Published in IET Science, Measurement and Technology
Received on 31st May 2011
Revised on 3rd April 2012
doi: 10.1049/iet-smt.2011.0126

ISSN 1751-8822

Numerical simulation of electrical machines by means of a hybrid parallelisation using MPI and OpenMP for finite-element method

S. Boehmer[1] T. Cramer[2] M. Hafner[1] E. Lange[1] C. Bischof[2] K. Hameyer[1]

[1]Institute of Electrical Machines, RWTH Aachen University, Aachen D-52056, Germany
[2]Center for Computing and Communication, RWTH Aachen University, Aachen D-52056, Germany
E-mail: stefan.boehmer@iem.rwth-aachen.de

Abstract: In this study, a hybrid parallelisation approach for the simulation of non-linear electromagnetic problems by means of the message passing interface (MPI) and the OpenMP application program interface for the finite-element method (FEM) is investigated. After an introduction, the metrics applied to evaluate the speedup and the efficiency are outlined. By parallelising the institute's in-house FEM-package '*i*MOOSE' either by MPI or by OpenMP, an evaluation basis for the hybrid approach is being founded. The hybrid parallelisation approach is being evaluated on the high-performance computing cluster of university's centre for computing and communication.

1 Introduction

Throughout the past decade, the increase of computational power was partially a result of increasing the processing frequency but mainly a consequence of massive parallelisation within the central processing unit (CPU). This multi-core architecture evolved rapidly within the past years and along with their research programs, the road maps of the large CPU manufacturers indicate a break through of more than a hundred cores for a single CPU within the next decade [1]. This massive parallelisation has already proven its potential for general purpose graphic processing units (GPGPU). The architecture classification of CPUs and GPGPUs according to [2] distinguishes two categories: multiple instructions multiple data (MIMD) for the CPU and single instructions multiple data (SIMD) for the GPGPU, respectively.

In theory, GPGPUs have an extensive performance benefit in comparison with CPUs because of its SIMD concept and a huge number of parallel processing units. However, in most real-world applications this performance is restricted by the memory bandwidth available to the GPGPU.

The motivation of this work is to evaluate the performance of the industrial standardised parallelisation paradigms MPI, OpenMP and their hybrid combination on high-performance computing clusters based on MIMD architectures to lay the foundation for a decision basis for future FEM-software design for the numerical simulation of electrical machines. The proposed parallelisation can be extended to SIMD architectures, which is not included in this work, since one of the underlying equation solvers will be supporting GPGPUs in the near future [3].

2 Metrics

In order to compare different parallelisation approaches, the following metrics are applied [4]. All measurements are based on the execution time $T(p)$ (Wall Clock Time) with p being the number of parallel processes. The sequential execution time $T(1)$ accounts for the exclusively consumed CPU-time as well as for peripheral access time caused by a sequential process. The parallel overhead is described by

$$T_{\mathrm{O}}(p) = pT(p) - T(1) \tag{1}$$

The speedup for a given number of p parallel processes or threads is defined as

$$S(p) = \frac{T(1)}{T(p)} \tag{2}$$

The general definition of the efficiency with respect to the computational resources is given by

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)} = \frac{1}{1 + (T_{\mathrm{O}}(p)/T(1))} \tag{3}$$

According to Amdahl's law [5] the theoretical maximum speedup for a given program in case of $p \to \infty$ is limited by its sequential fraction α

$$S(p) \leq S(\infty) \leq \frac{1}{\alpha} \tag{4}$$

The theoretical maximum speedup of a program with a sequential fraction of, for example, $\alpha = 0.1$ is $S(\infty) \leq 10$.

3 Parallelisation paradigms

Processor manufacturers state that the single core speed is not going to increase significantly in the next CPU generations. This arises from physical limitations, which are not expected to be overcome in the near future [6]. Owing to the upcoming microprocessor architectures it is crucial to exploit parallelisation to cut down computation time. Essentially, two approaches can be used for parallelisation on general purpose hardware: The OpenMP Application Program Interface [7] and the MPI [8].

3.1 OpenMP against MPI

OpenMP is designed to work on architectures having a shared memory address space and can be deployed to parallelise loops and computationally independent program fragments with a minimum of additional code. Critical memory access operations, for example, during the element-wise assembly of the system matrix within a parallel loop must be guarded by exclusive locking mechanisms, which guarantee that only one process writes to a certain entry of the matrix at the same time.

