A Parallelization of Finite Volume Method for Calculation of Gas Microflows by Domain Decomposition Methods

Kiril S. Shterev and Stefan K. Stefanov

Institute of Mechanics, Bulgarian Academy of Sciences,
Acad. G. Bonchev, Block 4, Sofia 1113, Bulgaria
{kshterev,stefanov}@imbm.bas.bg
http://www.imbm.bas.bg

Abstract. In this paper a parallel organization of a finite volume algorithm for calculation of two-dimensional compressible, viscous gas flows is presented. The problem is addressed to the new emerging area of the micro gas flows, taking place in Micro-Electro-Mechanical Systems (MEMS). Technically, the parallel algorithm is organized by using standard MPI, non-blocking communications and the latency of the communications is overlapped with useful work. The speedup was estimated on two clusters. The first cluster uses MYRINET interconnections (BG04-ACAD). The second uses conventional cards for interconnections (BG03-NGCC). Both clusters are a part of GRID-infrastructure of the European Research Area (ERA). An ideal speedup is obtained on BG04-ACAD for a number of processors up to 20 CPUs. The speedup obtained on BG03-NGCC is very good, however it depends on the mesh refinement.

Keywords: Finite volume method, gas microflows, parallel algorithms, GRID.

1 Introduction

The computational analysis of fluid dynamics problems depends strongly on the computational resources [11]. The computational demands are related mainly to: the CPU performance and the memory size. The considered in the paper example of calculation of a two-dimensional gas flow past a supersonic speed particle moving in planar microchannel is a typical problem demonstrating such kind of computational requirements. We consider a supersonic flow with Mach number equal to 2.43. The problem is described in details in [9]. The supersonic speed leads to the existence of a bow shock wave , which reflects from the channel walls. As a result past the particle we obtain a complex picture of interaction of reflected shock waves (see Fig. 1). The shock waves have significant gradients of velocities, pressure, and temperature. Thus, an accurate calculation of the flow requires a very fine or adaptive grid to be used. Calculations of this problem has been carried out for a set of gradually refined meshes. Finally, a mesh with 8000x1600 cells was found to give stable and accurate enough results. It is easy

I. Lirkov, S. Margenov, and J. Waśniewski (Eds.): LSSC 2009, LNCS 5910, pp. 523–530, 2010.

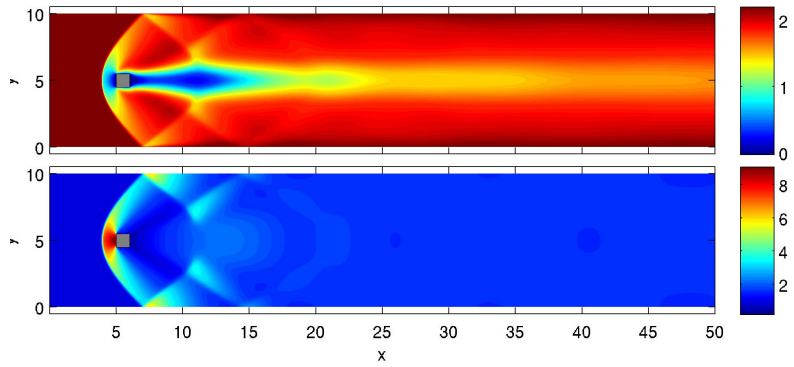

Fig. 1. Horizontal velocity (upper part) and pressure (lower part) fields calculated by parallel FVM [9]

to predict that a calculation of this problem on a single PC will take years. Obviously, a parallel organization of the computational process is required in order to overcome this difficulty. In the next section we present how this can be accomplished.

2 Parallel Organization

Parallel organizations of FVM algorithms are presented in many papers (for examples, see [12,6]). We presented here the main statements of the parallel organization of the finite volume method [8,11] (FVM) for calculation of the considered gas flow. The corresponding serial algorithm is presented in details in [10]. The same notations are used here.

