HYBRID MPI/OPEN-MP ACCELERATION APPROACH FOR HIGH-ORDER SCHEMES FOR CFD

MICHAL SACZEK, KAROL WAWRZAK,
ARTUR TYLISZCZAK AND ANDRZEJ BOGUSLAWSKI

Czestochowa University of Technology
Faculty of Mechanical Engineering and Computer Science
Armii Krajowej 21, 42-201 Czestochowa, Poland

(received: 19 April 2018; revised: 21 May 2018;
accepted: 26 June 2018; published online: 13 July 2018)

Abstract: The paper presents a hybrid MPI+OpenMP (Message Passing Interface/Open Multi-Processor) algorithm used for parallel programs based on the high-order compact method. The main tools used to implement parallelism in computations are OpenMP and MPI which differ in terms of memory on which they are based. OpenMP works on shared-memory and the MPI on distributed-memory whereas the hybrid model is based on a combination of those methods. The tests performed and described in this paper present significant advantages provided by a combination of the MPI/OpenMP approach. The test computations needed for verifying possibilities of MPI, Open-MP and Hybrid of both tools were carried out using an academic high-order SAILOR solver. The obtained results seem to be very promising to accelerate simulations of fluid flows as well as for application using high order methods.

Keywords: MPI, OpenMP, hybrid MPI/OpenMP, high-order methods

DOI: https://doi.org/10.17466/tq2018/22.3/e

1. Introduction

For ages researchers have struggled to understand precisely the complex structure of fluid flows. Impressive improvements and achievements in computer technology and the increasingly common use of numerical simulations have enabled more elaborate experiments thanks to which one can observe constant progress in the field of Computational Fluid Dynamics (CFD). The issue that has received much attention in the CFD community are high-order discretization methods that are irreplaceable in DNS (Direct Numerical Simulation) and LES (Large Eddy Simulation) studies focused on very deep and detailed analysis of the fluid flow problems. High-order methods are expected to open the door to a high solution accuracy at a lower cost, their present status and ways of application are described in the review paper [1]. In terms of the highest accuracy there are spectral and pseudo-spectral methods that are considered to be superior to other high-order discretization methods, however, they can be used mainly in simple computational domains.

One of the most attractive high-order methods is the compact difference method [2], which combines global approximation (spectral method) and flexibility of the computational domain (finite volume method). However, in this method the approximations of derivatives are obtained by solving a linear system of equations (tridiagonal or pentadiagonal) that creates huge difficulties in parallelization as it makes it necessary to execute the algorithm step by step in a serial fashion. Special methods have to be applied to divide a linear system of equations into independent computational groups. There are many techniques, such as the cyclic reduction method [3], the partitioning methods [4–7] and various implementations of the Gauss elimination method [8–10], which allow the linear system to be solved in a parallel way.

As shown in the exhaustive review paper [11] the dominant programming tools which are used to implement parallelism in CFD codes are OpenMP, MPI, hybrid OpenMP + MPI and CUDA. The first three approaches are discussed in the paper. The main difference between OpenMP and MPI relies on access to memory *i.e.*, OpenMP works on shared-memory [12] whereas the MPI is based on distributed-memory that means that the exchange of data requires communications between nodes [13]. Hybrid programming is based on the idea of using OpenMP threads to employ multiple cores in a certain socket/node and MPI to communicate among the sockets/nodes. Possible variants of OpenMP and MPI methods and their combination are presented in Figure 1. All of the tools

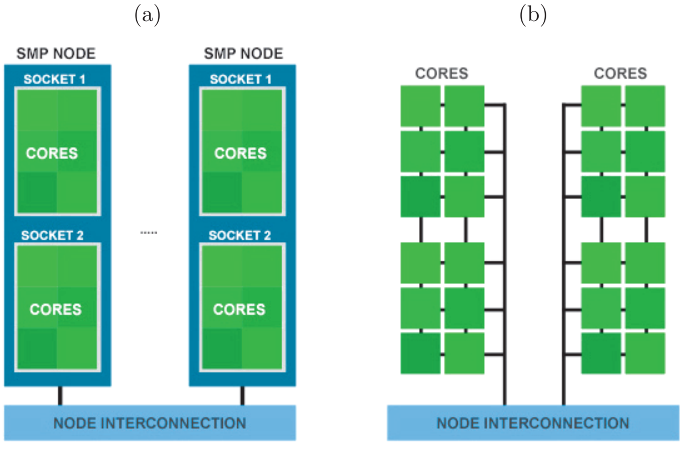

Figure 1. SMP (Symmetric multiprocessing) cluster: (a) – typical multi-socket multi-core cluster, (b) – mapped MPI processes to each core [14]

mentioned above have been widely used in many research studies in the CFD investigation: pure OpenMP [15, 16], pure MPI [17, 18], as well as the hybrid approach MPI + OpenMP [19, 20].

