

Development of an Optimized Parallel Numerical Electromagnetics Code (NEC) and Its Implementation on the Swiss-T1 and Eridan Parallel Supercomputers

A. Rubinstein, F. Rachidi

Swiss Federal Institute of Technology
Lausanne, Switzerland
{Abraham.Rubinstein,Farhad.Rachidi}@epfl.ch

M. Rubinstein

University of Western Switzerland
Yverdon, Switzerland
Marcos.Rubinsteion@eivd.ch

Abstract - We present a parallel version of the Numerical Electromagnetics Code (NEC). The original routines, both for matrix factorization and for the solution of the system of equations, have been substituted by equivalents using the Scalable Linear Algebra Package (SCALAPACK). The use of these routines requires the bi-dimensional block-cyclic distribution of matrices on a rectangular processor grid. The distribution of the matrices assures optimal load balance among the processors. An example of application to an electrically large structure is shown. The code is portable to any platform supporting MPI or PVM. It has been implemented in two parallel supercomputers featuring different architectures to test portability. Timing and memory results are presented.

1. INTRODUCTION

Despite all the advances in computing technology, electrically large structures are still a problem to EMC modeling. The Numerical Electromagnetics Code (NEC) [1] is one of the most popular programs used in EMC research and antenna design and modeling. Its application to small structures such as antennas and some more complex 3D models is possible on fast computers, with a good amount of RAM and a voluminous and fast hard drive. However, the use of NEC in modern EMC problems becomes less evident. Highly advanced electronic equipment, radio-communication devices, satellite based applications, etc. are being installed in rather large structures such as automobiles, ships, buildings or airplanes. Even though it is virtually impossible to use the original code in such cases, the theoretical core remains applicable to any structure. The only restriction is memory and computing power.

A parallelization of NEC is a good alternative to extend the use of the program to larger and more sophisticated models.

Excell et al. [2] presented a modified version of NEC to optimize performance on a four-processor Cray X-MP computer. This is a shared memory machine with a low degree of parallelism. Nitch and Fourié [3] re-implemented NEC using C++ and produced a modified version capable of running in parallel. The parallel version of NEC we present works on distributed memory parallel supercomputers and is portable to any platform supporting MPI or PVM.

This paper is organized as follows: Section (2) contains a brief introduction to the NEC program and its current limitations due to memory restrictions. In section (3) we explain the parallel tools and the approach adopted to

produce a parallel version of NEC. In section (4) we present some timing and memory results obtained on the Swiss-T1 [4] and Eridan [5] parallel supercomputers belonging to the Swiss Federal Institute of Technology in Lausanne. Finally, section (5) is dedicated to discussion and conclusions.

2. THE NUMERICAL ELECTROMAGNETICS CODE (NEC)

The Numerical Electromagnetics Code is a user-oriented computer code based on the method of moments and written in FORTRAN for the analysis of the electromagnetic response of antennas and other metal structures [1,6,7]. It has been widely used for radio communications testing as well as antenna design with great success. With its ability to represent models by means of wires, the code should also allow the simulation of very complex 3D structures [1].

NEC produces an interaction matrix representing the system of integral equations needed to obtain the currents and fields. The number of elements in this matrix depends on the number of segments and patches that conform the model to be evaluated. This matrix is then reduced using LU factorization and, together with the excitation vector, it produces the final solution to the integral equations [1]. For a model consisting of N segments, a good approximation to the amount of memory required by NEC is given by this expression:

$$Mem = N^2 \cdot 16 \quad (1)$$

Notice the square growth of the memory requirements. The required amount of memory becomes important on many current computers at around 2000 segments since, for that particular case, approximately 64 MB of RAM are necessary to complete program execution. With 128

MB of RAM, the number of segments that fit in memory won't even reach 3000. In order to run even larger models, virtual memory can be used but at a very high price in terms of time. The use of the hard disk, by means of a swap file, will slow down the execution of NEC to unpredictable values (see section 4). An 'out of core' routine is embedded into NEC to optimize performance for this kind of case. The matrix is cut in pieces that are stored on the hard disk, following a special pattern. The operating system can then use all available RAM and NEC will take care of disk swapping. The out of core routine requires four times the normal RAM and, as a consequence, enormous swap files are created.

In order to overcome this kind of problems, the only solution seems to be more memory. Having as much memory as needed is the best way to assure an optimal execution. Operating systems are not capable of managing all the memory one would be prepared to buy. A version of NEC compiled for a very large number of segments will fail to start, because operating systems can't fit it in memory.

3. PARALLEL NEC

Computer resources at the time NEC was written were very different from those available today. With almost 10000 lines, this code has been conceived for linear operation.