A domain decomposition (data partitioning) approach is used. The idea is a single instruction stream and multiple data stream (**SIMD**) to be used in accordance with Flinn's taxonomy. The realization was accomplished by using standard MPI (Message Passing Interface) [3] instructions. This approach makes possible running the code on symmetric multiprocessing (SMP) systems and computer clusters. Most of the modern supercomputers now are highly-tuned computer clusters using commodity processors combined with custom interconnects [5].

Here the send/received packages are of small size. The size of packages is less then 50KB, which is negligible for bandwidth 1Gbps. Therefore the bottleneck of the communications is the latency. There are a lot of communications in one iteration of FVM. To reach a high parallel efficiency non-blocking communications are used and latency is overlapped with calculations.

Figure 2 presents an example of decomposition of a domain into two processes. The halo region for process 0 contains a column of variables, some of them positioned in the centre and others on the face of the cells. The halo region for process 1 contains two columns of variables, placed in the centre and on the face

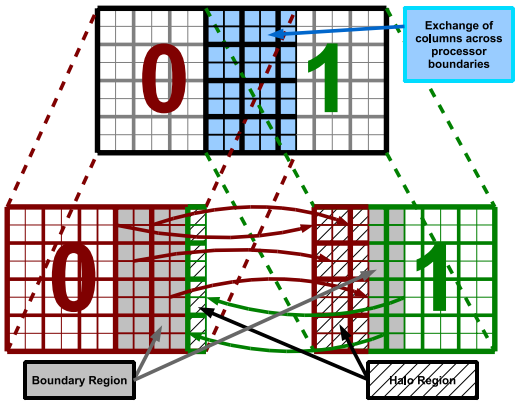

Fig. 2. Domain decomposition of FVM computational domain

of the cells. When halo regions are set in this way the minimum sent/received messages between neighbours is reached.

The steps in the parallel FVM are the same as in the serial FVM [10].

Parallel FVM

1. Initial guess of u, v, p, T and calculation of ρ, using equation of state.

Start loop 1

- **2.** Initial conditions for the current time step: $t = t + \Delta t$, $u^{(n-1)} = u$, $v^{(n-1)} = v$, $p^{(n-1)} = p$, $T^{(n-1)} = T$, $\rho^{(n-1)} = \rho$.
- **Start loop 2** (current time step calculations):

  - **3.** Evaluate flux and diffusion terms: F^x, F^y, D^{ux}, D^{uy}, D^{vx}, D^{vy}, D^{Tx}, D^{Ty}.
  - **4.1.** Evaluate pseudo velocity $\hat{u}$, coefficient d^u and coefficients a^{px}, b^{px} of pressure equation in the boundary region.
  - **4.2.** Initiate sending of boundary values of a^{px} and b^{px} to neighbours and initiate receipt of halo values of a^{px} and b^{px} from neighbours.
  - **4.3.** Calculate boundary values of pseudo velocity $\hat{v}$, coefficient d^v and coefficients for pressure equation a^{py}, b^{py}.
  - **4.4.** Initiate sending of boundary values of a^{py} and b^{py} to neighbours and initiate receipt of halo values of a^{py} and b^{py} from neighbours.
  - **4.5.** Calculate non-boundary values of pseudo velocity $\hat{u}$, coefficient d^u and coefficients for pressure equations a^{px} and b^{px}.
  - **4.6.** Calculate non-boundary values of pseudo velocity $\hat{v}$, coefficient d^v and coefficients for pressure equations a^{py} and b^{py}.
  - **4.7.** Wait for completion of sending of boundary values of a^{px}, b^{px}, a^{py}, and b^{py} and wait for completion of receipt of halo values of a^{px}, b^{px}, a^{py}, and b^{py}.