The modern multi-socket multi-core SMP cluster, shown in Figure 1 (a), allows a program to be run on various configurations [14]. Initial research, which is not presented in the paper, proved that the best configuration for a hybrid approach was to assign MPI to each socket and then to use every core available on that socket as OpenMP threads, as shown in Figure 2 (b). Another option is presented in Figure 2 (a), where every MPI process is assigned to nodes and then every core available on that node works as threads.

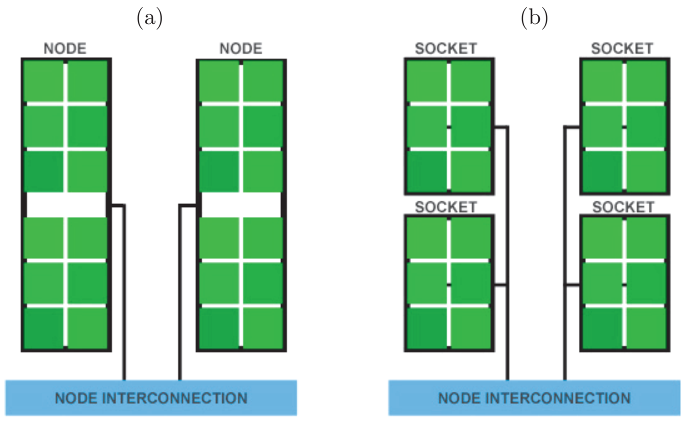

Figure 2. SMP cluster: (a) – mapped MPI to each node and OpenMP threads to each core on that node, (b) – mapped MPI processes to each socket and OMP threads to each core on that socket [14]

The last option is shown in Figure 1 (b) where the MPI process is assigned to every core available in the cluster, then threads can be pinned to the same core. This method is especially useful for pure MPI programs.

The aim of this work is to present a hybrid MPI + OpenMP parallelized algorithm for a high-order compact method in three spatial dimensions. The test program created for this research is included in the SAILOR solver used in various LES studies including free jet flows [21], multi-phase flows [22] and flames [23].

2. Numerical algorithm

The SAILOR code is based on a projection method for the pressure-velocity coupling. The time integration is performed by a predictor-corrector (Adams-Bashforth/Adams-Moulton) approach and the spatial discretization is based on the 6th order compact difference method for half-staggered meshes. The detailed description of the SAILOR code is presented in [24].

2.1. Compact Difference Scheme

The compact difference scheme allows calculating the derivatives of a general function f at the mesh node using the values of f and its derivative at the neighboring nodes. The general formula for the first derivative is given as [2]:

$$\begin{aligned}&\beta f'_{i-2}+\alpha f'_{i-1}+f'_i+\alpha f'_{i+1}+\beta f'_{i+2}=\\&a\frac{f_{i+1}-f_{i-1}}{h}+b\frac{f_{i+2}-f_{i-2}}{h}+c\frac{f_{i+3}-f_{i-3}}{h}\end{aligned}\tag{1}$$

where h represents the uniform distance between nodes. In the case of the 6th order accuracy used in this work, the coefficients are equal to: $\beta=0$, $\alpha=\frac{1}{3}$, $a=\frac{7}{9}$, $b=\frac{1}{36}$, $c=0$. For non-periodic problems it is necessary to reduce the order of approximation near the boundaries. For $i=2,N-1$ (N – number of mesh nodes) we used the 4th order formula ($\beta=0$, $\alpha=\frac{1}{4}$, $a=\frac{3}{4}$, $b=0$, $c=0$), whereas for $i=1,N$ we employed the asymmetric 3rd order approximations given as [2]:

$$f'_i+2f'_{i\pm1}=\pm\frac{1}{h}\left(-\frac{15}{6}f_i+2f_{i\pm1}+\frac{1}{2}f_{i\pm2}\right)\tag{2}$$

where the negative sign is for $i=N$. The Equation (2) written for every node leads to the system of equations:

$$\mathbf{A}f'=\mathbf{B}f\tag{3}$$

where matrix $\mathbf{A}$ is tridiagonal and matrix $\mathbf{B}$ is pentadiagonal. This system is shown in Equation (5), where P_0, P_1 and P_2 present a sample division of the system between 3 parallel tasks.

2.2. Right hand size

The right hand side (RHS) of Equation (5) has to be calculated to perform calculations for derivatives. Moreover, the RHS must be calculated at first, before the System (5) is solved. Despite the fact that the RHS is simple to calculate (multiplying the matrix and the vector) it requires communication as the domain is decomposed into independent subdomains (with distributed memory). Each subdomain (MPI process) has to send values of function f from two boundary nodes to their neighboring MPI processes and receive data from them.

