Heterogeneous Parallel Algorithm for Compressible Flow Simulations on Adaptive Mixed Meshes

Sergey Soukov(✉)

Federal Research Center Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Moscow 12504, Russia

Abstract. The paper describes a parallel heterogeneous algorithm for compressible flow simulations on adaptive mixed meshes. The numerical algorithm is based on an explicit second order accurate finite-volume method for Navier-Stokes equations with linear reconstruction of variables. Gas dynamics equations are discretized on hanging node meshes of tetrahedra, prisms, pyramids, and hexahedra. The mesh adaptation procedure consists in the recursive refinement and/or the coarsening of cells using predefined templates. Heterogeneous parallelization approach uses MPI + OpenMP + CUDA program model that provides portability across a hybrid supercomputers with multicore CPUs and Nvidia GPUs. A two-level domain decomposition scheme is used for the static load balance among computing devices. First level decomposition involves the mesh partitioning between supercomputer nodes. At the second level node subdomains are partitioned between devices. The Cuthill-McKee reordering is used to set cell local indexes in the MPI process computational domain to speed up CFD kernels for the CPU. The CUDA implementation includes a communication and computation overlap that hides the communication overhead. The heterogeneous execution scalability parameters are shown by the example of simulating a supersonic flow around a sphere on different adaptive meshes using up to 70 multicore CPUs and 105 GPUs.

Keywords: Heterogeneous computing · Unstructured mesh refinement · Parallel CFD

1 Introduction

The time and results of CFD simulations strongly depend on the type and size (number of nodes, elements, faces, etc.) of the computational mesh. Hybrid and mixed meshes [1] are most suitable for modeling flows around the complex shape bodies. Adaptive mesh refinement methods are often used to improve solution accuracy and decrease the mesh size [2]. The adaptive mesh is statically or dynamically refined directly to the flow features. Static adaptation is used for steady flow simulations. The launch of the parallel CFD program alternates with the solution analysis and mesh adaptation [3]. This approach can be used for multiple simulations with different flow parameters and the same starting coarse mesh. Dynamic adaptation techniques are applied for the simulations of unsteady flows [4, 5]. The mesh is refined during the time the solution is being calculated. For example, the fine cells zones are moved according to the current positions of the turbulent regions after each time integration step. In this case, the mesh adaptation routines are included directly to the main CFD code.

This paper describes a parallel algorithm for compressible flow simulations on hanging node mixed meshes. Heterogeneous parallelization approach is based on MPI + OpenMP + CUDA hybrid program model. The CUDA implementation includes a communication and computation overlap. A distinctive feature of the proposed algorithm in comparison with existing CFD codes is that it can be used both for simulations on hanging node meshes and arbitrary polyhedral meshes.

The paper is organized as follows. The mathematical model and numerical method are briefly outlined in Sect. 2. The parallel algorithm and details of the overlap mode and heterogeneous execution are described in Sect. 3. The results of the study of parallel efficiency and performance are presented in Sect. 4. And Sect. 5 contains some conclusions.

2 Mathematical Model and Numerical Method

A compressible viscous fluid is modeled by the Navier–Stokes equations:

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \nabla \cdot \boldsymbol{F} = 0, \tag{1}$$

where $\boldsymbol{Q} = (\rho, \rho u, \rho u, \rho u, E)^T$ is a vector of conservative variables, $\boldsymbol{F} = \boldsymbol{F}^C + \boldsymbol{F}^D$ is a flux vector that consists of convective and diffusive fluxes, respectively.

The Eq. (1) are discretized on an hanging node mixed meshes using a explicit cell-centered finite-volume method with a polynomial reconstruction. The starting conformal mesh combines tetrahedra, prisms, pyramids, and hexahedrons. The mesh adaptation procedure consists in the recursive refinement and/or the coarsening of cells using predefined templates (Fig. 1). A subset of the mesh cells are marked for dividing or agglomeration based on the adaption function, which is created from numerical solution data. After adaptation, neighboring cells can't to differ by more than one level of refinement.

