PARALLEL IMPLEMENTATION OF THE NUMERICAL ELECTROMAGNETICS CODE

D.C. Nitch and A.P.C Fourie
Department of Electrical Engineering
University of the Witwatersrand
South Africa

ABSTRACT

NEC2 is a 10 000 line, public domain, FORTRAN IV program for electromagnetic analysis. This program has been adapted for use on transputer networks of various dimension. The FORTRAN code has been modified and translated into OCCAM where necessary. Parallel algorithms were developed and implemented for each NEC2 function in order to optimise efficiency. Execution efficiencies in excess of 80% were attained.

1 INTRODUCTION

The Numerical Electromagnetics Code NEC2 (Burke, 1981a, 1981b, 1981c) was originally intended to run on Mainframe computers but has recently been ported to run on personal computers (PC). The main problem with running NEC2 on a PC is that it is exceedingly slow and insufficient memory limits the size of structures that can be analyzed. Stellenbosch University (Le Roux, 1988) compiled NEC to run on a single T800 INMOS transputer and showed that this ran much faster than the PC version. NEC2 was later extended (Nitch and Fourie, 1990) to run on a fixed 16 transputer network with only two of the main algorithms rewritten to execute in parallel. The following disadvantages were apparent:

- The matrix was returned in full to the host transputer which meant that problem size was limited by the memory on the host transputer.
- Many algorithms were still sequential.
- The transputer network was of fixed dimension

This paper presents solutions to these problems. It is acknowledged that the transputer (T800) is quite an old chip and is relatively slow. The parallel algorithms discussed in this paper, however, may also be applied to contemporary distributed-memory parallel processing machines.

2 BACKGROUND TO NEC

Structures are modelled by wire segments with options to include sources, loads and networks. These structures may be analyzed in various environments (free space, ground etc). Essentially NEC calculates the interaction between the N wire segments making up the structure and hence obtains an NxN matrix. An excitation vector is then calculated as a function of the sources. Solving this matrix equation yields the currents on each segment in the structure. Mathematically this may be expressed as :

$$\begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1N} \\ Z_{21} & Z_{22} & \cdots & Z_{2N} \\ . & . & . & \\ . & . & . & \\ . & . & . & \\ Z_{N1} & Z_{N2} & \cdots & Z_{NN} \end{bmatrix} \times \begin{bmatrix} I_1 \\ I_2 \\ . \\ . \\ . \\ I_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_N \end{bmatrix}$$

where Z_{ij} is the interaction between segment i and j and is a function of the wire geometry
I_i is the current on segment i, which is obtained by solving the equation.
E_i is the excitation on segment i which is calculated from the specified sources.

Once the currents on the segments are found, other electromagnetic characteristics such as electromagnetic (EM) fields may be found.

The sequence of possible events carried out by the program is as follows:

- The structure geometry is read from a file
- The Z-matrix is filled by calculating the interaction between segments. This requires N^2 operations each of which involves numerical integration.
- The Z-matrix is factorized which requires N^3 simple operations.
- The E-vector is calculated from the sources.
- Solve for currents which requires N^2 simple operations.
- The effect of I networks are found by I solve operations on the Z-matrix and modifications of the E-vector.

- A single EM field value is calculated by summing the effect of the structure currents at that point. This involves *MN* operations for M field points.

NEC allows structure symmetry to be exploited in order to reduce computational effort for filling and factoring the matrix. This involves alteration to the fill, factor and solve of the matrix which will be discussed in more detail.

3 PARALLEL ARCHITECTURE AND COMMUNICATION

The transputer (INMOS, 1989) is a single chip microprocessor with 4 communication links for direct communication to other transputers. The general architecture used for the parallel implementation is shown in Figure 1. It should be noted that this architecture may be generalised for processors that have more than 4 communication links.

