MPI based Parallel Method of Moments Approach for Microstrip Structures Analysis

FRANCISCO CABRERA
Departamento de Señales y Comunicaciones
Universidad de Las Palmas de Gran Canaria
Campus de Tafira S/N, Las Palmas 35017
SPAIN
fcabrera@dsc.ulpgc.es

EUGENIO JIMENEZ
Departamento de Señales y Comunicaciones
Universidad de Las Palmas de Gran Canaria
Campus de Tafira S/N, Las Palmas 35017
SPAIN
ejimenez@dsc.ulpgc.es

Abstract: In this paper we will present a parallel Method of Moments (MoM for short) technique using the MPI library. Here, the MoM is used to analyze microstrip structures. The main goals to achieve are efficient parallel coefficient computation and efficient linear equation system solving. The efficiency and accuracy of the parallel-processing MoM code is analyzed through several examples with two data distribution using ScaLAPACK library.

Key–Words: MoM, microstrip, parallel, MPI, ScaLAPACK

1 Introduction

The MoM [1] provides a numerical solution to linear equation problems. The form of the linear equation can be written as

$$F(g) = h \tag{1}$$

in which F is a linear operator, h is known, and g is to be determinated. Let g be expanded in a series of functions $g_1, g_2, g_3 \ldots$ in the domain of F

$$g \simeq c_1 g_1 + c_2 g_2 + \cdots + c_N g_N = \sum_{n=1}^{N} c_n g_n \tag{2}$$

where the c_n are constants. The f_n are called *expansion functions* or *basis functions*. Substituting (2) into (1) and using the linearity of F, one has

$$\sum_{n=1}^{N} c_n F(g_n) = h \tag{3}$$

If (3) represents an approximate equality, then the difference between the exact and approximate equation is

$$R = \sum_{n=1}^{N} c_n F(g_n) - h \tag{4}$$

which is called the *residual R* and has to be minimized. Now define a set of *testing functions* or *weighting functions*, w_m, in the range of F. Take the inner product between R and w_m and set all the weighted residuals equal to zero.

$$\langle w_m | R \rangle = 0 \tag{5}$$

Finally, substitute (4) into (5) to obtain

$$\sum_{n=1}^{N} c_n \langle w_m | F(g_n) \rangle = \langle w_m | h \rangle \quad m = 1 \cdots N \tag{6}$$

This set of equations can be written in matrix form as

$$[F_{mn}]\,[c_n] = [h_m] \tag{7}$$

in which F_{mn}, c_n and h_m can be written as

$$[F_{mn}] = \begin{bmatrix} \langle w_1|F(g_1)\rangle & \cdots & \langle w_1|F(g_N)\rangle \\ \langle w_2|F(g_1)\rangle & \cdots & \langle w_2|F(g_N)\rangle \\ \vdots & & \vdots \\ \langle w_N|F(g_1)\rangle & \cdots & \langle w_N|F(g_N)\rangle \end{bmatrix} \tag{8}$$

$$[c_n] = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} \quad [h_m] = \begin{bmatrix} \langle w_1|h\rangle \\ \langle w_2|h\rangle \\ \vdots \\ \langle w_N|h\rangle \end{bmatrix} \tag{9}$$

If $[F]$ is nonsingular, its inverse exists, and $[c]$ is given by

$$[c_n] = [F_{mn}]^{-1}\,[h_m] \tag{10}$$

2 MoM and microstrip geometries

In microstrip geometries, the Electric Field Integral Equation (EFIE for short) can be written as

$$\overline{E}^i(x,y) + j\omega\mu_0 L\left[\overline{\overline{G}}_{EJ}(x,y|x',y')\overline{J}_S(x',y')\right] = 0 \tag{11}$$

Let $\hat{x}$ and $\hat{y}$ components of $\overline{J}_S$ be expanded in a series of *basis functions* J_{x_j} and J_{y_j}

$$J_{Sx} = \sum_{j=1}^{N} A_{x_j} J_{x_j}(x',y') \tag{12}$$

$$J_{Sy} = \sum_{j=1}^{N} A_{y_j} J_{y_j}(x',y') \tag{13}$$

Substituting (12) and (13) into (11), the discretized EFIE is written as

$$E_x^i = -j\omega\mu_0\Big(\sum_{j=1}^{N} A_{x_j} L_{E_{xx}}\left[J_{x_j}\right] + \sum_{j=1}^{N} A_{y_j} L_{E_{xy}}\left[J_{y_j}\right]\Big) \tag{14}$$

$$E_y^i = -j\omega\mu_0\Big(\sum_{j=1}^{N} A_{x_j} L_{E_{yx}}\left[J_{x_j}\right] + \sum_{j=1}^{N} A_{y_j} L_{E_{yy}}\left[J_{y_j}\right]\Big) \tag{15}$$

Basis functions J_{x_j} and J_{y_j} are chosen as the product of two functions.

$$J_{x_j}(x,',y') = T_j(x')Q_j(y') \tag{16}$$

$$J_{y_j}(x,',y') = Q_j(x')T_j(y') \tag{17}$$

The longitudinal function T_j has a piecewise sinsuoidal or triangular behaviour while the transversal function Q_j has a constant distribution.

