Hybrid MPI/Open-MP acceleration approach for high-order schemes for CFD

Michal Saczek, Karol Wawrzak, Artur Tyliszczak, Andrzej Boguslawski
Faculty of Mechanical Engineering and Computer Science, Czestochowa University of Technology, Czestochowa, Poland
E-mail: michal.saczek@icis.pcz.pl

Abstract. The paper presents hybrid MPI+OpenMP (message Passing Interface/Open Multi-Processor) used for paralleled programs including high-order compact method. The main tools used to implement parallelism in computations are OpenMP and MPI which differ in terms of memory they are based on. OpenMP works on shared-memory and MPI on distributed-memory whereas hybrid model is based on combination of those methods. Tests performed and described in this paper present significant advantages provided by hybrid MPI/OpenMP. Testing computations needed for verifying possibilities of MPI, Open-MP and Hybrid of both are carried out using an academic high-order SAILOR solver. Obtained results seem to be very promising to accelerate simulations for fluid flows as well as for application using high order methods. The proposed approach was tested up to 96 cores with up to 4 nodes.

1. Introduction
For ages researchers have struggled to understand precisely a complex structure of fluid flows. Impressive improvements and achievements in computer technology and increasingly common use of numerical simulations have enabled more elaborate experiments thanks to which there can be observed constant progress in the field of Computational Fluid Dynamics (CFD). Issue that has received lots of attention in the CFD community are high-order discretisation 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 high solution accuracy with lower cost, their present status and ways of applications are described in the review paper [1]. In terms of accuracy there are spectral and pseudo-spectral methods that are considered superior to other high-order discretisations ones, however, they can be used mainly in simple computational domains.

One of the most attractive high-order method is compact difference one [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 linear system of equations (tridiagonal or pentadiagonal) that creates huge difficulties in parallelization as it makes it a necessity to execute algorithm step by step in a serial fashion. In order to divide linear system of equations into independent computational groups special methods have to be applied. There are many methods, such as cyclic reduction method[3], partitioning methods [4, 5, 6, 7] and various implementations of the Gauss elimination method [8, 9, 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. However, the CUDA tools is not included in the paper. The main difference between OpenMP and MPI is different access to memory, OpenMP works on shared-memory[12] whereas MPI is based on distributed-memory that means the exchange of data relies on communications between nodes[13]. Hybrid programming is based on the idea of using OpenMP threads to employ the multiple cores in a certain socket/node and MPI to communicate among the sockets/nodes. OpenMP, MPI methods and their combination are presented in Fig. 1. All of the tools mentioned above were widely used by many researches in CFD investigation: pure OpenMP[14, 15], pure MPI[16, 17], as well as hybrid approach MPI+OpenMP[18, 19].

![Figure 1. SMP (Symmetric multiprocessing) cluster: a - typical multi-socket multi-core cluster, b - mapped MPI processes to each core[20].](image1)

![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[20].](image2)

Modern multi-socket multi-core SMP cluster, showed in Fig. 1a, allows a program to be run on various configuration[20]. Initial research, which is not presented in the paper, proved that
the best configuration for hybrid approach is to assign MPI to each socket and then to use every core available on that socket as OpenMP threads, as is shown in Fig. 2b. Another option is shown on Fig. 2a, every MPI process is assigned to nodes and then every core available on that node works as threads. The last option is showed in Fig. 1a where MPI process is assigned to every core available in cluster then threads can be pined to the same core. This method is especially useful for pure MPI programs.

The aim of this work is to present hybrid MPI+OpenMP parallelized algorithm for high-order compact method in three spatial dimensions. The test program which was created for this research is included in 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 was presented in [24].

2.1. Compact Difference Scheme
The 6th order compact finite difference formula for the first order derivative on a collocated mesh is given as [2]:

\[
af_i' - f'_i + af_{i-1}' = \frac{b_1(f_{i+1} - f_{i-1})}{h} + \frac{b_2(f_{i+2} - f_{i-2})}{2h}
\]

where the coefficients are equal to \( a = 1/3, \ b_1 = 7/9, \ b_2 = 1/36 \). The equation (1) written for every node leads to the system of equations:

\[
Af' = Bf
\]

where the matrix \( A \) is tridiagonal.

