Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Parallel local mesh refinement for Code_Saturne

Ales Ronovsky[a], Tomas Karasek[a,*], Vit Vondrak[a], Charles Moulinec[b], Yvan Fournier[c]

[a]IT4I, VSB Technical University of Ostrava, Czech Republic
[b]STFC Daresbury Laboratory, United Kingdom
[c]EDF R&D, MFEE, Chatou, France

Abstract

Computational Fluid Dynamics (CFD) is one of the field which can fully utilize the capacity of existing HPC systems. There are many cases either from basic or applied research which are so complex that their numerical simulation with requested accuracy requires very fine representation of the computational domain. To solve certain problems numerical models consisting of hundred billions of cells are necessary. There are several approaches to create such huge meshes. One of them is based on global mesh refinement and is also known as mesh multiplication. This approach was already described in [1, 2]. Global refinement was already implemented into Code_Saturne enhancing its capability in terms of mesh refinement. Meshes with sizes of up to one hundred billion of cells were generated on the fly. Since there are many CFD problems where only local area is of interest (either areas close to boundaries, small geometrical entities or in regions with high gradient of solved quantities), local refinement is another approach for mesh creation. In this paper implementation of parallel local refinement applied to Code_Saturne is described. The bottleneck of local adaptive refinement is that it breaks load balancing of the original mesh and requires a lot of global communications. Strategy to re-partition the mesh before its refinement is a key issue for optimal resource utilization. To minimize the amount of data transferred among cores it is necessary to do most of the communication during the preprocessing step on the coarse mesh before refinement. Local mesh refinement strategy was tested and its scalability and performance within Code_Saturne were analysed. Results are presented in this paper.

1. Introduction

Creating very large meshes of billion cells is a challenge and still very few open-source parallel mesh generators are available nowadays. Such huge meshes cannot be easily stored, transferred and for many CFD solvers it is even very difficult to read them. An alternative is to use mesh multiplication (global refinement) firstly mentioned in [1, 2] to create very large meshes from an initial coarser mesh, already suitable for the CFD solver.

In global refinement, subdivisions of each of the cells of the original mesh into certain number of cells by using midpoints of its edges and centres of its quadrangle faces were used. This approach ensures global connectivity of the new vertices and helps to avoid unnecessary core-to-core communication (see Fig. 1). It also ensures that new coordinates and indices of vertices are computed in the same way for every single sub-domain of the whole mesh. When performing local refinement, cells of interest are split in the same way as for global refinement, but processor load balance is key issue here. It is also important to deal with transition between cells of interest and the surrounding cells.

Code_Saturne is used to demonstrate the performance of the algorithms. It is a multi-purpose CFD software, which has been developed by EDF-R&D since 1997. The code was originally designed for industrial applications and research activities in several scientific domains related to energy production. Code_Saturne has been released as open source in 2007 and is distributed under a GPL licence.

In the next section, details about full hybrid global mesh refinement and achieved results are shortly mentioned. Link between full hybrid refinement as a prerequisite for local adaptive mesh refinement is then discussed. Local mesh refinement could provide a strategy for future exascale computing where very huge meshes will be required for solving of complex CFD problems.

*Corresponding author.
e-mail. tomas.karasek@vsb.cz

2. Full hybrid mesh multiplication and used data-structures

Mesh multiplication is the process of modifying an existing mesh mainly to increase accuracy of the solution. During this process the size of the mesh i.e. the number of elements is increased and their shape may also be changed. In our implementation, we focus on regular mesh multiplication, where the higher accuracy of the solution is obtained by refining each element of the mesh. In case of 3D meshes, those could be created by hexahedral, tetrahedral, pyramidal and/or prismatic types of elements.

To ensure global connectivity of the new elements after refinement and to avoid unnecessary core-to-core communication it is necessary to create global numbering for each element during the preprocessing of the original mesh. Subdivision of cells of the original coarse mesh is done by using midpoints of its edges, its centres of rectangular faces and in case of hexahedral cells by using vertices in the centres of each cell.