- **Start loop 3:**
  - **5.1.** Calculate temperature and pressure in the boundary cells, using energy and pressure equations.
  - **5.2.** Initiate sending of the boundary values of p and T to neighbours and initiate receiving of halo values of p and T from the neighbours.
  - **5.3.** Calculate temperature and pressure in the non-boundary cells.
  - **5.4.** Wait for completion of the sending of boundary values of p and T and wait for completion of receiving of halo values of p and T.
  - **5.5.** Initiate sending of maximum residuals for the equations of pressure and energy from each process to process 0 and initiate receipt of maximum residuals for equations or pressure and energy from all processes in process 0.
  - **5.6.** Initiate array p_{old} for next iteration in **loop 3**: $p_{old} = p$.
  - **5.7.** Wait for completion of sending of maximum residuals for equations for pressure and energy and wait for completion of receipt of maximum residuals for equations for pressure and energy in process 0.
  - **5.8.** The **loop 3** continues until convergence criteria are not satisfied or the maximum number of iterations is not reached. Initiate sending and receipt of information to stop or continue **loop 3**.
  - **5.9.** Initiate array T_{old} for next iteration in **loop 3**: $T_{old} = T$.
  - **5.10.** Wait for completion of sending and receipt of information to stop or continue **loop 3**.
- **End loop 3.** At least 2 iteration have to be performed to ensure convergence. The detailed analysis of two-dimensional unsteady state pressure driven channel flow [10] showed that 2 iterations were enough to achieve a good accuracy.
- **6.1.** Calculate u in the boundary cells.
- **6.2.** Initiate sending of boundary values of u to neighbours and initiate receipt of halo values of u from neighbours.
- **6.3.** Calculate v in the boundary cells.
- **6.4.** Initiate sending of boundary values of v to neighbours and initiate receipt of halo values of v from neighbours.
- **6.5.** Calculate u in non-boundary cells.
- **6.6.** Initiate sending of maximum residuals for u from each process to process 0 and initiate the reception of the maximum residuals for u from all processes in process 0.
- **6.7.** Calculate v in the non-boundary cells.
- **6.8.** Initiate sending of maximum residuals for v from each processes to process 0 and initiate receipt of maximum residuals for v from all processes in process 0.
- **6.9.** Calculate the array containing information for $\sqrt{T}$. (This is useful as $\sqrt{T}$ is often used in algorithm).

- •• **6.10.** Wait for completion of sending of boundary values of u and v and wait for completion of receipt of halo values of u and v.
- •• **6.11.** Wait for completion of sending of maximum residuals for u and v and wait for completion of receipt of maximum residuals for u and v in process 0.
- •• **6.12.** Check conditions to stop **loop 2**.
- •• **6.13.** Initiate sending of information to continue or to stop **loop 2** in process 0 to all processes and initiate receipt of decision to stop or to continue **loop 2** in all processes.
- •• **6.14.** Calculate density ρ, using equation of state and p and T calculated in **loop 3**.
- •• **6.15.** Wait for completion of sending and receipt of information to stop or continue **loop 2**.

• **End loop 2**.

End loop 1: The calculation is stopped, when the end time or the stationary state is reached. If these criteria are not reached go to **step 2**.

Note: Equations for p and T in **loop 3** are calculated using Jacobi iterative method (see [7]).

The organization of sending and receiving messages is done fully manual in order to make possible the use of non-blocking communications. To this aim a structure `DomainDecomposition` is created. It contains information for every process. The following functions are used for calculation of tags and indices of requests of messages:

- `CalculateTagForSendVariableToNeighbour` and `CalculateIndexRequestForSendVariableToIJ0` calculate the tag and index of request of each message send to a neighbour. Both functions receive an unique index for each variable (`i_variable`) that must be send (for example the index for pressure is 1, for temperature is 2 and etc.) and the index of the neighbour (`i_neighbour`). Variable `zero_tag_for_this_IJ` is the maximum tag of all processes on the left (Fig. 2.).
- `CalculateTagForSendVariableToIJ0` and `CalculateIndexRequestForSendVariableToIJ0` calculate the tags and indices of request of the messages send to process 0. Both functions receive the following variables: `i_variable`; `N_exchanged_variables_with_neighbours` — the number of all variables, which are sent/received with neighbours; `N_neighbours` — the number of neighbours (for 2D case is 4); `N_actions`, which is equal to 2 (two possible actions — send and receive).
- `CalculateTagForSendVariableFromIJ0` and `CalculateIndexRequestForRecvVariableFromIJ0` calculate tags and indices of request of the messages send of process 0 to other processes, where `N_send_variables_to_IJ0` is the number of send variables to process 0.