2.3. Tridiagonal matrix algorithm

The tridiagonal matrix is a special type of a matrix that has the nonzero elements only on the main diagonal and two neighboring diagonals. When the RHS is calculated the system to solve is reduced to:

$$\mathbf{A}f'=\mathrm{RHS}\tag{4}$$

This system could be solved by the tridiagonal matrix algorithm (TDMA) also known as the Thomas algorithm. It could also be solved by the product of the inverse tridiagonal matrix and RHS ($f'=\mathbf{A}^{-1}\mathrm{RHS}$). Nevertheless, this method is more time-consuming than the TDMA. The TDMA is carried out in two steps:

$$\begin{array}{c}P_0\\ \\ \\ P_1\\ \\ \\ P_2\end{array}\begin{bmatrix} 1 & 2 & & & & & & & 0 \\ \frac{1}{4} & 1 & \frac{1}{4} & & & & & & \\ & \frac{1}{3} & 1 & \frac{1}{3} & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & \frac{1}{3} & 1 & \frac{1}{3} & & \\ & & & & & \ddots & \ddots & \ddots & \\ & & & & & & \ddots & \ddots & \ddots \\ & & & & & & \frac{1}{3} & 1 & \frac{1}{3} \\ & 0 & & & & & \frac{1}{4} & 1 & \frac{1}{4} \\ & & & & & & & 2 & 1 \end{bmatrix} \begin{bmatrix} f'_1 \\ f'_2 \\ f'_3 \\ \vdots \\ f'_{i-1} \\ f'_i \\ f'_{i+1} \\ \vdots \\ f'_{N-2} \\ f'_{N-1} \\ f'_N \end{bmatrix} = \frac{1}{h} \begin{bmatrix} -\frac{15}{6} & 2 & \frac{1}{2} & & & & & & & 0 & \\ -\frac{3}{4} & 0 & \frac{3}{4} & & & & & & & & \\ -\frac{1}{36} & -\frac{7}{8} & 0 & \frac{7}{8} & \frac{1}{36} & & & & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & & & \\ & & & -\frac{1}{36} & -\frac{7}{8} & 0 & \frac{7}{8} & \frac{1}{36} & & & \\ & & & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & & & -\frac{1}{36} & -\frac{7}{8} & 0 & \frac{7}{8} & \frac{1}{36} \\ & 0 & & & & & & & -\frac{3}{4} & 0 & \frac{3}{4} \\ & & & & & & & & \frac{1}{2} & 2 & -\frac{15}{6} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{i-1} \\ f_i \\ f_{i+1} \\ \vdots \\ f_{N-2} \\ f_{N-1} \\ f_N \end{bmatrix} \qquad (5)$$

- **forward** loops from 1 to n and preparing coefficients d and γ

$$\begin{cases} d_i = c_i / b_i & \text{where } i = 1 \\ d_i = (b_i - a_i \cdot d_{i-1}) / c_i & \text{where } i = 2,3,\dots,n \end{cases} \tag{6}$$

$$\begin{cases} \gamma_i = \mathrm{RHS}_i / b_i & \text{where } i = 1 \\ \gamma_i = (\mathrm{RHS}_i - a_i \cdot \gamma_{i-1}) / (b_i - a_i \cdot d_{i-1}) & \text{where } i = 2,3,\dots,n \end{cases} \tag{7}$$

- **backward** loops from n to 1 and getting the result x

$$\begin{cases} f'_i = \gamma_i & \text{where } i = n \\ f'_i = \gamma_i - d_i \cdot x_{i+1} & \text{where } i = n-1, n-2, \dots, 1 \end{cases} \tag{8}$$

and requires to be done step by step in a serial fashion.

2.4. Parallel TDMA "arrowhead" method

The arrowhead method [7] is one of the commonly used partitioning methods which allows solving Equation (4) in a parallel way. The method relies on reducing the initial system to a new system with diagonally independent blocks, as is presented in Figure 3.

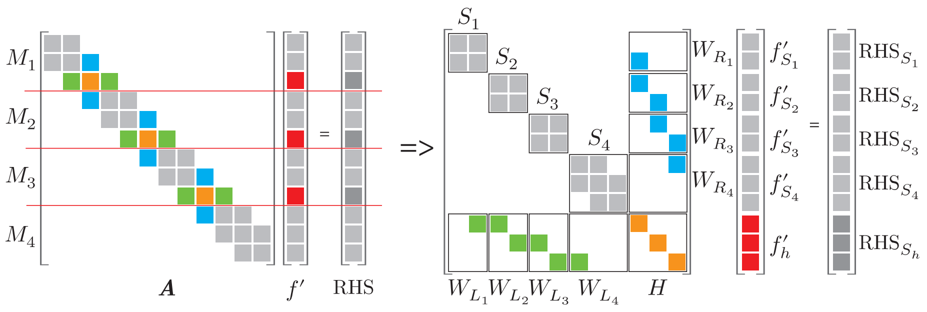

Figure 3. Tridiagonal system rearrangement into arrowhead form [7]

The matrix is rearranged in the following way:

- choose the number of subsystems M which can correspond to a number of MPI processes. Such subsystems are shown in Figure 3 and are separated by red lines;
- mark $M-1$ separations block-rows (orange, green, red and grey colors) for every subsystem without the last one;
- mark the corresponding block-column (blue);
- shift the marked separation block-rows to the bottom;
- shift the marked separation block-columns to the right.