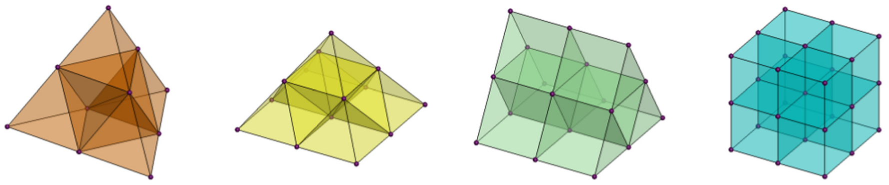
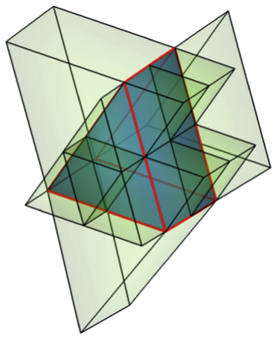


Fig. 1. Cells refinement templates.

During CFD simulations, non-conformal hanging node mesh is transformed to a conformal polyhedral mesh. If a cell is adjacent to four cells along a face, then this face is divided into four parts (Fig. 2).


a) prism and adjacent cells

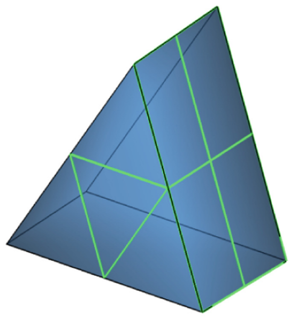

b) transformed prism

Fig. 2. Example of transforming a prism to a polyhedron.

Each polyhedral cell C_i has a volume $|C_i|$ and surface ∂C_i. In accordance with the finite volume method, a balancing relation is written for cells as:

$$\frac{\partial}{\partial t}\int_{C_i} \boldsymbol{Q} d\Omega + \oint_{\partial C_i} \boldsymbol{F} \mathrm{d}\mathbf{S} = 0. \tag{2}$$

The discrete values of the mesh function Q_i are defined at the cell-centroid $\boldsymbol{r}_i$ as integral average

$$Q_i = \frac{1}{|C_i|}\int_{C_i} Q d\Omega. \tag{3}$$

Surface ∂C_i of an internal cell consists of flat faces ∂C_{ij}. The geometry of the face ∂C_{ij} between cells C_i and C_j is set by the unit normal $\boldsymbol{n}_{ij}$, the face-midpoint $\boldsymbol{r}_{ij}$ and its square S_{ij}.

The discrete variant of (2) has the form

$$\frac{\partial \boldsymbol{Q}_i}{\partial t} + \sum\nolimits_{j \in I_i} \left(\boldsymbol{F}_{ij}^C + \boldsymbol{F}_{ij}^D\right) S_{ij} = 0, \tag{4}$$

where I_i is the set of C_i neighbors.

The convective flux $\boldsymbol{F}_{ij}^C$ is computed using a Riemann solver

$$\boldsymbol{F}_{ij}^C = \Phi\left(\boldsymbol{Q}_{ij}^L, \boldsymbol{Q}_{ij}^R, \boldsymbol{n}_{ij}\right), \tag{5}$$

where $\boldsymbol{Q}_{ij}^L$ and $\boldsymbol{Q}_{ij}^R$ are the values from the left and right states evaluated at the interface ∂C_{ij}. The implemented set of Riemann solvers [6, 7] Φ includes CUSP, Roe, Roe–Pike for subsonic flows; HLL, HLLE, HLLC, Rusanov and AUSMDV for supersonic flows.

It is assumed that the solution is piecewise linearly distributed over the control volume. In this case, the left and right states can be found from the relations

$$Q_{ij}^{L/R} = Q_{i/j} + \Psi_{\mathrm{i/j}}\left(\nabla Q_{\mathrm{i/j}} \cdot \left(\boldsymbol{r}_{ij} - \boldsymbol{r}_{i/j}\right)\right), \tag{6}$$

where $\nabla Q_{\mathrm{i}} = (\partial Q/\partial x, \partial Q/\partial y, \partial Q/\partial z)$ is the gradient of Q at the i-th cell center and Ψ denotes a limiter function [8, 9].