2.2. Basic tridiagonal matrix algorithm
The tridiagonal matrix is a special type of matrix that the nonzero elements only on the main diagonal \( b \) and two neighbors before (\( a \)) and after (\( c \)) main diagonal. A general system of equations showed in Fig. 3 is:

\[
Ax = RHS
\]

**Figure 3.** Tridiagonal matrix.
and it could be solved by tridiagonal matrix algorithm (TDMA) also known as the Thomas algorithm. It could also be solved by the product of inverse tridiagonal matrix and RHS ($x = A^{-1}RHS$). Nevertheless, the later method is more time-consuming than TDMA. The TDMA algorithm is carried out in two steps:

- **forward** loops from 1 to $n$ and preparing $d$ and $\gamma$ coefficients
  
  $d_i = c_i/b_i$ where $i = 1$

  $d_i = (b_i - a_i * d_{i-1})/c_i$ where $i = 2, 3, ..., n$

  $\gamma_i = rhs_i/b_i$ where $i = 1$

  $\gamma_i = (rhs_i - a_i * \gamma_{i-1})/(b_i - a_i * d_{i-1})$ where $i = 2, 3, ..., n$

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

  $x_i = \gamma_i$ where $i = n$

  $x_i = \gamma_i - d_i * x_{i+1}$ where $i = n - 1, n - 2, ..., 1$

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

2.3. Parallel TDMA "arrowhead" method

The arrowhead method [7] is one of partitioning methods which allows solving equation (3) in a parallel way. The method relies on reducing initial matrix to some new independent diagonal blocks and sparse auxiliary ones along the right and bottom side and a coupling supplementary block (Fig. 4). Reduced matrix is presented in Fig. 5.

![Figure 4. Initial tridiagonal block divide by number of processes[7.](./fig4.png)](./fig4.png)

![Figure 5. Matrix after rearranged to arrowhead form[7.](./fig5.png)](./fig5.png)

The matrix rearrangement is performed in the following way:

- define number of subsystems $M$ which can rely on number of MPI tasks. Every subsystem is shown in Fig. 4 marked by thin black square,

- mark $M - 1$ separations block-rows (orange, green, red and grey colors ) for every subsystem without last one and corresponding block-column (blue) what is shown in Fig. 4

- marked separation block-rows are shifted to the bottom

- marked separation block-columns are shifted to the right
Global formula of this system is written as

\[
\begin{pmatrix}
S & W_r \\
W_l & H
\end{pmatrix}
\begin{pmatrix}
x_s \\
x_h
\end{pmatrix}
=
\begin{pmatrix}
rhs_s \\
rhs_h
\end{pmatrix}
\] (7)

where \( S \) is subsystem, \( H \) - head (orange squares), \( W_l \) - wings (green and blue squares).

The solution of the system is given by the relations

\[
\begin{cases}
x_s = S^{-1}rhs_s - S^{-1}W_rx_h \\
x_h = (H - W_lS^{-1}W_r)^{-1}(rhs_h - W_lS^{-1}rhs_s)
\end{cases}
\] (8)

The system is solved in the following steps:

- preparing \( Z_k = (S^k)^{-1}W_l^k \) on each process, which leads to TDMA solution for \( S_kZ^K = W_l^k \),
- preparing \( H - \sum_{k=1}^M W_l^k Z^k \) on one process,
- preparing \( z_k = (S^k)^{-1}rhs_s^k \) on each process, which leads to TDMA solution for \( S_kz^K = rhs_s^k \),
- preparing \( rhs_h - \sum_{k=1}^M W_l^k z^k \) on one process.

