Parallel Mesh Multiplication for Code_Saturne

Pavla Kabelikova, Ales Ronovsky, Vit Vondrak[a]

Dept. of Applied Mathematics, VSB-Technical University of Ostrava, Tr. 17. listopadu 15, 708 00 Ostrava, Czech Republic

Abstract

In this whitepaper, a new mesh multiplication package developed for Code_Saturne is described. The package implements parallel global refinement of hexahedral meshes for Code_Saturne to allow creating meshes with more than 1 billion cells. This enables running Code_Saturne's extremely large CFD simulations on PRACE Tier-0 systems. The effectiveness of the implemented multiplication algorithm is demonstrated on practical examples, which were carried out on CURIE system at CEA.

1. Introduction and Goals of the Project

Code_Saturne is a multi-purpose Computational Fluid Dynamics (CFD) software, which has been developed by Electricité de France Recherche et Développement EDF-R&D since 1997. The code was originally designed for industrial applications and research activities in several fields related to energy production. Code_Saturne has been released as open source in 2007 and is distributed under a GPL licence. During the solution of PRACE-1IP project, Code_Saturne has been selected as an engineering community code and supported by developing teams from several project partners to enable its petascale capabilities.

The main bottleneck to enable petascaling of Code_Saturne is the time required to generate large meshes. This is the case even for relatively modest sizes, e.g. 10 million cells, if the geometry is very complex and boundary layers have to be meshed. Therefore, it is obvious that mesh generation of billion of cell meshes has to be parallel, but no open-source parallel mesh generators are yet available. Therefore, other routes must be followed and the one proposed by Code_Saturne developers deals with parallel global mesh refinement (or mesh multiplication), i.e. an initial mesh of about 100 million cells would be read by Code_Saturne and then each of its cells would be split. This process could be repeated several times in order for the Navier-Stokes solver to run on a several billion cell mesh, while post-processing would be carried out on the initial 100 million cell mesh.

According to Code_Saturne developers' requirements, the main aim of this project was to develop a parallel mesh multiplication package and integrate it to Code_Saturne to extend its capability to generate more than a billion cell meshes. In this whitepaper the basic ingredients of the implemented parallel mesh multiplication package will be described and its performance and scalability tests on a local cluster at VSB-TU Ostrava (ComSio) and on the CURIE system will be presented.

2. Overview of Mesh Multiplication

Mesh multiplication is the process of taking an existing mesh and changing its size, i.e. number and/or shape of the cells, to increase accuracy of the solution. In the case of Code_Saturne, we focused only on the multiplication of regular meshes, where the higher accuracy of the solution is attained refining each cell of the mesh. Since the main goal was to generate more than 1 billion cells meshes in 3D, we considered only meshes that contain hexahedral, tetrahedral, pyramidal and/or prismatic cells.

For an appropriate refinement, several methods are known to divide a single element in 2D as well as in 3D. We have used an approach, where each cell of the domain is divided into smaller cells using original cell vertices and midpoints on its edges. This ensures the global connectivity of vertices in the whole (global) refined mesh. The shape of the new mesh then depends on how the new midpoints are connected. The basic approaches are shown in Fig. 1. This type of refinement retains element types in case of hexahedral, tetrahedral and prismatic cells [1]. Therefore, this refinement produces smoother mesh of the same types of cells. Refining of the pyramidal cell type in the same manner leads to the mesh with pyramidal and tetrahedral cells, i.e. hybrid mesh (but the global vertex connectivity of the mesh is still held).