MPI relies on data distribution requiring an explicit communication between the parallel processes, which demands in case of FE analysis an a priori mesh decomposition. This decomposition allows for a locking free assembly of the system matrix but requires an explicit data exchange for solving and incorporating boundary constraints. The amount of additional code is considerably larger compared with OpenMP. MPI parallelisation gains a huge benefit, if the FE problem description exceeds the main memory capability of a single computing node, since such problems can be computed by using multiple computing nodes by MPI.

Parallelisation with MPI is done by multiple processes, whereas OpenMP uses multiple threads belonging to one process. Threads are light-weight processes sharing the same address space and therefore also global variables. This makes context switching between different threads of a process much faster than switching between different processes.

3.2 Hybrid parallelisation

In case of hybrid parallelisation, both paradigms OpenMP and MPI are combined to a joint parallelisation in order to exploit the advantages of both approaches. Parallelisation on one computing node is done by OpenMP, which theoretically benefits from the shared memory address space. In addition, MPI is used for the parallelisation between multiple computing nodes for better scalability of the computation.

4 Implementation details

The proposed hybrid parallelisation consists of a combined OpenMP and MPI parallelisation. Each paradigm is described separately in the following sections.

4.1 Parallelisation based on OpenMP

OpenMP exploits a shared address space memory parallelisation, such that every thread is capable of arbitrarily accessing the whole memory. In contrast to the MPI parallelisation, no data distribution is necessary. Anyhow, it must be ensured that the parallelised regions are thread-safe, particularly the access to the system matrix is a critical operation. Here, data race conditions must be avoided, which occur when multiple threads access the same location in the matrix without an appropriate control mechanism. Different approaches have been evaluated to guarantee such an exclusive access to the system matrix. Finally, data encapsulation in combination with OpenMP's critical sections wrapping the data access performed best during evaluation.

The following computationally expensive regions have been identified and parallelised:

- assembly of the system matrix;
- adding the Jacobian matrix;
- solving the system of equations;
- curl of the magnetic vectorpotential to compute the magnetic fluxdensity.

Parallelisation is done on loop level by adding OpenMP directives mainly in front of for-loops, for example, at the loop iterating over all elements during assembly of the system matrix.

4.2 Parallelisation based on MPI

The implemented MPI parallelisation employs domain decomposition of the FE-mesh for the required explicit data distribution. Therefore the FE-mesh is cut into multiple sub-meshes, which are assigned to the participating MPI processes, so that every involved process is exclusively working on a particular sub-mesh.

The domain decomposition of the FE-mesh is done element-wise so that each FE-element is assigned to exactly one sub-mesh. Formally, the decomposition of the complete mesh domain Ω into s sub-meshes Ω_i is given by

$$\bigcup_{i=0}^{s} \Omega_i = \Omega \quad \text{with} \quad \Omega_i \cap \Omega_j = \emptyset \quad \text{for } i \neq j \tag{5}$$

The mesh decomposition is interpreted as a graph-based problem and thus methods from graph theory can be applied to perform the decomposition. In a first step, the FE-mesh is transformed to a graph representation, that is, the dual graph of the mesh. Every element of the mesh is assigned to a single vertex in the dual graph. Two vertices in the dual graph are connected by an edge if the corresponding elements are adjacent. For 2D problems two elements are adjacent if they share a common edge in the mesh and in 3D if they share a common surface. Each vertex of the dual graph is weighted according to the number of degrees-of-freedom (DoFs) associated with its corresponding element of the FE mesh. Afterwards, the dual graph can be partitioned by several methods. In this work PARMETIS [9] is applied, which employs multilevel dual-graph partitioning methods. Finally, the mesh decomposition is determined from the partitioned dual graph by the unique mapping from vertices of the dual graph to mesh elements.

The mesh decomposition is the first fundamental component of the MPI parallelisation, parallel assembly of the system matrix is the second fundamental component. Since MPI relies on data distribution for all processes, similar to the partitioning of the FE-mesh the system matrix and the right-hand side vector have to be split up and stored on different processes as well. This is done row-wise so that every process holds a specific range of rows.

During the assembly of the system matrix one has to identify the DoFs located within the interior of a sub-mesh and the DoFs associated to the boundaries of the sub-meshes. As the latter ones interact with each other, the following logic enumeration is applied:

1. *All interior DoFs*: Process with rank zero handles its interior DoFs and maps them to the first rows of the matrix. All other processes enumerate their interior DoFs with increasing rank the same way.
2. *All boundary DoFs:* Enumeration by a common pattern.