2.1. Full hybrid pre-processing and Algorithm

In Code_Saturne, mesh information is handled by the *mesh_t* structure which can only operate with global numbering for vertices, faces and cells of the mesh. This structure of storing information is not sufficient for efficient mesh refinement. Therefore it was necessary to extend the existing mesh storage format so it could handle additional information about the mesh. During the pre-processing step of the algorithm information about the edges had to be gathered and cell to faces connectivity had to be build. Local and global indices of each edge were created. Some more numbering necessary for full hybrid refinement was created for each type of cell (tetrahedra, hexahedras, ...) and for quadrangle faces (border, interior) as well. This information is required to compute the new indices of the refined cells and the new created vertices during the refinement step of the algorithm in order to avoid communication amongst sub-domains, (see Fig. 2).

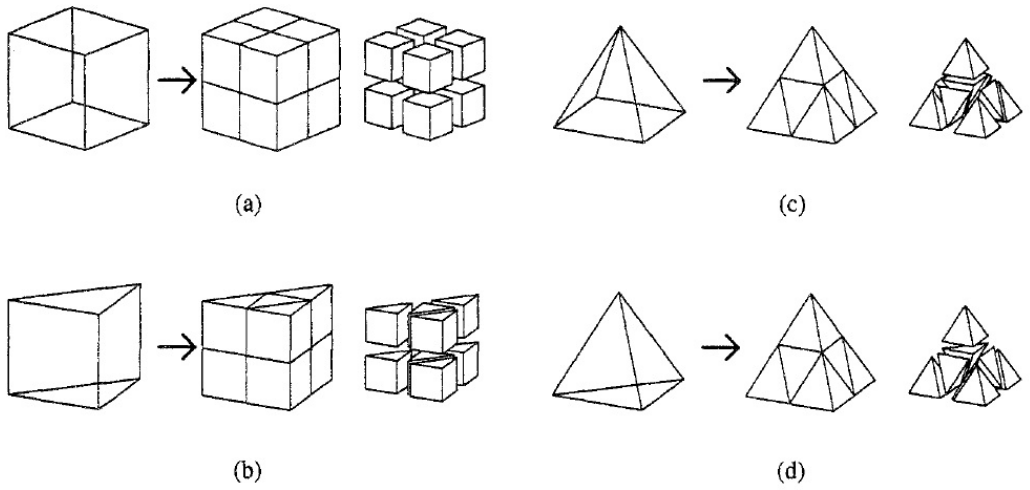

Fig. 1. 3D element refinement. (a) hexahedra, (b) prisms, (c) pyramids, (d) tetrahedra.

3. Tests and results for full hybrid refinement

Performance and scalability tests of the Parallel Mesh Multiplication algorithm have been carried out on different machines with different architectures. It contains ANSELM (Bullx), HECToR (CRAY XE6), Blue Joule and Mira (both IBM Blue Gene/Q). For most tests, the simulation was stopped after few time-steps to save computational resources, e.g. 10 time-steps for the 1.6 , 3.2 billion cell meshes and 5 time-steps for the 105 billion cell mesh. A 26B cell mesh case was also performed only for testing the refinement algorithm scalability, and no time-steps of the Navier-Stokes solver were run.

3.1. Test example

The configuration consists of staggered distributed tube bundles (see Fig. 3 and 4), where Large Eddy Simulations (LES) are performed. The setup is the one experimentally studied by Simonin and Barcouda [8], and the original mesh (or elemental pattern) is made of a whole tube in its centre and four quarter of tubes in the corners. It contains about 13 million cells (2-D cross-section: $100,040$ cells; 3^{rd} direction: 128 layers) and is denoted by 13M. Two other relatively coarse meshes are built, by mesh joining, the former by glueing two elemental patterns together to get an about 26 million cell mesh (26M) and the latter by glueing four elemental patterns to get an about 51 million cell mesh (51M).