Synchronizing data to one process and preparing coefficients for result:

- preparing \( x_h \) which is a result for \( M - 1 \) subsystems \( x_h = (H - W_lS^{-1}W_r)^{-1}(rhs_h - W_lS^{-1}rhs_s) \) which leads to TDMA solution for \( (H - W_lS^{-1}W_r)x_h = (rhs_h - W_lS^{-1}rhs_s) \),
- sending \( x_h \) by scattering and correction of result on every process \( x_s^k = z^k - Z^k x_h \).

This method is very simple to implement and provide speed-up, especially compared to basic TDMA method. It could be applied for other diagonals like five-diagonal matrix or seven-diagonal matrix.

### 2.4. Test program

The SAILOR code is parallelized using MPI library by dividing the computational domain in the one direction (\( y \)) as presented in the Fig. 6a. Within this approach derivatives in the others directions (\( x, z \)) are obtained in parallel way (the communication between MPI processes is not required). Each subdomain whereas the \( y \)-derivatives are computed using serial TDMA algorithm or the arrowhead method (the communication between MPI processes is required). As it was mentioned in the Introduction in order to present hybrid MPI+OpenMP algorithm the test program was created. The test program prepares analytic function and performs calculations of derivatives in all spatial directions (\( x; y; z \)), however, it does not carry out whole CFD simulation. To allow OpenMP threads cooperate with MPI the domain could be additionally divided. For \( y \)-derivatives (communication between MPI processes is required) division should be done in second direction as presented in the Fig. 6b. For another derivatives (communication between MPI processes in not required) division should be done in the same direction as MPI processes, as shown in the Fig. 6c.

### 3. Results

Tests were run on PLGRID platform, which allows about 2399 TFlops computational power. Running and compiling application was done by using Intel MPI library version 2018 Update 1 and OpenMP in version 5.0. Specification of cluster is shown in table 1. Tested domain had to be fitted to cluster specification and was performed for two mesh sizes, for seven different numbers of computational units and for four different numbers of nodes. The meshes were
structured and the distance between mesh points was constant. Details are shown in table 2. The tests were performed for three different cases: pure MPI tool and serial TDMA, pure MPI and parallel TDMA (arrowhead), hybrid (MPI/OpenMP) and parallel TDMA (arrowhead).

Speed-up is measured by function: \( P = \frac{T^1}{T'} \), where \( T^1 \) is time for particular configuration, \( T' \) is time for basic TDMA solver on one computation unit.

As it can be observed in Figs 7 and 8 the hybrid solution offers great performance especially for many CPUs and it is less efficient for smaller number of CPUs. For minimum number of CPUs it offers similar efficiency that serial TDMA with pure MPI. The maximum performance what was achieved for hybrid solution is:

- for 240x240x240 mesh size:
  - about 7 times faster than serial TDMA and pure MPI
  - about 4.7 times faster than parallel TDMA method and pure MPI
- for 480x480x480 mesh size:
  - about 10 times faster than serial TDMA and pure MPI

Figure 6. Domain dividing: a - to MPI processes, b - to MPI processes and OpenMP in second direction, c - to MPI processes and OpenMP in the same direction.

Table 1. Specification of the test cluster.

| Processors     | Intel E5-2680v3 |
|---------------|-----------------|
| Socket per node | 2               |
| Cores per socket | 12              |
| Clock rate    | 2,50 GHz        |
| Memory size   | 128GB           |
| Nodes         | 2160            |
| Network       | Infiniband FDR 56Gb/s |

Table 2. Number of nodes for computational units and meshes.

| computational units | 1 | 6 | 12 | 24 | 48 | 60 | 96 |
|---------------------|---|---|----|----|----|----|----|
| 240x240x240         | 1 | 1 | 1  | 1  | 2  | 3  | -  |
| 480x480x480         | 1 | 1 | 1  | 1  | 2  | 3  | 4  |
Figure 7. Results for mesh size 240x240x240 points.

Figure 8. Results for mesh size 480x480x480 points.