The algorithm for gradient calculation is based on the Green-Gauss approach. For the original mesh cells, the linear reconstruction coefficients are calculated as

$$\nabla Q_{\mathrm{i}} = \frac{1}{|C_i|}\sum\nolimits_{j\in I_i}\left(g_{ij}^{i}Q_i + g_{ij}^{i}Q_j\right)\boldsymbol{n}_{ij}S_{ij}. \tag{7}$$

Geometrical coefficients g_{ij}^{i} and g_{ij}^{j} are inversely to the distances from the cell-centroids to the face plane and satisfy $g_{ij}^{i} + g_{ij}^{j} = 1$.

For reconstruction over transformed polyhedra, dummy control volumes are used. Dummy cell $C_i^{'}$ vertices are placed at the points of intersection between ∂C_{ij} planes and lines connecting the polyhedra centroid $\boldsymbol{r}_i$ and adjacent cells centroids $\boldsymbol{r}_j$. The volume of $C_i^{'}$ is bounded by triangulation of the convex hull of its vertices. The values of variables at the vertices and faces midpoints are computed by linear interpolation on segments and triangles. Figure 3 shows reconstruction control volume for transformed prism (Fig. 2). The dummy control volume approach can also be used to calculate gradients in the conformal mesh cells.

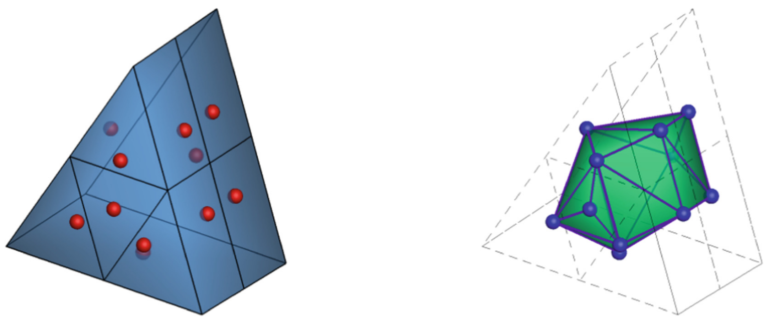


Fig. 3. Dummy control volume for transformed prism.

In order to evaluate the diffusive fluxes $\boldsymbol{F}_{ij}^{D}$ first derivatives of the velocity components and temperature at face-midpoints are computed as the combination of the averaged gradients $\overline{\nabla Q_{ij}}$ and the directional derivatives:

$$\overline{\nabla Q}_{ij} = \frac{\nabla \boldsymbol{Q}_{\mathrm{j}} + \nabla \boldsymbol{Q}_{\mathrm{j}}}{2}, \tag{8}$$

$$\nabla Q_{\mathrm{ij}} = \overline{\nabla Q}_{ij} - \left[\overline{\nabla Q}_{ij}\mathbf{t}_{\mathrm{ij}} - \frac{\mathrm{Q}_{\mathrm{j}} - \mathrm{Q}_{\mathrm{i}}}{\mathrm{l}_{\mathrm{ij}}}\right]\mathbf{t}_{\mathrm{ij}}. \tag{9}$$

Here $\mathrm{l}_{\mathrm{ij}} = |\boldsymbol{r}_j - \boldsymbol{r}_i|$ represents the distance between the cell-centroids, and $\mathbf{t}_{\mathrm{ij}} = (\boldsymbol{r}_j - \boldsymbol{r}_i)/\mathrm{l}_{\mathrm{ij}}$ is the unit vector along the line connecting them.

The boundary conditions are specified by setting the appropriate flow variables $\boldsymbol{Q}^R_{i\Gamma}$ or fluxes $\boldsymbol{F}_{i\Gamma}$ at the midpoints of the faces $\partial C_{i\Gamma}$ at the computational domain boundary Γ.

Finally, explicit multi-stage Runge–Kutta (RK) time integration schemes are used.

3 Parallel Algorithm and Implementation

3.1 Sequential Time Integration Step Algorithm

Each explicit RK step include multiple execution of the following operations:

- Reconstruction kernel – to get gradients at the cell centers, a loop over the cells;
- Fluxes kernel – calculation of convective and diffusive fluxes through inner and boundary faces, a loop over the faces;
- Update kernel – summation of fluxes along adjacent faces and calculation of the flow variables at a new RK scheme stage or time step, a loop over the cells.