Testing functions are chosen to be the same as *basis functions* (Galerkin method). Finally, after setting all the weighted residuals equall to zero, one has

$$-j\omega\mu_0\Big(\sum_{j=1}^{N} A_{x_j}\big\langle L_{E_{xx}}\left[T_j(x')Q_j(y')\right]\big|T_i(x)Q_i(y)\big\rangle + \sum_{j=1}^{N} A_{y_j}\big\langle L_{E_{xy}}\left[Q_j(x')T_j(y')\right]\big|T_i(x)Q_i(y)\big\rangle\Big) = \big\langle E_x^i\big|T_i(x)Q_i(y)\big\rangle \quad i = 1\ldots N \tag{18}$$

$$-j\omega\mu_0\Big(\sum_{j=1}^{N} A_{x_j}\big\langle L_{E_{yx}}\left[T_j(x')Q_j(y')\right]\big|Q_i(x)T_i(y)\big\rangle + \sum_{j=1}^{N} A_{y_j}\big\langle L_{E_{yy}}\left[Q_j(x')T_j(y')\right]\big|Q_i(x)T_i(y)\big\rangle\Big) = \big\langle E_y^i\big|Q_i(x)T_i(y)\big\rangle \quad i = 1\ldots N \tag{19}$$

Again, this set of equations can be written in abreviated matrix form as

$$\begin{pmatrix} Z_{xx}^{ij} & Z_{xy}^{ij} \\ Z_{yx}^{ij} & Z_{yy}^{ij} \end{pmatrix}_{2N\times 2N} \begin{pmatrix} I_x^i \\ I_y^i \end{pmatrix}_{2N\times 1} = \begin{pmatrix} V_x^j \\ V_y^j \end{pmatrix}_{2N\times 1} \begin{matrix} i = 1..N \\ j = 1..N \end{matrix} \tag{20}$$

where each one of the elements of the Z matrix in (20) can be written as a convolution integral

$$\int dx' \int dx \int dy' \int F_j(x',y')F_i(x,y)G_\alpha(x-x',y-y')dy \tag{21}$$

where F_j and F_i are *basis* and *testing functions* and G_α are components of the microstrip Green function (Sommerfeld integrals). An in-house developed *Sommerfeld* library [2] is used to compute these G_α functions.

3 Parallel MoM technique

In many MoM problems, the matrix fill time is the dominant factor, as each matrix element typically involves the computing of a numerical integral of a complex function. Usually a $2N$ fold numerical quadrature of a complex function has to be computed, being N the geometry dimension (vg $N = 1$ for wires, $N = 2$ for planar geometries and $N = 3$ for 3-D geometries). This type of problem is typically easy to parallelize as the computation of any matrix element is completely indepedent of the value of any other matrix element.

As structures become electrically large [3], the matrix factorization begins to dominate the overall solution time. The memory and computation requirements for these large structures can become daunting as grow as $O[N^3]$ being N the number of unknowns

In a fully parallelized MoM, the F matrix is never stored into the memory of any processor but blocks of it. Therefore, filling and factorizacion steps are closely linked.

3.1 Parallelized scheme

Now we are going to distribute the computation and the factorization of the Z matrix among a set of $nproc$ processors. There is a main processor which collects the geometry data input, distributes the work and collects and stores

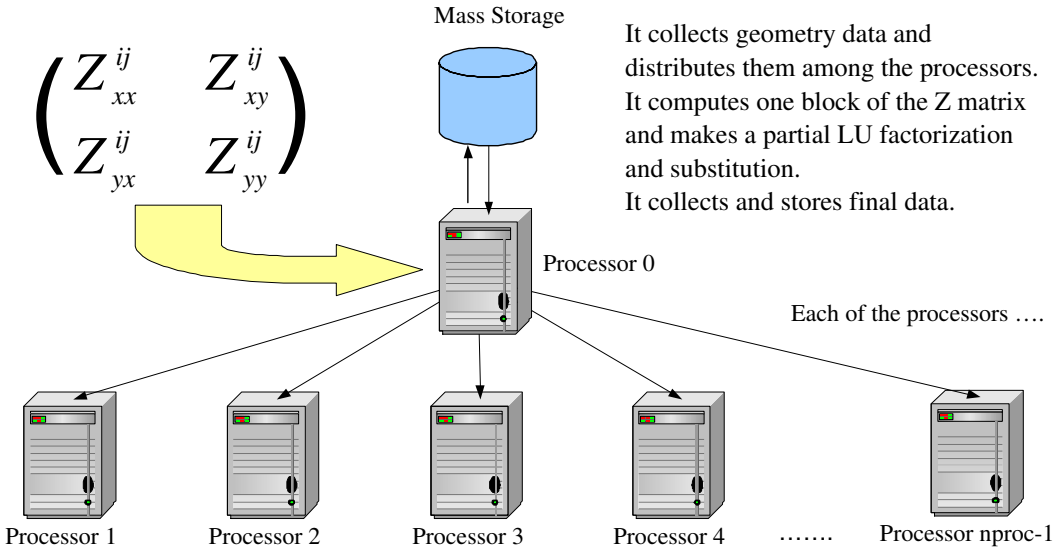


Figure 1: Work distribution among processors.

final data. Each of the processors, including the main one, computes and factorizes one block of the Z matrix. This is shown in figure 1.

3.2 The *MPI* paradigm

One of the most successful parallel computational models is the message-passing model [4, 5, 6]. This model posits a set of processors that have only *local* memory but are able to communicate with other processes by sending and receiving messages. *MPI* [7, 8],which stands for Message Passing Interface, is an attempt to collect the best features of the message-passing systems that have been developed over the years, improve them when appropriate, and standardize them.