We present the results of an implementation of a multilevel mesh multiplication (global refinement) algorithm for multi-billion cell mesh simulations. The mesh multiplication algorithm starts from a coarse mesh and subdivides each of its cells into a given numbers of cells, subdivides the boundaries and computes the

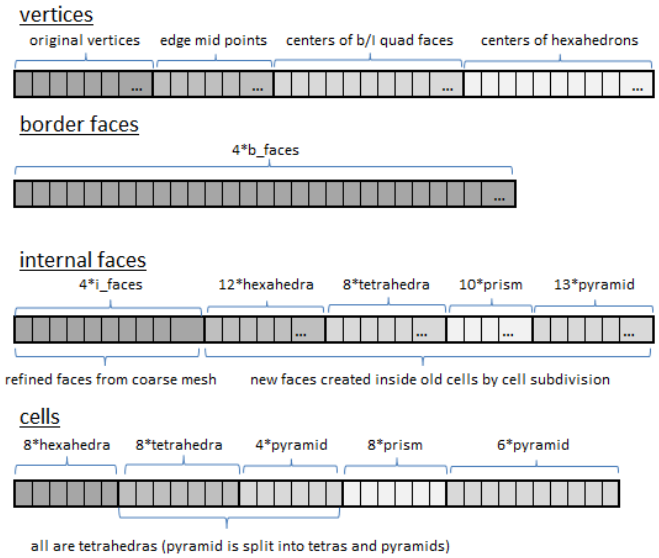

Fig. 2. Elements re-ordering, computing of indices of elements in full refined mesh

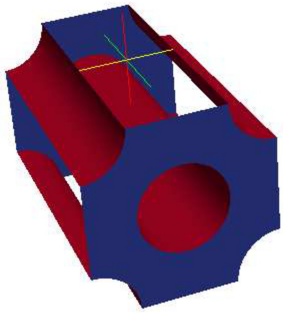

Fig. 3. One single elemental pattern (13M)

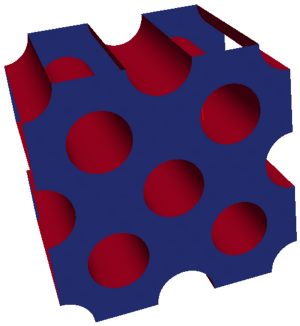

Fig. 4. Four elemental patterns joined (51M)

global indices of the new elements. There is no limitation for the number of refinements to be computed by the algorithm, apart from the memory available on the machine. Up to 4 levels of refinement are used in the tests showed here. Meshes containing about 6.6 , 13 and even 105 billion cells (6.6B, 13B and 105B respectively) were generated on the fly using this technique. Some timings obtained for the BlueJoule machine are presented in Tables 1, 2 and more results obtained for the 105B case on Mira are shown in Table 3.

Table 1. Performance for two elemental pattern joined (Blue Joule) up to 1.6 billion with 10 time steps computed

Two elemental patterns: 26M $\rightarrow$ 1.6B				
no. mpi tasks	ref. levels	mesh refinement (s)	avg. time / step (s)	total computation (s)
16384	2	3.35	96	1114
32768	2	2.79	55	633
65536	2	2.7	26	511

4. Adaptive mesh generation

4.1. Introduction

The refinement problem can be described as a technique involving insertion of additional vertices to the mesh in order to produce meshes leading to higher accuracy of the solution. Adaptive mesh refinement is used to effectively solve numerical systems of partial differential equations while saving computational resources. Instead of uniformly processing a mesh, which could be created by the full refinement algorithm, we place more nodes to the areas, where we anticipate large local error of the solution or which are of some importance for the final solution.

Table 2. Performance of four elemental patterns joined (Blue Joule) for 3.2 billion cells with 10 time steps and 26 billion cells with no time step computed

Four elemental patterns: 52M → 3.2B, 52M → 26B				
no. mpi tasks	ref. levels	mesh refinement (s)	avg. time / step (s)	total computation (s)
16384	2	4.13	191	2109
32768	2	3.55	95	1110
65536	2	2.79	68	1080
16384	3	23.7	-	-
32768	3	14.5	-	-

Table 3. Performance for two elemental patterns (Mira) up to 105 billion cells

Two elemental patterns: 26M → 105B				
no. mpi tasks	ref. levels	mesh refinement (s)	avg. time / step (s)	total computation (s)
262144	4	8.39	701	3797
524288	4	8.92	376	2273

Different methods exist to perform mesh adaptation, as for example described in [9]. We used a technique, where the mesh is modified to locally increase its mesh size depending on the class of CFD problem used and the type of domain. The new mesh is built according to a required mesh size and precision. This is a very effective process because the mesh size is well adapted to the particular problem and required accuracy of the solution.

