Application of Integral Equation and Hybrid Techniques to the Parallel Computation of Electromagnetic Fields in a Distributed Memory Environment

Ulrich Jakobus

Institut für Hochfrequenztechnik, University of Stuttgart,
Pfaffenwaldring 47, D-70550 Stuttgart, Germany
E-Mail jakobus@ihf.uni-stuttgart.de

ABSTRACT. **This paper describes the parallelization of the method of moments and hybrid code FEKO for execution on massively parallel supercomputers with a distributed memory as well as on clusters of connected workstations. The parallel implementation of the different phases of the solution process, e.g. matrix fill, solution of the system of linear equations, and near– and far–field computation is discussed in detail. Several results for different applications are given and the achieved performance is presented.**

1 Introduction

The method of moments (MoM) originally developed by Harrington in 1968 [1] remains a very popular technique for solving numerous electromagnetic radiation and scattering problems involving metallic or dielectric scattering bodies.

As compared with other numerical techniques such as the finite difference (FDTD) or finite element method (FEM), the main advantage lies with the fact that for most MoM formulations only boundaries must be discretized, i.e. instead of a three–dimensional discretization for FDTD or FEM and other techniques, only a two–dimensional discretization of boundaries is necessary for 3D problems. Thus the number of unknowns for the MoM is usually a magnitude smaller when compared to FDTD or FEM. However, the matrix is not sparse but dense so the memory requirement is rather critical, especially for higher frequencies.

In this paper, we consider the computer code FEKO, which has been developed at the University of Stuttgart and represents a rather comprehensive MoM and hybrid implementation. FEKO has been adapted such that it is capable of executing in parallel on massively parallel supercomputers as well as on clusters of connected workstations with a large distributed memory. In such a distributed memory environment, every node (e.g. workstation) has its own local memory which is not accessible by the other nodes as opposed to the shared memory concept where all processes have access to a global bank of memory. The terms node, processor, or processing element (PE) are used interchangeably in the following.

A general overview of parallel processing techniques in the area of computational electromagnetics (CEM) can be found in [2, 3]. The parallel FEM is described in [4, 5] and a parallel implementation of FDTD is presented in [6]. Asymptotic high frequency methods are considered in [7, 8]. Several authors have already described a parallel MoM formulation [9, 10, 11, 12, 13, 14, 15], but in this paper we put special emphasis also on achieving reasonable performance on heterogeneous clusters of workstations by implementing a dynamic load balancing scheme. FEKO also offers some hybrid extensions for higher frequencies and special geometries, and the parallelization of these parts is also considered. The hybrid extensions allow a reduction of the matrix size, so for the hybrid techniques high performance computing (HPC) is mainly used to reduce the computation time. Only when the conventional MoM is applied (e.g. for validation of the hybrid results), we quite often face problems with up to 40000 unknowns or even more, and then HPC is a must.

Our parallel implementation of FEKO is based on the *Message-Passing Interface* (MPI) standard [16], to which an excellent introduction can be found in [17]. Alternatively PVM *(Parallel Virtual Machine)* might be used as well. For massively parallel supercomputers (the results in this paper were obtained on an Intel Paragon and a CRAY T3E) the MPI functions and subroutines are usually provided in an optimized library by the manufacturer. But especially for clusters of connected workstations some public domain implementations are also available. We have compared MPICH [18] and LAM *(Local Area Multicomputer)*. MPICH was chosen for performance and compatibility reasons.

The general structure of FEKO including the hybrid extensions is presented in Section 2. Section 3 describes in detail the parallelization of the different phases of the solution process for the MoM part of FEKO, while Sec-

tion 4 deals with the parallelization of the hybrid extensions. Finally, some results demonstrating the performance of the parallel implementation are presented in Section 5.

2 General structure of the sequential version of FEKO

2.1 Conventional MoM

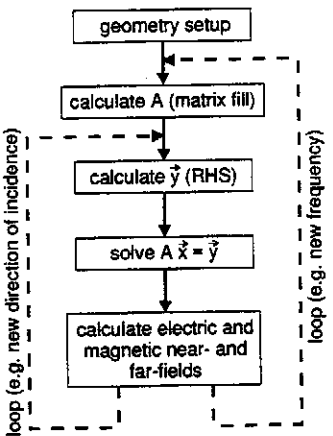

Fig. 1: Simplified structure of the MoM part of FEKO.

The simplified flow chart of the MoM part of FEKO is shown in Fig. 1. Some additional blocks must be included when considering hybrid extensions as described in Section 2.2.

The first block in Fig. 1 entitled *geometry setup* includes the allocation of memory, reading the geometrical data from an input file, the setup of the geometry and some optional error checks.

After this, the system matrix A is computed. Let N be the number of unknowns, then the CPU–time requirement for this phase is proportional to N^2. After the computation of the elements of the right–hand side (RHS) vector $\vec{y}$, the system of linear equations $A\,\vec{x} = \vec{y}$ can be solved resulting in the current vector $\vec{x}$. For very large matrices this step dominates the CPU time since there is an N^3 dependency. Once the currents are known, near- and far–fields can be computed. Several loops are possible as indicated in Fig. 1 by the dashed arrows.