The statement blocks of the loops are data independent and are considered as the parallelism units in the CPU (OpenMP) and GPU (CUDA programming model) CFD kernel implementations.

3.2 Distributed-Memory MPI Parallelization

The problem of static load balancing between MPI processes is solved based on the domain decomposition method. Each MPI process updates the flow variables for the cells of the polyhedral mesh subdomain (Fig. 4).

Graph partitioning subroutines [10, 11] work with an unstructured node-weighted dual graph. It is a graph in which nodes are associated with mesh polyhedrons, and edges are adjacency relations by shared faces. The polyhedron weight is set as the integer equal to the number of faces. As a result of decomposition, each mesh cell gets the rank of the MPI process, the subdomain of which it belongs to, and each process gets a list of its own cells.

The set of own cells is divided into subsets of inner and interface cells. The interface cells spatial scheme stencil includes cells from neighbouring subdomains or halo cells. Thus, the local computational domain of the MPI process joins up its own and halo cells. The interface cells of one subdomain are halo cells for one or more neighboring subdomains.

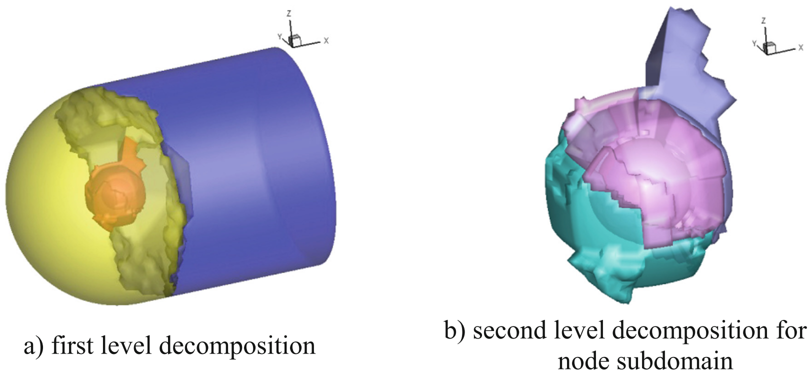

a) first level decomposition
b) second level decomposition for red node subdomain

Fig. 4. Two-level decomposition.

The local indexes of the computational domain cells are ordered according to their type: inner, interface, halo. In addition, Cuthill-McKee reordering [12] is used to set indexes within inner and interface cell groups. This approach optimizes the data location in RAM and improves the CFD kernels performance. The halo cell indexes are ordered by the MPI process ranks. Figure 5 shows the distribution of cells by type for the computational domain, built around the 8×8 cells subdomain of a structured grid.

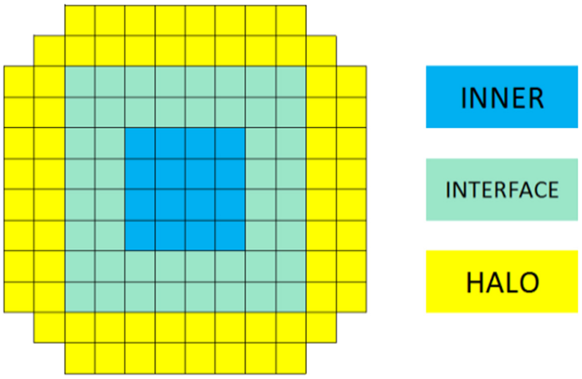


Fig. 5. Computational domain structure in 2D.

Parallel execution of the RK stage starts with asynchronous MPI exchanges. The data transfer includes only the flow variables. Halo cells gradients and interface/halo face fluxes computations are duplicated. Persistent requests for messages are created before the simulation starts by the subroutines MPI_Send_Init and MPI_Recv_Init. At the synchronization point, the MPI process packs the interface data to the send buffers, starts collecting requests (MPI_Startall routine) and then waits for all given requests to complete (MPI_Waitall routine). Halo data is received directly to the calculation data arrays without intermediate bufferization.

3.3 Heterogeneous Execution and Overlap Mode

Each MPI process is bound either to fixed CPU cores number or to one core (host) and one GPU (device). The sizes of the local computational domains are proportional to the device performance. The relative performance ratio between CPU and GPU is determined experimentally by the CFD kernels execution time.