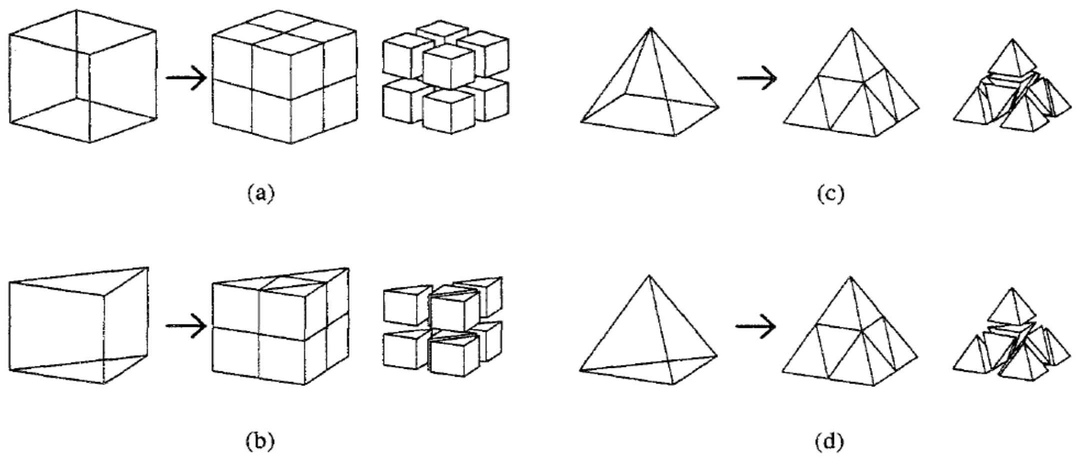

Fig. 1: 3D element refinement. (a) hexahedras, (b) prisms, (c) pyramids, (d) tetrahedras.

In the first step, we have implemented the refinement algorithm for hexahedral cells. Each hexahedral cell is refined into 8 smaller hexahedral cells by dividing in half along all axes (Fig. 1(a)). The step-by-step algorithm for hexahedral-type meshes refinement is described below:

<u>Mesh refinement algorithm (hexahedral mesh):</u>

1. Input: coarse mesh.
2. Create new vertex in the middle of each edge.
3. Create new vertex in the centre of each boundary face.
4. Refine all boundary faces (edge vertex connected to boundary face centre vertex).
5. Create new vertex in the centre of each inner face.
6. Refine all inner faces (edge vertex connected to inner face centre vertex).
7. Create new vertex in the middle of each cell.
8. Refine all cells (face centre vertexes connected to cell middle vertex).
9. Output: refined mesh.

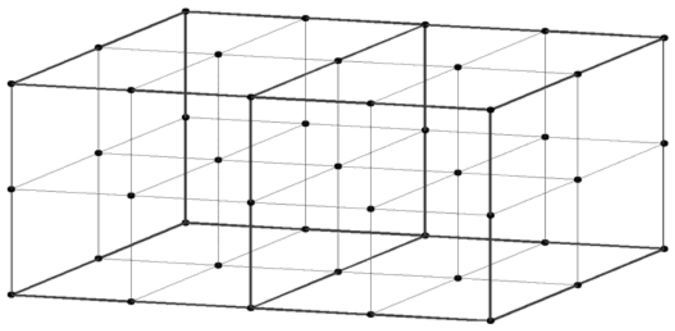

Fig. 2: Refinement of two elements (hexahedra)

3. Parallel Mesh Multiplication for Code_Saturne

The Code_Saturne is the EDF's general purpose Computational Fluid Dynamics (CFD) software. The basic capabilities of Code_Saturne enable the handling of either incompressible or expandable flows with or without heat transfer and turbulence. Dedicated modules are available for specific physics such as radiative heat transfer, combustion (gas, coal, heavy fuel oil, ...), magneto-hydrodynamics, compressible flows, two-phase flows (Euler-Lagrange approach with two-way coupling), or atmospheric flows. Code_Saturne is portable on all Linux and UNIX platforms tested so far (HP-UX, Solaris, Cray, IBM Blue Gene, Tru64, ...). It runs in parallel using MPI on distributed memory machines (clusters, Cray XT, IBM Blue Gene, ...). Developed since 1997 at EDF R&D, it is based on a co-located Finite Volume approach that accepts meshes with any type of cell (tetrahedral, hexahedral, prismatic, pyramidal, polyhedral, ...) and any type of grid structure (unstructured, block structured, hybrid, conforming or with hanging nodes, ...).

A mesh stored in Code_Saturne mesh_t format is used as a coarse mesh input to our parallel mesh multiplication routines (described in the previous section). This mesh is obtained during the Code_Saturne preprocessing and/or partitioning phase. If the mesh is not distributed yet, one of the inner partitioning routines of Code_Saturne is used to distribute it.

The process of mesh multiplication is then executed in parallel on distributed parts of the mesh, i.e. each parallel process takes care of its own part of the mesh, but the faces on the subdomain interface are shared with more processors. At this moment, it is guaranteed that the refined mesh does match on subdomain interfaces for hexahedral elements (and the other regular cell types are in progress).

