 Introduction

EULAG (Eulerian/semi-Lagrangian fluid solver) is an established numerical model for simulating thermo-fluid flows across a wide range of scales and physical scenarios [12]. The EULAG model has the proven record of successful applications, and excellent efficiency and scalability on the conventional supercomputer architectures [18]. Currently, the model is being implemented as the new dynamical core of the COSMO weather prediction framework. The multidimensional positive definite advection transport algorithm (MPDATA) is among the most time-consuming calculations of the EULAG solver [5–16].

In our previous works [19,20], we proposed two decompositions of 2D MPDATA computations, which provide adaptation to CPU and GPU architectures. In the paper [20], we developed a hybrid CPU-GPU version of 2D MPDATA to fully utilize all the available computing resources. The next step of our research was to parallelize the 3D version of MPDATA. It required to develop a different approach [14] than for the 2D version. In paper [13], we presented an analysis of resources usage in GPU, and its influence on the resulting performance. We detected the bottlenecks and developed a method for the efficient distribution of computation across CUDA kernels.

The aim of paper [15] was to adapt 3D MPDATA computations to clusters with GPUs. Our approach was based on a hierarchical decomposition including the level of cluster, as well as an optimized distribution of computations between GPU resources within each node. To implement the resulting computing scheme, the MPI standard was used across nodes, while CUDA was applied inside nodes. We presented performance results for the 3D MPDATA code running on the Piz Daint cluster equipped with NVIDIA Tesla K20x GPUs. In particular, the sustained performance of 138 Gflop/s was achieved for a single GPU, which scales up to more than 11 Tflop/s for 256 GPUs.

In this paper, we focus on improving scalability of our code running across nodes of a GPU cluster. The focus of the MPI parallelization is on overlapping inter- and intra-node data transfers with GPU computations. Our approach is based on decomposing a computational domain into two unequal parts corresponding to either data-dependent or data-independent parts, where the stream processing is provided to overlap the data transfers with the data-independent part of computations. Since MPDATA consists of a set of stencils, the data-dependent part corresponds to a halo area of the computational domain. Moreover, we develop a performance model to estimate an appropriate number of nodes for achieving the highest scalability.

The rest of the paper is organized in the following way. Section 2 outlines related works, while an overview of 3D MPDATA is presented in Sect. 3. The parallelization of MPDATA is discussed in Sect. 4, focusing on the cluster level. The proposed performance model is introduced in Sect. 5. The experimental validation of this model is presented in Sect. 6, while Sect. 7 describes conclusions and future work.

2 Related works

The new architectures based on modern multicore CPUs and accelerators, such as GPU [6–14] and Intel Xeon Phi [17], offer unique opportunities for modeling atmospheric

processes significantly faster and with accuracy greater than ever before. In the last few years, papers devoted to the effective adaptation of the EULAG model to CPU and GPU architectures have appeared. The CPU-based implementations of solving partial differential equations in 3D space for Numerical Weather Prediction (NWP) problems were presented in [18]. As the GCR elliptic solver is one of the most significant part of the EULAG model, several techniques for porting this solver to a cluster with multicore CPUs have been studied in [5], based on combining the MPI and OpenMP standards.

Reorganizing stencil calculations to take full advantages of GPU clusters has been the subject of much investigation over the years. In particular, the fluid dynamics simulations of turbulence performed using up to 64 GPUs were presented in [10], where a second-order staggered-mesh spatial discretization was coupled with a low storage three-step Runge–Kutta time advancement and pressure projection. The performance results achieved on two GPU clusters, and a CPU cluster was discussed and compared. This paper shows that synchronization can have a profound effect on the GPU performance.

Taking into account the complexity of current computing clusters, the performance evaluation and modeling becomes crucial to enable the optimization of parallel programs. Designing and tuning MPI applications, particularly at large scale, requires understanding the performance implications of different choices for algorithm and implementation options [8]. In paper [1], the authors proposed an approach based on a combination of static analysis and feedback from a runtime benchmark for both communication and multithreading efficiency measurement. The proposed model collects some statistics, including communication latency and overheads, and analyzes MPI routines and OpenMP directives to estimate the performance of algorithms.