The computer code FEKO supports metallic surfaces and wires as well as dielectric bodies. For the latter, two different formulations based on the surface and volume equivalence principle, respectively, have been implemented. Different basis functions $\vec{f}_n^{\,i}$ are involved:

- Linear roof–top basis functions $\vec{f}_n^{\,1}$ according to Rao, Wilton and Glisson [19] defined on flat triangular patches are used for metallic surfaces.
- Along metallic wires, overlapping triangular basis functions $\vec{f}_n^{\,2}$ are employed to represent the line current.
- Special basis functions $\vec{f}_n^{\,3}$ are required to model a current flow from metallic wires to surfaces at attachment points.
- An application of the surface equivalence principle to homogeneous dielectric scattering bodies leads to equivalent electric and magnetic surface current densities $\vec{J}$ and $\vec{M}$, respectively. Similar to metallic scattering problems, the surface of dielectric bodies are also subdivided into electrically small triangular patches. Basis functions $\vec{f}_n^{\,4}$ identical to $\vec{f}_n^{\,1}$ are used to represent $\vec{J}$, and for the magnetic surface current $\vec{M}$ a new set of vectorial basis functions $\vec{f}_n^{\,5}$ have been developed in Ref. [20].
- For inhomogeneous dielectric bodies the volume equivalence principle can be used in FEKO. The dielectric volume is subdivided into non–uniform cuboidal cells and within each cell three pulse basis functions $\vec{f}_n^{\,6}$ and three pulse basis functions $\vec{f}_n^{\,7}$ are employed in the three coordinate directions. $\vec{f}_n^{\,6}$ represents the equivalent electric volume current density and is required only for regions where the permittivity ε is not identical to the free space permittivity ε_0. The magnetic current is expanded into basis functions $\vec{f}_n^{\,7}$, and these basis functions are required only when $\mu \neq \mu_0$.

In addition to these 7 different basis functions $\vec{f}_n^{\,i}$, $i = 1 \ldots 7$, corresponding weighting functions $\vec{w}_m^{\,i}$ must be defined in order to convert the set of coupled integral equations into a system of linear equations. In general, we use the Galerkin procedure with $\vec{w}_m^{\,i} = \vec{f}_n^{\,i}$.

Fig. 2 shows the general structure of the MoM matrix A. The 7×7 sub-matrices A_{ij} associated with the combinations of basis functions $\vec{f}_n^{\,j}$ and weighting functions $\vec{w}_m^{\,i}$ can easily be identified. Note, however, that usually for an application to general radiation and scattering problems only some of these sub-matrices will be present. For most of the metallic problems, the sub-matrix A_{11} will be dominant in size, i.e. $N_1 \gg N_2, N_3$, and $N_4 = N_5 = N_6 = N_7 = 0$.

2.2 Hybrid extensions

FEKO is not only a pure MoM code, but includes some hybrid features allowing an efficient application also to high–frequency problems. The general idea is to reduce the number of unknowns N by applying a coupling of

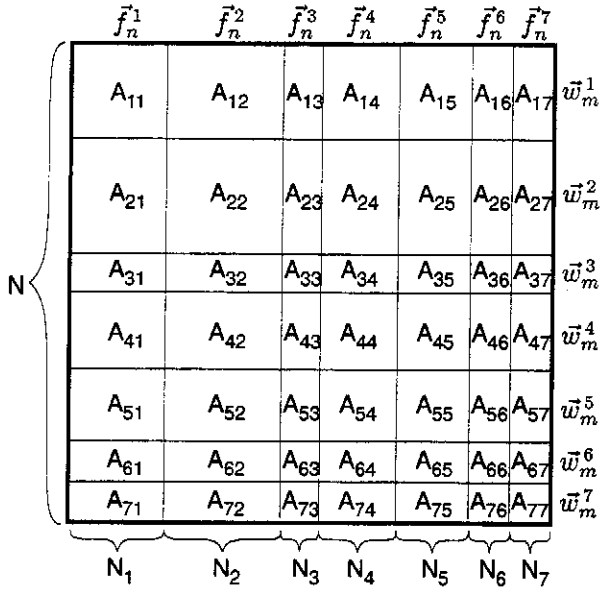

Fig. 2: Matrix A consisting of 7×7 sub-matrices A_{ij} of different size.

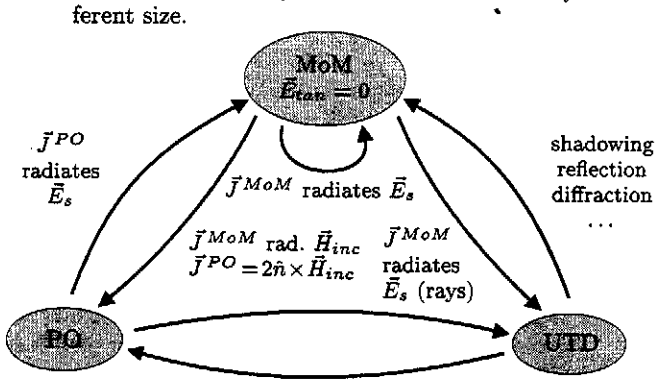

Fig. 3: Hybrid method combining MoM, PO and UTD.

the MoM with high–frequency techniques such as Physical Optics (PO) with correction terms [21, 22, 23], Fock currents [24], or diffraction theory (UTD) [25]. The different coupling mechanisms are illustrated in Fig. 3.

Some examples considered in the references cited above demonstrate the drastic reduction of memory requirement of the hybrid technique as compared to an application of the MoM alone. There are, of course, some additional computations introduced by the coupling mechanisms depicted in Fig. 3, so that the flow chart of FEKO in Fig. 1 becomes more complicated, nevertheless there is also a considerable reduction in the CPU–time achievable.

Another hybrid extension of FEKO is the possible replacement of the free–space Green's function by a special Green's function for a homogeneous, layered dielectric sphere [26]. This technique can be used to optimize the performance of mobile telecommunications antennas radiating close to the operator's head. Only the antenna structure with the case must be discretized, hence the number of unknowns N can be kept relatively small.