3.1. Mesh partitioning and mesh distribution

In the Code_Saturne package, several mesh partitioning routines are implemented, as well as a possibility of using some of standard mesh partitioners like Metis, Zoltan, etc. In parallel environment, ParMetis package can be also used.

An important issue is to protect optimal load balancing after the mesh refinement process. The implemented mesh multiplication algorithms for structured meshes protect the load balancing of the original partitioning into subdomains. Therefore, the problem of the optimal load balancing has to be solved applying suitable partitioning tool before the mesh multiplication phase. Thus the final decomposition to subdomains must be done in advance before calling the multiplication routines and any further partitioning must be omitted.

3.2. Mesh analysis and data preparation

In Code_Saturne, various formats of meshes are supported. For our purposes, the NOPO/SIMAIL (INRIA/SIMULOG) format is used. This is a semi-portable Fortran binary file, which handles tetrahedral, prismatic and hexahedral cells in a Cartesian coordinate system. Independently of the used format, all meshes are preprocessed into the mesh_t structure, which contains all necessary information.

As the mesh stored in Code_Saturne mesh_t format is used as a coarse mesh input to our parallel mesh multiplication routines, some additional preprocessing is needed to work with this structure efficiently. At first, the cell types and their numbers, as well as the numbers of the triangular or rectangular faces, have to be computed. The preparation phase of the following computation consists of pre-computation of the memory required for the new structures, computation of the intervals of the new global indexes of the new mesh, and allocation of memory for the new mesh.

Since the Code_Saturne mesh_t structure handles only data structures of the mesh that is not sufficient for the efficient mesh refinement, it was necessary to extend the current mesh storage format by adding vertex-edge and edges-to-face information. Specifically, the second step of our algorithm “Create new vertex on each edge” requires indexation of edges, because the Code_Saturne mesh_t structure contains indexes just for faces, vertexes and cells. For cell refinement and the following indexation of new objects, we also needed to build the face-to-cell list, which was not included in original Code_Saturne structures.

3.3. Subdomain mesh refinement

The subdomain mesh multiplication in Code_Saturne is processed by our multiplication algorithm, which is described in Section 2 “Overview of Mesh Multiplication”. The mesh refinement is executed in parallel on distributed parts of the mesh and it is done using local variables. During the final global re-indexation, special care has to be given to faces on the subdomain interfaces that are shared with other processors. Details will be described later. Very fine meshes can be obtained applying the implemented mesh multiplication algorithm recursively. If required, the next levels of refinement can be processed in the same way as the first level.