By this mapping from DoFs to rows in the equation system the matrix layout equals an arrow matrix. Let s be the number of processes and n the total number of DoFs

$$\begin{pmatrix} A_1 & & & & F_1 \\ & A_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & A_s & F_s \\ F_1^{\mathrm{T}} & F_2^{\mathrm{T}} & \cdots & F_s^{\mathrm{T}} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \\ b_n \end{pmatrix} \begin{array}{cc} \text{rank } 0 & \text{submesh } 1 \\ \text{rank } 1 & \text{submesh } 2 \\ \vdots & \vdots \\ \text{rank } s-1 & \text{submesh } s \\ \text{all ranks} & \text{boundary DoFs} \end{array} \quad (6)$$

Matrix $\boldsymbol{A}_i$ contains the portion of all interior DoFs of the sub-mesh i, which are associated with the process of rank $i-1$. The portion of all boundary DoFs is contained in the matrices $\boldsymbol{F}_i$ and $\boldsymbol{D}$, which are assembled by all processes. Owing to this enumeration every process can iterate independently over the elements of its associated sub-mesh, generate the element matrices and add them to the system matrix.

Hence, the MPI parallelisation is done on a higher level of abstraction with respect to the source code structure in comparison with the OpenMP parallelisation. By means of the domain decomposition, the whole computation is done in parallel whereas only several for-loops are parallelised by the OpenMP approach.

5 Results

To evaluate the implemented parallelisation mainly problems from electrical machine modelling have been taken into account to consider the relative motion of stator and rotor since this is a crucial task for parallelisation of electromagnetic field problems. The topological change of the FE-mesh has to be addressed by the domain decomposition, which is described in detail by Boehmer *et al.* [10]. The considered numerical problems are stated in Table 1 and contain different numbers of DoFs. Note that the speedup (2) is independent from the FE-formulation, which can be quasi-static or transient. All measurements have been done on the high-performance computing cluster of the university's centre for computing and communication equipped with Intel Xeon X5570 'Nehalem EP' computer systems connected via InfiniBand.

Table 1 Evaluated test cases

Test case	# DoFs	Description
PMSM2D	89.587	synchronous machine 2D
TEAM20	255.804	TEAM20 test problem 2D
PMSM3D	805.206	synchronous machine 3D
LARGE3D	9050.911	conductor in free space

5.1 Parallelisation based on OpenMP

The measured speedups of the studied problems with up to eight threads are shown in Fig. 1. The maximum speedup measured with eight threads varies from 1.9 to 2.5 and increases with respect to the problem size. There are mainly two reasons for these rather small speedup values. First, according to Amdahl's law (4) the sequential code fraction limits the maximum speedup of the computation. The sequential fraction of the problem PMSM2D is about 11% resulting in a theoretical maximum speedup of nine with an infinite number of threads. Second, the iterative solving process scales badly for the applied solver library, which is not in the focus of this work.

5.2 Parallelisation based on MPI

The MPI parallelisation is evaluated with up to 64 processes, which equals eight computing nodes equipped with eight cores. Owing to the parallel execution of all processes, a sequential code fraction limiting the speedup as occurred at the OpenMP parallelisation cannot be observed (Fig. 2).

