On Parallelization of the OpenFOAM-Based Solver for the Heat Transfer in Electrical Power Cables

Raimondas Čiegis, Vadimas Starikovičius, and Andrej Bugajev

Vilnius Gediminas Technical University,
Saulėtekio al. 11, LT-10223, Vilnius, Lithuania
{raimondas.ciegis,vadimas.starikovicius,andrej.bugajev}@vgtu.lt

Abstract. In this work, we study the parallel performance of OpenFOAM-based solver for heat conduction in electrical power cables. The 2D benchmark problem is used for our tests. The parallelization approach used in OpenFOAM-based solver is described and a basic scalability analysis is done. Results of computational experiments on a cluster of multicore computers are presented and the parallel efficiency and scalability of the solver are analyzed.

Keywords: OpenFOAM, parallel algorithms, domain decomposition, MPI, multicore.

1 Introduction

The knowledge of heat generation and distribution in and around the high-voltage electrical cables is necessary to optimize the design and exploitation of electricity transferring infrastructure. Engineers are interested in maximum allowable current in different conditions, optimal cable parameters, cable life expectancy estimations and many other engineering factors.

Presently applicable IEC standards for the design and installation of electrical power cables are often based on the analytical and heuristic formulas.Obviously, these formulas cannot accurately account for the various conditions under which the cables are actually installed and used. They estimate the cable's current-carrying capacity (so-called *ampacity*) with significant margins to stay on the safe side [3]. The safety margins can be quite large and result in 50-70% usage of actual resources. A more accurate mathematical modelling is needed to meet the latest technical and economical requirements and to elaborate new, improved, cost-effective design rules and standards.

When we need to deal with mathematical models for the heat transfer in various media (metals, insulators, soil, water, air) and non-trivial geometries, only the means of parallel computing technologies can allow us to get results in an adequate time. To solve numerically selected models, we develop our numerical solvers using the OpenFOAM package [4]. OpenFOAM is a free, open source CFD software package. It has an extensive set of standard solvers for popular

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part I, LNCS 8805, pp. 1–11, 2014.

CFD applications. It also allows us to implement our own models, numerical schemes and algorithms, utilizing the rich set of OpenFOAM capabilities [5]. Adapting OpenFOAM library to specific applications still requires theoretical analysis of selected algorithms and nontrivial selection of optimal data structures for the implementation of required algorithms. Examples of such projects are described in [1, 2].

The important consequence of this software development approach is that numerical solvers can automatically exploit the parallel computing capabilities already available in the OpenFOAM package. A detailed analysis on implementation of some types of parallel algorithms for GPU processors is done in [6].

Scalability and performance of parallel OpenFOAM solvers based on MPI for various applications are investigated in [9, 10]. Computational experiments are done on homogeneous distributed parallel platforms with up to 1024 cores. It is noted in [10] that the scalability and efficiency of parallel OpenFOAM solvers is not very well understood for many applications when executed on massively parallel systems. An extensive experimental analysis of OpenFOAM selected applications is done in Prace project. A few CFD applications with different multi-physics models are approximated by FVM on mainly fully structural 3D meshes. Mesh partition is done by using Simple and Scotch tools. The presented experimental results are showing a good OpenFOAM scaling and efficiency performance on IBM Blue Gene Q and Hewlett Packard C7000 parallel systems up to 2048- 4096 cores. It is noted that such results are expected when balancing between computation, message passing and I/O work is good.

In this work, we study and analyze the parallel performance of OpenFOAM-based solver for heat conduction in electrical power cables. The main goal is to consider the scalability and efficiency of the developed parallel solver in the case when the parallel system is not big, but it consists of non homogeneous multicore nodes. The mesh is adaptive and it is partitioned by using Scotch method. Then load balancing techniques must be used in order to optimize the parallel efficiency of the solver. The second aim is to investigate the sensitivity of parallel preconditioners with respect to the number of processes.

In Section 2, we describe the benchmark problem used for all numerical tests. In Section 3, we describe our OpenFOAM-based solver and discuss the parallelization approach employed in the OpenFOAM package. The theoretical scalability analysis of the parallel algorithm is presented in Section 4 . In Section 5, we present and analyze the obtained results on parallel efficiency and scalability of the solver. Finally, some conclusions are drawn in Section 6.