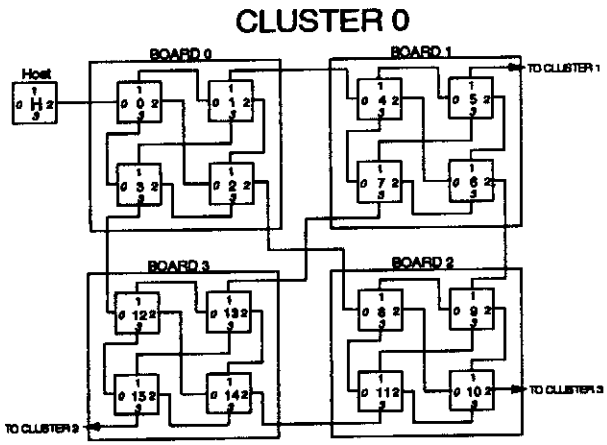


Figure 1: The transputer network.

The reasons for choosing this architecture are :

- the path from any processor to any other processor in the network is minimized when compared to other networks investigated (various meshes, hypercubes etc.).
- the algorithm controlling the communication is simple.
- extending or reducing the network dimension is easy. This extension or reduction may, in most cases, be achieved by adding or removing single processors from the network.

Three general communication strategies were required:

- Broadcast from host to network.
- Send from all transputers to host.
- Broadcast from any transputer to the rest of the network.

Each transputer has knowledge of the network size and a number identifying its position in the network. From this information it is possible to deduce the network interconnections for any processor. All unconnected links are ignored when executing the algorithms described below.

3.1 Broadcast from host to network.

All transputers wait on link 0 for a message. Upon receipt:

- the main transputer on a board whose identity is (*proc.id REM 4 = 0*) sends the message on links 1 to 3. (The variable *proc.id* is the number identifying the processor in the network, whilst the *REM* function calculates the remainder of the division of *proc.id* and *4*.)
- otherwise if the processor is not the main transputer on a board, then send out to next board or cluster if connected.

3.2 Send from all transputers to host

All transputers send their own message out on link 0. Transputers listen on links 1 to 3 for messages which will be routed through them and passed on through link 0.

3.3 Broadcast from any transputer to network

The broadcasting transputer sends its message on all four links. Other transputers redirect the message in the following way:

- If the broadcaster is on the same board as the receiver then only redirect to other boards or clusters
- If the broadcaster is not on the same board as the receiver then the receiving processor redirects the message to the rest of the processors on its board and to connected clusters (not other connected boards).

4 IMPLEMENTATION OF PARALLEL ALGORITHMS

4.1 Matrix filling

Inherently the matrix filling requires each transputer to have knowledge of the structure and the environment in which it is situated. This information is broadcast from the host to the transputer network. This enables each transputer to calculate any matrix element independently of other transputers. It is important to decide on which part of the matrix each processor should fill. The following points require consideration :

- Each processor should calculate approximately the same number of matrix elements.

- Each processor should calculate that part of the matrix which it requires for later operations. The reason being that the matrix occupies a large portion of the memory and hence it is difficult to reshuffle efficiently. In the previous implementation (Nitch and Fourie, 1990), the matrix was returned to the host for reordering. This method obviously did not make efficient use of the distributed memory.

The matrix was filled by rows because:

- both column and row distribution reduce the communication overhead during matrix factoring when compared to the overhead when the matrix is divided into blocks.
- later factoring of the cyclic block matrix (as a result of structure symmetry) requires that it be in row form to reduce communication overhead.

Wrap mapping of rows was employed for load balancing in the factoring algorithm since the diagonal elements of the Z-matrix were generally the largest.

4.2 Matrix factoring

The matrix in NEC is solved using Gaussian elimination with back substitution. The parallel implementation of this algorithm was based on the algorithm presented in a paper by Giest and Romine (1988) and is shown in Figure 2.

```
FOR k = 0 TO n-1
  determine pivot row using parallel search
  update permutation vector
  IF (I own pivot row)
    broadcast pivot row
  ELSE
    receive pivot row
  ENDIF
  FOR (all rows i>k that I own)
    l_ik := a_ik/a_kk
    FOR j = k+1 TO n-1
      a_ij := a_ij - l_ik a_kj
    ENDFOR
  ENDFOR
ENDFOR
```

Figure 2: The parallel algorithm presented by Geist and Romine.

Where *l* is a temporary vector housing the pivot column.

