Parallel WZ Factorization on Mesh Multiprocessors

R. Asenjo
M. Ujaldon
E.L. Zapata

September 1993
Technical Report No: UMA-DAC-93/07

Published in:

J. Microprocessing and Microprogramming
vol. 38, no. 5, 1993, pp. 319-326

University of Malaga
Department of Computer Architecture
C. Tecnologico • PO Box 4114 • E-29080 Malaga • Spain

PARALLEL WZ FACTORIZATION ON MESH MULTIPROCESSORS[*]

R. Asenjo, M. Ujaldón and E.L. Zapata

Dept. Arquitectura de Computadores, University of Málaga, Plaza El Ejido. 29013 Málaga. SPAIN.

***Abstract**. We present a parallel algorithm for the QIF (Quadrant Interlocking Factorization) method, which solves linear equation systems using WZ factorization. The parallel algorithm we developed is general in the sense that it does not impose any restrictions on the size of the problem and that it is independent from the dimensions of the mesh. The result is an efficient algorithm using half the messages of the equivalent parallel LU algorithm. Finally, we have compared the QIF algorithm in multiprocessor architectures with mesh topology and hypercube topology, obtaining similar calculation times for both architectures. This last aspect confirms the communication redundancy of the hypercube topology, which raises hardware costs without any significant improvement in the efficiency of the algorithms.*

1. Sequential QIF algorithm

The solution of a linear equation system is a problem which arises in many situations, whether scientific (simulations, solution of differential equation systems), economic (resource assignment, econometric models in general) and engineering (passive electronic circuits), etc.

The QIF (*Quadrant Interlocking Factorization*) algorithm, introduced by Evans and Hatzopoulos in 1979 [EvHa79], is a numerical method for finding a solution for systems of the type $Ax=b$, where $\boldsymbol{A}$ is a non singular matrix of dimensions ($N\times N$), $\boldsymbol{x}$ is an unknown column vector, and $\boldsymbol{b}$ is the independent term vector provided. The QIF method is based on the WZ factorization of the system matrix, A = WZ. The main advantage of this factorization is that it presents a complexity order half of the one in the LU decomposition, due to the fact that it performs the simultaneous evaluation of two columns or two rows. A detailed description of this algorithm can be found in [EvHa79], [EvHa80], [EHN81], [Evan82] and [Hatz82]. We will just summarize its basic steps.

Matrices W and Z into which we are going to factor matrix A, are:

$$W = \begin{bmatrix} 1 & & \mathbf{0} & & & 0 \\ w_{1,0} & 1 & & & 0 & w_{1,N-1} \\ w_{2,0} & w_{2,1} & 1 & 0 & w_{2,N-2} & w_{2,N-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ w_{N-3,0} & w_{N-3,1} & 0 & 1 & w_{N-3,N-2} & w_{N-3,N-1} \\ w_{N-2,0} & 0 & & & 1 & w_{N-2,N-1} \\ 0 & & \mathbf{0} & & & 1 \end{bmatrix}$$

$$Z = \begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & \cdots & z_{0,N-3} & z_{0,N-2} & z_{0,N-1} \\ & z_{1,1} & z_{1,2} & \cdots & z_{1,N-3} & z_{1,N-2} & \\ & & z_{2,2} & \cdots & z_{2,N-3} & & \\ & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \\ & & z_{N-3,2} & \cdots & z_{N-3,N-3} & & \\ & z_{N-2,1} & z_{N-2,2} & \cdots & z_{N-2,N-3} & z_{N-2,N-2} & \\ z_{N-1,0} & z_{N-1,1} & z_{N-1,2} & \cdots & z_{N-1,N-3} & z_{N-1,N-2} & z_{N-1,N-1} \end{bmatrix}$$

where the elements of W and Z are:

$$w_{i\,j} = \begin{cases} 1, & i = j \\ 0, & 0 \le i \le \dfrac{(N-1)}{2} \qquad i+1 \le j \le N-i-1 \\ 0, & \dfrac{N}{2} \le i \le N-1 \qquad N-i-1 \le j \le i-1 \\ w_{i\,j} & otherwise \end{cases} \qquad \mathbf{(1)}$$

$$z_{i\,j} = \begin{cases} z_{i\,j} & 0 \le i \le \dfrac{(N-1)}{2} \qquad i+1 \le j \le N-i-1 \\ z_{i\,j} & \dfrac{N}{2} \le i \le N-1 \qquad N-i-1 \le j \le i-1 \\ 0 & otherwise \end{cases} \qquad \mathbf{(2)}$$