– about 6.7 times faster than parallel TDMA method and pure MPI

It can also be observed that pure MPI with serial TDMA accelerates with increasing computational units. That is because measured speed-up includes preparing RHS, creating $f$ and performing calculations in the second and third spatial directions. It is also important that both approaches serial TDMA and parallel TDMA method with pure MPI slow down after some "sweet point" depending on mesh size.

4. Conclusions
There is a great demand for fast and efficient systems in today’s word. Hybrid MPI/OpenMP approach is supposed to enhance the speed of calculations for high order methods, what has been proved by the investigations performed. Comparison of all three sorts of approaches shows
the important advantages of the hybrid model over the pure MPI with serial TDMA and pure MPI with parallel TDMA. It is worth mentioning that the size of computational domain directly influences the performance of calculations. The tests indicate that the larger domain is used the larger acceleration is achieved. In the case of computational units below 12 it is sufficient to use pure MPI with parallel TDMA, however, to gain best performance in the case of computational units over 12 taking advantage of the hybrid MPI/OpenMP approach is preferable. As a part of future work this approach will be implemented to whole CFD simulation. Moreover other approaches of performing calculations for high-order methods on heterogeneous clusters with different numbers of socket and core per socket may be compared.

Acknowledgements.
The research was supported by Polish National Science Centre, project no. 2016/21/B/ST8/00414. The computations were carried out using 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** 811-845
[2] Lele S 1992 *J. Comput. Phys.* **103** 16-42
[3] Hockney R 1965 *J. ACM* **12** 16-42
[4] Wang H 1981 *ACM Trans. Math. Softw.* **7** 170-183
[5] Mattor N, Williams T, and Hewett D 1995 *Parallel Comput.* **21** 1769-1782
[6] Sun H, Zhang H, and Ni L 1992 *IEEE on Trans.Comput.* **41** 286-296
[7] Belov P, Nugumanov E and Yakovlev S 2015 *Preprint arXiv* 1505.06864
[8] Stone H 1975 *ACM Trans.Math. Softw.* **1** 289-307
[9] Rao S C S 2008 *Parallel Comput.* **34** 177-197
[10] Qin J and Nguyen D 1998 *ADV ENG SOFTW* **29** 395-397
[11] Afaq A, Ansari Z, Faizabadi A, and Ramis M 2017 *ARCH COMPUT METHOD E* **24** 337-363
[12] Dagum Land Menon R 1998 *IEEE Comput. Sci. Eng.* **5** 46-55
[13] William D, Lusk E and Skjellum A 1999 *Using MPI: Portable Parallel Programming With The Message-Passing Interface* **1**
[14] Amritkar A, Deb S and Tafti D 2014 *J.Comput.Phys*
[15] Mavriplis D 2002 *Int. J. High Perform. Comput. Appl.* **16** 395-407
[16] Maknickas A, Kačianauskas A, Kačianauskas R, Balevičius R, and Algis Džingys A 2006 *Informatica* **17** 207-224
[17] Jia R and Sundén B 2004 *Comput. Fluids* **33** 57-80
[18] Mininni P, Rosenberg D, Reddy R, and Pouquet A 2011 *Parallel Comput.* **37** 316-326
[19] Gropp W, Kaushik D, Keyes D, and Smith B 2001 *Parallel Comput.* **27** 337-362
[20] Rabenseifner R, Hager G, and Jost G 2009 In *Proc Euromicro Int Conf Parallel Distrib Netw Based Process* **17** 427-436
[21] Wawrzak K, Bogusławski A, and Tyliszczak A 2015 *Flow Turbul. Combust.* **95** 437-459
[22] Aniszewski W, Bogusławski A, Marek M, and Tyliszczak A 2012 *J. Comput. Phys.* **231** 7368-7397
[23] Wawrzak A and Tyliszczak A 2017 *ARCH MECH* **69** 157-175
[24] Tyliszczak A 2014 *J. Comput. Phys.* **276** 438-467