2 Benchmark Problem

As a benchmark problem in this research we solve the heat conduction problem for electrical power cables directly buried in the soil. It is also assumed that the thermo-physical properties of the soil remain constant, i.e. the moisture

transfer in the soil is not considered. Such a simplified problem is described by the following well-known mathematical model:

$$\begin{cases} c\rho\dfrac{\partial T}{\partial t} = \nabla\cdot(\lambda\nabla T) + q, & t \in [0, t_{max}], \boldsymbol{x} \in \Omega, \\ T(\boldsymbol{x}, 0) = T_b, & \boldsymbol{x} \in \Omega, \\ T(\boldsymbol{x}, t) = T_b, & \boldsymbol{x} \in \partial\Omega, \\ T, \lambda\nabla T \ \text{ are continuous}, & \boldsymbol{x} \in \Omega, \end{cases} \tag{1}$$

where $T(\boldsymbol{x}, t)$ is the temperature, $c(\boldsymbol{x}) > 0$ is the specific heat capacity, $\rho(\boldsymbol{x}) > 0$ is the mass density, $\lambda(\boldsymbol{x}) > 0$ is the heat conductivity coefficient, $q(\boldsymbol{x}, t, T)$ is the heat source function due to power losses, T_b is the initial and boundary temperature. Coefficients λ, c, ρ are discontinuous functions. Their values can vary between metallic conductor, insulators and soil by several orders of magnitude [3].

In this work, we have used 2D geometry for our benchmark problem. Three cables are buried in the soil as shown in Figure 1. The red area is metallic conductor, the blue area is an insulator and the gray area marks the soil.

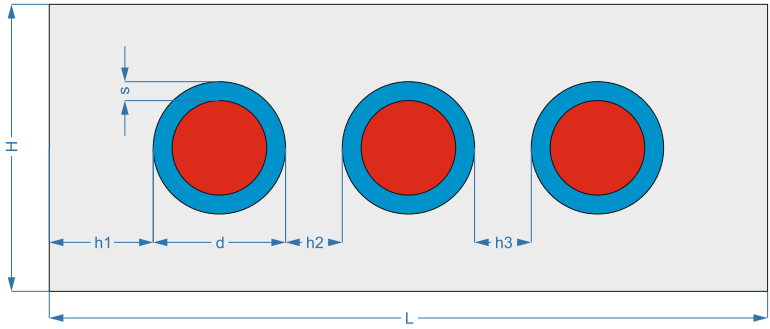

Fig. 1. 2D geometry of benchmark problem: three cables in the soil

OpenFOAM (Open source Field Operation And Manipulation) [4] is a C++ toolbox (library) for the development of customized numerical solvers for partial differential equations (PDEs). For benchmark problem (1) we obtain a numerical solver by a modification of the standard *laplacianFoam* solver, adding variable problem coefficients. OpenFOAM uses the Finite Volume Method (FVM) with co-located arrangement of unknowns [5].

Two important sub-tasks should be solved accurately for this type of approximations. First, the exact fluxes of a solution are orthogonal to the boundary of finite volumes, thus for numerical fluxes this property must be approximated accurately. In our solver this problem is solved by using a proper Delaunay triangulation of the domain. We note that OpenFOAM tool proposes iterative interpolation type orthogonalization techniques. Second, a proper interpolation

should be used for definition of discontinuous coefficients λ in Laplacian term, namely *harmonic*.

Then for the 2D benchmark problem (1) by using Delaunay type triangulation we obtain FVM discretization with the four point stencil. In 3D case the uniform mesh is applied in the additional third dimension and the three-point stencil is used to approximate the fluxes in this direction. Resulting systems of linear equations with symmetric matrices are solved by preconditioned conjugate gradient method with the diagonal Incomplete-Cholesky (DIC) preconditioner.

3 Parallel OpenFOAM-Based Solver

Parallelization in OpenFOAM is robust and implemented at a low level using MPI library. Solvers are built using high level objects and, in general, don't require any parallel-specific coding. They will run in parallel automatically. Thus there is no need for users to implement standard steps of any parallel code: decomposition of the problem into subproblems, distribution of these tasks among different processes, implementation of data communication methods. A drawback of such automatic tools is that the user has very limited possibilities to modify the generated parallel algorithm if the efficiency of the OpenFOAM parallel code is not sufficient.