Implementing this algorithm on a network of transputers requires that, for good load balancing, the rows of the matrix are wrapped onto the processors (i.e for a network of 16 processors the first processor should have rows 1, 17, 33 etc the second processor should have rows 2, 18, 34 and so on). The reason for the wrapped row mapping is that once a processor has operated on the pivot column (column k), row k is not used in any calculation for the completion of the algorithm. The work load of the processor is therefore reduced. Thus to ensure that the processors each have equal loads throughout the execution of the algorithm, a wrapped row mapping is employed.

The communication between processors during the computation involves only the broadcast and reception of the pivot column.

4.3 Matrix solve

Consider the lower triangular linear system

$$Lx = b$$

where L is a lower triangular matrix of order n
b is the right-hand side vector of dimension n
x is the unknown solution vector

The serial solution of this system may be represented by the code

```
FOR i = 1 TO n
  FOR j = 1 TO i-1
    b_i = b_i - x_j L_ij
  ENDFOR
  x_i = b_i/L_ii
ENDFOR
```

Parallel matrix solve algorithms have been developed by Guangye and Coleman (1988), and Heath and Romine (1988), and others. The matrix solve routine used in this implementation was based on an algorithm presented by Heath and Romine (1988) and is shown below :

```
FOR j = 1 TO n
  IF (I have row j) THEN
    x_j = b_j/L_jj
  ENDIF
  fan-out (x_j, map(j))
  FOR (all rows i>j that I own)
    b_i = b_i - x_j L_ij
  ENDFOR
ENDFOR
```

The function *map(j)* relates a processor to the row j. Thus the line *fan-out* (x_j, *map(j)*) sends the message x_j to the processor with row j.

4.4 Field calculations

Given below is the sequential code used in NEC for the far field calculation.

```
FOR phi = 1 TO noOfPhiPoints
  FOR theta = 1 TO noOfThetaPoints
    FOR i = 1 TO N
      calculate field at (phi,theta) due to current
i
        vectorially add field at (phi, theta)
    ENDFOR
    write out field at (phi, theta)
  ENDFOR
ENDFOR
```

At the end of a single pass through the inner loop, the results are written to disk.

4.4.1 Field calculation code for the network processors

There are a number of ways one can split up the nested *FOR* loops for execution in parallel. Either the phi, the theta or both loops may be divided amongst the processors. The number of theta and number of phi points are not necessarily the same for each radiation pattern request. Thus splitting the phi or theta loops could produce very poor performance figures.

The method that is employed finds the total number of radiation points for the field calculation and hence reduces it to a single loop. These points are divided amongst the processors and each processor finds the field at these points. Implementing this method requires the decoding of radiation pattern point number to the (theta, phi) point in space.

```
FOR points = 1 TO noOfPhiPoints*noOf-
ThetaPoints
  calculate phi
  calculate theta
  FOR i = 1 TO N
   calculate field at (phi,theta) due to current
i
    vectorially add field at (phi, theta)
  ENDFOR
  write out field at (phi, theta)
ENDFOR
```

Splitting the loop amongst transputers in order to achieve parallel execution has some difficulties. The transputers in the network do not have access to disk and memory will be wasted if all the fields are stored. The loop must hence be further subdivided such that a specified number of fields are calculated before information is relayed to the host.

An algorithm employing this subdivision was developed for the transputer network.

4.4.2 Field calculation code for the host processor

The principle behind the algorithm for the host processor is as follows :

```
FOR i = 1 TO noOfGroupsOfPoints
   request network to find radiation pattern
for
  group of points
  FOR p = 1 TO noOfProcessors
   FOR j = 1 TO noOfPointsPerProcessor
      receive radiation pattern
      write result to disk
   ENDFOR
  ENDFOR
ENDFOR
```

The deficiency in this algorithm is that it has two main serial components. The first is the request to the network to find the radiation pattern for a group of points and the second is the writing of the results to disk. Thus the network waits for the host to write the results to disk before computing the next set of fields.

This serial component can be masked by buffering the radiation patterns received from the host. A request to find the next set of radiation pattern points may be made before writing the present results to disk. Thus at the expense of memory, the parallel execution can be sped up. An algorithm employing such a buffering scheme was implemented in the parallel NEC.

4.5 Networks

Networks are evaluated in NEC through the use of a small network matrix with dimension equal to the number of networks.