3 Parallel implementation of the MoM in FEKO

3.1 Geometry setup

As briefly described in Section 2.1, the part of this phase of the solution process consuming the most time is the search for connected wires or edges between triangular patches or connections between wires and surface elements etc. Some acceleration techniques such as the spatial partitioning technique borrowed from computer graphics [27] together with boxing algorithms have been implemented in order to avoid, for example, the comparison of two triangular patches which are located far away. Hence, for large structures, the CPU time spent in this phase is rather small when compared to the matrix fill, matrix solve and field calculations. Therefore, after the allocation of memory (parallel by all processes), the input file is read only by one process. After setting up the geometry, the data are sent to all other processes using the `MPI_TYPE_STRUCT`, `MPI_TYPE_COMMIT`, `MPI_BCAST` and the `MPI_TYPE_FREE` subroutines. Using these subroutines enables us to send whole Fortran COMMON blocks which is much more efficient than sending single variables or arrays.

As an option it has also been implemented to preprocess the geometry on a PC or workstation. The connection information and other computed data are written to a file, which can be read later by the parallel job together with the original input file.

3.2 Matrix fill

When trying to parallelize the setup of the matrix, i.e. the computation of matrix elements a_{mn}, $m, n = 1 \ldots N$ with $N = \sum_{i=1}^{7} N_i$ (see Fig. 2), special care must be taken in order to preserve the advantages of a possible symmetry of the structure or of some efficient matrix fill techniques. Exploiting symmetry allows to reduce the number of unknowns from N to a smaller value $\tilde{N}$. If the available main memory permits, then in a first step an $\tilde{N} \times N$ matrix is constructed which is later compressed to a square $\tilde{N} \times \tilde{N}$ matrix. The advantage of this procedure is that we can use some symmetry relations between the matrix elements in a row to further reduce the CPU–time. However, if not enough main memory is available for the $\tilde{N} \times N$ matrix, it is also possible to directly use

only an $\tilde{N} \times \tilde{N}$ array at the expense of a slightly increased CPU–time.

Different storage schemes for dense matrices, to the effect that the partitions can be assigned to different processors, are described e.g. in [28, Section 5]. When exploiting symmetry, when using efficient matrix fill techniques, or when using hybrid techniques where a modification of the matrix elements is necessary because of the coupling between MoM– and asymptotic region, it is is highly advantageous to keep a whole row of the matrix on one node.

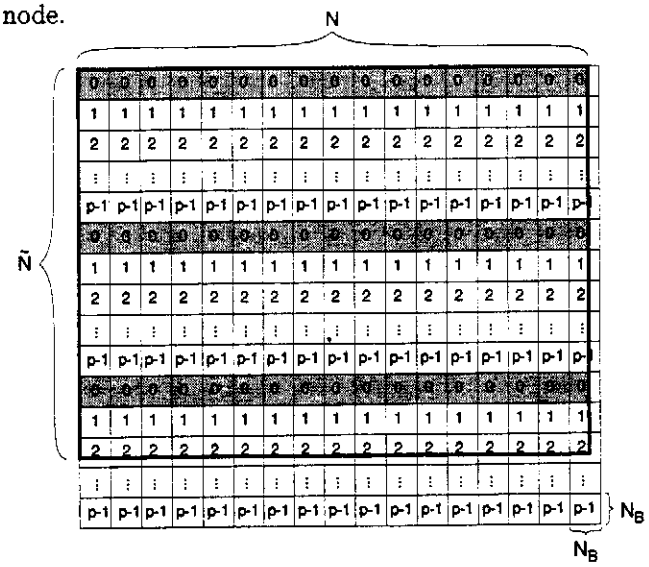

Fig. 4: Distributed storage scheme of the $\tilde{N} \times N$ matrix on p nodes (the numbers in the blocks indicate the nodes).

Hence, for a parallel computation and a distributed storage of the elements a_{mn}, the resulting rectangular matrix with $\tilde{N}$ rows and N columns is split among the p processors according to Fig. 4 in a one–dimensional block–cyclic row distribution. One square block consists of $N_B \times N_B$ elements with a block size in the range $1 \leq N_B \leq \lceil \frac{\tilde{N}}{p} \rceil$ (the notation $\lceil x \rceil$ represents the smallest integer value that is greater than or equal to x), see below (especially Fig. 10) for some considerations concerning an optimal choice of N_B.

The CPU time required for the computation of a single matrix element differs over a wide range by a factor of about 10 or even more, since various adaptive integration and testing schemes are applied depending on the distance $|\vec{r}-\vec{r}\,'|$ of source and observation point $\vec{r}\,'$ and $\vec{r}$, respectively. Also in some cases when $\vec{r}$ is located within the integration domain or very close to it, singular or quasi singular integrals are evaluated analytically. But on average, the time required to compute an element a_{mn} of the sub-matrix A_{11} (see Fig. 2) on an IBM RS 6000 workstation is in the range of $120 \ldots 250\,\mu$s. Experience has shown that the CPU time to compute the matrix elements in different blocks of N_B rows and N columns of the sub-matrix A_{11} is about the same (averaging effect).

However, we cannot expect this any more when different sub-matrices are involved. For a matrix row with weighting functions $\vec{w}_m^1$ (sub-matrices A_{1j}) the required CPU–time is most likely not equal to the CPU–time for a matrix row with different weighting functions $\vec{w}_m^j$, $j = 2 \ldots 7$. To overcome this problem, we have introduced a special mapping function allowing us to exchange matrix rows when the original matrix according to Fig. 2 is mapped to the matrix underlying the distributed storage scheme in Fig. 4. The mapping function is constructed so that each node contains about the same number of rows from all the sub-matrices, see Section 5 for an example (Figs. 12 and 13).