If the CFD kernels are running on a GPU, parallel execution includes additional device-to-host and host-to-device data transfers before and after the MPI exchange. Interface flow variables are packed to intermediate buffers on the device and then copied to the host MPI send messages buffers. Halo data received to intermediate host buffers, which are copied to device data arrays.

The execution of the kernels on the GPU allows to overlap communication and computation. The inner cells are computed simultaneously with the halo update operation, then the interface cells are computed using the updated halo. In this case, host intermediate buffers are allocated in page-locked (or pinned) memory. Two asynchronous CUDA streams are created on the device. First stream used as a queue for halo update procedures execution. CFD kernels are placed to the second stream. The algorithm for executing the RK stage on the GPU in overlap mode is shown in Fig. 6.

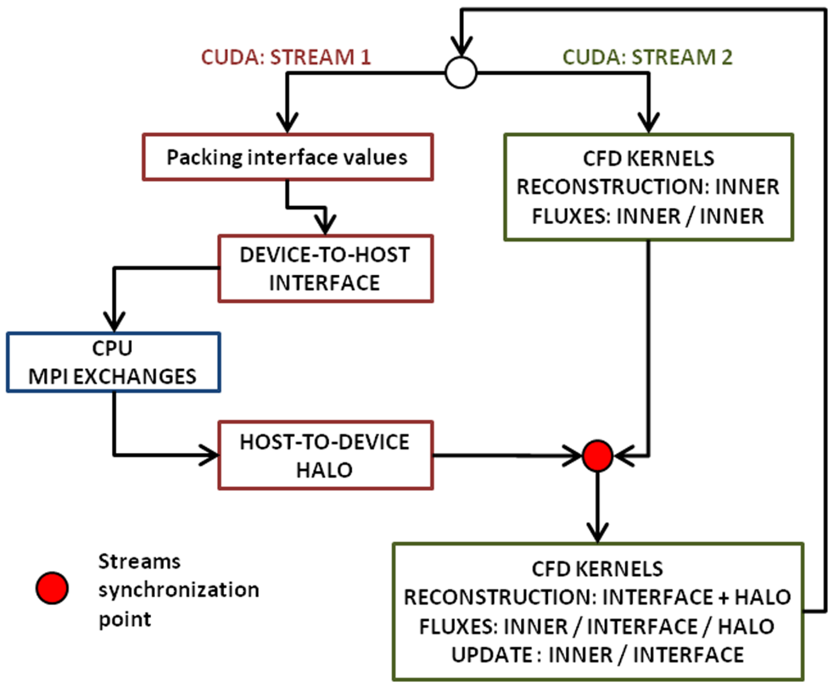
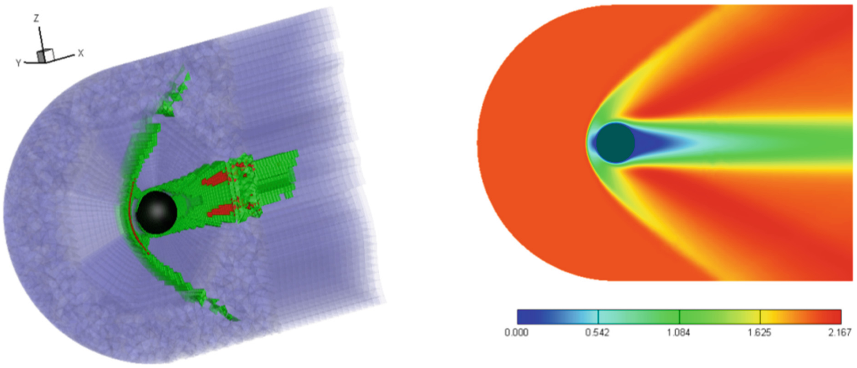

CUDA: STREAM 1
CUDA: STREAM 2
Packing interface values
DEVICE-TO-HOST INTERFACE
CPU MPI EXCHANGES
HOST-TO-DEVICE HALO
CFD KERNELS RECONSTRUCTION: INNER FLUXES: INNER / INNER
CFD KERNELS RECONSTRUCTION: INTERFACE + HALO FLUXES: INNER / INTERFACE / HALO UPDATE : INNER / INTERFACE
Streams synchronization point

Fig. 6. RK stage execution algorithm.