The following steps need to be performed for the networks in the structure:-

```
Generate the RHS vector of the network matrix
equation (Step a)
Fill network matrix (Step b)
FOR i = 1 TO noOfNetworks
   Solve the Z-matrix to get a modification vector
(Step c)
  use modification vector to adapt network matrix
(Step d)
ENDFOR
Factor the network matrix (Step e)
Modify RHS of network matrix equation (Step f)
Solve network equations for voltage across ports of
those networks without voltage soures. (Step g)
```

The computationally time consuming portion of this solution is filling the network matrix (steps b, c and d). Steps a and b are carried out by the host while the transputer network is filling and factoring the Z-matrix. Step c requires the use of the factorised Z-matrix to find the modification vector. While c is performed in parallel on the network, d can be made to execute concurrently on the host.

Arranging the code in this manner enables the host processor and the network to work in parallel. First, the host processor fills the network matrix while the network fills and factors the much larger Z-matrix. Then the network finds the modification vector (using the parallel algorithm for solving discussed before), while the host processor uses a previously computed modification vector to update the network matrix. Using this technique the time required to fill the network matrix is approximately equal to the time required to find the modification vectors.

The remaining steps are comparatively fast. There is little point in factoring the network matrix on the transputer network since the matrix is generally of small dimension (30x30) and the efficiency of the network when factorising such matrices is low. The solution of a matrix of this dimension is not very time consuming.

4.6 Cyclic Block matrices

Memory and Computation time is saved when the structure being simulated is symmetric. The time required to fill the matrix is reduced since only the interactions between those segments in the first symmetric section and the structure are calculated. Hence the filling routine is simplified to :

```
FOR i = 1 TO noOfSegs
  FOR j = 1 TO noOfSegsInSymSection
    find interaction between segments i and j i.e
Zij
  ENDFOR
ENDFOR
```

The resulting matrix is structured as shown below.

$$[A_1 \quad A_2 \quad \dots \quad A_M] \times \begin{bmatrix} I_1 \\ I_2 \\ \cdot \\ \cdot \\ \cdot \\ I_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_N \end{bmatrix}$$

where $$A_i = \begin{bmatrix} Z_{11} & Z_{12} & \dots & Z_{1S} \\ Z_{21} & Z_{22} & \dots & Z_{2S} \\ \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \\ Z_{S1} & Z_{S2} & \dots & Z_{SS} \end{bmatrix}$$

and s is the number of segments in a symmetric section.

The solution of this system of equations is accomplished using the following steps:

- Each submatrix is combined using the formula $A_i = \sum_{k=1}^{M} S_{ik} A_k$.

  where A_i is the i^{th} submatrix.
  S_{ik} are factors calculated according to the type of symmetry.
  M is the number of submatrices
- Each submatrix is factored.
- The excitation vector is filled in the normal fashion.
- Each submatrix equation is solved.
- The resulting solutions are combined using $I_i = \sum_{k=1}^{M} S_{ik} I_k$ to find the currents on the segments.

  where I_i is the solution to the i^{th} submatrix equation.

Execution of these steps in parallel may be considered in two sections, namely, the filling and the solution of the submatrices.

The filling of the matrix is split up by asking each processor in the network to fill specific rows. It is important that processors fill rows of the Z-matrix, since the formula used to combine the submatrices operate on the rows of the submatrices. Thus, once the matrix has been filled, each processor can combine the elements of its portion of the submatrices without having to communicate with other processors.

Solving the submatrix equations is accomplished by sequentially factoring and solving each of the submatrix equations on the network. The resulting solutions are then combined using $I_i = \sum_{k=1}^{M} S_{ik} I_k$ to give the currents on the structure.

5 PARALLEL PERFORMANCE

In assessing the performance of an algorithm on a network of transputers, it is useful to compare the time taken to complete the task on the network to the time taken on one

processor. Thus in gauging the performance of the parallel NEC, the efficiency and speedup of the computation were calculated.