An interesting technique for performance modeling is presented in paper [4], which investigates how application performance can be modelled based on the number of memory page faults. The performance model to estimate the communication performance of parallel computers is proposed in [2], using the LogGP model as a conceptual framework for evaluating the performance of MPI communications. One of the main objectives of this work is to compare the performance of MPI communications on several platforms.

In paper [3], the authors focused on estimating the performance of stencil computations, where the expected execution time is evaluated based on time required for communication and computation, as well as overlapping time for those two operations. The authors proposed a method for prediction of the execution time for large-scale systems. On the contrary, one of the main advantage of our approach is that we do not need to measure the computation time and communication time separately. This permits for avoiding interference in the application that could damage the performance results, and allows us to increase the accuracy of modeling.

3 Overview of 3D MPDATA

The MPDATA algorithm is the main module of the multiscale fluid solver EULAG [12–16]. MPDATA solves the continuity equation describing the advection of a nondiffusive quantity Ψ in a flow field, namely

```
for(k=1; k <= n3m; ++k)
 for(j=1; j <= mp; ++j)
  for(i=1; i <= ilft; ++i)
   f1(i,j,k)=donor(c1*x(i-1,j,k)+c2,c1*x(i,j,k)+c2,v1(i,j,k));
for(k=1; k <= n3m; ++k)
 for(j=jbot; j <= mp; ++j)
  for(i=1; i <= np; ++i)
    f2(i,j,k)=donor(c1*x(i,j-1,k)+c2,c1*x(i,j,k)+c2,v2(i,j,k));
for(k=2; k <= n3m; ++k)
 for(j=1; j <= mp; ++j)
  for(i=1; i <= np; ++i)
   f3(i,j,k)=donor(c1*x(i,j,k-1)+c2,c1*x(i,j,k)+c2,v3(i,j,k));
for(k=1; k <= n3m; ++k)
 for(j=1; j <= mp; ++j)
  for(i=1; i <= np; ++i)
   x(i,j,k)=x(i,j,k)-(f1(i+1,j,k)-f1(i,j,k)
          +f2(i,j+1,k)-f2(i,j,k)+f3(i,j,k+1)-f3(i,j,k))/h(i,j,k);
```

Fig. 1 Part of 3D MPDATA stencil-based implementation

$$\frac{\partial \Psi}{\partial t} + \mathrm{div}(V\Psi) = 0, \tag{1}$$

where V is the velocity vector. The spatial discretization of MPDATA is based on a finite-difference approximation. The algorithm is iterative and fast convergent [16]. In addition, it is positive defined, and by appropriate flux correction [16] can be monotonic as well. This is a desirable feature for advection of positive definite variables, such as specific humidity, cloud water, cloud ice, rain, snow, and gaseous substances.

The MPDATA scheme belongs to the class of the forward-in-time algorithms, which assume iterative execution of multiple time steps. The number of required time steps depends on a type of simulated physical phenomenon, and can exceed even few millions. A single MPDATA step requires five input arrays, and returns one output array that is necessary for the next time step. Each MPDATA time step is determined by a set of 17 computational stages, where each stage is responsible for calculating elements of a certain array. These stages represent stencil codes which update grid elements according to different patterns. We assume that the size of the 3D MPDATA grid determined by coordinates i, j, and k is $n \times m \times l$.

The stages are dependent on each other: outcomes of prior stages are usually input data for the subsequent computations. Every stage reads a required set of arrays from the memory, and writes results to the memory after computation. In consequence, a significant memory traffic is generated, which mostly limits the performance of novel architectures [9]. Figure 1 shows a part of the 3D MPDATA implementation, consisting of four stencils.

4 MPDATA Parallelization on GPU Cluster

The single-node parallelization is described in our previous papers [13–15]. In particular, our idea of adaptation to a single GPU relies on an appropriate compression of GPU kernels. It increases hardware requirements for CUDA blocks, and decreases

```
for(k=1; k<l-1; ++k) {
 q1=fmax(NO(0.),v3M(i,j,k+1))*xM(i,j,k)+fmin(NO(0.),v3M(i,j,k+1))*xM(i,j,k+1);
 xP[ijk]=x[ijk] - ( fmax(NO(0.0),v1M(i+1,j,k)) * xM(i,j,k)
                   +fmin(NO(0.0),v1M(i+1,j,k)) * xM(i+1,j,k)
                   -fmax(NO(0.0),v1M(i,j,k))   * xM(i-1,j,k)
                   -fmin(NO(0.0),v1M(i,j,k))   * xM(i,j,k)
                   +fmax(NO(0.0),v2M(i,j+1,k)) * xM(i,j,k)
                   +fmin(NO(0.0),v2M(i,j+1,k)) * xM(i,j+1,k)
                   -fmax(NO(0.0),v2M(i,j,k))   * xM(i,j-1,k)
                   -fmin(NO(0.0),v2M(i,j,k))   * xM(i,j,k) + q1-q0)/h[ijk];
  q0=q1; ijk+=M; }
```