OpenFOAM employs a common approach for parallelization of numerical algorithms – domain decomposition. The mesh and its associated fields are partitioned into sub-domains, which are allocated to different processes. Parallel computation of the proposed finite FVM algorithm requires two types of communication: local communications between neighboring processes for approximation of the Laplacian term on the given stencil and global communications between all processes for computation of scalar products in DIC iterative method.

OpenFOAM employs a zero-halo layer approach [6], which considers cell edges on sub-domain boundaries as boundary and applies a special kind of boundary condition.

OpenFOAM supports four methods of domain decomposition, which decompose the data into non-overlapping sub-domains: simple, hierarchical, scotch and manual [4]. In all parallel tests the mesh is partitioned by using Scotch library [7]. Scotch is a library for graph and mesh partitioning, similar to well-known Metis library. It requires no geometric input from the user and attempts to minimize the number of boundary edges between sub-domains. The user can specify the weights of the sub-domains, what can be useful on heterogeneous clusters of parallel computers with different performance of processors. We will use this possibility in our computational experiments.

4 Scalability Analysis of the Parallel Algorithm

In this paper we are interested to investigate the efficiency of the parallel algorithm generated by the OpenFOAM tool with respect to load balancing and data communication costs. Thus in all numerical tests (and in the scalability

analysis) we compute 10 time steps with the fixed constant number of iterations for solving systems of linear equations – 1000. In this way, we ensure that the same amount of work is done in all parallel tests, despite the possible differences in convergence due to parallel preconditioning and different roundoff errors, including data communication subroutines. The influence of mesh partitioning on parallel preconditioners is investigated in the last subsection of computational experiments.

Let us estimate the costs of the sequential algorithm to compute a solution at one time step as

$$W = (c_1 + 1000c_2)J,$$

where J is the total number of finite volumes in the mesh, c_1 estimates the costs of computation of the coefficients of the discrete system, c_2 estimates the costs of one DIC iteration.

Next let us estimate the complexity of the proposed parallel algorithm. For simplicity of theoretical scalability analysis let us assume that p homogeneous processes are used in computations. Then computation costs of the parallel algorithm can be estimated as

$$T_p^1 = (c_1 + 1000c_2)\lceil J/p \rceil.$$

Two assumptions have been used in derivation of this estimate. First, we have assumed that a perfect load balancing of sizes of partitioned mesh parts is achieved, this assumption usually is very accurately satisfied for meshes partitioned by Metis or Scotch libraries. Second, we are not taking into account that $c_1 = c_1(p)$, $c_2 = c_2(p)$ may depend on p and they can decrease for $p > 1$ due to a better cashing of smaller size discrete subproblems. In fact, such a behaviour will be illustrated by results of computational experiments. In the theoretical scalability analysis we are considering the worst case scenario.

Next we will estimate costs of communication among processors. As was stated above the implementation of the given parallel algorithm requires local send/receive of data between neighbour processes and global communication in computation of scalar products for Krylov type iterations. We assume that the largest number of data items sent between neighbour processes can be estimated as $c_3\sqrt{J}$ and let M be the largest number of neighbours for some process. Then the communication costs can be estimated as [8]

$$T_p^2 = 1000\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big].$$

Here α denotes the message startup time and β is the time required to send one element of data. Coefficients $1 \le r(M) \le M$ and $\log p \le R(p) \le p$ define the parallel efficiency of local data exchange and global reduce operations. The values of $r(M)$ and $R(p)$ depend on the implementation of MPI functions and on interconnection network of a parallel computer. For example, for a simple implementation of `MPI_ALLREDUCE` function, when all processors send their local values to the master processor, which accumulates results and broadcasts the sum to all processors, $R(p) = p$. On the 2D mesh network this function can be implemented with $R(p) = c\sqrt{p}$.

Thus the total complexity of the parallel algorithm is equal to

$$T_p = (c_1 + 1000c_2)\lceil J/p \rceil + 1000\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big].$$

The scalability analysis of any parallel algorithm finds the rate at which the size of the sequential algorithm W needs to grow up with respect to the number of processes p in order to maintain a fixed efficiency of the parallel algorithm $E_p = W/(pT_p)$. For a given efficiency E the isoefficiency function $W = g(p, E)$ is defined by the implicit equation [8]:

$$W = \frac{E}{1-E}\, H(p, W)\,. \tag{2}$$

The total overhead of the proposed parallel algorithm is given by

$$\begin{aligned} H(p, W) &:= pT_p - W \\ &= (c_1 + 1000c_2)\big(p\lceil J/p \rceil - J\big) + 1000p\big[r(M)(\alpha + \beta c_3\sqrt{J}) + R(p)\big]. \end{aligned}$$

Let us assume that the effects of load disbalance and start-up time of communications are negligible. Then it follows from (2) that asymptotical isoefficiency functions due to local and global communications both have the same order $W = O(p^2)$.