In our implementation we are working with solvers based on finite volume discretisation and using tetrahedral cell meshes. The local refinement involves two main tasks: Subdivision of the target tetrahedra and transition between zones with different levels of refinement.

The technique we are using is sometimes called h-method or h-refinement (see [10] or [11]). There are several approaches for subdividing tetrahedra, see [12] for bisection or [13] for 8-Tetrahedra Longest Edge partition. Another type of tetrahedral refinement widely used in numerical computations is based on the standard partition. The original tetrahedron is divided into eight sub-tetrahedra by using midpoints of its edges. It is done by cutting four corners by planes through the midpoints of the edges and subdividing the remaining octahedron into 4 tetrahedra by selecting one of the three possible internal diagonals. We are using standard partition as a template for a full refined tetrahedron (tetra-6), see Fig. 5.

4.2. Error cell detection

During the preprocessing stage of the initial mesh, the solver needs to compute an error indicator to tag cells which should be refined. There are many methods used for error indicators such as methods based on geometry of the domain, a-priory or a-posteriori error estimators for a particular solver and used method. The objective of the work presented in this paper was not to create an algorithm for error detection. Consequently, we are using for our testing purposes a simple selection method, where we force the algorithm to refine some particular cells of our choice. This could of course be changed in the future, using any method returning an error indicator based on any known estimation.

4.3. Algorithm basics and refinement templates

The refinement process starts with tagging all the elements of level#n based on an error criterion. During this process all the elements are marked, either as error elements (elements to be refined), transition elements (elements sharing an edge with an error element) or as elements which should not be refined. Transition and error elements are then assign to level #n+1. Subdividing transition elements lead to splitting theirs angle and this could lead to a damageable degradation of the mesh quality in the worse case scenarios. To avoid splitting any angle in the initial mesh more than once for more levels of local refinement, we are using nodal neighbourhood. It means that in each level of adaptive refinement all the cells connected to the error elements by nodes are fully refined too. Transition elements are connected to nodal neighbours, not directly to error elements. For refining the elements of the mesh we use mid-edge subdivision. Each triangle is split into four homothetic triangles, each tetrahedron is split into 4 homothetic elements and 4 non-homothetic ones. Special point must be made regarding the transition elements. They need to ensure a conformal transition between two different levels of refinement. In 2D, a transition element could be only a triangle with a single hanging node.

For that it is necessary to connect the pending node to the opposite vertex to split the triangle into two. In 3D case, the transition element could be tetrahedron with a single edge to be split (tetra-1), two opposite edges (tetra-2) or a single face (tetra-3). This needs to be done by using different templates. A single edge case is split into two tetrahedra, two opposite edges and single face are split into four tetrahedra, see Fig.4.3.