The global formula of this system is written as

$$\begin{pmatrix} S & W_r \\ W_l & H \end{pmatrix} \begin{pmatrix} f'_s \\ f'_h \end{pmatrix} = \begin{pmatrix} \mathrm{RHS}_s \\ \mathrm{RHS}_h \end{pmatrix} \tag{9}$$

where S is the subsystem, H – the head (orange squares), W_{lr} – the wings (green and blue squares).

The solution of the system is given by the relations

$$\begin{cases} f'_s = S^{-1}\mathrm{RHS}_s - S^{-1}W_r f'_h \\ f'_h = \left(H - W_l S^{-1} W_r\right)^{-1}\left(\mathrm{RHS}_h - W_l S^{-1}\mathrm{RHS}_s\right) \end{cases} \tag{10}$$

The system is solved in the following steps:

- compute $Z_k = \left(S^k\right)^{-1} W_l^k$ on each process, which leads to the TDMA solution for $S_k Z^K = W_l^k$;
- compute $H - \sum_{k-1}^{M} W_l^k Z^k$ on the root process;
- compute $z_k = \left(S^k\right)^{-1} \mathrm{RHS}_s^k$ on each process, which leads to the TDMA solution for $S_k z^K = \mathrm{RHS}_s^k$;
- compute $\mathrm{RHS}_h - \sum_{k-1}^{M} W_l^k z^k$ on the root process.

Synchronize the data to master the process and:

- compute f'_h which is a result for $M-1$ subsystems $f'_h = \left(H - W_l S^{-1} W_r\right)^{-1}\left(\mathrm{RHS}_h - W_l S^{-1}\mathrm{RHS}_s\right)$ which leads to the TDMA solution for $\left(H - W_l S^{-1} W_r\right) f'_h = \left(\mathrm{RHS}_h - W_l S^{-1}\mathrm{RHS}_s\right)$;
- scatter f'_h and correct the result on every process $x_s^k = z^k - Z^k x_h$.

This method is very simple to implement and allows one to reduce the computation time of the TDMA. Moreover, the arrowhead algorithm could be applied in another diagonal system with a five-diagonal matrix or a seven-diagonal matrix.

2.5. Test program

The solver to simulate fluid flow problems has to make many different calculations. However, one of its most important tasks is approximation of derivatives. The SAILOR code decomposes the domain using the MPI library, in one (y) of the three directions (x, y, z), as presented in Figure 4. Assuming for instance that the number of nodes is $N \times K \times M$, each MPI process has $N \times K_p \times M$ nodes, where p is the number of MPI processes and $K_p = K/p$. In this approach the calculations of derivatives (using TDMA) in directions (x, z) are obtained without any exchange of data. Each MPI process has all needed nodes in these directions, therefore, communication between MPI processes is not required. Unfortunately, communication between nodes is necessary to make calculations of derivatives in the decomposed direction. In this direction, each MPI process has some of the nodes only. Furthermore, communication is required for the TDMA as well as for the RHS calculations. The test program was created to present a hybrid MPI+OpenMP algorithm. The test program performs the calculations of derivatives in all spatial directions (x, y, z). The subdomains should be additionally divided to allow OpenMP threads to cooperate with MPI processes. It is important to mention that this division is logical which means that there are no additional subdomains created but it enables threads to work on individual parts of a subdomain. For instance, if the subdomain size is $N \times K_p \times M$ then each OMP thread that takes part in an MPI process performs calculations for the $N/th \times K_p \times M$ or $N \times K_p/th \times M$ nodes, where th is the number of the OMP threads. Moreover, it does not matter which thread performs the calculations for individual parts of the subdomain because this tool works on a shared memory (OpenMP threads have access to the same memory in the same MPI process). For y-derivatives (communication between MPI processes is required) the division should be made in the second direction as presented in Figure 5 (a). For other derivatives (communication between MPI processes in not required) the division should be made in the same direction as the MPI processes, as shown in Figure 5 (b).

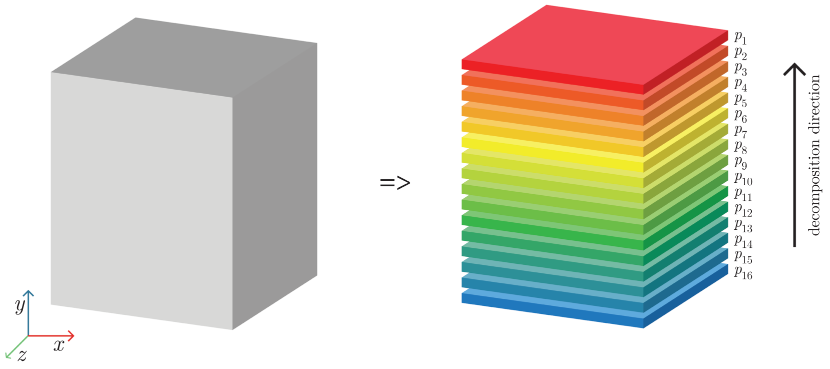

Figure 4. Domain decomposition into 16 subdomains in the first direction (y)