Table 1. Analysis of the mesh decomposition algorithm: M_p denotes the largest number of neighbours, N_p denotes the maximum number of elements communicated between two processes and NT_p denotes the largest total number of elements sent by some process

	$J = 32000$	$J = 128000$	$J = 512000$	$J = 1018488$	$J = 2048000$
M_2	1	1	1	1	1
N_2	208	442	1386	3011	4854
M_4	3	2	3	3	3
N_4	148	466	1038	2383	3057
NT_4	342	826	2348	4023	6752
M_8	5	5	7	7	7
N_8	279	279	836	1224	2395
NT_8	681	681	2432	3851	5915

In Table 1 we present a basic information on the quality of the mesh decomposition algorithm. The load balancing of sizes of subproblems was very close to optimal, thus we restrict to analysis of data communicated among processes. Here M_p denotes the largest number of neighbours for some process, N_p denotes the maximum number of elements communicated between two processes and NT_p denotes the largest total number of elements sent by some process to its neighbours.

The results in Table 1 show that $O(\sqrt{J})$ is also a realistic estimate of the total number of elements sent by one process to its neighbours and the dependence of this number on p is very weak.

5 Parallel Performance Tests and Analysis

The computations were done on the Vilkas cluster at Vilnius Gediminas technical university. It consists of eight Intel Quad i7-860 processors with 4 cores (2.80 GHz) per node and eight Intel Quad Q6600 processors with 4 cores (2.4 GHz) per node. They are interconnected via Gigabit Smart Switch (http://vilkas.vgtu.lt). We also note that the sub-cluster of Intel Quad i7-860 processors is not fully homogeneous. Thus the Vilkas cluster is quite heterogeneous and therefore additional weighted load balancing is included into mesh distribution step.

In Table 2 we present CPU times of computational experiments with different nodes of the cluster. Here $i7$-x and q-x denote the x-th Intel Quad i7-860 and Intel Quad Q6600 processor, respectively. The number of iterations for solving systems of linear equations by using DIC preconditioner is fixed to 1000. In the case of Quad Q6600 nodes we have presented results only for the fastest and slowest nodes.

Table 2. CPU times of the sequential algorithm for different sizes of the problem and different processors of Vilkas cluster: $i7$-x and q-x denote the x-th Intel Quad i7-860 and Q6600 processor, respectively

	$J = 128000$	$J = 254892$	$J = 512000$	$J = 1018488$	$J = 2048000$
$i7$-0	36.7	82.7	201.9	415.9	909.2
$i7$-1	36.6	82.9	202.0	415.5	893.3
$i7$-2	36.7	82.4	198.6	413.0	887.9
$i7$-3	35.2	77.7	177.0	362.3	772.0
$i7$-4	36.8	82.4	198.3	415.8	887.6
$i7$-5	36.8	83.6	199.0	412.6	874.4
$i7$-7	36.1	81.1	190.7	390.2	839.3
$i7$-8	36.9	83.2	200.3	417.8	868.1
q-8	66.2	151.3	336.5	677.5	1441
q-13	66.3	153.9	341.3	678.2	1442

Two important conclusions follow from the presented computational results. First, due to memory cashing effects, the CPU time of the OpenFOAM solver increases over-linearly with respect to the size J of the discrete problem. On the basis of these experimental results we propose the following computation time prediction model for the parallel OpenFOAM solver

$$T_p^1(J) = \max_{x \in G} T_0(x, J/p), \tag{3}$$

where G is set of p processors used to solve the problem of size J and $T_0(x, J/p)$ denotes the CPU time of the sequential algorithm applied for problem of size J/p on the x-th processor.

The second conclusion is that Intel Quad i7-860 processors are approximately 1.6 times faster than Q6600 processors. In addition some Intel Quad i7-860 processors are up till 1.15 times faster than the remaining processors. Thus a weighted load balancing technique can reduce the global CPU time of the parallel solver.

Next we present results of computational experiments, which confirm both conclusions. In Table 3 CPU times of the parallel OpenFOAM algorithm are given for different sizes of the discrete problem and different sets of processors.