A part of source code containing functions for calculation of tags and indices of requests of messages:

```
inline int CalculateTagForSendVariableToNeighbour(
 const int& i_variable, const int& i_neighbour){
  return (zero_tag_for_this_IJ + i_variable * N_neighbours
      + i_neighbour);};
inline int CalculateIndexRequestForSendOrRecvVariableNeighbour(
 const int& i_variable, const int& i_neighbour,
 const int& send_or_recv_data){
  return(i_variable * N_actions * N_neighbours
   + send_or_recv_data * N_neighbours + i_neighbour);};
inline int CalculateTagForSendVariableToIJ0(
 const int& i_variable){
  return (zero_tag_for_this_IJ
          + N_exchanged_variables_with_neighbours * N_neighbours
          + i_variable);};
inline int CalculateIndexRequestForSendVariableToIJ0(
 const int& i_variable){
  return(N_exchanged_variables_with_neighbours * N_actions
         * N_neighbours + i_variable);};
inline int CalculateTagForSendVariableFromIJ0(
 const int& i_variable){
  return (zero_tag_for_this_IJ
   + N_exchanged_variables_with_neighbours * N_neighbours
   + N_send_variables_to_IJ0 + i_variable);};
inline int CalculateIndexRequestForRecvVariableFromIJ0(
 const int& i_variable){
  return(N_exchanged_variables_with_neighbours * N_actions
         * N_neighbours + N_send_variables_to_IJ0 + i_variable);};
```

3 Speedup Analysis

The speedup of the parallel FVM for calculations of gas microflows, was estimated on two clusters. The first cluster uses myrinet [4] interconnection (BG04-ACAD). The second cluster uses conventional cards for interconnection (BG03-NGCC). The characteristics of the clusters are shown in Table 1. They are a part of a GRID-infrastructure [2] of the European Research Area (ERA). The tests were submitted by user certificate [1].

The speedup is calculated as $S_n = T_s/T_{par}$ and the efficiency is calculated as $E_n = S_n/n$, where n is the number of cores (CPUs), S_n is the speedup, when n-cores are used, T_s is the time for sequential run of the program (run on a single core), T_{par} is the time of code run on n-cores, E_n is the efficiency, when n-cores are used.

The test problem mentioned in the introduction is a calculation of supersonic flow past a small square shaped particle moving in a microchannel, Fig. 1. The test calculations were carried out on two meshes: 500x100 and 1000x200 cells. The calculation of the case (500x100 cells) was completed for around 2.5 hours on a single core.

Table 1. The characteristics of the clusters BG04-ACAD and BG03-NGCC

Cluster:	**BG04-ACAD**	**BG03-NGCC**
Numbers of cores:	80	200
CPU model:	AMD Opteron(tm) Processor 250	Intel(R) Xeon(R) CPU E5430
CPU GHz:	2.4 GHz	2.66 GHz
Cache size:	1024 KB	6144 KB
RAM per core:	2 GB	2 GB
Interconnection:	Myrinet	Conventional cards

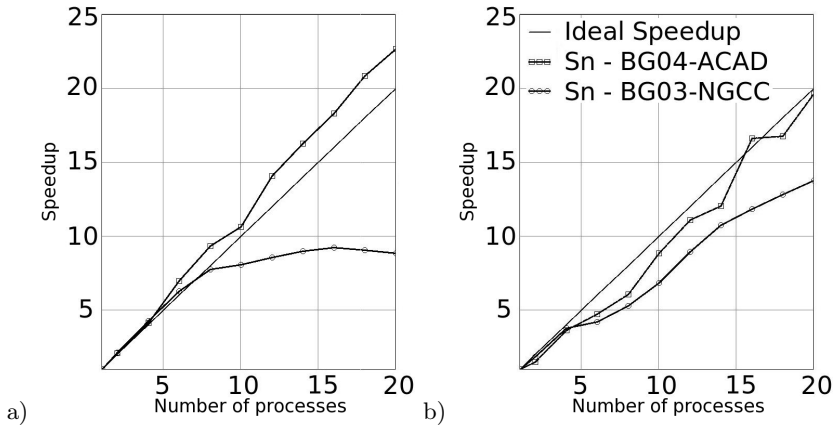

Fig. 3. The speedup of the clusters BG04-ACAD and BG03-NGCC, for meshes: a) 500x100 cells and b) 1000x200 cells

The results concerning the speedup are shown in Fig. 3. The speedup for BG04-ACAD was calculated by taking the minimal run time obtained from 6 runs of each case. The speedup for BG03-NGCC was calculated by taking the minimal time obtained from 11 runs of each case. The tests on BG03-NGCC needed more runs, than those performed on BG04-ACAD, because there was a larger variation in the run times obtained from BG03-NGCC due to the use of conventional card for interconnection.