4 Parallel Efficiency and Scalability

The study of the parallelization efficiency, the overlap mode and the heterogeneous execution performance was carried out during numerical simulation of the supersonic flow around a sphere at Mach number 2 and Reynolds number 300. The mesh adaptation algorithm and the numerical simulation results discussed in [13]. The computational mesh is refined at the detached shock wave position, near the isosurface with a local Mach number ($M = 1$) and near the recirculation region boundary (Fig. 7a). Figure 7b illustrates the flow structure.



a) adaptive mesh structure b) Mach number field

Fig. 7. Flow around a sphere.

Coarse and fine adaptive hanging node meshes were used for testing (Table 1). Both meshes contain cells of the first and second refinement levels.

Table 1. Computational meshes parameters.

Parameter	Coarse mesh	Fine mesh
Number of nodes	452288	3486628
Number of tetrahedral	224696	1588900
Number of hexahedrons	306713	2463616
Number of prisms	99622	881851
Number of pyramids	9741	54911
Total number of cells	640772	4989278

The parallel program was run on a heterogeneous supercomputer K-100 with asymmetric nodes. Two Intel Xeon X5670 CPUs and three NVIDIA 2050 GPUs are available on one node to run the program.

4.1 MPI Parallelization and Overlap Mode

Figure 8 shows parallel efficiency graphs for a coarse mesh in the GPU mode (3 MPI processes per node). Blue line - results of pure computations without MPI exchange. The minimum parameter value (24 GPU and about 25000 mesh cells per device) is 87%. Here, the CFD kernels execution time increases due to domain decomposition imbalance and duplication of calculations at halo cells. Explicit MPI exchanges (red line) visibly increase the program execution time. For a group of 24 devices, communications are comparable to the computation time. Parallel efficiency drops to 64% and the speedup turns out to be 1.6 times lower than the expected value. In overlap mode, MPI exchanges are successfully hidden behind computations. Parallel efficiency (green line) differs by only 2–5% compared to the results for CFD kernels.

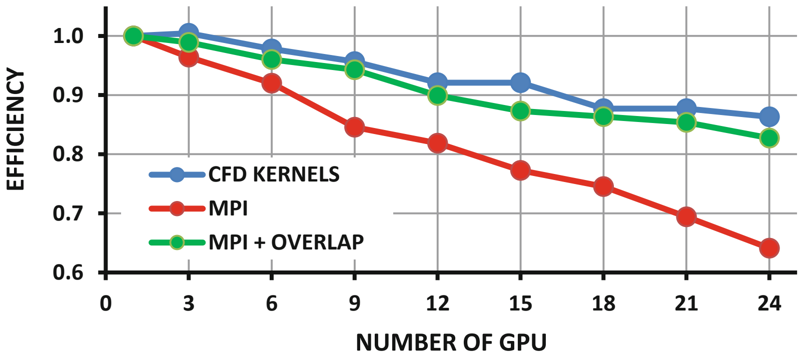


Fig. 8. Parallel efficiency. (Color figure online)

4.2 Heterogeneous Execution

To measure the efficiency of a heterogeneous mode, the values of absolute and relative performance are used. The absolute performance P_N^{mode} on N nodes for specific mode (CPU only, GPU only, heterogeneous CPU + GPU) is equal to the number of RK time integration steps per second. The relative performance R_N^{mode} is calculated as

$$R_N^{mode} = \frac{P_N^{mode}}{P_N^{CPU} + P_N^{GPU}}. \qquad (10)$$

In CPU only mode, each MPI process occupies 3 cores to run OpenMP threads (4 MPI processes per supercomputer node). The configuration for running 6 processes per node in a heterogeneous mode is shown in Fig. 9. Here, MPI processes and OpenMP threads get strong affinity to CPU cores.