Another possible drawback of the described parallel matrix filling technique is that it is not very easy to implement a dynamic load balancing scheme. In the startup phase of the parallel version of FEKO, when the values of N, $\tilde{N}$ and the number of nodes p is known, a suitable block size N_B is selected and remains fixed. On a cluster of connected workstations, where the different processors are not exclusively assigned to a certain user, it might happen, that during the computation of the matrix elements another user starts some jobs on one of the workstations and, consequently, the parallel job on this node takes much longer than on the other workstations. There is currently no way to react to this situation, i.e. the other nodes have to wait (barrier) until the slowest machine has also finished its calculations. We have some ideas on how to overcome this problem, e.g. the faster nodes calculate some more matrix elements which were initially assigned to the slower machine and will then send them as a whole block to the slower machine. But this extension has not yet been implemented.

Note that this problem does not exist on massively parallel supercomputers where the different PEs are all of the same speed and the nodes are exclusively assigned to one single process. There are only some small differences in the required CPU–time on the different nodes due to the fact that we use adaptive integration techniques with a variable number of integration points or that the elements of the different sub-matrices A_{ij} are based on different equations, see Figs. 11, 12 and 13 below in Section 5 for some performance results.

3.3 Solution of the system of linear equations

A review of parallel matrix solvers for CEM applications is given in [29], the paper [30] concentrates on a parallel LU algorithm.

As already mentioned in the introduction, the whole parallelization of FEKO is based on the MPI standard. Several portable linear algebra packages based on MPI exist and their suitability has been compared.

PIM [31] provides iterative solvers such as CG, BiCG, BiCGStab, QMR, or GMRES. However, convergence studies with the sequential version of FEKO have shown that convergence is only satisfactory for some dielectric structures treated with the volume equivalence principle (sub-matrices A_{66}, A_{67}, A_{76}, A_{77} in Fig. 2). For metallic structures, which in contrast to the Fredholm integral equation of the second kind for these dielectric problems is based on a Fredholm integral equation of the first kind (EFIE), convergence is rather poor. However, some recent publications (e.g. [32, 33, 34]) show, that by using suitable preconditioners the convergence of the above mentioned iterative schemes can be improved significantly.

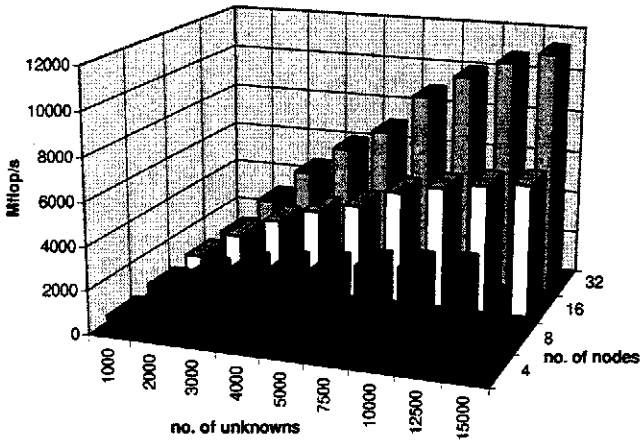

Fig. 5: Performance of the LU factorization on the CRAY T3E using ScaLAPACK (data from [35]).

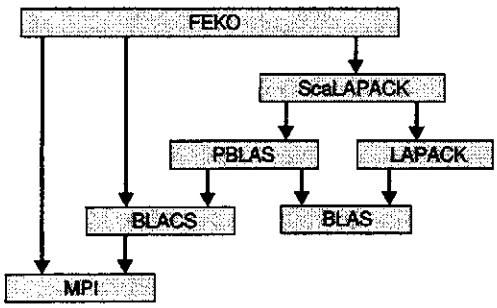

Fig. 6: Call graph of the main program FEKO and the libraries.

Quite often we are also interested in a solution for multiple right hand sides $\vec{y}$ (loop indicated by the dashed arrow in Fig. 1), therefore an LU factorization with subsequent back–substitution is preferred here.

One possible candidate for the solution of the system of linear equations might be PETSc [36], but motivated by the performance of ScaLAPACK [37, 38, 39] (see Fig. 5 and Refs. [38, 35] for more performance details) we chose the latter package, which is the parallel and distributed memory version of LAPACK [40]. According to the call graph in Fig. 6, ScaLAPACK requires the BLAS library, its parallel version PBLAS [41] as well as BLACS [42]. Besides the MPI calls in FEKO to the MPI library (e.g. MPICH on our workstation cluster), there are also direct calls to BLACS and ScaLAPACK subroutines. In ScaLAPACK, we use the subroutine `PZGETRF` to perform the LU factorization of the matrix A, `PZGECON` to get an estimate of its condition number, and `PZGETRS` to obtain the solution by back–substitution.

At the end of 1997 an alternative to ScaLAPACK became available. The PLAPACK [43] code claims to outperform ScaLAPACK for larger problem sizes, we are currently investigating this package as well.

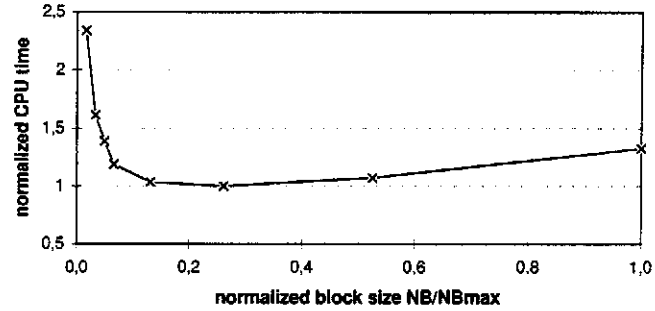

Fig. 7: Typical dependency of the normalized CPU–time required for matrix solve (LU factorization with ScaLAPACK) as a function of the normalized block size $\frac{N_B}{N_{Bmax}}$ with $N_{Bmax} = \lceil \frac{N}{p} \rceil$.