Table 3. CPU times of the parallel OpenFOAM algorithm for different sizes of the problem and different sets of processors: $i7$-x and q-x denote the x-th Intel Quad i7-860 and Q6600 processor, respectively

	J=128000	J=254892	J=512000	J=1018488	J=2048000
$i7$-0, $i7$-1	18.8	39.3	86.7	204.9	420.2
$i7$-3, $i7$-7	18.4	38.3	86.1	193.3	394.4
q-8, q-9	30.7	73.2	162.0	337.5	689.5
$i7$-3, q-8	25.5	67.41	155.4	331.6	683.6
$i7$-0, $i7$-1 $i7$-2, $i7$-4	11.8	20.4	41.3	89.4	209.4
$i7$-3, $i7$-5 $i7$-7, $i7$-8	11.4	20.4	40.9	90.0	204.3
q-8, q-12 q-9, q-13	23.4	37.9	79.4	168.5	347.6
8 $i7$ nodes	8.8	13.6	22.4	43.6	94.3
8 q nodes	22.1	28.0	44.3	86.7	179.5
16 nodes	17.3	20.5	26.2	42.7	85.4

It follows from the presented results that for two largest size discrete problems the CPU time is shorter when all 16 processors of the cluster are used.

Since Vilkas cluster is heterogeneous and consists of two types of nodes, a load balance of computational tasks can be improved by using the weighted mesh partitioning algorithm. In Table 4 CPU times of the parallel OpenFoam algorithm are given for different relative weights assigned to processors. Here $i7(w)$ and $q(w)$ denote Intel Quad i7-860 or Q6600 processors and w denotes the relative speed of this processor.

It follows from these results that adaptive mesh distribution algorithm improves the load balancing and CPU time decreases 1.3 times for the largest discrete problem.

Table 4. CPU times of the parallel OpenFoam algorithm for the adaptive mesh decomposition algorithm. Here $i7(w)$ and $q(w)$ denote Intel Quad i7-860 or Q6600 processors and w denotes the relative speed of this processor.

	J=512000	J=1018488	J=2048000	J=8192000
$i7$-3(1), q-8(1) q-9(1)	102.0	222.0	461.1	
$i7$-3(2), q-8(1) q-9(1)	89.7	189.9	381.3	
$i7$-3(1.87), q-8(1) q-9(1)	88.8	183.0	371.8	
8 $i7(1)$, 8 $q(1)$	26.2	47.7	85.4	363.6
8 $i7(1.5)$, 8 $q(1)$	24.3	35.5	69.4	288.2
8 $i7(1.6)$, 8 $q(1)$	24.5	35.7	66.1	279.9

Next we have investigated the efficiency of the parallel solver when 2 and 4 cores per node are used in computations. The first conclusion is that only 2 cores are giving a reasonable speed-up of computations. Thus in Table 5 we present CPU times of the parallel OpenFoam algorithm for different numbers of processors and 2 cores per node. The size of the problem is $J = 2048000$ elements. The case 1×1 provides CPU time for the sequential algorithm.

Table 5. CPU times of the parallel OpenFoam algorithm for different numbers of processors n_d and $n_c = 2$ cores per node. The size of the problem is $J = 2048000$ elements. The case 1×1 provides CPU time for the sequential algorithm.

	1×1	1×2	2×2	4×2	8×2
$i7$-3	772.0	566.5	288.2	142.7	64.8
q-9	1441	1161	573.6	270.2	122.7

Two conclusions follow from the presented results. First, the scalability of the parallel algorithm is still good for clusters with multicore nodes. This scaling follows very similar trends as for one core per node. The second conclusion states that the retardation coefficient $\mu(n_c) > 1$ should be included into the estimate of computation costs of the parallel algorithm

$$T_p^1 = (c_1 + 1000\mu(n_c)c_2)\lceil J/p \rceil.$$

It depends on the maximum number of cores n_c per processor. This coefficient should be taken into account, since the shared-memory structure can become a bottleneck when too many cores try to access the global memory of a node simultaneously.

In the next series of computational experiments we have solved the largest problem with $J = 8192000$ elements. 16x2 processes of two types of nodes were used in computations. Since Vilkas cluster is heterogeneous, a load balancing strategy is applied in the weighted mesh partitioning algorithm. In Table 6 CPU times of the parallel OpenFoam algorithm are given for different relative weights assigned to processors.