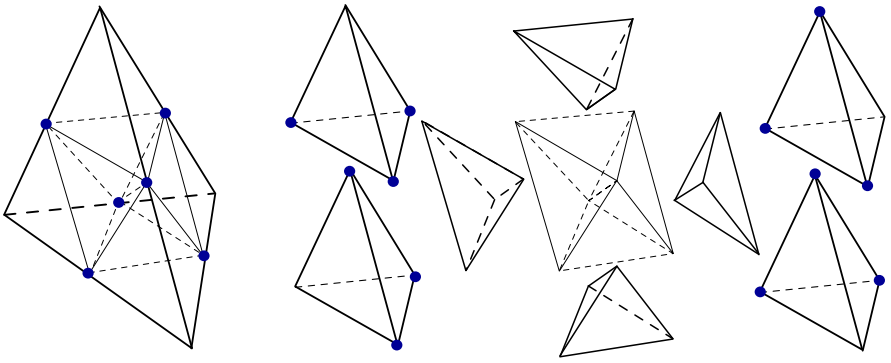

Fig. 5. 3D tetrahedra templates

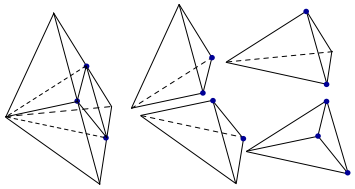

Fig. 6. Single face template

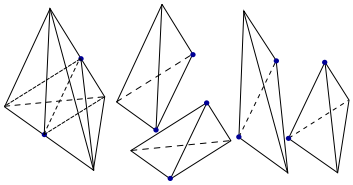

Fig. 7. Two edges template

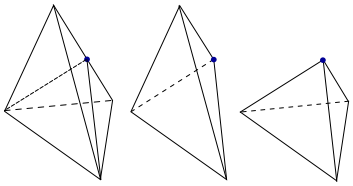

Fig. 8. Single edge template

To identify transition elements, we mark all the edges of elements that could be refined (error elements) as edges to refine (error edges). To ensure best quality after the refinement we are using templates which refine only tetrahedra with faces with one or all error edges. At this point, we need to investigate, if there is no element having a face with two error edges. If yes, we need to mark this element also as an error one and repeat this process until there are only correct elements. At the end of this step, we identify transition elements, which share single edge, two edges or a face with any nodal neighbour. The next step of the algorithm is to refine all the elements using a particular template. The final mesh is created by adding all the unrefined elements and all the children of the transition elements. All the children of the refined and of the nodal neighbours are investigated in the next level of the adaptive mesh generation in the same way until the finest level is reached, where the rest of the elements are added as well.

4.4. Extended data structures

As we mentioned earlier for full hybrid refinement, it was necessary to extract some more information about the mesh and to augment the original *cs_mesh_t* structure with more information. For adaptive mesh refinement we had to augment this structure even further. For huge meshes addressing exascale problems using thousands or even millions of processors, it is necessary to perform all the communication on the coarser mesh with much smaller amount of data to transfer in order for every single processor to work independently. Prior to this, some preprocessing has to be performed. A new structure called *cs_mesh_adapt_t* is created, containing a lot of new information about the mesh. In the first step, we create edges in the same way as for full hybrid refinement. Afterwards, we need to identify the error cells and mark all cells, faces and edges as ones to be either refined or kept. We need to create local and global numbering for all types of those elements, which is then used for unique index computation of new elements (cells, faces and vertices), as seen Fig. 9.

Adaptive tetrahedral refinement algorithm on all processors:

1. Input: coarse mesh.
2. Pre-processing:
   (a) Create local/global numbering:
      - Basic: vertices, cells, interior and border faces.
      - Edges.
   (b) Create face to edge connectivity.
   (c) Define cells.
3. Determine error cells:
   (a) Mark all error cells.
   (b) Mark all error faces and edges.
   (c) Mark all transition cells, faces.
   (d) Create local/global numbering:
      - Error edges, faces, cells.
      - Non-error edges, faces, cells.
      - Transition cells, faces.
4. All non-refined faces and cells from coarse mesh are copied to new mesh and re-indexed.
5. Refinement:
   (a) Create a new vertex in the middle of each error edge.
   (b) Refine all the error faces.
   (c) Refine all the transition faces.
   (d) All new faces inherit family and group from parent.
6. Cell refinement (of each single error and transition cell):
   (a) Preparation:
      - Order faces of the cell to ensure positiveness of normal vectors.
      - Prepare indices of vertices.
7. Cell subdivision:
   (a) Refine the error cells.
      - Refine error tetrahedra by standard template.
      - Refine transition tetrahedra by particular template.
   (b) Create new interior faces.
   (c) Assign proper face to cell connectivity to each new face and cell.
8. Output: refined mesh.