Efficiency and speedup are defined as follows :-

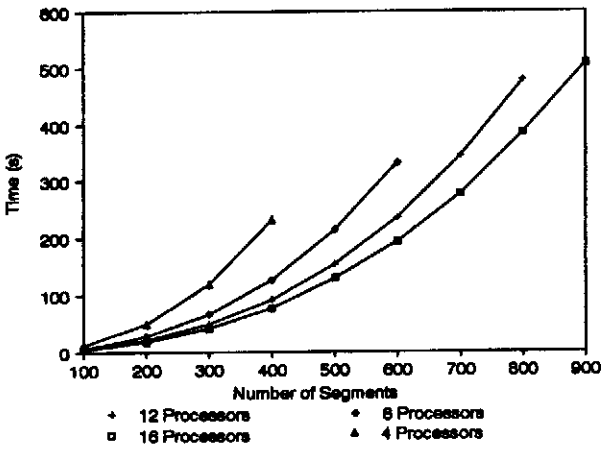
$$Efficiency = \frac{time\ taken\ to\ complete\ task\ on\ one\ processor}{(time\ taken\ to\ complete\ task\ on\ p\ processors) \times p}$$

$$Speed\ Up = \frac{time\ taken\ to\ complete\ task\ on\ one\ processor}{time\ taken\ to\ complete\ task\ on\ p\ processors}$$

where the time taken to complete the task on p processors is made up of the time spent communicating between processors and the time spent doing the computation.

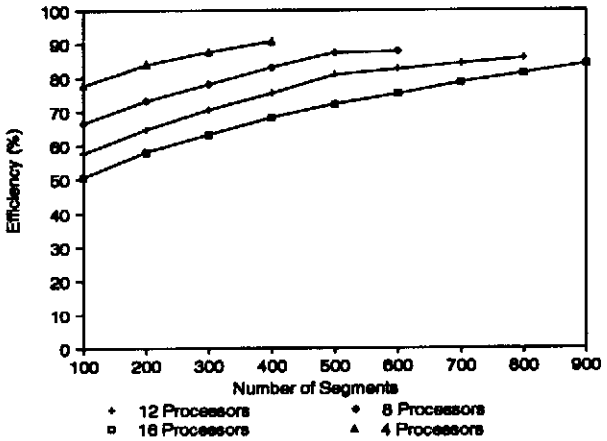
Calculating the efficiency and speedup of the parallel NEC requires the times for simulating problems on both a single processor and on the network. Since full use of the distributed memory is used in the simulations, a single transputer with the same amount of memory as the network should be used. However, a processor with this amount of memory was not available. It is possible to predict the time that it would take for a single processor to do a simulation. Thus the efficiency and speedup graphs use some predicted values.

Graph 1 shows the times taken to simulate structures of various electrical size on processor networks consisting of 4, 8, 12, and 16 processors.

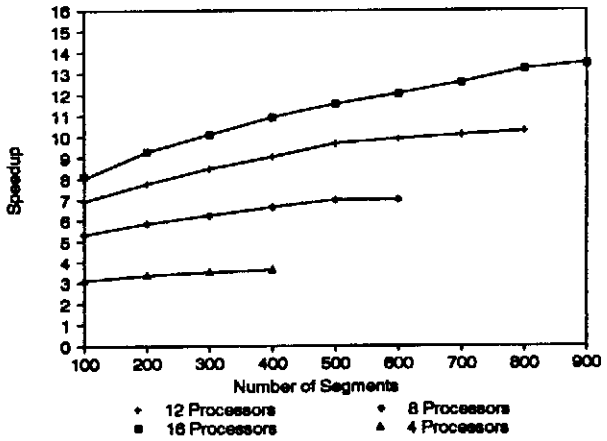


Graph 1 : The times taken to simulate structures of varying electrical size.

Graph 2 gives the efficiency of the transputer networks and graph 3 gives the speedup of the simulations.



Graph 2 : The efficiency of the transputer network.



Graph 3 : The speedup of various transputer networks for structures of varying electrical size.