Table 6. CPU times of the parallel OpenFoam algorithm for the adaptive mesh decomposition algorithm. Here $i7(w)$ and $q(w)$ denote Intel Quad i7-860 or Q6600 processors and w denotes the relative speed of this processor. The size of the discrete problem $J = 8192000$.

	$i7(1), q(1)$	$i7(1.4), q(1)$	$i7(1.5), q(1)$	$i7(1.6), q(1)$	$i7(1.7), q(1)$
T_{32}	279.9	229.7	216.6	206.3	202.4

Up to this point all results were obtained by fixing the number of linear solver iterations to 1000. In practice the number of iterations is calculated dynamically to fit the convergence tolerance requirement. Since we are using the conjugate gradient method with the diagonal Incomplete-Cholesky (DIC) preconditioner, the number of iterations may depend on the number of processes p. For computational tests we take $J = 2048000$. Results of computational experiments show that the time of computations is proportional to the number of iterations and the number of iterations weakly depends on the number of processes. So for different numbers of processes p we calculate the number of iterations that are performed to achieve the tolerance equal to 10^{-6}. The results in Table 7 show that the increased number of iterations lowers the efficiency, this number can occasionally also drop. But in general efficiency of parallel preconditioners is not sensitive to changes of p.

Table 7. The average number of iterations per time step for different number of processes p. The size of the discrete problem $J = 2048000$, the tolerance parameter for solver is equal to 10^{-6}.

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Average number of iterations	1015.6	1140.1	1142.6	1448.2	1401.6
T_p	803.7	452	236.1	135.6	91.5

In the case of $p = 16$ processes we have used 8 nodes and 2 cores per node.

6 Conclusions

It is shown that for a developed OpenFOAM solver the scaling and efficiency performance on Vilkas cluster is good up to 32 cores when I/O effects are neglected and load balancing is used for mesh partition.

Smaller sizes of distributed discrete sub-problems enable a better caching and give a sub-linear speed-up for computational part of the parallel algorithm.

It is important to test the effects of I/O costs when balancing between computation and I/O parts of the algorithm is not good, for example when a solution should be saved every 5-10 time steps. It is well known that OpenFOAM I/O libraries are based on standard MPI I/O routines and they are introducing quite big overheads.

A hybrid MPI and OpenMP parallel model can be attractive in the case of parallel systems with a big number (12 or 16) of cores per node.

References

1. Higuera, P., Lara, J., Losada, I.: Realistic wave generation and active wave absorption for Navier–Stokes models: Application to OpenFOAM. Coastal Engineering 71(1), 102–118 (2013)
2. Petit, O., Bosioc, A., Nilsson, H., Muniean, S., Susan-Resigo, R.: Unsteady simulations of the flow in a swirl generator using OpenFOAM. International Journal of Fluid Machinery and Systems 4(1) (2011), doi:10.5293/IJFMS.2011.4.1.199
3. Makhkamova, I.: Numerical Investigations of the Thermal State of Overhead Lines and Underground Cables in Distribution Networks. Doctoral thesis, Durham University (2011)
4. OpenFOAM, `http://www.openfoam.org`
5. Weller, H.G., Tabor, G., Jasak, H., Fureby, C.: A Tensorial Approach to Computational Continuum Mechanics Using Object-oriented Techniques. Journal of Computational Physics 12(6), 620–631 (1998)
6. AlOnazi, A.: Design and Optimization of OpenFOAM-based CFD Applications for Modern Hybrid and Heterogeneous HPC Platforms. Master thesis, University College Dublin (2013)
7. Chevalier, C., Pellegrini, F.: PT-Scotch: A Tool for Efficient Parallel Graph Ordering. Parallel Computing 34(6-8), 318–331 (2008)
8. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to parallel computing: design and analysis of algorithms. Benjamin/Cummings, Redwood City (1994)
9. Piscaglia, F., Montorfano, A., Onorati, A.: Development of fully-automatic parallel algorithms for mesh handling in the OpenFOAM-2.2.x technology. SAE Technical Paper 2013-24-0027 (2013), doi:10.4271/2013-24-0027
10. Rivera, O., Fürlinger, K., Kranzlmüller, D.: Investigating the scalability of OpenFOAM for the solution of transport equations and large eddy simulations. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) ICA3PP 2011, Part II. LNCS, vol. 7017, pp. 121–130. Springer, Heidelberg (2011)