Fig. 2 Four MPDATA stencils compressed into a single GPU kernel

the GPU occupancy. However, it allows for the reduction of the number of temporary arrays, and thereby it decreases the traffic between GPU shared and global memories, which is the main performance bottleneck. In consequence, the best configuration of the MPDATA algorithm is compression of 17 stencils into four GPU kernels. Figure 2 presents a single kernel which is obtained [15] after compression of the four MPDATA stencils shown in Fig. 1.

In the resulting single-node parallelization, the 2.5D blocking technique [9] is used as well that allows to reduce the GPU global memory traffic and provide elimination of some common sub-expressions. In this paper, we only challenge the issue of achieving the scalability across many GPU nodes.

The performance results obtained in paper [15] shown that on a single GPU, the 3D MPDATA code is profitable for meshes greater or equal to $64 \times 64 \times 16$. To keep many GPUs busy, we need to process even greater grids ($256 \times 256 \times 64$ or greater). When using the EULAG model for NWP purposes, typical simulations contain grids from $500 \times 250 \times 60$ to $2000 \times 2000 \times 120$. Moreover, the grid size l in dimension l is much smaller than the first two grid sizes m and n, where usually $l \leq 128$. Therefore, to provide parallelization of 3D MPDATA on a cluster with GPUs, it is sufficient to map the 3D MPDATA grid on a 2D mesh of size $r \times c$ (Fig. 3). In the EULAG code, MPDATA is interleaved with other algorithms in each time step. So after every MPDATA call (one time step), we need to update halo regions between neighboring subdomains. The analysis of MPDATA data dependencies shows that the halo regions of size 3 on each side of a subdomain are sufficient in a 2D model of communications (Fig. 3).

In consequence, the MPDATA grid is partitioned into subdomains of size $n_p \times m_p \times l$, where each node is responsible for computing within a single subdomain, and $n_p = n/r; m_p = m/c$.

The cluster-based parallelization scheme proposed in this paper takes into account the following constraints: (i) the MPDATA algorithm needs to update the output array x after each time step; (ii) data that needs to be updated corresponds to halo regions located on four sides of the subarray x_s corresponding to a subdomain; and (iii) the data flow scheme is as follows: a halo region from a source cluster node is transferred from the GPU global memory to the host memory, then to the host memory of the neighboring cluster node, and finally to the GPU global memory of this node. In the basic MPI+CUDA implementation [15], all the operations in point (iii) are performed

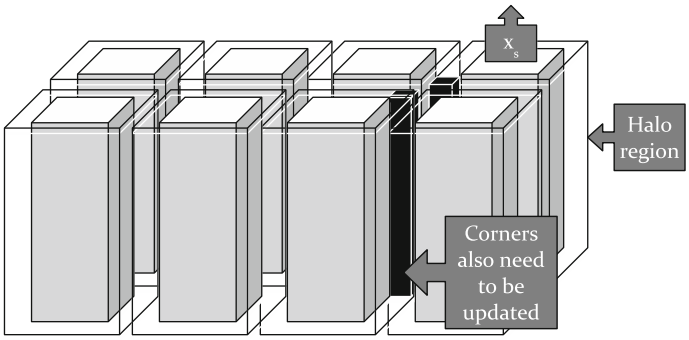

Fig. 3 2D decomposition of computational domain into subdomains

```
sendBodyToGPU();
for(int ts=0; ts<timeStep; ++ts) {
  sendBordersToGPU();  computeBordersGPU();
  startUpdateGPU(); // CUDA:start transfers of halos from GPU memory to host
  startBodyGPU(); // CUDA: start computing body
  stopUpdateGPU(); // CUDA: wait for receiving halo regions from GPU
  updateNodes(); // MPI: exchanging halos between nodes and updating them
  stopBodyGPU(); //CUDA: wait for finalizing body computation }
recvBodyFromGPU();
```

Fig. 4 Scheme of routine calls representing the overlapping strategy in the improved version

sequentially, with no computation/communication overlap, which limits the scalability. In this paper, we modify the basic scheme to provide overlapping data transfers and GPU computation. For this aim, we propose to divide all the computations within each subdomain into two parts. The first one is responsible for computing the halo regions (data-dependent part), while the second part is responsible for computing the interior part (or body) of the subarray x_s (data-independent part).