A super linear speedup was reached on the cluster BG04-ACAD for the case with mesh 500x100 cells, Fig. 3. a). The reason for this is the cache effect. The program, run on a single core, needs around 14.5 MB of memory. The increase of process number leads to a decrease of the used memory of the program and, in the case with 20 processes, all data can be placed in the cache memory of the processors (Table 1.). The case with 4 times more cells (1000x200) needs around 58 MB on a single core, Fig. 3. b), respectively the available cache memory is not enough to hold all data and a super linear speedup cannot be reached. The speedup for BG03-NGCC is very good Fig. 3. a), even considering the small problem on mesh 500x100 cells. For a finer mesh (1000x200 cells) the speedup Fig. 3. b) is excellent, nevertheless that BG03-NGCC uses conventional cards

for interconnection. The speedup line is not very smooth (Fig. 3) because of the weak interference with other runs simultaneously performed by other users on the clusters.

4 Conclusions

The non-blocking communications make possible an excellent speedup to be reached. A super linear speedup was reached on the cluster with myrinet interconnection (BG04-ACAD). A very good speedup was reached on the cluster BG03-NGCC for the case of mesh with 500x100 cells and an excellent speedup — on mesh (1000x200). These results show that the good parallel organization makes the algorithm efficient even when run on clusters with conventional cards for interconnection.

Acknowledgments. The authors appreciate the financial support of NSFB DO 02-115, Module 1. The author K. Shterev also appreciates the financial support of THE EUROPEAN SOCIAL FUND, OPERATIONAL PROGRAMME HUMAN RESOURCES DEVELOPMENT, GRANT No: BG051PO001/07/3.3-02-55/17.06.2008. The results were obtained using the grid sites of the South East European Regional Operating Centre of the EGEE project (`http://public.eu-egee.org`).

References

1. Bulgarian Academic Certification Authority (BG.ACAD|CA), `http://ca.acad.bg`
2. Enabling Grids for E-sciencE (EGEE), `http://www.eu-egee.org`
3. MPI Forum, `http://www.mpi-forum.org`
4. Myricom Inc., `http://www.myri.com`
5. Supercomputers, `www.top500.org`
6. Bui, T.T.: A parallel finite-volume algorithm for large-eddy simulation of turbulent flows. Computers & Fluids 29, 877–915 (2000)
7. Em, K.G., Kirby II, R.M.: Parallel Scientific Computing in C++ and MPI. A Seamless Approach to Parallel Algorithms and their Implementation. University Press, Cembridge (2003)
8. Patankar, S.V.: Numerical heat transfer and fluid flow. Hemisphere Publishing Corporation, New York (1980)
9. Shterev, K.S., Stefanov, S.K.: Finite Volume Calculations of Gaseous Slip Flow Past a Confined Square in a Two-Dimensional Microchannel. In: Proceedings of the 1st European Conference on Microfluidics, La Societe Hydrotechnique de France (2008)
10. Shterev, K.S., Stefanov, S.K.: Pressure based finite volume method for calculation of compressible viscous gas flows. Journal of Computational Physics 229, 461–480 (2010), doi:10.1016/j.jcp.2009.09.042
11. Versteeg, H.K., Malalasekra, W.: An Introduction to Computational Fluid Dynamics: The Finite Volume Method, 2nd edn. Prentice Hall, Pearson (2007)
12. Wang, P., Ferraro, R.D.: Parallel multigrid finite volume computation of three-dimensional thermal convection. Computers & Mathematics with Applications 37, 49–60 (1999)