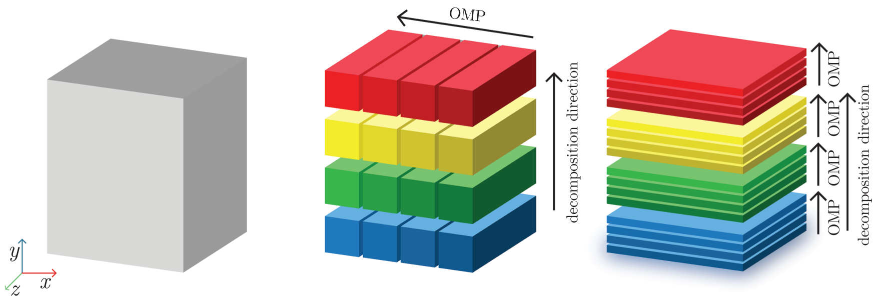

Figure 5. Domain decomposition into 4 subdomains in the first direction (y) and into 4 subdomains in: (a) – the second direction (x), (b) – the same direction (y)

3. Results

The tests were run on the PLGRID platform with the computational power of about 2399 TFlops. The program was run and compiled using the Intel MPI library version 2018 Update 1 and OpenMP, version 5.0. Taking into consideration the cluster specification shown in Table 1 specific meshes have to be chosen with the number of nodes allowing an equal division between the parallel tasks. As can be observed, each node of the cluster in this specification consists of 2 physical processors with 12 cores each. However, for the end user, these processors are divided into four logical processors with six cores. Hence, the mesh sizes have to be a multiple of six. The investigations were carried out for six mesh sizes and for different numbers of cores. The meshes were structured and the distance between the mesh points was constant. Details are shown in Table 2. The tests were performed for three different cases: a pure MPI tool and a serial TDMA, a pure MPI and a parallel TDMA (arrowhead), hybrid (MPI/OpenMP) and parallel TDMAs (arrowhead). Moreover, the analysis of these cases includes the workload time of cores that is essential to calculate derivatives in two directions (x,z) where communication is not required and in three directions(x,y,z) where communication is necessary in one of them (y). Furthermore, the workload time needed to calculate the RHS and the time that is essential to calculate the TDMA are investigated separately.

Table 1. Test cluster specification

processors	Intel E5-2680v3
socket per node	2
cores per socket	12
clock rate	2.50 GHz
memory size	128 GB
nodes	2160
network	Infiniband FDR 56 Gb/s

Table 2. Nodes needed for cores and meshes

cores	1	6	12	24	48	60	96	192
$M_1 - 240 \times 240 \times 240$	1	1	1	1	2	3	—	—
$M_2 - 240 \times 480 \times 240$	1	1	1	1	2	3	4	—
$M_3 - 480 \times 480 \times 480$	1	1	1	1	2	3	4	—
$M_4 - 240 \times 960 \times 240$	1	1	1	1	2	3	4	8
$M_5 - 480 \times 960 \times 480$	1	1	1	1	2	3	4	8
$M_6 - 960 \times 960 \times 960$	1	1	1	1	2	3	4	8

There are two important metrics for parallel systems: speedup and efficiency. Speedup is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors. Efficiency is defined as the ratio of speedup to the number of processors. The efficiency measures the fraction of time for which a processor is usefully utilized.

The speedup is measured by function: $S = T'/T^1$, where T^1 is the runtime for a basic TDMA solver on one computation unit and T' is the runtime for a particular configuration on p processors. Efficiency is measured by the function $E = S/p$.

As can be observed in Figures 6–7 the hybrid solution offers excellent performance especially for many cores and it is less efficient for smaller numbers of cores. For a minimum number of cores, it offers a similar efficiency as a serial TDMA with a pure MPI. Moreover, it can be seen that the hybrid approach as well as MPI with a parallel TDMA is closer to the linear speedup when a bigger mesh is used. An MPI with a parallel TDMA is characterized by better performance than the hybrid approach for small numbers of cores. Moreover, it achieves a linear speedup. Another interesting issue that is worth noting is the so-called "breakpoint". It is a point where two solutions offer the same speedup. The breakpoint moves slightly to the right along with a bigger mesh, however, it is still between 12 and 24 cores for the investigated meshes. The maximum performance achieved for the hybrid solution is:

- for M_1:
  - about 4.5 times faster than serial TDMA and pure MPI
  - about 3.1 times faster than parallel TDMA method and pure MPI
- for M_2:
  - about 6.7 times faster than serial TDMA and pure MPI
  - about 3.7 times faster than parallel TDMA method and pure MPI
- for M_3:
  - about 6.2 times faster than serial TDMA and pure MPI
  - about 5.4 times faster than parallel TDMA method and pure MPI
- for M_4:
  - about 9.4 times faster than serial TDMA and pure MPI
  - about 5.11 times faster than parallel TDMA method and pure MPI
- for M_5:
  - about 9 times faster than serial TDMA and pure MPI
  - about 7.8 times faster than parallel TDMA method and pure MPI
- for M_6:
  - about 8.3 times faster than serial TDMA and pure MPI
  - about 6.6 times faster than parallel TDMA method and pure MPI