Figure 10 shows a graph of relative performance for a coarse mesh. The ratio of values practically does not change for a different number of nodes. The relative heterogeneous performance is about 83%. The difference between R_N^{HET} and $R_N^{CPU} + R_N^{GPU} = 1$ is partly due to the fact that 3 cores are used to control the execution of CFD kernels

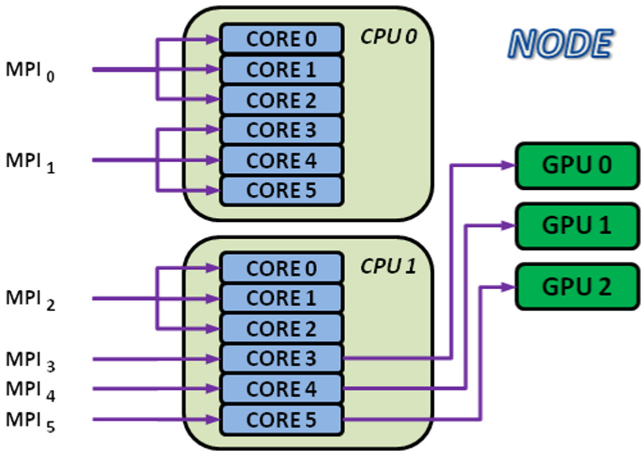


Fig. 9. Program launch configuration.

on devices. In this case, the ideal value is $R_N^{HET} = 0.75R_N^{CPU} + R_N^{GPU} \approx 0.91$. Thus, the real R_N^{HET} differs from the theoretical maximum value by only 8%.

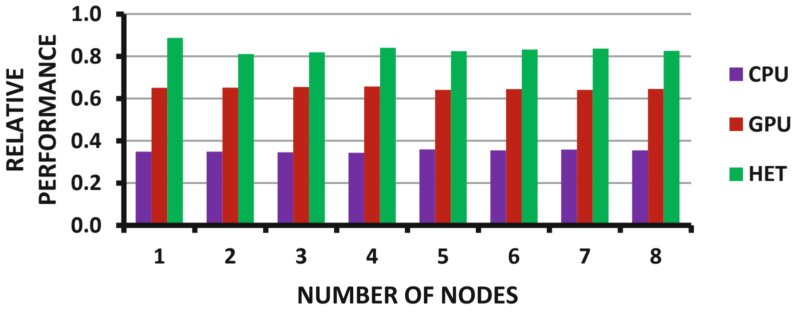


Fig. 10. Relative performance.

4.3 Parallel Program Scaling

The study of the program scaling was carried out for a fine 5 million cells mesh. Figure 11 shows a speedup graph for heterogeneous parallelization (green line). For comparison, there is a graph of ideal linear speedup (red dashed line). The performance ratio of the CPU and the GPU has a fixed value of 1:1.6. The low current efficiency of the CFD algorithm implementation for the GPU is explained by the use of the same parallelism units.

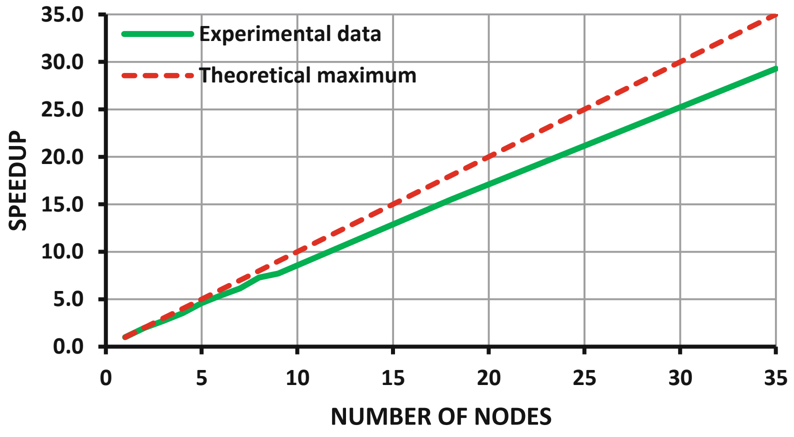


Fig. 11. Speedup for fine mesh. (Color figure online)

The speedup for 35 nodes (70 CPUs, 105 GPUs, 210 processes MPI group size) is 29.3 times, which corresponds to a parallel efficiency of 84%. The absolute performance is $P_N^{HET} \approx 56$ steps per second. That is, it takes 0.0179 s for one time integration step or 0.0045 s for RK stage (include MPI exchange).