The elements of W and Z can be evaluated in n steps, $0 \le k < n$, where $n = (N-1)/2$. In the k-th step we obtain rows k and N-k-1 of matrix Z, and columns k and N-k-1 of W. In each step, the elements of Z are evaluated according to the following equation:

$$\begin{aligned} z_{k\,j} &= a_{k\,j} \\ z_{N-k-1\,j} &= a_{N-k-1\,j} \end{aligned} \qquad k \le j \le N-k-1 \qquad \mathbf{(3)}$$

In order to obtain the elements of W we will have to solve the following linear systems (2×2):

[*]**This work was supported by the Ministry of Education and Science (CICYT) of Spain underproject TIC-92-0942**

$$\begin{aligned} z_{kk}w_{jk} + z_{N-k-1,k}w_{j,N-k-1} &= a_{jk} \\ z_{k,N-k-1}w_{jk} + z_{N-k-1,N-k-1}w_{j,N-k-1} &= a_{j,N-k-1} \end{aligned} \qquad k<j<N-k-1 \qquad \textbf{(4)}$$

After step k, we must update, in preparation for the next step, the rows of A. In order to do this we need the elements of Z and W that were calculated in step k.

$$a_{ij} = a_{ij} - w_{ik}z_{kj} - w_{i,N-k-1}z_{N-k-1,j} \qquad k<i,j<N-k-1. \qquad \textbf{(5)}$$

After the WZ factorization, the next step consists in obtaining the solution vector x. For this, we first solve systems $Wy=b$ and then system $Zx=y$. Vector y is evaluated in n steps, and components y_k, y_{N-k-1} are produced in the k-th step,

$$\begin{aligned} y_k &= b_k \\ y_{N-k-1} &= b_{N-k-1} \end{aligned} \qquad \textbf{(6)}$$

and the rest of the elements of vector b are updated as a preparation for the following step:

$$b_j = b_j - w_{jk}y_k - w_{j,N-k-1}y_{N-k-1} \qquad k<j<N-k-1 \qquad \textbf{(7)}$$

Finally, for obtaining vector x, we solve $Zx=y$, taking into account two cases; N even or N odd. If N is odd, we first obtain the central element of vector x,

$$x_n = \frac{y_n}{z_{nn}} \qquad \textbf{(8)}$$

update the rest of vector y and continue with the next stage:

$$y_j = y_j - x_n z_{jn} \qquad 0\le j\le N-1, \quad j<>n. \qquad \textbf{(9)}$$

The rest of the elements of vector x are obtained in k steps, $n < k \le 0$, by solving the linear systems (2 × 2) where elements x_k and x_{N-k-1} are computed:

$$\begin{aligned} z_{kk}x_k + z_{k,N-k-1}x_{N-k-1} &= y_k \\ z_{N-k-1,k}x_k + z_{N-k-1,N-k-1}x_{N-k-1} &= y_{N-k-1} \end{aligned} \qquad (n-1)\ge k\ge 0 \qquad \textbf{(10)}$$

Vector y is updated before starting the next iteration,

$$y_j = y_j - x_k z_{jk} - x_{N-k-1}z_{j,N-k-1} \qquad \begin{matrix} 0\le j<k \\ (N-k)\le j\le N-1 \end{matrix} \qquad \textbf{(11)}$$

If N is even, all the components of vector x are obtained in pairs by solving linear system (10) and updating the vector according to equation (11).