It is a *library not a language*. It specifies the names, calling sequences and results from functions to be called from Fortran/C/C++ programs. *MPI* is an specification not a particular implementation. A correct *MPI* program should be able to run on all *MPI* implementations without change [9, 10]. A minimal message interface between two processes should be built using two primitives: send and receive.

For the sender, the things that must be specified are the data to be communicated and the destination process to which the data is to be sent. The minimal way to describe data is to specify a starting address and a length (in bytes) and a destination field (usually an integer).

On the receiver's side, the minimum arguments are the address and length where received data is going to be placed and a variable to be filled in with the identity of the sender.

Although this minimum interface could be adequate for some applications, one key notion is missing; *matching*. A process must be able to control which messages it receives. This is the type or *tag* of the message. Finally it is useful for the receiver specify a maximum message size (the actual length) for a given *tag*. Therefore, our minimal message interface has become: `send(address, length, destination, tag)` and `receive(address, length, source, tag, actlen)` That basic interface is implemented in *MPI* using `MPI_send` and `MPI_receive`

3.3 Matrix filling

The way the Z matrix is filled depends on the way it is factorized. In order to fill the matrix, we need to analyse the microstrip structure. This structure is split in several patches, which is divided into four subdomains, with four vertices (figure 2).

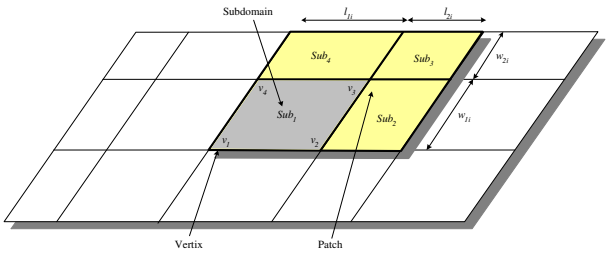


Figure 2: Microstrip structure discretization.

In this scheme, each processor computes a block of the Z matrix. Those blocks are chosen so for a given pair of indexes (i, j) the four parameters Z_{xx}^{ij}, Z_{xy}^{ij}, Z_{yx}^{ij} and Z_{yy}^{ij} are computed in the same processor and there are no

communication among processors.

In figure 3 pseudocode to fill the matrix is shown. The couples between patches is computed as the integrals showed in (21) and the number of the couple is shown in [11].

```
do i = 1,...,N/2
  do j = 1,...,N/2
    Get dimensions patch(i)
    Get dimensions patch(j)
    Z_xx(i,j) = couple(1)+couple(4)
    Z_xy(i,j) = couple(2)
    Z_yx(i,j) = couple(3)
    Z_yy(i,j) = couple(5)+couple(6)
  end do
end do
```

Figure 3: Pseudocode for parallel matrix fill.

3.4 Matrix factorization

Each processor partially factorizes (LU) the same block that this processor has filled previously. Obviously, while factorizing the matrix, processors need to share information among them.

The programmer always tries to minimize communications between processors and to equalize the load among them. We have studied several ways to factorize the matrix and each one of those ways is linked to a matrix filling scheme. In this paper we are going to show two schemes: the cyclic one-dimensional data distribution and the cyclic two-dimensional data distribution.

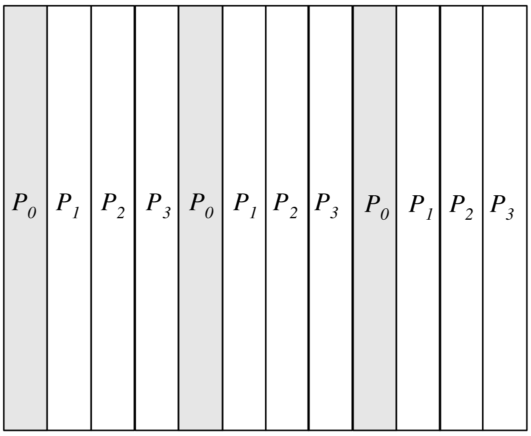


Figure 4: Cyclic one-dimensional data distribution. 4 processors, 3 cycles.

The one-dimensional data distribution can be in rows or in columns. Our selection has been in column (figure 4) to minimize the communication between processors. In this distribution, one processor has to wait for the previous processors. With the cyclic distribution, each processor works only with a data block and the wait time is minimized. In figure 5 pseudocode for cycling one-dimensional parallel factorization is shown.

```
do c = 1,...,ncycles
  do l = 1,...,id(proc,c)
    Update columns proc = l
    LU(l) factorization
    V(l) factorization
  end do
    do k = 1,...,ncol(id(proc,c))
      LU(k) factorization
      V(k) factorization
      Send column(k) to
      the rest of id(proc,c)
    end do
    Send V data to id(proc+1,c)
      do l = id(proc+1,c),...,nproc
    Update columns proc = l
      LU(k) factorization
      V(k) factorization
    end do
end do
```

Figure 5: Pseudocode for cyclic one-dimensional parallel LU factorization.

However, this distribution is not efficient when the matrix is very large. To equalize the load among processors, they work in a cyclic way as it is shown in figure 4. Due to data dependencies in LU factorization, there are processors that have to wait for other processors to finish. That idle time can be minimized if each processor works with a relatively small matrix block.