The resulting scheme of routine calls is shown in Fig. 4. This scheme corresponds to the improved version of MPI+CUDA implementation on GPU clusters. Here, the $sendBodyToGPU$ routine is responsible for sending the interior part (body) of the subarray x_s from the host memory to the GPU global memory. The remaining parts of this subarray, corresponding to the halo regions, are send by $sendBordersToGPU$ routine. The $computeBordersGPU$ routine calls the required GPU kernels, and computes the interior part of the subdomain. The $startUpdateGPU$ and $stopUpdateGPU$ routines are responsible for data transfers of halo regions from the GPU global memory to the host memory. These routines are performed within a single CUDA stream. The next pair of routines: $startBodyGPU$ and $stopBodyGPU$ are responsible for computing the body of the subarray x_s, and are performed within another CUDA stream. Since CUDA streams are processed asynchronously, those operations are overlapped. Finally, the $updateNodes$ routine is an MPI part of the application; it is responsible for exchanging the halo regions between nodes and updating them. This routine is managed by the host, and is processed asynchronously with CUDA streams.

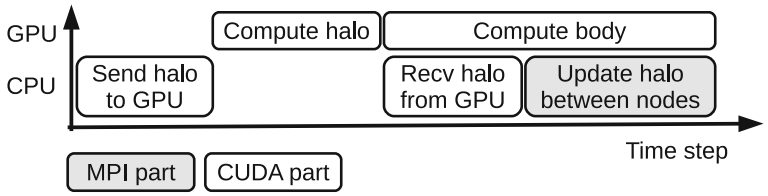

Fig. 5 Executing MPDATA on cluster with overlapping communication and computation

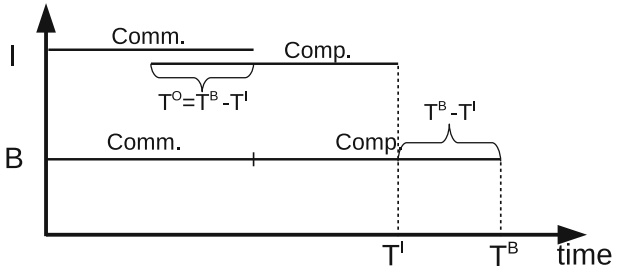

Fig. 6 Timings for executing basic (B) and improved (I) schemes on a cluster node

The approach presented above implies some advantages and disadvantages. It allows us to hide the exchange of the halo regions behind GPU computations responsible for calculating the interior parts (Fig. 5). In this scheme, the routines responsible for transferring halo regions from the GPU to the host and exchanging them between nodes using MPI are overlapped with processing the interior part of a subdomain. In consequence, the communication penalty is reduced, and the scalability of MPDATA running on clusters can be improved. However, the computations performed by a single GPU kernel are now divided into two GPU kernels. This results in some delays of computation that can reduce the performance for a small number of nodes.

5 Performance modeling of MPDATA on GPU cluster

In our performance model, we assume the weak scalability approach [7]. In this approach, the size of MPDATA grid increases with increasing the number of nodes, so the grid size per node is constant. The primary goal of our model is to estimate the minimum number N_p of nodes that allows for achieving the maximum performance R_{max} [Gflop/s], for a given size of grid per node.