It was already mentioned above and will again be shown below (see Fig. 10) that the block size N_B is a very important parameter for an efficient matrix fill. However, also the CPU–time required for the solution of the system of linear equations depends on N_B (see Fig. 7 for a typical relation we found from experiments).

3.4 Parallel field computation

Normally, when the conventional MoM is applied to electrically large structures, the CPU times for matrix fill and solving the system of linear equations are dominant (proportional to N^2 and N^3, respectively). The time required to compute electric and magnetic near– and far–fields, which is proportional to N and to the number of observation points, remains relatively small in this case.

However, with the hybrid extensions as described in Section 2.2, we are able to reduce N drastically, so that

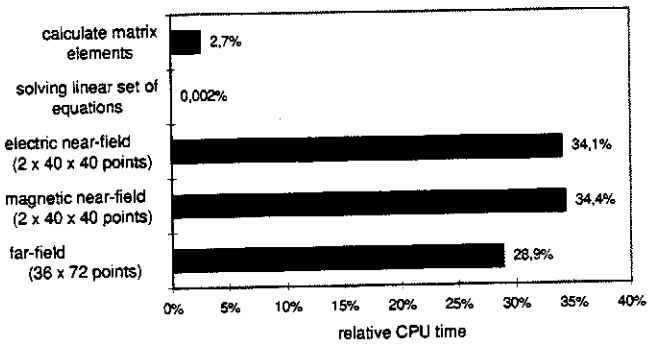

Fig. 8: CPU time requirement for the different phases of the solution process for an analysis of a mobile telephone radiating close to the human head (special Green's function).

the time required to compute the fields might become dominant. This is illustrated in Fig. 8, where the relative CPU–time for the different phases of the solution process is given for an analysis of a mobile telephone radiating at a frequency of 900 MHz close to the human head. The free–space Green's function has been replaced by the Green's function for a layered dielectric sphere [26]. One plane of symmetry was used, so that the $N_1 = 189$, $N_2 = 5$ and $N_3 = 6$ basis functions lead to $\tilde{N} = 99$ unknowns. The near–field was computed inside the head in two planes ($\vec{E}$ is required for the specific absorption rate SAR), and the far–field was calculated on a spherical surface with 36×72 observation points in order to compute the radiated power.

A master/server concept was chosen for the parallelization, which allows an almost perfect dynamic load balancing: The master process distributes the tasks to the remaining $p - 1$ server processes, which compute the near– or far–field. As soon as one server process has finished its calculation, it reports the result back to the master process. By this message, the master process is notified that the server process is ready and the master sends the coordinates of another observation point $\vec{r}$ to the server process. By this method, it is guaranteed that the server processes are always busy with numerical computations, i.e. server processes running on faster workstations don't have to wait for other processes running on slower machines.

Some possible drawbacks of this method are:

- The master process has to buffer the results it receives from the server processes before writing them to the output file, because the order how the results are received may be quite arbitrary. In the output file, however, the results shall be printed in the correct order specified by the user, e.g. with increasing x when the field strength is computed along the x–axis.
- From p processes only $p - 1$ are actually involved in the calculation. However, the loss of efficiency by a factor of $\frac{p-1}{p}$ is small for large values of p. Also on clusters of connected workstations we usually start the master and one server process on the same workstation, so that p processes are running on $p - 1$ workstations and all processors of these workstations are involved in the field calculation.

4 Parallelization of the hybrid extensions

The hybrid extensions by PO and UTD, as briefly summarized in Section 2.2, lead to a drastically reduced size of the matrix. The main changes concerning the parallelization as compared to the conventional MoM are that during the calculation of the matrix elements a_{mn} all the coupling effects between the different regions (MoM/PO/UTD) as indicated in Fig. 3 have to be taken into account. Therefore, with reference to the total solution time, the matrix filling remains relatively time consuming. The time for the matrix solve, however, becomes negligible for most applications.

For the matrix we keep the one–dimensional block–cyclic row distribution scheme, so that an entire row of the matrix resides on a node. After computing the MoM interaction, the modification due to PO or UTD can be performed locally on this node without any communication, since we keep the required geometry information for ray–tracing etc. on each node.

Especially concerning the coupling of MoM with UTD, the time for near– or far–field computations might be dominant for complex geometries with many surfaces and edges, since for every observation point ray–tracing must be performed in order to find possible ray paths with reflections and diffractions between source and observation point. The parallelization of the field computation as described in Section 3.4 can be applied without

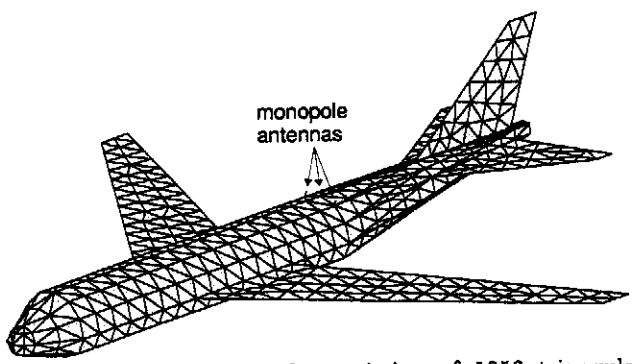

Fig. 9: Model of an aircraft consisting of 1256 triangular patches and 36 wire elements.

any modification to the MoM/UTD hybrid method as well. The implemented dynamic load balancing scheme is very important in this case, since for different locations of the observation point the number of ray paths might vary from zero to several hundreds or even thousands for complex geometries, which is also reflected in large variations of the required CPU–time for the ray–tracing procedure and the subsequent field computation.