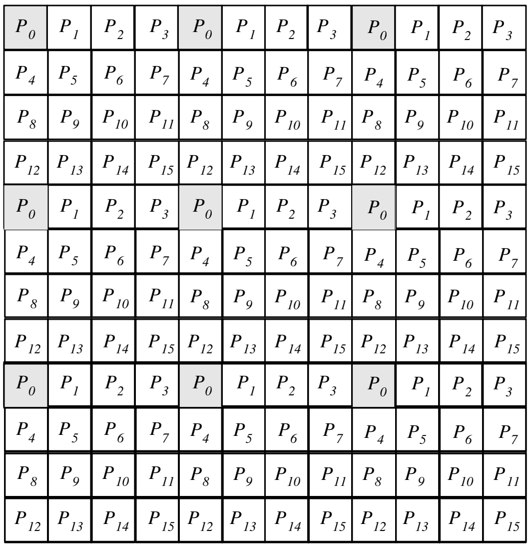


Figure 6: Cyclic two-dimensional data distribution. 16 processors, 3 cycles.

To maximize per processor performance, there is an optimum block size which depends on the matrix size and processors's cache memory. A block size smaller than optimum means more communications and less computations. A block size greater than optimum means more out-of-core computations and less speed.

In figure 7 pseudocode to run in each processor is shown. In this figure $ncycles_{col}$ is the number of cycles per column (3 in figure 6), $proc_{col/fil}$ is the position of a processor in a row or a column (it varies between 1 and 4 in figure 6) and $id(proc_{col/row}, cycle_{col/row})$ identifies the position in the matrix of processor $proc_{col/row}$ in cycle number $cycle_{col/row}$.

```
do c_col = 1,...,ncycles_col
  c_row = 1
  do l = 1,...,id(proc_col, c_col)
    Update rows(id(proc_row, c_row))
    Get pivots(id(l, c_col))
    LU(l) factorization
    V(l) factorization
  end do
  if (id(proc_row, c_col) ≤ id(proc_col, c_col)) then
    do k = 1,...,ncol(id(proc_col, c_col))
      Swap rows(id(proc_row, c_row))
      LU(k) factorization
      V(k) factorization
      Send column(k) to
      the rest of id(proc_row, c_col)
    end do
    Send V data to id(proc_col + 1, c_col)
  end if
  if (id(proc_row, c_col) = id(proc_col, c_col)) then
    crow=crow+1
  if (c_col = nciclos_col) then
    if (id(proc_row, c_col) > id(proc_col, c_col)) then
      do l = id(proc + 1, c_col),...,nproc_col
      Swap rows (id(proc_row, c_row))
    end do
  else
    do l = id(proc_row + 1, c_col),...,nproc_col
      Update rows(id(proc_row, c_row))
      Get pivots(id(l, c_col))
      LU(l) factorization
      V(l) factorization
    end do
  end if
end do
```

Figure 7: Pseudocode for cyclic bidimensional parallel LU factorization.

3.4.1 ScaLAPACK, BLAS and linear algebra software.

This pseudocode only works if $ncycles_{col} = ncycles_{row}$ and the number of processors is a perfect square (as in example shown in figure 6). Dealing with a non perfect square number of processors complicates the code quite a lot. Instead of re-coding the algorithm, we looked for mature well established code that was able help us.

ScaLAPACK [12, 13](*Scalable Linear Algebra Package*) is an optimized library of linear algebra subroutines that run in cluster environments. Its main components are:

- A low level BLAS (*Basic Linear Algebra Subprogram*) optimized for single processor.
- A communication library for BLAS subroutines BLACS (*Basic Linear Algebra Communication Subprogram*) that uses some message passing model (MPI or PVM).
- A parallel version of BLAS library PBLAS (*Parallel Basic Linear Algebra Subprogram*) that uses the aforementioned libraries.

In figure 8, the hierarchy of these libraries are shown. *ScaLAPACK* uses a two-dimensional cyclic data distribution that helped us to easily map our matrix data distribution to *ScaLAPACK's* one.

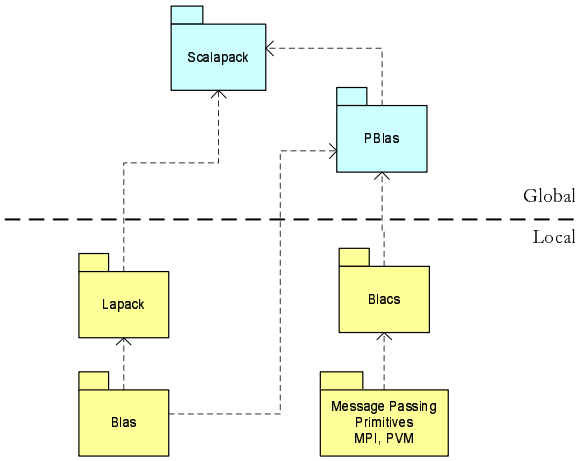


Figure 8: Library Scalapack Hierarchy.

4 Results

In this section, we investigate the efficiency of the proposed parallel processing. The evaluation was carried out on the GIC cluster "Maxwell". The cluster details are as follows:

- Number of processors: 23 Intel Pentium IV Prescott 2.8 GHz, 1GB RAM
- Network: 100 Mb/s
- Operating system: Linux 2.6 kernel series, Bproc based cluster
- Tools: GNU tools (gcc, g77,...)
- MPI: LAM-MPI
- BLAS: ATLAS

In figure 9 the Maxwell cluster is shown in the front and rear view. This cluster is situated in 2 frames.

(a) Front panel (b) Rear panel

Figure 9: Cluster Maxwell.