When building the model, we base on Fig. 6, which illustrates timings for the basic and improved schemes, showing overlapping communication with computation in the improved version. Then based on Fig. 7, the following assumptions could be formulated for our performance model:

- for some number of nodes, communication takes less time than computation;
- then with increasing the number of nodes, the algorithm execution is scalable until the communication time does not exceed the computation time, so communications are hidden behind computations;

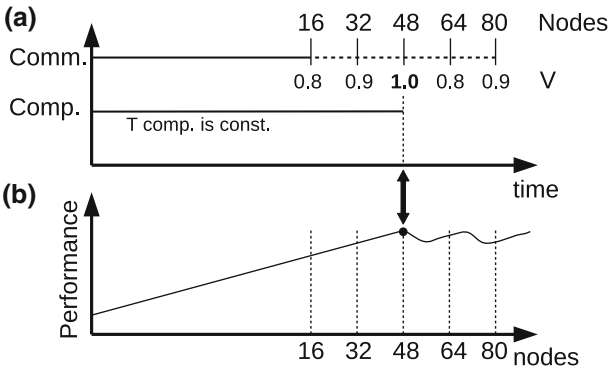

Fig. 7 Influence of increasing the number of nodes on overlapping communication with computation (**a**) and MPDATA performance (**b**)

– after achieving the full overlapping communications with computations for some number of nodes, a decrease in performance is expected for a higher number of nodes, because transfers take more time than computations.

The input values for the model are time measurements of tests performed for the basic and improved versions of MPDATA. These tests are executed for a different number N of nodes. In our case, we define the sampling interval from $N_1 = 1$ to $N_5 = 128$, so $N_i \in \{1, 16, 32, 64, 128\}$, $i = 1, \ldots, 5$. The tests are performed assuming the constant grid size per node that corresponds to the weak scalability case. In the basic version of MPDATA, computations and communications are processed sequentially, while in the improved version, overlapping is provided to increase the scalability at the cost of some overhead due to separation of GPU kernels. For each measurement i, this allows us to estimate the overlapping parameter

$$V_i = T_i^O / T_i^I \tag{2}$$

as the ratio of time T_i^O of overlapping communication with computation to the execution time T_i^I for the improved version. Then, to preserve the weak scalability assumption, the value of overlapping T_i^O is expressed in the following way: $T_i^O = T_i^B - T_i^{IC}$, where T_1^B is the execution time required by the basic version, and T_i^{IC} is the execution time for the improved version, without the overhead caused by separation of GPU kernels. To estimate T_i^{IC}, we assume that this overhead is proportional to the MPDATA execution time, so

$$T_i^{IC} = T_i^I - T_i^I * H, \tag{3}$$

where the overhead ratio H is expressed by the following equation which takes into account execution of two versions of MPDATA on a single node:

$$H = (T_1^I - T_1^B)/T_1^I. \tag{4}$$

Table 1 Execution times measured for the basic and improved versions, as well as overlapping, depending on number of nodes

Nodes	1	16	32	64	128
MPDATA					
Basic t (s)	4.165	4.719	5.244	6.361	8.935
Improved t (s)	4.4	4.5	4.54	4.55	4.62
Overlapping (%)	0.000	10.208	20.848	45.143	98.739
MPDATA-1 kernel					
Basic t (s)	2.576	3.117	3.683	4.845	7.188
Improved t (s)	2.679	2.708	2.712	2.772	4.769
Overlappping (%)	0.000	18.948	39.649	78.628	Out of scale
MPDATA+1 kernel					
Basic t (s)	5.621	6.163	6.741	7.884	10.117
Improved t (s)	8.889	9.168	9.256	9.292	10.02
Overlappping (%)	0.000	3.987	9.593	21.612	37.733

Since these measurements are performed for a single node, we can estimate the computation overhead H without communication, since the halo regions are not exchanged between nodes.

The next step is to define the number N of nodes as a function of the overlapping parameter V_i. For this aim, we use a linear regression method that allows us to define our function as $N(V_i) = 1.27722 * V_i + 3.5$ with the mean squared error MSE $= 4.1$. The final step is to calculate the value of $N(100) = 131.222$. It means that the algorithm should allow for achieving the best performance using around 131 nodes.

To validate the effectiveness of our model, we study two more cases. In the second case, we turned off one of four GPU kernels to simulate the algorithm with the same memory traffic but faster calculations. In the third case, we executed one of the kernels twice, simulating the algorithm with slower computations. The obtained results are shown in Table 1, where the overlapping data transfers with computations are expressed as the percentage of the MPDATA execution time.

It is expected that the optimal numbers of nodes for each case are respectively around 131, 80, and 328, with the MSE error of 4.1, 0.1, and 14.9. The highest accuracy is achieved for the second case, where the estimated optimal number of nodes is within the sampling interval (1 . . . 128). The highest error is achieved for the third test, where the optimal number of nodes is much more higher than the sampling interval.