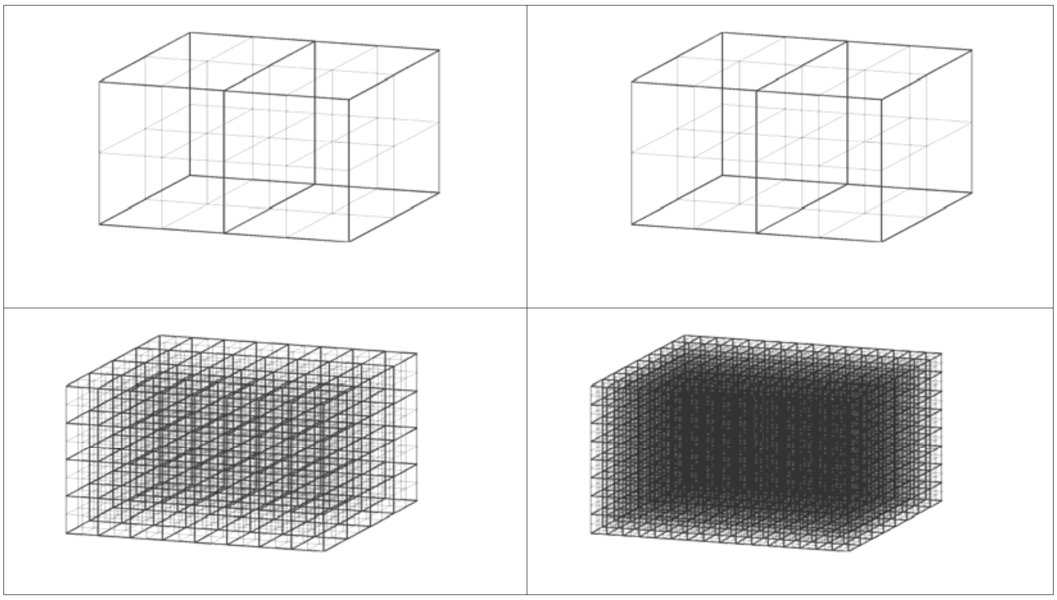
Fig. 3: Multilevel mesh refinement

3.4. Global indexation

Application of the parallel mesh multiplication on a distributed mesh duplicates some surface subdomain entities (vertices, edges and faces) in multiple processes. This implies reconfiguration of their global indexing. This task requires parallel communication between the neighbouring subdomains. Optimization of this inter-subdomain communication uses the knowledge of adjacency of subdomains and some additional information like cells on boundaries and their neighbouring cells in neighbouring subdomains. This information is set-up during Code_Saturne preprocessing and partitioning phase to extract all necessary information.

4. Tests and Results

Performance and scalability tests of the implemented parallel mesh multiplication package have been done on two different test examples on a local cluster at VSB-TU Ostrava (ComSio) and on the CURIE cluster at CEA. The local cluster ComSio is a blade cluster with nodes equipped with 4 CPU cores and 2GB RAM per core. The CURIE cluster has nodes with 16 CPU cores and 4GB of memory per core.

4.1. First test example

As an initial example, a mesh with 700 cells and 1562 vertices was used as the coarse mesh. Applying six levels of refinement, a mesh with more than 180 million cells was obtained in less than 7 seconds (using 28 CPU cores). Table 1 collects all performance tests and demonstrates the scalability of the given example on the ComSio cluster and shows the time needed to mesh refinement (without global indexation). The sixth level of refinement cannot be computed on one core due to lack of memory. The seventh level has too high memory requirements for the Comsio cluster.

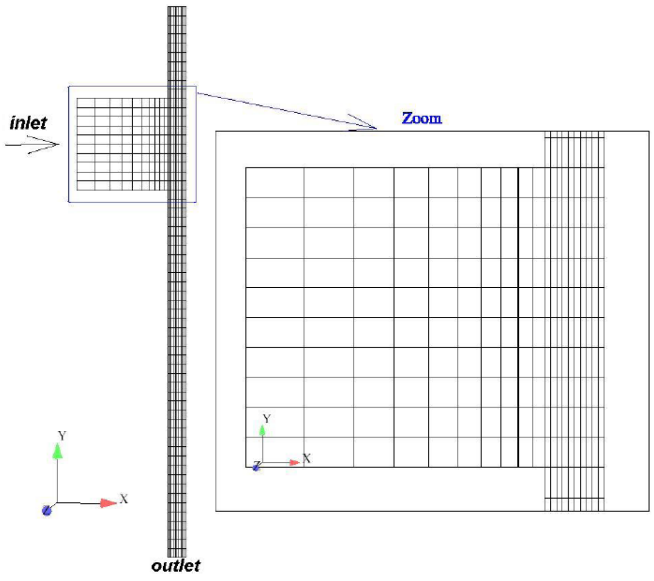

Fig. 4: First test example

Table 1: Performance test on local ComSio cluster

Parameters of given mesh			Number of used cores						
level of refinement	no. of cells	no. of vertices	1 core	2 cores	4 cores	8 cores	16 cores	24 cores	28 cores
			time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]
0	700	1562	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
1	5600	8883	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
2	44800	57605	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
3	358400	408969	0,36	<0.5	<0.5	<0.5	<0.5	<0.1	<0.1
4	~2.87M	~3M	2,88	<0.5	<0.5	<0.5	<0.5	<0.5	<0.5
5	~22.9M	~23,7M	22,75	12,5	6,3	3,16	1,49	1,11	0,85
6	~183,5M	~186,6M	---	157,3	46,8	22,5	12,7	8,23	6,81

Table 2 shows the scalability of the test example on the CURIE cluster. The times in the table show the pure time needed for the mesh refinement. The seventh level of refinement was carried out on 128 cores only due to high memory requirements per 1 core.