4.1 Parallel processing efficiency metrics

The efficiency of parallel-processing code depends not only on how we develop the code, but also on the computer hardware and networking equipment. We employ the conventional definition of scalability or speedup of the parallel processing code as

$$S_p = \frac{T_s}{T_p} \tag{22}$$

where T_s is the simulation time when the entire problem is simulated using a single processor, and T_p is the simulation time in the p-processors parallel processing. During the simulations, we had to define a new figure of merit that we called *parallel speedup*. It is used when the entire problem does not fit in one single processor. It is defined as

$$PS_p = \frac{T_{1c}}{T_p} \tag{23}$$

where T_{1c} is the simulation time using p processors but the data distribution has only one cycle per row and column (the worst case).

An alternative measurement to the speedup is the efficiency concept. The efficiency measures the degree of use of the system. This expression can be written with the similar parameters that were described at (22).

$$E_p = \frac{S_p}{p} \tag{24}$$

4.2 Simulated Microstrip Structures

The analyzed structures are 2D microstrip structures such as a meander line, a patch antenna and a 3dB Hybrid. Speedup, Efficiency, optimum size block, topology grid and kiviat graph are shown in this section.

The meander lines have several utilities in microstrip technologies. Many meander lines can be used to make a delay time using zigzag trace to match the timing between two or more signals [14]. Other meander lines are used as pass-band filters or antennas which resonant frequency depends on the separation of the lines and the number of zigzag lines.

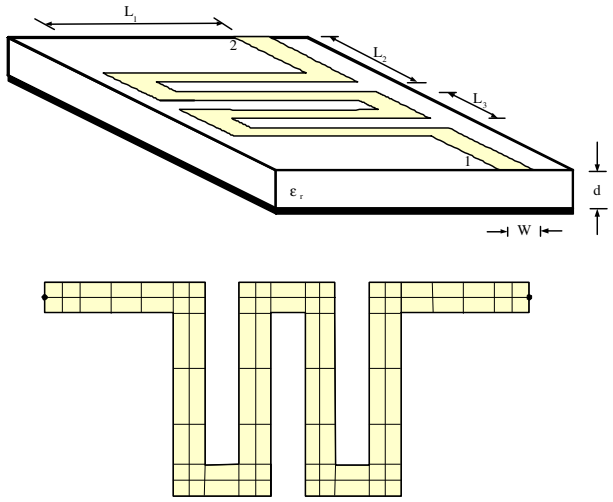


Figure 10: Meander line.

The dimensions of the meander line (figure 10) are the following: (lengths) L_1 = 18.58 mm, L_2 = 20.44 mm, L_3 = 5.08 mm, (width) W = 1.86 mm, height of the substrate d = 0.833 mm and dielectric constant ϵ_r = 3.26. This structure has been simulated with several meshes from 200 patches to 6400 patches (Table 1).

	Mesh 1	*Mesh 2*	*Mesh 3*
Patches	200	800	6400
Subdomains	388	1550	12400
Vertices	600	2398	19176
Excitations	4	4	8
Ports	2	2	2

Table 1: Meander Meshs.

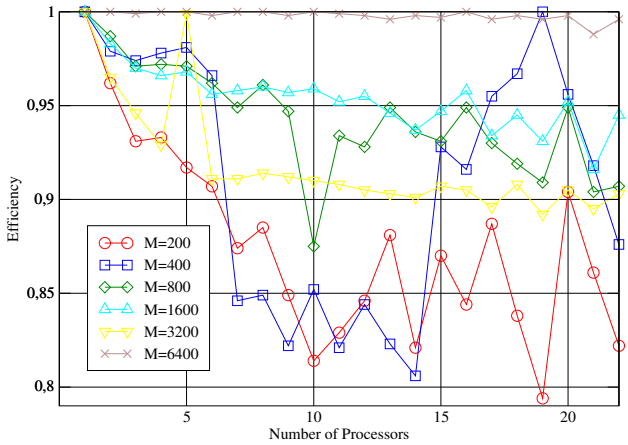


Figure 11: Efficiency of the matrix filling.

The matrix filling is tested with the meander line. The

number of processors has been increased from 1 to 23. The theoretical efficiency is 1. It is shown in figure 11 the efficiency of the matrix filling. The theoretical efficiency is 1. When the number of patches is increasing, the efficiency is close to 1. In figure 12, it is shown the speedup. The theoretical slope is P.

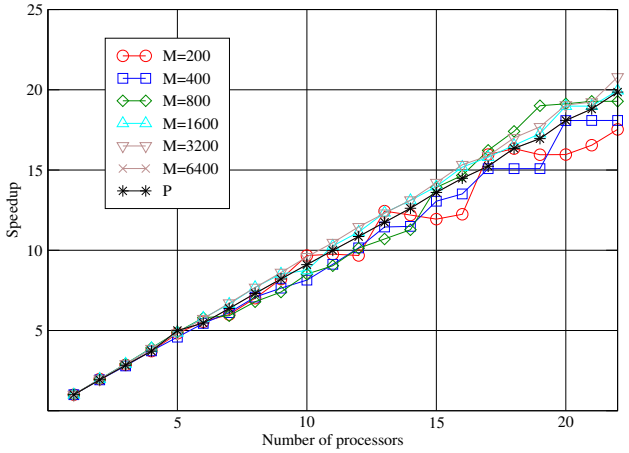


Figure 12: Speedup of the matrix filling.

Other simulated structure is a patch microstrip antenna [15]. A patch antenna consists of a metal patch constructed on a dielectric substrate over a ground plane. This kind of antennas are popular because they are easy to fabricate and modify.