6 Performance results and experimental validation of the model

All the experiments are performed on the Piz Daint supercomputer [11]. This machine is located in the Swiss National Supercomputing Centre. Currently, it is ranked 7th in the TOP500 list (November 2015 edition). Each node is equipped with one 8-core 64-bit Intel SandyBridge CPU clocked 2.6 GHz, and one NVIDIA Tesla K20X GPU with 6 GB of GDDR5 memory, and 32 GB of host memory. The nodes are connected by

Table 2 Weak scalability results of MPDATA algorithm on Piz Daint cluster

# Nodes	# Grid elements	Basic time (s)	Basic perf. (Gflop/s)	Improved time (s)	Improved perf. (Gflop/s)
1	2^{24}	4.165	138	4.40	131
2	2^{25}	4.211	273	4.43	260
4	2^{26}	4.288	537	4.50	511
8	2^{27}	4.430	1040	4.50	1022
16	2^{28}	4.719	1953	4.50	2048
32	2^{29}	5.244	3516	4.54	4063
64	2^{30}	6.361	5798	4.55	8100
128	2^{31}	8.935	8256	4.62	15,964
256	2^{32}	13.057	11,299	9.13	16,162

the "Aries" proprietary interconnect from Cray. The software environment includes the MPICH V6.2.2 implementation of MPI, and SUSE Linux with the gcc v.4.3.4 compiler and CUDA v.6.5. In all the tests, we use the compilation flag `-O3`. The current setup does not allow MPI applications to utilize the GPUDirect RDMA technology to speedup communications between nodes.

Table 2 shows the performance results of weak scalability analysis, where the number of grid elements and number of GPUs increases twice in successive experiments. The achieved results correspond both to the basic and improved versions. In the improved code, each kernel is called twice: for the first time, it computes the halo region, while for the second time, it computes the body of computation domain. This is required to provide overlapping GPU computations with communications.

As we can see, there are some overheads caused by the distribution of the GPU computation into two parts. These overheads are compensate using 16 or more nodes, where the improved version outperforms the basic one. Based on these results, we can conclude that the current MPDATA implementation scales well up to 128 nodes, when the sustained performance is almost 16 Tflop/s.The efficiency radically decreases for 256 nodes.

For a more exhaustive validation of the proposed model, we execute the MPDATA algorithm for different number of nodes. We study all three cases included in Table 1, but due to shortage of space only the first case is presented in Fig. 8. In accordance with our model, we can observe that the performance of MPDATA raises sharply until the range from 128 to 136 nodes. Thus the maximum performance of $R_{\max} = 16.372$ Tflop/s is obtained for 136 nodes.

For the second case without one kernel call, the optimal number of nodes is 80, as it was predicted. Finally, in the third case with an additional kernel, the optimal number of nodes is 316 which is within the range of error from the predicted value. We can observe a similar scalability behaviour in all three cases. The reason is that the scalability is limited by the memory traffic, which is constant in all cases. Then, for example, when the computations run faster per a single node because of a reduced number of

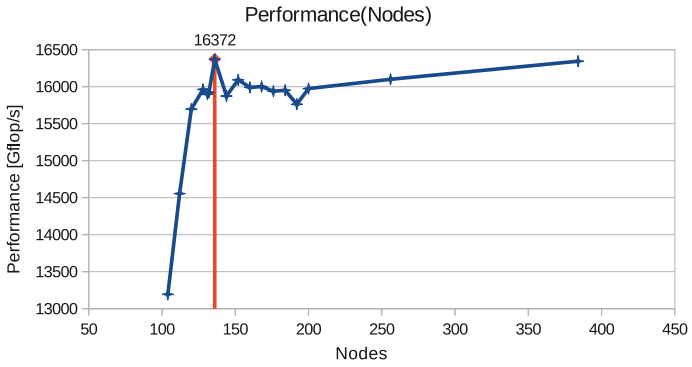

Fig. 8 Validation of performance model for MPDATA with the same grid size per node

operations in the second case, then the algorithm achieves the peak performance using a less number of nodes than in the first case.