Table 2: Performance test on CURIE cluster

Parameters of given mesh			Number of used cores					
level of refinement	no. of cells	no. of vertices	32 cores		64 cores		128 cores	
			time [s]	no. of vertices	time [s]	no. of vertices	time [s]	no. of vertices
0	700	1562	<0.1	72	<0.1	42	<0.1	24
6	183.5M	186.6M	4.12	6.0M	2.5	3.2M	1.2	1.6M
7	1.5B	1.5B	---	---	---	---	8.05	12M

Second test example

In the second example, the mesh corresponds to a stratified flow in a T-junction. There are two inlets, a hot one in the main pipe and a cold one in the vertical nozzle. The volume flow rate is identical in both inlets. It is chosen small enough so that gravity effects are important with respect to inertia forces. Therefore, cold water creeps backwards from the nozzle towards the elbow until the flow reaches a stable stratified state. The mesh used here contains 16320 elements. Again, we can use "arbitrary many" levels of refinement.

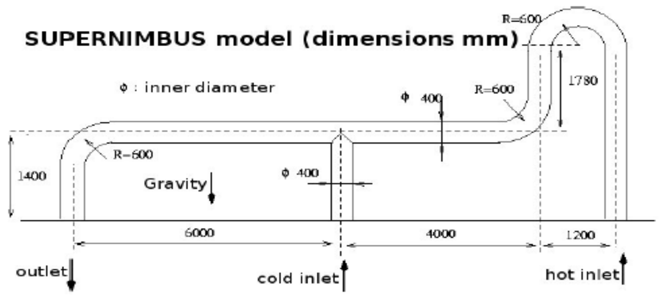

Fig. 5: Second test example

Table 3 presents the scalability of the second example on the CURIE cluster. The times in the table show the pure time needed for the mesh refinement. The sixth level of refinement was carried out just on 128 cores because of the high memory requirements on a single core.

Table 3: Performance test on CURIE cluster

Parameters of given mesh			Number of used cores					
level of refinement	no. of cells	no. of vertices	32 cores		64 cores		128 cores	
			time [s]	no. of vertices	time [s]	no. of vertices	time [s]	no. of vertices
0	16320	18070	<0.1	850	<0.1	550	<0.1	280
3	8.4M	8.5M	<0.2	280k	<0.1	150k	<0.1	76k
4	66.8	66M	1.5	2.2M	<1	1.1M	<1	600k
5	0.5B	0.5B	12	17M	6	8.6M	3.8	4.4M
6	4.3B	4.3B	---	---	---	---	28	34M

5. Conclusions

In our contribution to the Code_Saturne project, we have developed a new package for mesh multiplication for hexahedral meshes. This package has been integrated into Code_Saturne and its functionality has been tested on several numerical examples on two types of clusters. The tests prove good parallel scalability of the presented solution. Using the CURIE cluster, we have obtained a refined mesh with more than 1 billion cells and vertices in less than 30 seconds.

Future improvements of the optimization of the global indexation using other Code_Saturne internal structures and/or methods can lead to more effective computation of very fine meshes in smaller memory and communication requirements.

In a future work, multiplication of other types of meshes has to be implemented to complete the full mesh multiplication functionality. While the multiplication of regular tetrahedral and prismatic meshes is straightforward and can be done very easily, the multiplication of pyramidal or hybrid meshes needs more effort.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources [CURIE, France].

References

[1] M.L.Staten and N.L. Jones; Local refinement of three-dimensional finite element meshes, Engineering with Computers (1997), 13: 165-174

[2] Code_Saturne® : a Finite Volume Code for the Computation of Turbulent Incompressible Flows – Industrial Applications, Archambeau, F. Machitoua, N. and Sakiz, M (2004), International Journal on Finite Volumes, Vol. 1, 2004.

[3] Developing Code_Saturne® for Computing at the Petascale, Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A.G. Sunderland, J.C. Uribe, in Press for Computers and Fluids Journal, Special Edition 2011.

[4] Development of a Two-velocities Hybrid RANS-LES Model and its Application to a Trailing Edge Flow. J. Uribe, N. Jarrin, R. Prosser, D. Laurence, Journal of Flow Turbulence and Combustion (DOI: 10.1007/s10494-010-9263-6), 2010.

[5] A stress-strain lag eddy viscosity model for unsteady mean flow, A.J. Revell, S. Benhamadouche, T.J. Craft, D.R. Laurence, Int. J. Heat Fluid Flow, 2006.

Appendix: Steps of the subdomain mesh refinement algorithm