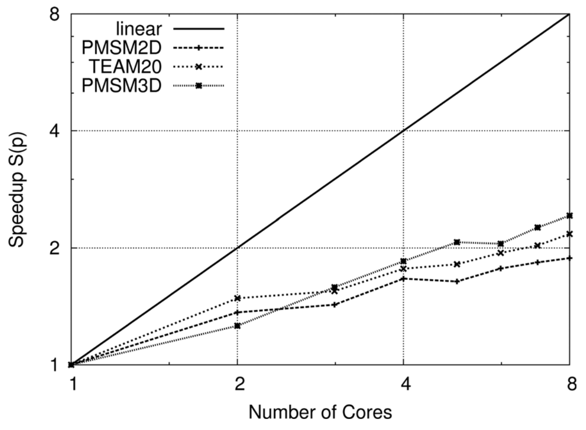

Fig. 1 *OpenMP speedup*

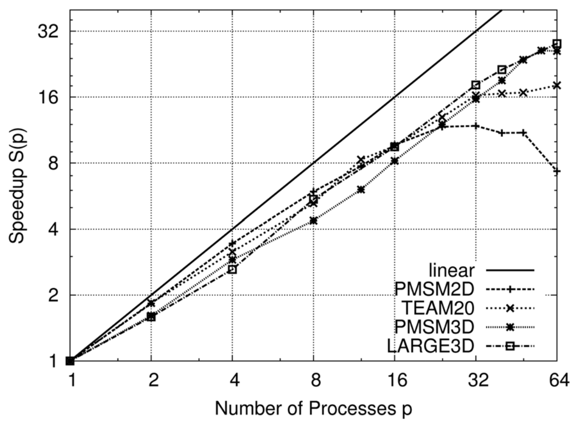

Fig. 2 *MPI speedup*

Fig. 2 also shows the impact of the number of DoFs on the maximum speedup. The smallest test case PMSM2D reaches its maximum speedup already at 24 processes. If the number of processes is further increased, the speedup decreases, which implies that the communication overhead between the involved processes outweights the benefit gained by parallel execution. In contrast to the OpenMP parallelisation the limiting factor is the explicit communication during execution. If one process handles less than a certain number of DoFs the parallelisation is not efficient any more. The maximum measured speedup increases with the number of DoFs and the overall maximum speedup of 28 is measured for the test case LARGE3D at 64 processes. This means that a simulation requiring almost a full month in the sequential case can be done in approximately 1 day using eight computing nodes with eight cores on each node.

The parallelisation is further analysed by splitting up the logical parts of solving and the remaining part of the computation. To do so, the time consumed during solving is measured independently from the total runtime. The results for test case PMSM3D are shown exemplarily in Fig. 3. The speedup of the solving is quite low compared to the remaining computation for all considered number of cores. The latter is nearly linear, in the range from two to eight cores it is almost ideal. The low speedup of the solving can be explained by the applied iterative algorithm for solving the system of equations. In this work, the conjugate-gradient algorithm is applied in combination with symmetric successive overrelaxation as block preconditioner. This results in the problem that the required number of iterations of the solving algorithm increases with *p* depending on the problem investigated owing to the behaviour of the preconditioner, which is known from various works. The relative growth of the required number of iterations as a function of the number of processes for all test cases related to the sequential case is shown in Fig. 4. The number of iterations for test case PMSM3D increases by a factor of almost 1.5 from one to four processes. Thus, also if one iteration of the solving algorithm scales linearly, the speedup of the complete solving decreases by this factor in contrast to the linear speedup.

5.3 Hybrid parallelisation

The hybrid parallelisation decomposes the discretisation of the problem to a given number of *p* sub-meshes. Within each process the matrix assembly is performed by the OpenMP parallelisation. The applied equation solver is LIS [11] which is based on a hybrid parallelisation of OpenMP and MPI. For the hybrid parallelisation up to 80 cores distributed over ten computing nodes are used. The speedup is analysed by assigning one core to every thread and running three different hybrid configurations (HC):

- HC1: MPI only with one thread for each process.
- HC2: Hybrid with two threads for each process.
- HC3: Hybrid with four threads for each process.

The speedup of the smallest test case PMSM2D is shown in Fig. 5. As seen before in Fig. 2 the MPI only parallelisation, that is, HC1 variant, reaches the maximum speedup at 32 cores. The HC2 variant delivers a near identical speedup, which increases further up to 48 cores. The hybrid approach allows to increase the scalability for higher number of cores whereas the speedup of the MPI only parallelisation is already declining. The HC3 variant does not gain a better speedup at the considered up to 64 cores.

The results of the test case TEAM20 are shown in Fig. 6. The HC1 variant reaches its maximum speedup at 48 cores and declines with further growing number of cores. The speedup of both hybrid variants HC2 and HC3 stays below the speedup of HC1 from 2 to 64 cores. If the number of cores is increased up to 80 both outperform the speedup of the MPI parallelisation, which is declining. However, they do not reach the maximum speedup of the MPI parallelisation. So the hybrid parallelisation does not make sense for this specific test case and the considered up to 80