7 Conclusions and future work

For the Piz Daint cluster equipped with NVIDIA Tesla K20 GPUs, the proposed approach to parallelization of the MPDATA algorithm allows us to achieve a weak scalability up to 136 nodes. The obtained performance exceeds 16 Tflop/s in double precision. On the cluster level, the key optimization is hiding the inter- and intra-node communication behind the GPU computation. In consequence, our improved code is almost twice faster than the basic one, without overlapping communication and computation. For the improved code, the separation of GPU kernels generates some overheads that are compensated when using 16 or more nodes, where the improved version outperforms the basic one. The proposed performance model permits us to predict the scalability of MPDATA based on estimating the overlapped part of data transfers.

In future works, the particular attention will be paid to implementation of MPDATA using OpenCL to ensure the code portability across different devices, as well as development of autotunig mechanisms aiming at providing the performance portability.

References

1. Adhianto L, Chapman B (2007) Performance modeling of communication and computation in hybrid MPI and OpenMP applications. Simul Model Pract Theory 15(4):481–491
2. Al-Tawil K, Moritz C (2001) Performance modeling and evaluation of MPI. J Parallel Distrib Comput 61(2):202–223

3. Barker K, Davis K, Hoisie A, Kerbyson D, Lang M, Scott P, Sancho J (2009) Using performance modeling to design large-scale systems. Computer 42(11):42–49
4. Cai J, Rendell A, Strazdins P (2008) Performance models for cluster-enabled OpenMP implementations. Comput Syst Archit Conf, pp 1–8
5. Ciznicki M et al (2014) Elliptic solver performance evaluation on modern hardware architectures. In: Proceedings of the PPAM 2013, Lecture notes in computer science 8384:155–165
6. Datta K, Kamil S, Williams S, Oliker L, Shalf J, Yelick K (2009) Optimization and performance modeling of stencil computations on modern microprocessors. SIAM Rev 51(1):129–159
7. Hager G, Wellein G (2011) Introduction to high performance computing for scienceand engineers. CRC Press, London
8. Hoefler T, Gropp W, Thakur R, Traff J (2010) Toward performance models of MPI implementations for understanding application scaling issues. In: 17th European MPI Users Group Meeting, EuroMPI 2010, Lecture notes in computer science 6305:21–30
9. Kamil S, Husbands P, Oliker L, Shalf J, Yelick K (2005) Impact of modern memory subsystems on cache optimizations for stencil computations. In: Proceedings of the 2005 workshop on memory system performance, pp 36–43
10. Khajeh-Saeed A, Perot JB (2012) Computational fluid dynamics simulations using many graphics processors. Comput Sci Eng 14(3):10–19
11. PizDaint & PizDora. http://www.cscs.ch/computers/piz_daint/index.html
12. Prusa JM, Smolarkiewicz PK, Wyszogrodzki AA (2008) EULAG, a computational model for multiscale flows. Comput Fluids 37:1193–1207
13. Rojek K, Szustak L, Wyrzykowski R (2014) Performance analysis for stencil-based 3D MPDATA algorithm on GPU architecture. Proc PPAM 2013 8384:145–154
14. Rojek et al K (2015) Adaptation of fluid model EULAG to graphics processing unit architecture. Concurr Comput Pract Exp 27(4):937–957
15. Rojek K, Wyrzykowski R (2015) Parallelization of 3D MPDATA algorithm using many graphics processors, parallel computing technologies. Lect Notes Comp Sci 9251:445–457
16. Smolarkiewicz P (2006) Multidimensional positive definite advection transport algorithm: an overview. Int J Numer Methods Fluids 50:1123–1144
17. Szustak L et al (2015) Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci Program 2015. doi:10.1155/2015/642705
18. Wojcik D et al (2012) A study on paralllel performance of the EULAG F90/F95 Code. In: Proceedings of the PPAM 2011, Lecture notes in computer Science 7204:419–427
19. Wyrzykowski R, Szustak L, Rojek K, Tomas A (2013) Towards efficient decomposition and parallelization of MPDATA on hybrid CPU-GPU cluster. In: Proceedings of the LSSC 2013, lecture notes in computer science 8353:457–464
20. Wyrzykowski R, Rojek K, Szustak L (2014) Parallelization of 2D MPDATA EULAG algorithm on hybrid architectures with GPU accelerators. Parallel Comput 40(8):425–447