These results show that the size of the mesh directly influences the maximum speedup. In Figure 7 it can be observed that a smaller size of the mesh in the second (x) and third(z) direction enhances the speedup for the hybrid and the MPI with a parallel TDMA approach. This results from reducing the distance between the data stored in the memory. As is mentioned in the previous section the domain is logically divided into OMP threads. Hence, the speedup for the hybrid approach is smaller than the MPI with a parallel TDMA.

To check the solver's scalability the investigations were performed for cases where the derivatives were calculated in the second and third direction (communication between nodes is not needed). The results are presented in

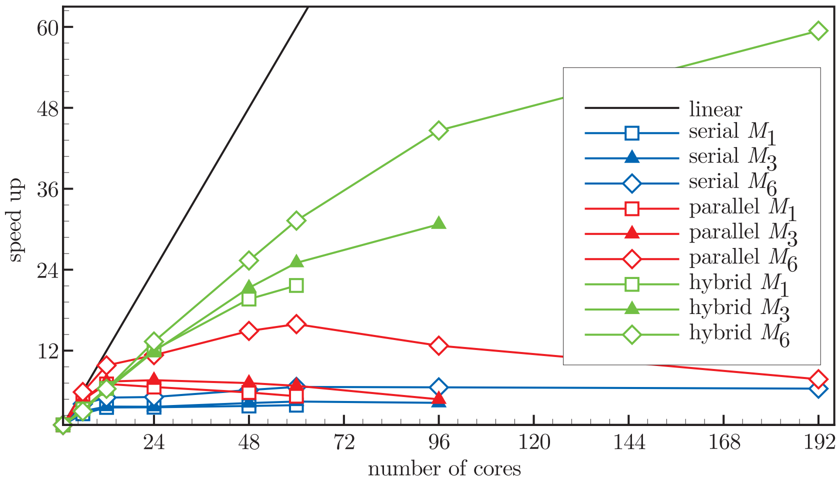

Figure 6. Speedup for three different cases and three different mesh sizes, where every direction has the same size

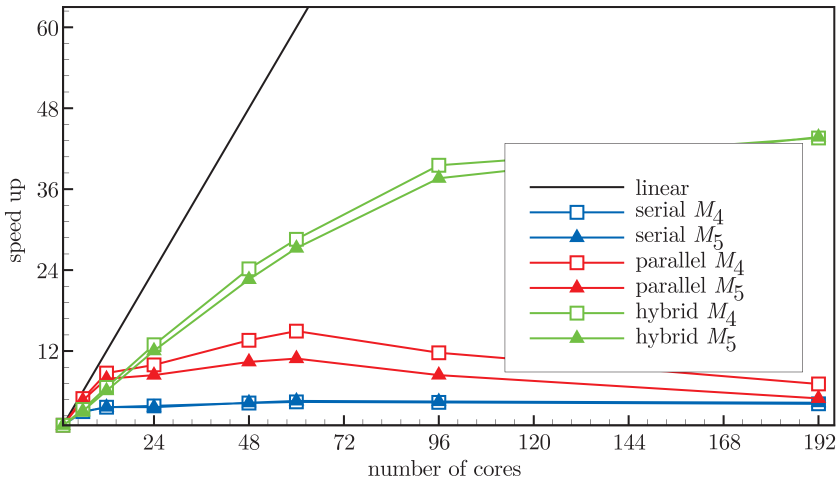

Figure 7. Speedup for three different cases and two different mesh sizes, where first direction has different size than second and third

Figures 8–9. In this investigation it was only the MPI and hybrid approaches that were taken into account because the calculations of derivatives in the second and third directions (x,z) performed using the MPI with a serial TDMA and the MPI with a parallel TDMA are obtained in the same way. As can be observed the scalabilities are on the same level. The higher efficiency on 60 cores for the MPI approach is due to the fact that the computational potential of the node is not fully utilized for the hybrid approach. 3 nodes (72 cores) have to be used to run the program using 60 cores. Moreover, the hybrid approach is pinned to the so-called numa nodes (a numa node is allocated on logical processors) where each numa node reserves the 1/2 size of cache memory of a physical processor, whereas, the