5 Examples and parallel performance

For benchmarking purposes, the simple model of an aircraft as shown in Fig. 9 was used. The structure is excited by 3 small monopole antennas on top of the fuselage. The coupling between the antennas was of interest, but the far–field radiation pattern and the near–field were computed as well. The model consists of 1256 triangular patches and 36 wire elements, leading to $N_1 = 1850$ rooftop basis functions $\vec{f}_n^1$ on triangular patches, $N_2 = 33$ basis functions $\vec{f}_n^2$ on wire elements, and $N_3 = 20$ basis functions $\vec{f}_n^3$ at connection points. There is one plane of symmetry, so that the total number of $N = N_1 + N_2 + N_3 = 1903$ unknowns can be reduced to $\tilde{N} = 975$ ($\tilde{N}_1 = 940$, $\tilde{N}_2 = 22$, and $\tilde{N}_3 = 13$).

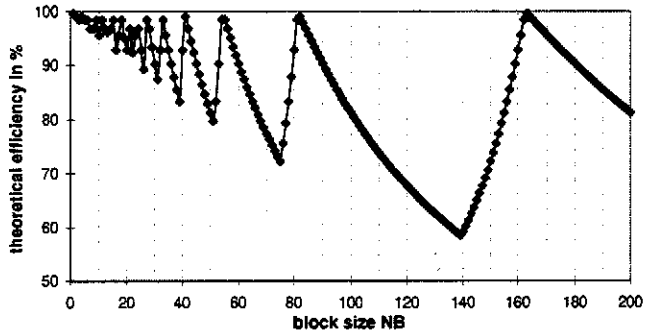

Fig. 10: Theoretical efficiency for the matrix filling process on $p = 6$ nodes with $\tilde{N} = 975$ rows as a function of the block size N_B.

For the first investigation, we used $p = 6$ nodes on an Intel Paragon. The matrix with $\tilde{N} = 975$ rows and $N = 1903$ columns is distributed among the PEs according to Fig. 4 with a block size N_B in the range $1 \leq N_B \leq \lceil \frac{\tilde{N}}{p} \rceil = 163$. If $N_B = 163$ is selected, the first 5 nodes have 163 rows of the matrix each, while the last process stores only the remaining 160 rows. This means that during matrix fill the last process has to wait $3T$ where T denotes the time to compute a matrix row (here we assume that the time T is the same for all the matrix rows). The theoretical efficiency based on the fact that some processes have to wait for others to finish the computations is then

$$\frac{5 \cdot 163 \cdot T + 1 \cdot 160 \cdot T}{6 \cdot 163 \cdot T} = \frac{975}{6 \cdot 163} = 99.69\,\%$$

(communication times etc. have been neglected). However, a block size of for instance $N_B = 140$ results in 7 blocks (6 blocks with 140 rows on nodes $1 \ldots 6$ and 1 last block with 135 rows on node 1). The theoretical efficiency is then much lower, namely

$$\frac{1 \cdot 275 \cdot T + 5 \cdot 140 \cdot T}{6 \cdot 275 \cdot T} = \frac{975}{6 \cdot 275} = 59.09\,\%.$$

The dependency of this theoretical efficiency on the block size N_B is plotted in Fig. 10. For our example here, useful block sizes are $163, 82, 55, 41, 33, 27, \ldots$. From this set the final choice is made automatically so that we expect the matrix solve to be most efficient (based on experiences on the different target machines, see Fig. 7). For the following investigations the maximum block size $N_B = 163$ is selected.

Fig. 11 shows the time required to compute the matrix elements on each of the 6 nodes. As already mentioned in Section 3.2, the CPU time required to compute a single matrix element a_{mn} differs over a wide range, since some of the integrals contain singularities whilst others are evaluated numerically with adaptive integration

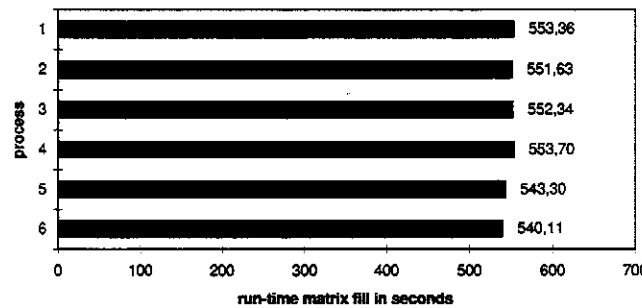

Fig. 11: Run–time for the matrix fill on the different nodes for $p = 6$ processes (without using the efficient fill technique), $\tilde{N} = 975$, $N = 1903$.

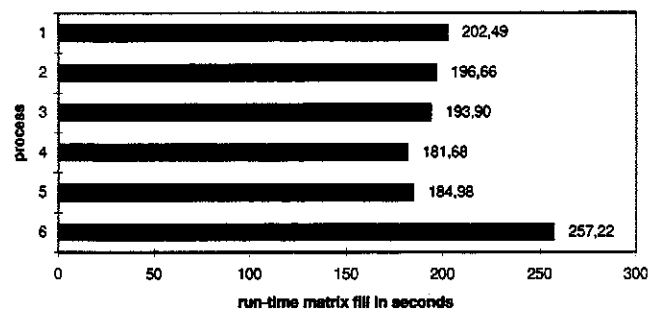

Fig. 12: Run–time for matrix fill on the different nodes for $p = 6$ processes (using the efficient fill technique but no mapping function), $\tilde{N} = 975$, $N = 1903$.