It is much easier to write a parallel application from scratch than to transform a linear code into parallel. In cases when only a part of the code can be parallelized, the final program might be even slower than the original one. If the problem to be solved is general enough, some existing library can be found, adapted and applied.

NEC can be globally and very roughly divided in two parts. The first part reads geometrical information and calculates coefficients to produce G for the matrix equation $[G][I]=[E]$ to be solved. The second part solves this equation by means of the Gauss-Doolittle numerical method. This method has been divided in two routines; one does a LU decomposition of G [1]. The factorized matrix is used with E to determine the solution of the system. The major computational effort is factoring G into L and U . In fact, computation of the elements of the matrix G and the solution of the matrix equation are the two most time-consuming steps in computing the response of a structure, often accounting for over 90% of the computation time [1].

Using LAPACK one can create a linearly optimized version of NEC. Such version has been implemented and tested with success. Linear Algebra Package (LAPACK), is a very powerful linear algebra library designed for workstations, vector supercomputers, and shared memory parallel computers. It provides routines for

solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided [8]. There are several versions of LAPACK, specially optimized for different types of processors. If the right version is chosen, an optimized NEC will perform from 10 to 15 times faster than the original.

Scalable LAPACK (SCALAPACK) is a library that includes a subset of LAPACK routines redesigned for distributed memory parallel computers [9]. The parallel version of the NEC program was created following the same criteria used for the LAPACK optimization of NEC but using SCALAPACK. SCALAPACK assumes matrices are laid out in a two-dimensional block cyclic decomposition among the processors. This means that matrices G and E have to be cut into smaller sub-matrices that will be local to each processor.

In order to obtain good performance, a processor grid as square as possible should be used. This is not always possible as the number of processors available may vary from one machine to another. An automatic calculation of the squarest processor grid is done at the beginning of parallel NEC and is then used to declare and start the parallel environment.

The SCALAPACK routines require a message-passing environment such as MPI (Message Passing Interface) or PVM (Parallel Virtual Machine) [10,11]. Communications usually take place through a high-speed network but message passing should be minimized to decrease runtime.

Instead of generating G and then distributing, the original matrix creation routine was modified in order to distribute G block-cyclically as it is being calculated. Every processor on the grid holds the information needed in order to calculate the elements of G . One function will indicate to each processor if the element i, j being calculated corresponds to its local sub-matrix. If this is the case, a subroutine will calculate the new set of i, j for the same element to be assigned to the local matrix. At the end of the matrix generation routine, each processor holds its local version of the interaction matrix.

The dimensions of the interaction matrix array are assigned at runtime based on the calculated dimensions of the local sub-matrices, which depend on the number of processors in use. The code adapts automatically to the environment and to the number of processors in use. The dimensions of the rest of the arrays are still indicated at compilation time. However, the maximum number of segments (MAXSEG) and the maximum core storage (MAXMAT) should be identical and equal or higher than the number of segments on the model to be run.

Memory requirements for each one of the local sub-matrices are inversely proportional to the number of processors. For example, a model consisting of 11500 segments requires approximately 2 GB of RAM running on only 1 processor. If 16 processors with 128 MB of RAM each are used, the memory gets shared and the model can be successfully executed without swapping matrix information between memory and disk. As a general rule, an estimation of the memory requirements at a local node for the parallel version of NEC can be calculated with this expression:

$$Mem = \frac{N^2 \cdot 16}{P} \quad (2)$$

where P is the number of processors being used at runtime.

The LU decomposition in NEC is done in a routine called FACTR. After this routine is executed, the interaction matrix G can be expressed as:

$$[G] = [L][U] \quad (3)$$

and the matrix equation becomes:

$$[L][U][I] = [E] \quad (4)$$

The solution is then computed in the routine SOLVE, solving for F by forward substitution in

$$[L] = [F][E] \quad (5)$$

and solving for I by backward substitution in

$$[U] = [I][F] \quad (6)$$

Both the FACTR and SOLVE routines have been substituted by SCALAPACK equivalents.

In the case of the routine FACTR, a local sub matrix is passed as an argument. In order to execute the LU decomposition all processors need to have information about the original global matrix, the number of processors running as well as the local dimensions of the

sub matrices. All this information is calculated at runtime and does not need to be entered by the user.

For the routine SOLVE, the already decomposed and distributed G is passed as an argument. However, vector E needs to be block cyclically distributed as well. This task is carried out by a routine that automatically assigns values and indices. After solution I is found, data is put back together by means of a block cyclic composition routine so that normal program execution can continue and solutions can be printed to the output file.