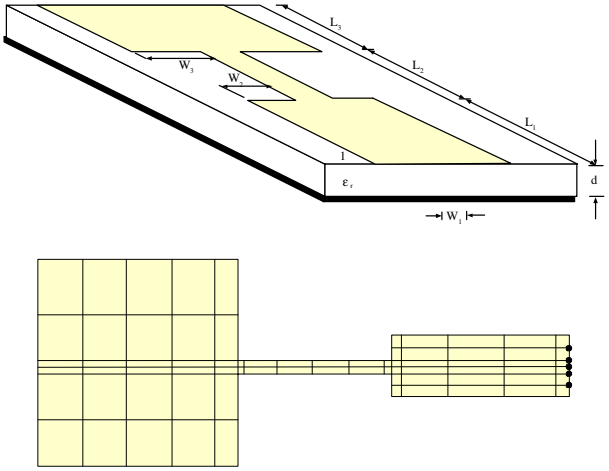


Figure 13: Patch Antenna.

This structure is made up of a patch square, a quarter wavelength transformer to match the antenna to 50 ohm and a microstrip line with this impedance. The dimensions of the antenna (figure 13) are: (lengths) L_1 = 50 mm, L_2 = 30 mm, L_3 = 56.2 mm, (widths) W_1 = 37.5 mm, W_2 = 0.45 mm, W_3 = 3.31 mm, height of the substrate d = 1.57 mm and dielectric constant ϵ_r = 3.86. The meshs of this structure are collected in table 2.

The size of block in the matrix factorization and the topology grid are tested with this structure. In figure 14,

	Mesh 1	*Mesh 2*	*Mesh 3*
Patches	6400	12800	22500
Subdomains	8986	17912	31590
Vertices	15788	31472	55504
Excitations	4	4	8
Ports	1	1	1

Table 2: Patch Antenna Meshs.

speedup is shown vs block size and for three processor grids (16 processors). The square arrangement is clearly better than the other two.

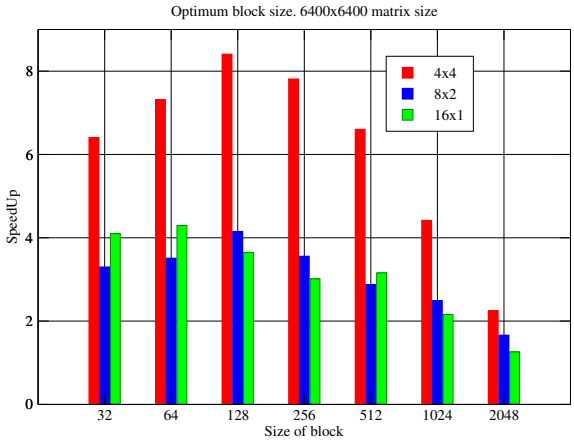


Figure 14: Optimum block size and speedup for three processor grids.

In figure 15, parallel speedup vs block size is shown for three matrix sizes and for a 5x4 processor topology grid. The optimum block size is 128 for the three meshs. The performances are very good when the size of the matrix is very large.

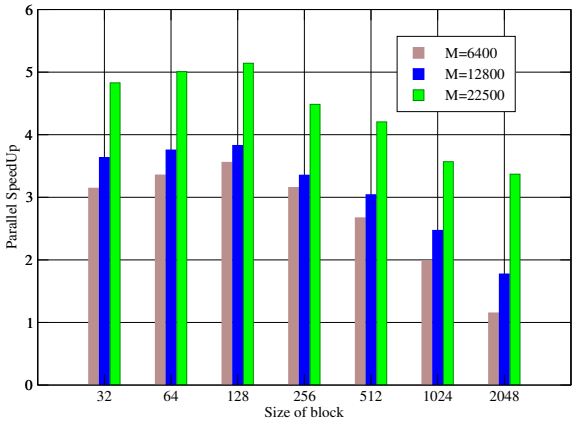
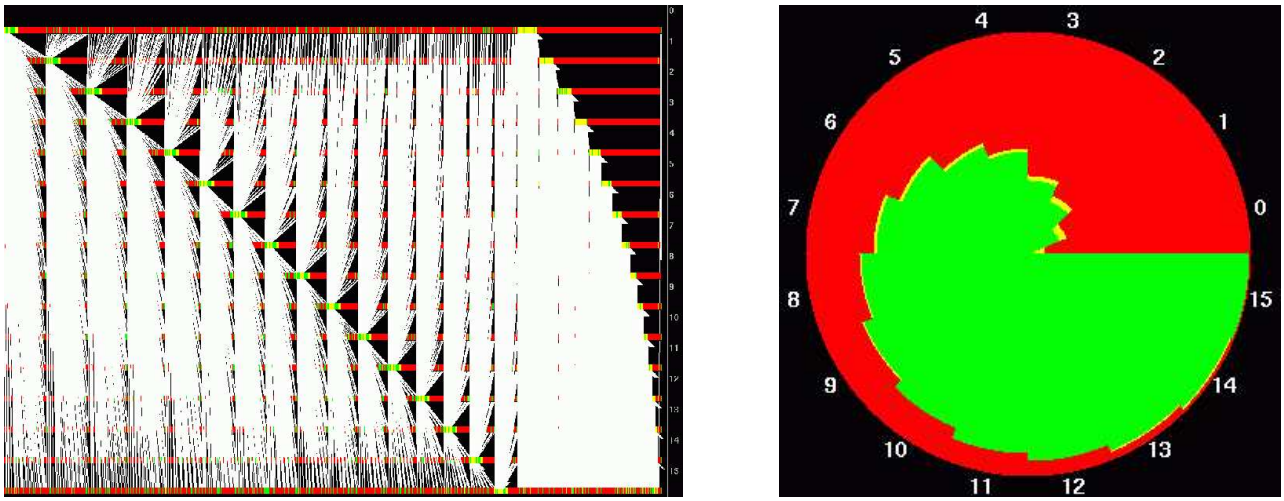


Figure 15: Optimum block size, 5x4 processor grid.