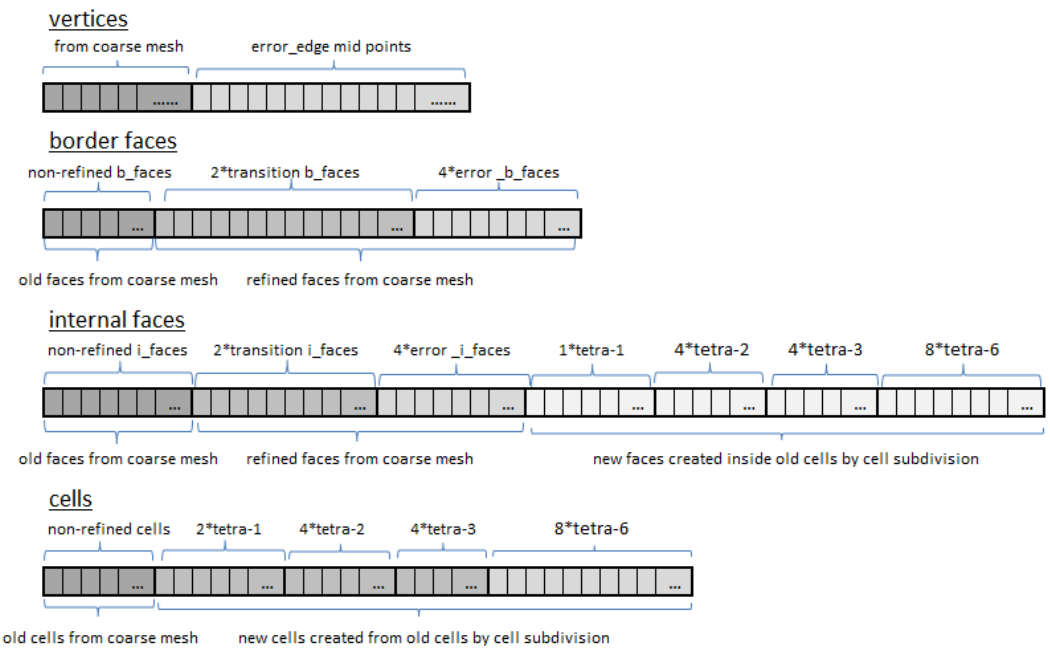

Fig. 9. Elements re-ordering, computing of indices of elements in locally refined tetrahedral mesh

Table 4. Performance of the cubic cavity example on ANSELM:

Adaptive cubic cavity case: 14M → 16M				
no. mpi tasks	partitioning	mesh refinement (s)	avg. time step (s)	total computation (s)
128	44.7	29.9	105	1513
256	63	41.1	85	1231
512	101	55.3	165	1852
1024	179	70.8	210	2543
Full refinement cubic cavity case: 14M → 111M				
no. mpi tasks	partitioning	mesh refinement (s)	avg. time step (s)	total computation (s)
128	37.1	8.27	701	7386
256	48.2	9.1	405	4904
512	73	10.9	297	3125
1024	177	29.8	583	6520
Adaptive cubic cavity case: 111M → 126M				
no. mpi tasks	partitioning	mesh refinement (s)	avg. time step (s)	total computation (s)
128	71	65.8	712	7623
256	81.5	80.2	426	5120
512	116	96.5	321	3542
1024	186	127	605	7035

5. Tests and results for adaptive refinement

Performance and scalability tests of the Parallel Adaptive Mesh Multiplication algorithm have been carried out on ANSELM (Table 4). The test case here simulates the laminar flow in a lid-driven cavity (cubic box with a horizontal constant velocity imposed at the top wall). The original mesh was created from a unstructured tetrahedral mesh using full mesh multiplication algorithm from Section 2. The mesh after one level of uniform refinement has 14 million cells (14M), 111 million cells (111M) after two levels. A 14M mesh was locally refined after single step of adaptive refinement to 16M, 111M mesh to 126M. Non-weighted partitioning was performed by Morton space-filling curve algorithm. Simulation was stopped after 10 time-steps. For better comparison of effectiveness of local adaptive mesh refinement is the last case refined using full mesh refinement from 14M to 111M. The accuracy of solutions of 14M→16M case and 14M → 111M was the same on the cells of importance.