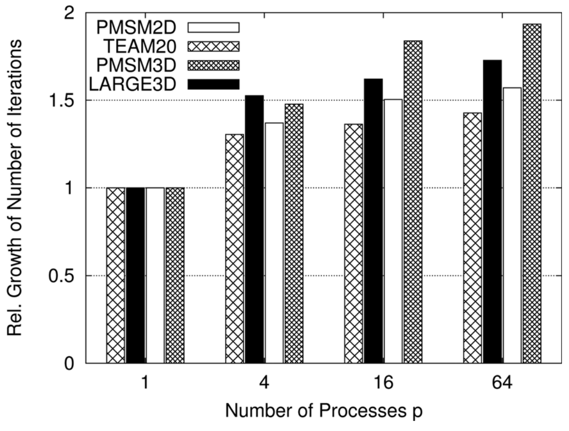

Fig. 4 *Increased number of CG-iterations*

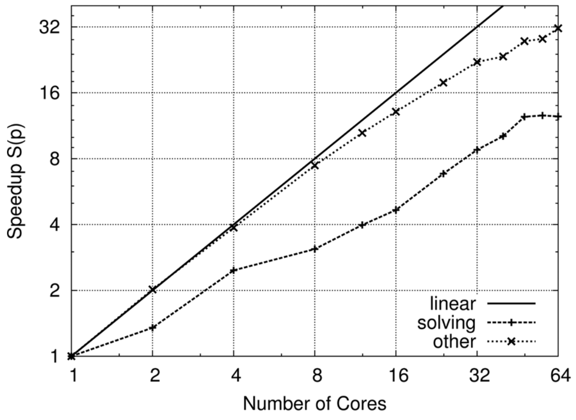

Fig. 3 *MPI speedup PMSM3D*

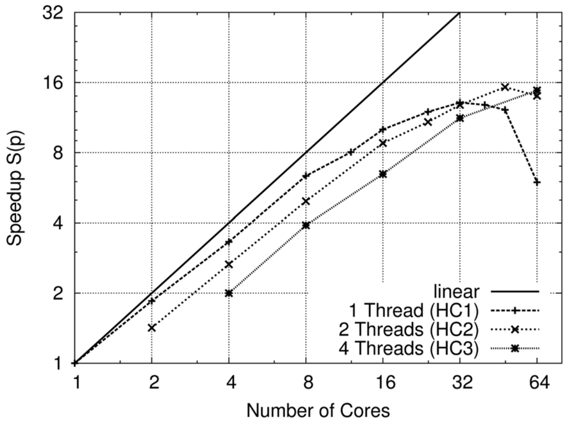

Fig. 5 *Hybrid PMSM2D*

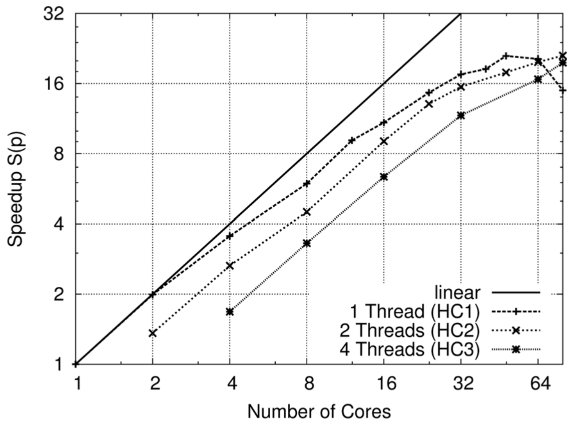

Fig. 6 *Hybrid TEAM20*

cores, but possibly is advantageous at a higher number of cores.

The measurement of the test case PMSM3D in Fig. 7 shows that the hybrid parallelisation HC2 delivers a benefit in speedup above 64 cores in comparison to the HC1 variant. This benefit is a consequence of the declining performance of the MPI parallelisation because of the increasing communication overhead between the MPI processes.

Fig. 8 shows the speedup of the largest test case LARGE3D. In contrast to all other test cases there is no stagnation or declining of the HC1 variant for a higher number of cores. Thus the hybrid parallelisation cannot gain any benefit and the speedup remains below the MPI only one. Predicting both hybrid curves on basis of the results of the other test cases, an advantage for the hybrid variant will occur at a high number of cores. This is already signified at 80 cores where the speedup of the hybrid variant HC2 approaches to the MPI variant HC1.