4. TIMING AND MEMORY RESULTS

Our testing has been carried out using one PC with a PIII running at 750 MHz with Windows NT 4.0 and 256 MB of RAM and 2 distributed memory machines, the Swiss-T1 [4] and Eridan [5]. The Swiss-T1, the sixth fastest supercomputer in Switzerland, features 35 Compaq DS20E dual Alpha processor boxes at 500 MHz each. They all have 1 GB of RAM and two 9.1 GB SCSI-U2 hard disks. Two of those boxes are used as interactive front-end, another one for upgrades, development and maintenance and the other 32 are used as calculation nodes. This 64-processor calculation part is connected by a Tnet network, which is a reliable, high performance, low latency network to build commodity-supercomputing clusters. The Eridan supercomputer has 128 MIPS R14000 RISC super-scalar CPU's running at 500 MHz each and distributed among 32 nodes. Each processor has 512 Mbytes of RAM for a total of 64 Gbytes. Total capacity of hard disks goes up to 1.5 TB for system and users.

The first NEC input file used for testing consisted of 6753 segments. The Eridan computer was only used to test portability of the code and no timing or memory results are presented here. On the Swiss-T1, the same input file was executed with 4, 8, 16, 24, 32 and 36 processors. Table 1 shows the relation of the number of processors with matrix filling, matrix factorization and run times, local memory and total memory.

Table 1 – *Timing and memory as a function of the number of processors for a 6753 segments test case.*

Number of Processors	Matrix Filling Time	Matrix Factorization time	Run Time (2 freq. Steps)	Mem/processor (MB)	Total Memory (MB)
1 – PC	10 min	6 h 47 min	14 h 6 min	62*	2790
4 – T1	2 min 56 sec	5 min 24 sec	16 min 7 sec	193.9	769.34
8 – T1	2 min 42 sec	2 min 45 sec	11 min 37 sec	104.4	816.04
16 – T1	2 min 34 sec	1 min 31 sec	8 min 52 sec	57.28	883.76
24 – T1	2 min 50 sec	1 min 39 sec	8 min 29 sec	39.55	905.81
32 – T1	2 min 33 sec	1 min 27 sec	8 min 16 sec	33.72	1019.45
36 – T1	3 min	1 min 3 sec	8 min 34 sec	28.45	988.57

*Approximation based on a core storage of 2000 segments.

On the PC, we used NEC-Win Professional and we specified that the maximum core storage should be limited to 2000 segments since the total model requires about 700 MB. This would leave enough space for the operating system and services and should allow flawless calculation using the out of core routine included in NEC.

One first thing to notice is the fact that the use of only 4 processors reduces run time to a small fraction of the original, only 1.9%. Of course, this is mainly due to the fact that the PC case is forced to use disk swapping. Per processor memory behaves as expected. It is important to notice that the memory used by each single processor depends on the particular dimensions of its local matrix and that the values displayed on the table are the biggest. The total memory for each parallel configuration is higher than the PC theoretical requirements because the rest of the arrays that are used in the parallel version need to be declared with the total size of the model.

In figures 1,2 and 3, we present the matrix filling time, matrix factorization time and the total run time as a function of the number of processors. From these figures, we observe that the matrix filling is not scalable and, the use of too large a number of processors can compromise the total run time. We still see that matrix factorization is even faster as the number of processors increases but this gain in speed is cancelled by the time loss in matrix filling.

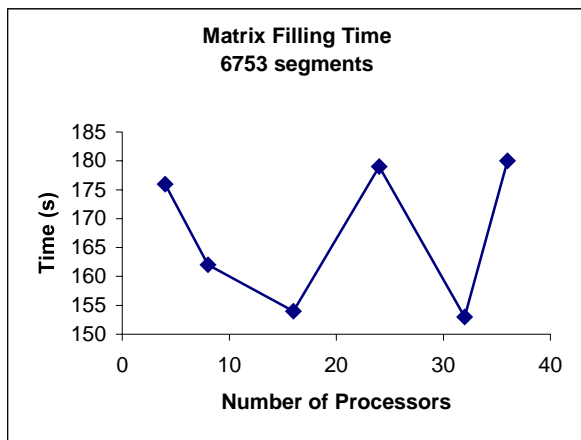


Fig. 1 – Matrix filling time as a function of number of processors.

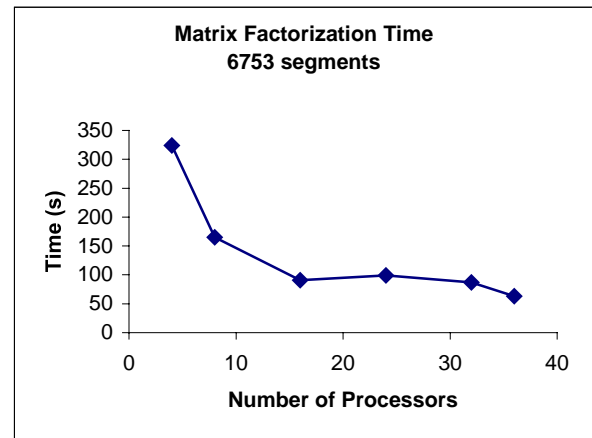


Fig. 2 – Matrix factorization time as a function of number of processors.