MPI processes which are pinned to cores can use the whole cache memory of the processor (the cache memory is divided dynamically between the running cores). Hence, in the case when 60 MPI processes are used on 3 nodes (maximum 72 cores) 1/5 of them have twice as much of the cache memory. Furthermore, the workload times of the RHS as well as the TDMA are measured independently on every process, and thus, the efficiency is better, if several of the MPI processes work faster. Another interesting issue which can be observed is the super efficiency of the MPI approach when executed on 6 and 12 cores, as shown in Figure 8. This super efficiency is achieved owing to the decomposition of the domain into 6 and 12 subdomains, placed on different MPI processes. Thus, the distance between the data stored in the memory is smaller. In contrast, the domain is not divided in the hybrid approach for 6 (1 MPI process and 6 OMP threads) cores, whereas it is divided into two subdomains only for 12 (2 MPI processes and 6 OMP threads each) cores. It is also worth mentioning that 6 and 12 MPI processes are pinned to cores of two physical processors (3 or 6 MPI processes per processor) and as a result, the whole cache memory of processors is utilized on the node. By contrast, in the hybrid approach, 6 cores are pinned to one physical processor, hence, only half of the cache memory of one processor is used. Summing up, if there is too much data for a particular core, the utilization of the computational potential is limited and the super efficiency cannot exist.

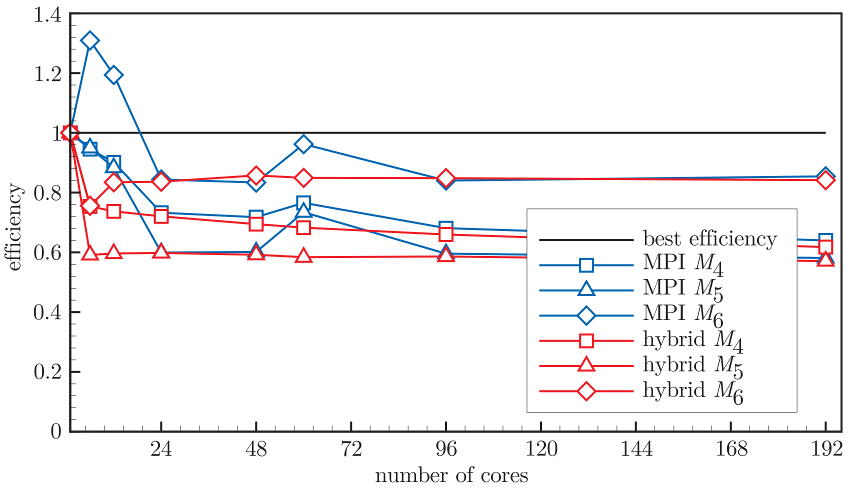

Figure 8. RHS calculation efficiency for hybrid and MPI approach without communication, for three different meshes

The last important part of analysis is a comparison of the efficiency of the TDMA calculations for five cases:

- MPI approach in two directions (x,z)
- Hybrid approach in two directions (x,z)
- MPI approach with serial TDMA in three directions (x,y,z)

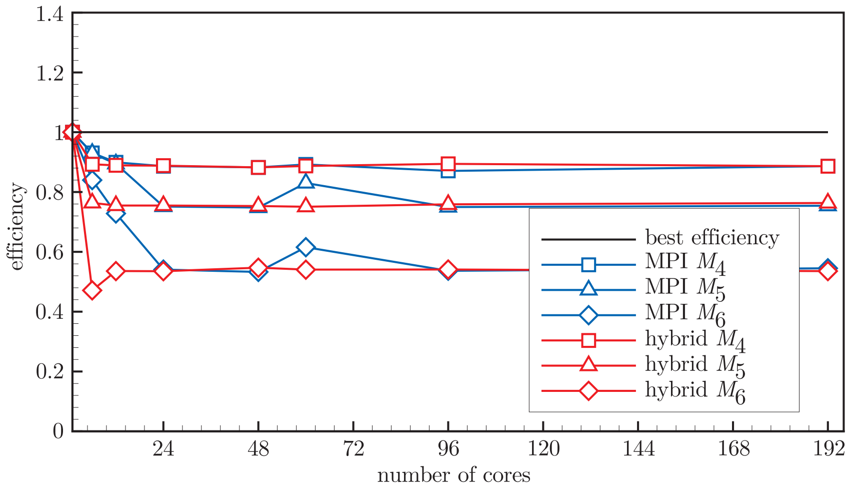

Figure 9. TDMA calculation efficiency for hybrid and MPI approach without communication, for three different meshes

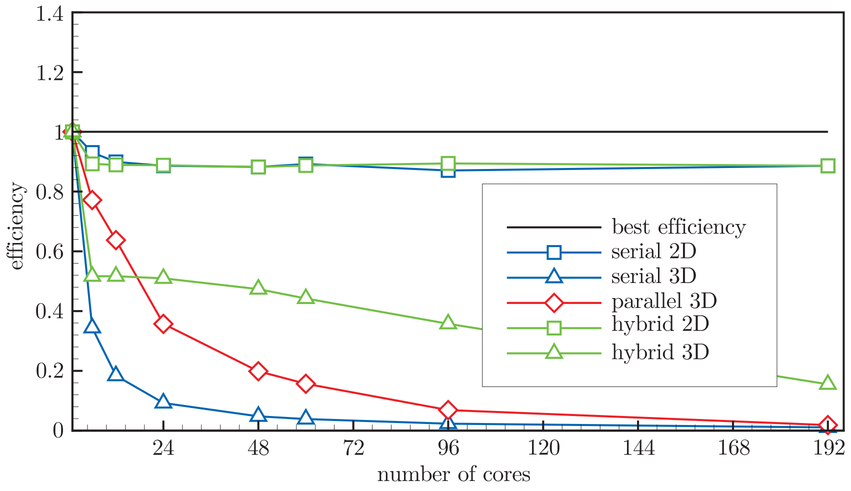

Figure 10. TDMA calculation efficiency for hybrid and MPI approach for Mesh3 without and with communication in two and three directions TDMA calculation efficiency for hybrid and MPI approach for M_3 without and with communication in two and three directions