6. Summary

It should be noted that the scalability of the whole process is strongly dependent on n.cells per core ratio and total number of time steps computed. Partitioning of the mesh and local mesh refinement need lot of global communication, but the time spent for preprocessing the mesh is nearly equal to time of single time step computation. So if we are computing certain number of time steps on the same locally refined mesh it is very efficient to use this algorithm instead of full refined mesh, where we need to have much bigger mesh for the same accuracy.

The time spent in the algorithms used in Code_Saturne for both full uniform and local mesh multiplication have proven to be modest in comparison with the time of whole computation (at most a couple of minutes compared to hours or days).

7. Conclusion

In our contribution to the Code_Saturne project, we enhanced capability of mesh multiplication package to be able to create full hybrid meshes up to hundreds of billion cells and adaptive tetrahedral meshes with up to hundreds of million cells. More testing will be performed on other architectures and on different and/or more complex (hybrid) meshes.

To be able to work with full hybrid meshes and to fully utilize all possibilities of mesh multiplication package in Code_Saturne it will be necessary to implement other templates into local adaptive mesh refinement.

Local mesh refinement could provide strategy for future exascale computing where very huge meshes will be required for solving of complex CFD problems.

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-312763. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. The authors also thank the Hartree Centre for using their resources.

This publication was also supported by the Projects of major infrastructures for research, development and innovation of Ministry of Education, Youth and Sports (LM2011033), the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Parallel Mesh Multiplication and its Implementation in Code_Saturne. A. Ronovsky, P. Kabelikova, V. Vondrak, C. Moulinec, Civil-Comp Proceedings ISSN 1759-3433. (doi:10.4203/ccp.101.11)
2. P. Kabelikova, A. Ronovsky, V. Vondrak; Parallel Mesh Multiplication for Code_Saturne, PRACE project (FP7/2007-2013)
3. C. Moulinec; M. J. B. M. Pourqui; B. J. Boersma; T. Buchal; F. T. M. Nieuwstadt Direct Numerical Simulation on a Cartesian Mesh of the Flow through a Tube Bundle. International Journal of Computational Fluid Dynamics, Volume 18, Issue 1 January 2004 , pp. 1 - 14
4. M.L.Staten and N.L. Jones; Local refinement of three-dimensional finite element meshes, Engineering with Computers (1997), 13: 165- 174
5. 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.
6. 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.
7. Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. G. Sunderland and J. C. Uribe. Comp. & Fluids, **45**, Issue 1, (2011), pp. 103-108.
8. O. Simonin and M. Barcouda. 4^{th} Int. Symp. on Apps of Laser Anemometry to Fluid Mechanics. (1988).
9. P.J. Frey, P.L. George, Mesh Generation. Applications to Finite Element Methods, London, ISTE, 2008.
10. R.E. Bank, PLTMG, A Software Package for Solving Elliptic Partial Differential Equations, User Guide 6.0, SIAM, Philadelphia, 1990.
11. M.C. Rivara, Local modification of meshes for adaptive and/or multigrid finite element methods, J. Comput. Appl. Math. 36 (1991) 79?9.
12. J. Bey, Simplical grid refinement: on Freudenthal's algorithm and the optimal number of congruence classes, Numer. Math. 85:1 (2000), 1-29
13. A. Plaza and G.F.Carey, Local refinement of simplicial grids based on the skeleton, App. Num. Math. 32 (2000), 195-218
14. S.H. Lo, D. Wu, K.Y. Sze, Adaptive meshing and analysis using transitional quadrilateral and hexahedral elements, Finite Elem. Anal. Des. 46 (2010) 2-6.
15. G. Nicolas, F. Arnoux-Guisse, O. Bonnin, Adaptive meshing for 3D finite element software, in: Proceedings of the IX international conference on finite elements in fluids, new trends and applications, Venezia, Italy, October , 1995, pp. 15-0.
16. M.T.Jones and P.E.Plassmann, Parallel algorithms for adaptive mesh refinement, SIAM J. Sci. Comp. 18(1997), 686-708