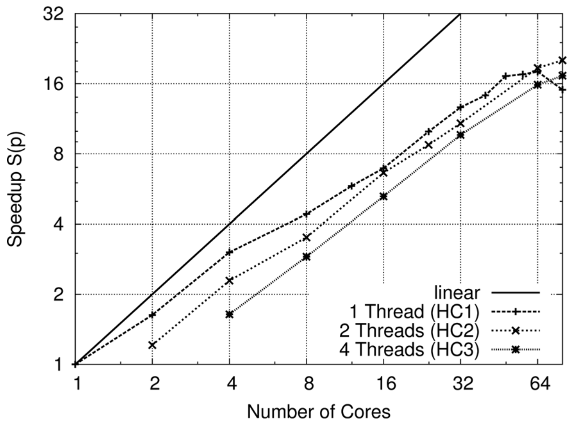

Fig. 7 *Hybrid PMSM3D*

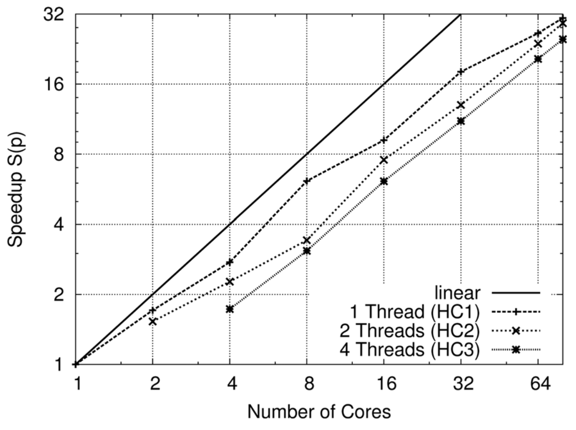

Fig. 8 *Hybrid LARGE3D*

6 Discussion and conclusions

This paper gives an evaluation of the numerical simulation of electrical machines by means of a hybrid parallelisation of the FEM-package *i*MOOSE using the standardised paradigms OpenMP and MPI. A comparison between OpenMP and the MPI parallelisation shows a benefit for the MPI parallelisation. This can be explained by the sequential code fraction of the OpenMP implementation, which does not exist for the MPI variant. Depending on the number of cores, the hybrid parallelisation gains a benefit. Especially, in case of the MPI speedup stagnation, the hybrid approach performs advantageously. This stagnation of the MPI speedup arises, when the communication overhead between the processes outweights the benefit of the parallelisation. This break-even point is particularly dependent on the problem size, respectively, the number of DoFs. As long as this point is not reached, the MPI parallelisation should be prefered. The advantage of the hybrid approach will especially become of interest throughout the forthcoming years with CPUs unifying hundreds of cores being available. The evaluation shows that the solving process of the system of equations limits the possible speedup owing to increasing number of required iterations with increasing number of used cores. To obtain a higher speedup of the parallelisation it is therefore required to improve the solving process and especially the preconditioning phase, for example, by using multigrid methods [12]. Another possibility in the future is to include GPGPUs for solving the systems of equations.

7 References

1 Various: ‘Intel technology journal volume 13, issue 4: addressing the challenges of tera-scale computing’ (Intel Press, 2009)
2 Flynn, M.J.: ‘Some computer organizations and their effectiveness’, *IEEE Trans. Comput.*, 1972, **21**, (9), pp. 948–960
3 Balay, S., Buschelman, K., Gropp, W.D., *et al.*: ‘PETSc Web page’, http://www.mcs.anl.gov/petsc
4 Grama, A., Karypis, G., Kumar, V., Gupta, A.: ‘Introduction to parallel computing’ (Addison Wesley, 2003, 2nd edn.)
5 Amdahl, G.M.: ‘Validity of the single processor approach to achieving large scale computing capabilities’. Proc. Spring Joint Computer Conf. on AFIPS’67, Atlantic City, New Jersey, 18–20 April 1967, p. 483
6 Flynn, M.J., Hung, P.: ‘Microprocessor design issues: Thoughts on the road ahead’, *IEEE Micro*, 2005, **25**, pp. 16–31
7 OpenMP Architecture Review Board. OpenMP Application Program Interface, 3.0 edition, Mai 2008, http://www.openmp.org/mp-documents/spec30.pdf
8 Message Passing Interface Forum. MPI-2 Extensions to the Message Passing Interface, 2.0 edition, 1997, http://www.mpi-forum.org
9 Karypis, G., Kumar, V.: ‘Parallel multilevel graph partitioning’. Tenth Int. Parallel Processing Symp. Proc. IPPS’96, 1996, pp. 314–319
10 Boehmer, S., Lange, E., Hafner, M., Cramer, T., Bischof, C., Hameyer, K.: ‘Mesh decomposition for efficient parallel computing of electrical machines by means of fem accounting for motion’, *IEEE Trans. Magn.*, 2012, **48**, (2), pp. 891–894
11 Nishida, A.: ‘Experience in developing an open source scalable software infrastructure in Japan’. Computational Science and its Applications – ICCSA 2010 (*LNCS*, **6017**), 2010, pp. 448–462
12 Hiptmair, R.: ‘Multigrid method for maxwell’s equations’, *SIAM J. Numer. Anal.*, 1999, **36**, (1), pp. 204–225