- MPI approach with parallel TDMA in three directions (x,y,z)
- Hybrid approach with parallel TDMA in three directions (x,y,z)

As can be seen in Figure 10 the efficiency for TDMA in two directions (x,z) is characterized by excellent scalability. Unfortunately, the efficiency rapidly goes down, when TDMA calculations in three directions are taken into consideration. It is only for the hybrid approach that the efficiency decreases slower than in the other cases. It is worth mentioning that in the hybrid approach for 6 cores, the domain is divided in the logical way only. Therefore, it will be always worse for a smaller number of cores. On the other hand, it reduces the amount of data to send and receive and it can offer much better performance for larger numbers of cores. As is shown in Figure 10 the major limitation of the entire solution algorithm is the TDMA parallel method which is efficient for small numbers of cores only.

4. Conclusions

There is great demand for fast and efficient computational systems in today's word. The hybrid MPI/OpenMP approach is supposed to enhance the speed of calculations for high order methods, and this has been proved in the present work. A comparison of three approaches shows the important advantages of the hybrid model over the pure MPI with a serial TDMA and the pure MPI with a parallel TDMA. It is worth mentioning that the size of the computational domain directly influences the performance of calculations. The tests indicate that the larger the domain used, the larger acceleration is achieved.

Acknowledgements

The research was supported by the Polish National Science Centre, Project No. 2016/21/B/ST8/00414. The computations were carried out using the PL-Grid Infrastructure.

References

[1] Wang Z, Fidkowski K, Abgrall R, Bassi F, Caraeni D, Cary A, Deconinck H, Hartmann R, Hillewaert K, Huynh H, Kroll N, May G, Persson P, Leer B and Visbal M 2013 *Int. J. Numer. Methods Fluids* **72** 811
[2] Lele S 1992 *J. Comput. Phys.* **103** 16
[3] Hockney R 1965 *J. ACM* **12** 16
[4] Wang H 1981 *ACM Trans. Math. Softw.* **7** 170
[5] Mattor N, Williams T and Hewett D 1995 *Parallel Comput.* **21** 1769
[6] Sun H, Zhang H and Ni L 1992 *IEEE on Trans.Comput.* **41** 286
[7] Belov P, Nugumanov E and Yakovlev S 2015 *Preprint*, arXiv 1505.06864
[8] Stone H 1975 *ACM Trans.Math. Softw.* **1** 289
[9] Rao S C S 2008 *Parallel Comput.* **34** 177
[10] Qin J and Nguyen D 1998 *Adv. Eng. Softw.* **29** 395
[11] Afzal A, Ansari Z, Faizabadi A and Ramis M 2017 *Arch. Comput. Method E* **24** 337
[12] Dagum Land Menon R 1998 *IEEE Comput. Sci. Eng.* **5** 46
[13] William D, Lusk E and Skjellum A 1999 *Using MPI: Portable Parallel Programming with the Message-Passing Interface 2nd edition*, MIT Press, Cambridge
[14] Rabenseifner R, Hager G and Jost G 2009 *Proc. Euromicro Int. Conf. Parallel Distrib. Netw. Based Process* **17** 427
[15] Amritkar A, Deb S and Tafti D 2014 *J. Comput. Phys.* **256** 501
[16] Mavriplis D 2002 *Int. J. High Perform. Comput. Appl.* **16** 395
[17] Maknickas A, Kačeniauskas A, Kačianauskas R, Balevičius R and Algis Džiugys A 2006 *Informatica* **17** 207
[18] Jia R and Sundén B 2004 *Comput. Fluids* **33** 57
[19] Mininni P, Rosenberg D, Reddy R and Pouquet A 2011 *Parallel Comput.* **37** 316
[20] Gropp W, Kaushik D, Keyes D and Smith B 2001 *Parallel Comput.* **27** 337
[21] Wawrzak K, Boguslawski A and Tyliszczak A 2015 *Flow Turbul. Combust.* **95** 437

[22] Aniszewski W, Boguslawski A, Marek M and Tyliszczak A 2012 *J. Comput. Phys.* **231** 7368
[23] Wawrzak A and Tyliszczak A 2017 *Arch. Mech.* **69** 157
[24] Tyliszczak A 2014 *J. Comput. Phys.* **276** 438