The next shape is the 3dB hybrid structure [16]. Hybrid couplers are the special case of a four-port directional coupler that is designed for a 3-dB (equal) power split. Hy-



(a) Communication Timeline (b) Kiviat graph

Figure 16: Trace Data, 16x1 processor.

brids come in two types, 90 degree or quadrature hybrids, and 180 degree hybrids.

The dimensions of the hybrid (figure 17) are: (lengths) L_1 = 12.163 mm, L_2 = 9.75 mm, L_3 = 3.96 mm, (width) W = 2.413 mm, height of the substrate d = 0.794 mm and dielectric constant ϵ_r = 2.2. The values of the meshs are collected in table 3.

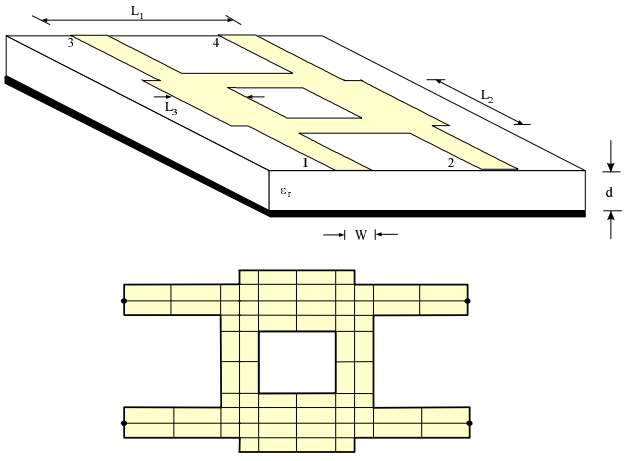


Figure 17: 3dB Hybrid.

In figure 18, it is obtained the same results than the patch antenna. The optimum block size is 128 for a 4x4 processor topology grid.

The load balance with are tested with this structure. Figure 16 shows the Kiviat graph and the communication line for a one-dimensional cyclic data distribution, 16x1 processor grid, and M=22500 (45000x45000 matrix size). Figure 19 shows the Kiviat graph for a bidimensional cyclic data distribution, 4x4 processor grid, and the same matrix size. The Kiviat graph shows the processor load and a nearly circular shaped graph means the load is balanced among processors.

	Mesh 1	*Mesh 2*	*Mesh 3*
Patches	6400	12800	22500
Subdomains	9888	19776	34762
Vertices	13952	27904	49050
Excitations	4	4	8
Ports	4	4	4

Table 3: Hybrid Meshs.

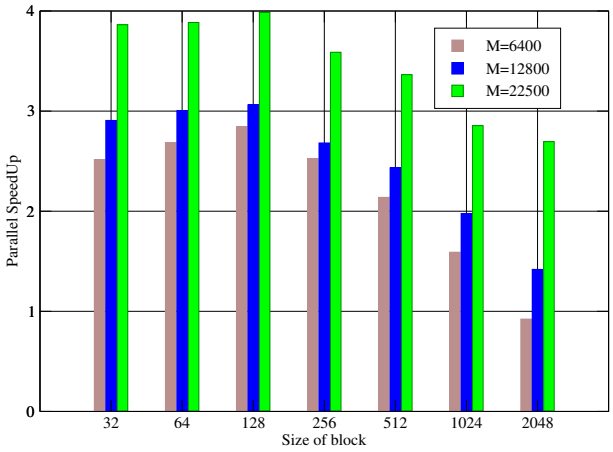
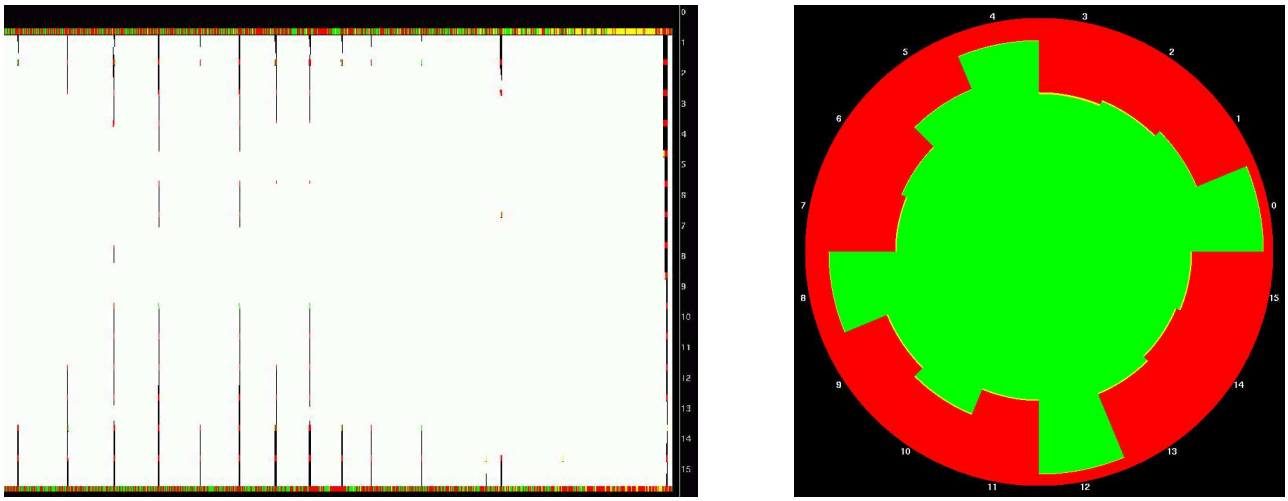


Figure 18: Optimum block size, 4x4 processor grid.