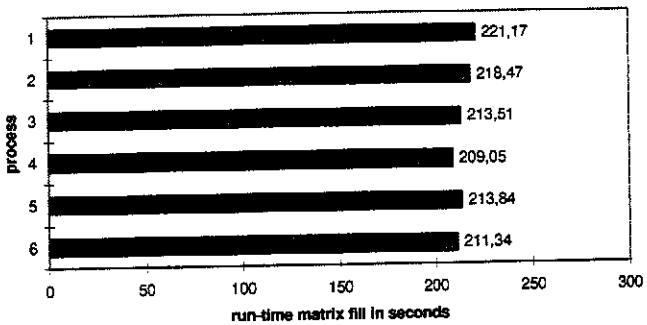

Fig. 13: Run–time for the matrix fill on the different nodes for $p = 6$ processes (using the efficient fill technique and a special mapping function to interchange matrix rows), $\tilde{N} = 975$, $N = 1903$.

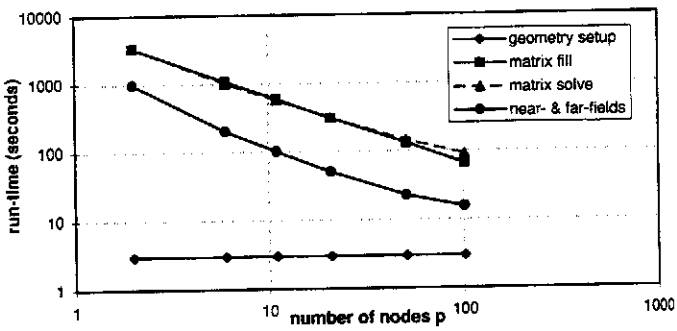

Fig. 14: Run–time of the various phases of the parallelized MoM solution for the example of the plane depicted in Fig. 9 as a function of the number of processes (no usage of symmetry and no efficient matrix fill), $N = 1903$.

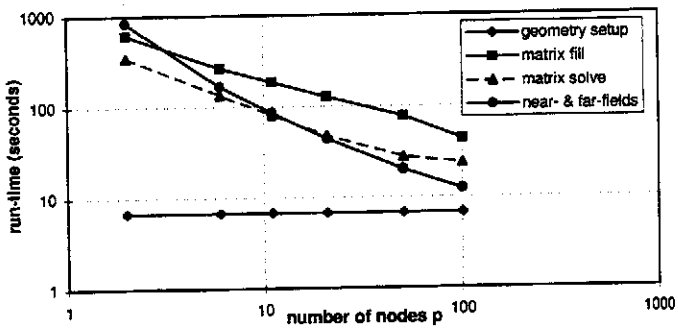

Fig. 15: Run–time of the various phases of the parallelized MoM solution for the example of the plane depicted in Fig. 9 as a function of the number of processes (usage of symmetry and efficient matrix fill), $\tilde{N} = 975$, $N = 1903$.

schemes. But it can be seen from Fig. 11 that on average all the 6 processes take about the same time to compute the matrix elements. The difference is only in the range of a few percent.

If, however, the efficient matrix fill technique is used, the situation changes as shown in Fig. 12. Of course the run times are reduced. The factor on the nodes 1 to 5 is 2.86 on average, which is quite satisfactory when compared to the reduction factor of 3.07 for the sequential version (using no symmetry, the reduction factor is usually somewhat higher, e.g. for the sequential version it is 3.32 for this particular example). The high ratio of $\frac{2.86}{3.07} \approx 0.93$ indicates that in most cases all the matrix elements to which the computed integrals (loop over triangular patches) have a contribution are kept on the same node. If we increase the number of processors, then this ratio will decrease. But even for $p = 100$ nodes (block size only $N_B = 10$), the ratio is still 1.75 on average, the maximum for one node is 2.62.

From Fig. 12 it can be seen that with the efficient fill technique the last process 6 takes about 30 % longer than the other nodes. The reason for this is that apart from the last 125 rows of the sub-matrices A_{1j} ($j = 1 \ldots 3$, see Fig. 2), the six sub-matrices A_{2j} and A_{3j} ($j = 1 \ldots 3$) with 22 and 13 rows, respectively, are kept entirely on this node and it takes longer to compute these elements.

The mapping function discussed in Section 3.2 can be used to overcome this problem. We also distribute the sub-matrices A_{2j} and A_{3j} to the different nodes, so that for instance for our example considered here with $N_B = 163$ the first node contains 157 consecutive rows of the sub-matrices A_{1j}, 4 rows of A_{2j} and 2 rows of A_{3j}. The resulting CPU–time requirement on the different nodes is depicted in Fig. 13. As opposed to Fig. 12 without mapping function, the total CPU time for the matrix fill can be decreased from 257.2 sec to 221.2 sec.

Another investigation is depicted in Figs. 14 and 15. We have executed FEKO again on the Intel Paragon, but now using a variable number p of nodes. The run–time of the different phases of the solution process according to the flow–chart in Fig. 1 has been measured and is plotted in Fig. 14 (symmetry was not used and no efficient matrix fill) and in Fig. 15 (usage of symmetry and efficient matrix fill) as a function of p. As explained in Section 3.1, the geometrical setup is performed sequentially by one process, i.e. the run–time for this part is constant. The other run–times show a satisfactory decrease with increasing p. The ideal case would be a decrement by a factor of 10 with 10 times more nodes, i.e. a linear diagonal line in the double logarithmic diagrams.