5 Conclusions

This paper describes a heterogeneous CFD algorithm for compressible flow simulations on adaptive mixed meshes. This algorithm can be successfully used in computations using static adaptation mesh methods, when the solution analysis and mesh refining routines are not included in the main CFD code. The algorithm software implementation has high scalability and parallel efficiency. In the future, the CUDA CFD kernels will be optimized as described in [14] and dynamic adaptation methods will be added.

Acknowledgments. The work has been funded by the Russian Science Foundation, project 19-11-00299. The results were obtained using the equipment of Shared Resource Center of KIAM RAS (http://ckp.kiam.ru).

References

1. Liou, M.-S., Kao, K.-H.: Progress in grid generation: from chimera to DRAGON grids. In: NASA Technical Reports, 19970031503, pp. 1–32, National Aeronautics and Space Administration (1994)
2. Powell, K., Roe, P., Quirk, J.: Adaptive-mesh algorithms for computational fluid dynamics. In: Hussaini, M.Y., Kumar, A., Salas, M.D. (eds.). Algorithmic Trends in Computational Fluid Dynamics. ICASE/NASA LaRC Series. Springer, New York (1993). https://doi.org/10.1007/978-1-4612-2708-3_18
3. https://www.ansys.com/products/fluids/ansys-fluent

4. Antepara, O., Lehmkuhl, O., Chiva, J., Borrell, R.: Parallel adaptive mesh refinement simulation of the flow around a square cylinder at Re = 22000. Procedia Eng. **61**, 246–250 (2013). https://doi.org/10.1016/j.proeng.2013.08.011
5. Schwing, A., Nompelis, I., Candler, G.: Parallelization of unsteady adaptive mesh refinement for unstructured Navier-Stokes solvers. In: AIAA AVIATION 2014 -7th AIAA Theoretical Fluid Mechanics Conference. American Institute of Aeronautics and Astronautics Inc. (2014). https://doi.org/10.2514/6.2014-3080
6. Toro, E.F.: Riemann Solvers and Numerical Methods for Fluid Dynamics. Springer, Heidelberg (2009). https://doi.org/10.1007/b79761
7. Wada, Y., Liou, M.-S.: A flux splitting scheme with high resolution and robustness for discontinuities. In: AIAA Paper 94-0083
8. Barth, T.J.: Numerical aspects of computing high-Reynolds number flows on unstructured meshes. In: AIAA Paper 91-0721 (1991)
9. Kim, S., Caraeni, D., Makarov, B.: A multidimensional linear reconstruction scheme for arbitrary unstructured grids. In: Technical report. AIAA 16th Computational Fluid Dynamics Conference, Orlando, Florida. American Institute of Aeronautics and Astronautics (2003)
10. Golovchenko, E.N., Kornilina, M.A., Yakobovskiy, M.V.: Algorithms in the parallel partitioning tool GridSpiderPar for large mesh decomposition. In: Proceedings of the 3rd International Conference on Exascale Applications and Software (EASC 2015), pp. 120–125. University of Edinburgh (2015)
11. Schloegel, K., Karypis, G., Kumar, V.: Parallel multilevel algorithms for multi-constraint graph partitioning. In: Bode, A., Ludwig, T., Karl, W., Wismüller, R. (eds.) Euro-Par 2000 Parallel Processing: 6th International Euro-Par Conference Munich, Germany, August 29 – September 1, 2000 Proceedings, pp. 296–310. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44520-X_39
12. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the ACM 24th National Conference, pp. 157–172 (1969)
13. Soukov, S.A.: Adaptive mesh refinement simulations of gas dynamic flows on hybrid meshes. Dokl. Math. **102**, 409–411 (2020). https://doi.org/10.1134/S1064562420050427
14. Gorobets, A., Soukov, S., Bogdanov, P.: Multilevel parallelization for simulating turbulent flows on most kinds of hybrid supercomputers. Comput. Fluids **173**, 171–177 (2018). https://doi.org/10.1016/j.compfluid.2018.03.011