(a) Communication Timeline

(b) Kiviat graph

Figure 19: Trace Data, 4x4 processor.

5 Conclusion

We have presented a parallel MoM analysis of microstrip structures. Our main goal has been to improve the efficiency of the process modifying two parameters: processors grid topology and optimum block size. Some code has been in-house developed and later modified to suit to the *ScaLAPACK* standard library with two schemes, one dimensional cyclic data distribution and bidimensional cyclic data distribution.

Three examples have been presented to demonstrate the scalability of the problem with several block sizes. Finally, it has shown the best results have been obtained with a square topology grid, the optimum block size and using the second scheme.

Acknowledgment

This work has been partially supported by the R&D Spanish National Projects TEC2007-67520-C02-02 and TEC2005-07010-C02

References:

[1] Roger F. Harrington, *"Matrix Methods for Field Problems"*, *Proc IEEE*, vol. 55, no 2, pp. 136-149, February 1967

[2] Eugenio Jimenez, Francisco Cabrera, *"Sommefeld: a library for computing Sommerfeld integrals"*, *IEEE Ant.and Prop. Symposium*, Baltimore, pp. 966-969, July 1996.

[3] Y. Zhang, T.K. Sarkar, H. Moon, A. De and M.C. Taylor, *"Solution of Large Complex Problems in Computational Electromagnetics using Higher Order Basis in MOM with Parallel Solvers"*, *IEEE Ant. and Prop. Symposium*, Honolulu, HI, pp. 5620-5623, June 2007

[4] Izzatdin Aziz, Nazleeni Haron, Low Tan Jung, Wan Rahaya Wan Dagang, *"Parallelization of Prime Number Generation Using Message Passing"*, *WSEAS Transactions on Computers*, Issue 4, Vol 7, pp. 291-303, April 2008

[5] D. Prabu, V. Vanamala, Anshu Garg, Sanjeeb Kumar Deka, R. Sridharan, B.B. Prahlada Rao, N. Moharam, *"Development of 64-bit Message Passing Interface for Large Scale Computing System"*, *WSEAS Transactions on Computer Research*, Issue 2, Vol 2, pp. 147-155 Feb. 2007

[6] Haroon-Ur-Rashid, Shi Feng, Ji Weixing, Qiao Baojun, *"TriBA - A Novel Scalable MPI based Architecture for High Performance Distributed Parallel Computing"*, *WSEAS Transactions on Computers*, Issue 12, Vol 6, 1161-1167 Dec 2007

[7] http://www.mpi-forum.org

[8] W. Gropp, E. Lusk, and A. Skjellum, *"Using MPI: Portable Parallel Programming with the Message-Pasing Interface, Second Edition"*, Cambridge, MA, MIT Press, 1999

[9] Wenhua Yu, Yongjun Liu, Tao Su, Neng-Tien Hunag, Raj Mittra, *"A robust parallel conformal finite-difference time-domain processing package using the MPI library"*, *"Antennas and Propagation Magazine, IEEE"*, Vol. 47, no. 3, pp. 39-59, June 2005

[10] C. Guiffaut and K. Mahdjoubi, *"A Parallel FDTD Algorithm Using the MPI Library"*, *Antennas and Prop-*

agation Magazine, IEEE, Vol. 43, no 2, pp 94-103, April 2001

[11] Francisco Cabrera, Eugenio Jimenez, *"Analysis of Irregular Microstrip Structures using a Full Wave MoM Scheme"*, *Millenium Conference on Antennas & Propagation*, Davos, Switzerland, pp 20, Mar 2000

[12] http://www.netlib.org/scalapack

[13] Y. Zhang, T.K. Sarkar,A. De, N. Yilmazer, S. Burinttramart, M. Taylor, *" A cross-platform parallel MoM code with ScaLAPACK solver"*, *IEEE Ant. and Prop. Symposium*, Honolulu, HI, pp. 2797-2790, June 2007

[14] Barry J. Rubin, Bhupindra Singh *"Study of Meander Line Delay in Circuit Boards"*, *IEEE Trans. on Microwave Theory and Techniques*, Vol. 48, n. 9, pp 1452-1460, Sep. 2000

[15] L. Zhu, E. Yamashita, I. Joishi, *"Generalized Modeling of Microstrip-Fed Patch Antennas Using an Equivalent Delta Voltage Source Backed by a Perfect Electric Wall"*, *IEEE Antennas and Propagat. Society International Symposium*, Baltimore, pp. 1082-1085, Jul 1996.

[16] D. M. Sheen, S. M. Ali, M. D. Abouzahra, J. A. Kong, *"Application of the Three-Dimensional Finite Diference Time-Domain Method to the Analysis of Planar Microstrip Circuits"*, *IEEE Trans. Antennas and Propagat.*, vol 38, pp. 849-857, Jul. 1990