The example of the plane is relatively small to be considered a real HPC problem. However, we have used this problem to optimize the code and perform a large number of tests. Some data for larger problems are also available. For instance, for EMC purposes we analyzed the surface currents and field distribution inside a car when a plane wave is exciting the structure with $N = 15864$ basis functions. The model has no symmetry, therefore also

Table 1: Wall clock times for an EMC investigation of a car on a CRAY T3E with $N = \tilde{N} = 15864$ unknowns.

	64 nodes	128 nodes	ratio
geometry setup	8.12 sec	7.56 sec	1.074
matrix fill	390.23 sec	195.91 sec	1.992
matrix solve	1174.48 sec	616.86 sec	1.904
near– & far–fields	39.85 sec	20.66 sec	1.929
total solution time	1616.14 sec	844.77 sec	1.913

$\tilde{N} = 15864$. The resulting wall clock times are tabulated in Table 1. The last column gives the ratio of the times for the runs on 64 and 128 nodes, respectively. A value of 2 is expected for perfect scaling, our values show a very promising scaling factor (except for geometry setup which has not been parallelized).

For near– and far–field calculations, the performance of the dynamic load balancing scheme, i.e. the automatic adaptation to the different CPU speeds and to the actual load of the workstations, shall also be demonstrated. The problem used to illustrate the scheme is the analysis of a mobile communications antenna radiating close to the user's head. The hybrid MoM/Green's function technique as briefly discussed in Section 2.2 is applied. For this kind of formulation the CPU–time for the field computation is dominant, see Fig. 8. But also for some of the hybrid formulations, e.g. when the time consuming ray–tracing of the UTD must be performed, the CPU–times for near– and far–field computations and matrix fill dominate over the matrix solve time.

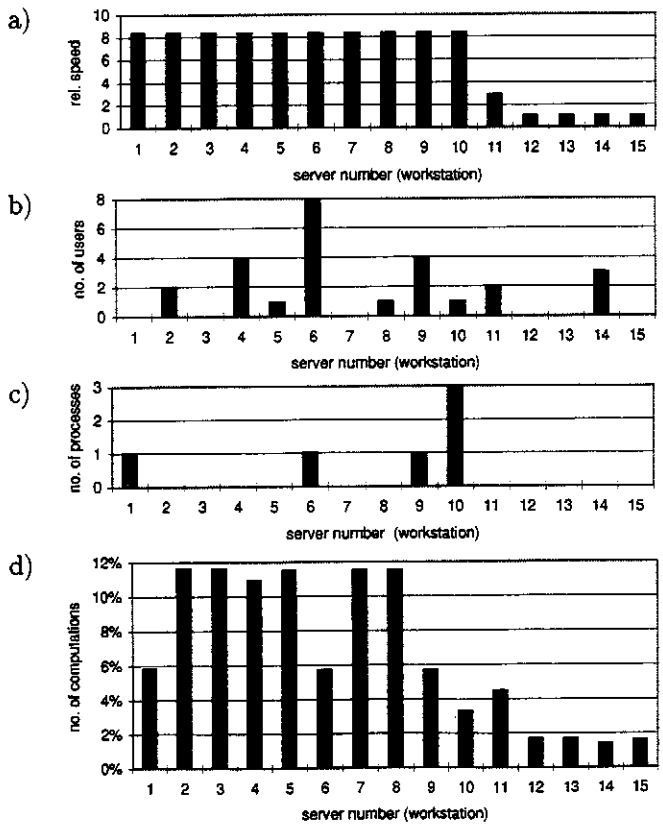

Fig. 16: Near– and far–field calculation with FEKO on a heterogeneous cluster of 15 different workstations during office hours with other users present in the network.

FEKO was executed on a heterogeneous cluster of 15 different workstations with different speeds (see Fig. 16 a) for relative speed) during office hours, so that there were also other users logged in (see Fig. 16 b) for the number) and there were also some background processes from other users running on some of the machines (see Fig. 16 c)).

Fig. 16 d) shows the relative number of computations carried out by the different server processes running on the 15 workstations. Even though there were no other background jobs on the last 4 workstations no. 12 to 15, they calculated the near– and far–field for only about 1.5% of the observation points each. The reason for this can be seen from Fig. 16 a): These 4 workstations are rather slow when compared to the other ones, so that the master process automatically takes this speed difference into account. Another example to demonstrate the dynamic load balance: If one compares the first two workstations of the same model and same speed, one can see that the process 1 computed only about half the number of field points when compared to process 2. Again Fig. 16 c) can be used to explain this behavior: There was one background process by another user on workstation 1.

Results for the speedup on a homogeneous cluster of identical workstations (no. 1 to 10 in Fig. 16 a)), which were exclusively available to the FEKO job during the test, i.e. there were no other users present and no back-

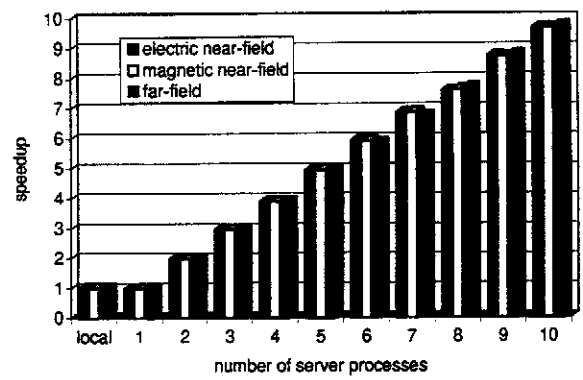

Fig. 17: Speedup for the near– and far–field calculation with a special Green's function on a homogeneous cluster of 10 IBM RS 6000 (Model 43P) workstations.