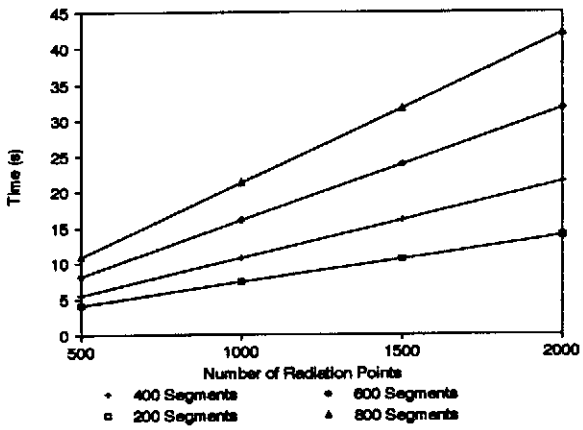
5.1 Radiation Pattern Performance

Analysis of the performance of the transputer network when calculating radiation patterns is difficult since there are many factors influencing the performance. These factors include :

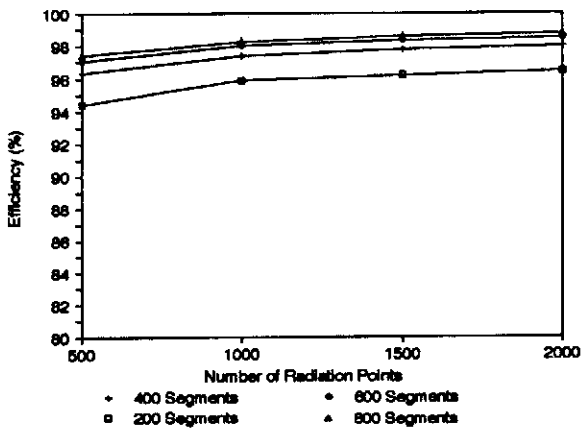
- the number of radiation pattern points requested.
- the number of segments in the structure.
- the number of processors in the network.
- the number of points returned to the host at a time.
- the speed of the disk.

When computing a large number of radiation pattern points for a structure consisting of a few segments there is a bottle-neck at the disk since the network calculates the points faster than they can be output to disk.

The performance of the far field algorithms are illustrated in graphs 1 and 5. Graph 4 shows the time required to find a number of radiated fields on structures of varying dimension on 16 processors. Graph 5 gives the efficiency of the process.



Graph 4 : The time required to find a radiation pattern on a 16 processor network.



Graph 5 : The efficiency of the radiation pattern calculation on a 16 processor network.

6 CONCLUSION

The speedup attained by the transputer network indicate that it is possible to significantly reduce the execution time of NEC by distributing the program onto a network of processor and executing the code in parallel. Full use of the distributed memory was made by careful consideration of the operations to be performed on the largest data structure (the interaction matrix).

7 REFERENCES

Burke G.J., Poggio A.J. (1981a) *"Numerical Electromagnetics Code (NEC2) - Method of Moments; Part I : Program Description - Theory"*, San Diego : Naval Oceans Systems Center, Tech Doc 116.

Burke G.J., Poggio A.J. (1981b) *"Numerical Electromagnetics Code (NEC2) - Method of Moments; Part II : Program Description - Code"*, San Diego : Naval Oceans Systems Center, Tech Doc 116.

Burke G.J., Poggio A.J. (1981c) *"Numerical Electromagnetics Code (NEC2) - Method of Moments; Part III : Program Description - User Guide"*, San Diego : Naval Oceans Systems Center, Tech Doc 116.

Giest G.A., Romine C.H. (1988) "Parallel LU Factorization", *SIAM J. Sci. Statist. Comput.*, Vol.9, No.4, pp.639-649.

Guangye L.I., Coleman T.F. (1988) "A Parallel Triangular Solver for a Distributed Memory Multiprocessor", *SIAM J. Sci. Statist. Comput.*, Vol.9, No.3, pp.485-502.

Le Roux J.J. (1988) "Numerical Electromagnetics Computation using the INMOS T800 Transputer on an Olivetti M24 Personal Computer", *Applied Computational Electromagnetics Society Journal and Newsletter*, Vol.3, No.2, pp.88-94.

Heath M.T., Romine C.H. (1988) "Parallel Solution of Triangular Systems on Distributed Memory Multiprocessors", *SIAM J. Sci. Statist. Comput.*, Vol.9, No.3, pp.589-600.

Nitch D.C., Fourie A.P.C. (1990) "Adapting the Numerical Electromagnetics Code to Run in Parallel on a Network of Transputers" *ACES Journal*, Vol.5, No.2, pp.76-86.