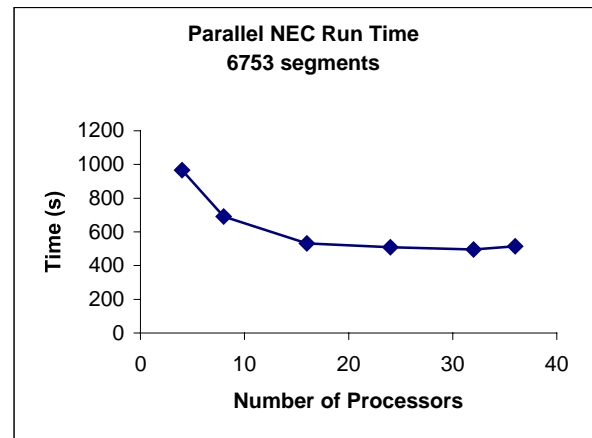


Fig. 3 – Total run time as a function of number of processors.

A NEC file consisting of 23449 segments was also tested. With this amount of segments, approximately 8.4 GB of RAM are required. This example cannot be executed on a PC, unless disk swapping is used. For such case over 35 GB of disk space would be used. The same file was launched on the PC using NEC-Win Professional and we stopped program execution after 168 hours. No solution was produced. Table 2 summarizes information about matrix filling time, matrix factorization time, total run time and memory.

Table 2 - Timing and memory as a function of the number of processors for a 23449 segments test case.

Number of Processors	Matrix Filling Time	Matrix Factorization time	Run Time (1 freq. Step)	Mem/processor (MB)	Total Memory (MB)
1 – PC	*****	*****	Stopped after 168 h	*****	~36000
32 – T1	35 min 50 sec	32 min 10 sec	68 min 24 sec	282.72	8940.32

5. DISCUSSIONS AND CONCLUSIONS

The Numerical Electromagnetics Code, one of the most important codes in the EMC research activities, has been parallelized using SCALAPACK, a scalable library for distributed memory supercomputer clusters.

The code is portable to any computer platform supporting MPI or PVM parallel environments. It can be run on heterogeneous clusters of computers running on several versions of UNIX, LINUX and WINDOWS.

The construction of the matrix has been modified in order to produce local sub-matrices ready to be used by the also substituted FACTR and SOLVE routines.

The possibility to run very complex models without the use of disk swapping produces very important speed improvements. Memory sharing also allows the execution of models that are impossible to run on a single processor machine.

The amount of processors to be used in calculation must be carefully selected. The appropriate selection of the number of processors is the subject of further study.

Acknowledgments – This work was carried out as part of the GEMCAR project, a collaborative research project supported by the European Commission under the competitive and Sustainable Growth Programme of Framework V (EC contract G3RD-CT-1999-00024).

REFERENCES

1. G. Burke and A. Poggio, 'Numerical electromagnetics code - method of moments', Livermore CA: Lawrence Livermore National Laboratory, Report No. UCID-18834, 1981.
2. P. S. Excell, G. J. Porter, Y. K. Tang and K. W. Yip, 'Re-working of two standard moment-method codes for the execution on parallel processors', International Journal of Numerical Modelling: Electronic Networks, Devices and fields. Vol 8, 1995.
3. D.C. Nitch, A.P.C. Fourié, 'Parallel implementation of NEC' Applied Computational Electromagnetics Society (ACES) Journal, Vol 9, No 1, p51-57, March 1994.
4. 'The Swiss-T1 parallel supercomputer', <http://tone.epfl.ch>.
5. 'The Parallel calculation server Eridan', <http://sewww.epfl.ch/SIC/SE/servcentraux/origin.html>.
6. A. J. Poggio and E. K. Miller, 'Computer Techniques for Electromagnetics', Chap. 4, Edited by R. Mittra, Summa, 1987.
7. T. R. Ferguson, T. H. Lehman and R. J. Balestri, 'Efficient Solution of Large Moments Problems: Theory and Small Problem Results', IEEE Transactions on Antenna Propagation, Vol AP-24, No 2, pp 230-235, March 1976.
8. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, 'LAPACK users' guide – Third edition', SIAM, 1999.
9. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, 'SCALAPACK users' guide', SIAM, 1997.
10. 'The Message Passing Interface (MPI) standard homepage', <http://www-unix.mcs.anl.gov/mpi/>.
11. 'Parallel Virtual Machine homepage', http://www.epm.ornl.gov/pvm/pvm_home.html.