The QIF algorithm for solving the linear system $Ax=b$ can thus be implemented by means of the following procedures:

```
void QIF()                /* Solves the system Ax=b */
  {
WZ;                                  /* A=WZ */
vector_y;                            /* Wy=b */
if ((N%2)!=0) {
    central_element;
    vector_x (n); }                  /* Zx=y */
else vector_x (n+1);
  }
```

We first carry out the decomposition of matrix A into the two matrices W and Z. The elements of these two matrices are defined in equations (1) and (2). The elements which are not 0 or 1 are obtained by means of equations (3), (4) and (5). In order to obtain vector y we solve the system $Wy=b$ using equations (6) and (7). When obtaining vector x we consider two cases: if the dimension of matrix A is odd, we first calculate the central element and update the rest of vector y by means of equations (8) and (9), and then obtain the rest of the elements of x by means of equations (10) and (11). The number of times equations (10) and (11) are evaluated is $n+1$ for N even, and n for N odd.

Finally, the numerical stability of the method can be improved by introducing one of the two possible pivoting strategies, partial pivoting or complete pivoting [KaSc87], [HaEv88].

2. Parallel QIF algorithm

A mesh is a two dimensional array of processing elements (PEs). In a generic mesh with $X\times Y$ PEs ($X\ge 1$, $Y\ge 1$), each one of them is connected by means of its channels N, S, E, and W to its four neighboring PEs, except the border processors which only use 3 channels and the 4 corner PEs which only use 2. The index assigned to each processor is a function of its position (i,j) in the mesh ($0\le i<X$, $0\le j<Y$) and is given by the following formula: PE=$j\cdot X+i$.

The basic operation for the conversion of sequential algorithms into parallel algorithms is the fragmentation of the nested loops so that different iterations are processed in different processors. This implies that the variables participating in the algorithms must be distributed among the processors. The distribution must be carried out so as to permit the solution of problems whose sizes are not related to the dimensions of the mesh.

The method we have followed for the partition/projection of the algorithms onto a mesh is similar to the one developed by Zapata et al. [ZRP90][Rive90] for hypercubes and consists in the following steps: 1) Analysis of the sequential algorithm at the loop level, detecting data and control dependencies. The maximum number of independent nested loops (*do all*) defines the number of dimensions in the algorithmic space. 2) If we find a single parallelizable loop, the whole mesh will be associated to it. If the dimension of the algorithmic space is two, the columns of the mesh will be assigned to one loop and the rows to the other. In any other case, (dimension higher than two) we project the two loops from which we extract the most parallelism, minimize communications and maintain a good load balance onto the mesh. The rest of the loops are sequentialized. 3) Distribution among the PEs of the variables participating the algorithm according to the indexing mode and the type of distribution chosen. 4) Design of the parallel algorithm. 5) Optimization of the algorithm through the iteration of steps 2 and 3.

In step 2 we have chosen the parallelization of 2 loops and the sequentialization of the rest, increasing data redundance, but reducing the communication load. A similar simplification was adopted by Gupta and Banerjee [GuBa92]. To limit the number of dimensions to two does not usually imply a loss of effective parallelism, and on the other hand, most real scientific algorithm have less than three dimensions.

2.1 Data distribution and broadcasting

The sequential QIF algorithm is divided into two parts: WZ factorization of matrix A, and solution of the system WZx = *b*. The second part is structured in two stages, solution of system *Wy*=*b* and solution of *Zx*=*y*.

In the description of the QIF algorithm an inherent dependence between the iterations can be observed. In iteration *k*, the evaluation of columns *k* and *N*-*k*-1 of *W* and of rows *k* and *N*-*k*-1 of *Z*, equations (3) and (4), require updating matrix *A* in the previous iteration (*k*-1) by means of equation (5). This dependence will hinder the parallelization of the QIF algorithm. The maximum number of independent loops is two, (2-partition) and for a generic mesh of *X*×*Y* PEs, the rows of the mesh, *X*, will be associated to the rows of the matrices, and the columns, *Y*, of the mesh to the columns of the matrices.

On the other hand, we can save memory and accelerate the calculation process by means of an *in place* implementation of the algorithm. Initially, matrix *Z* stores matrix *A* and vector *y* contains the components of vector *b*. This way, equations (3) and (6) will not have to be evaluated, and the process of updating matrix *A*, equation (5), and vector *b*, equation (7), will be carried out directly in matrix *Z* and in vector *y*, respectively.

In solving the system WZx=*b*, we find two nested loops corresponding to the index of the iteration and the number of elements in vectors *x* and *b*. The first one of these loops is not parallelizable as there is an inherent dependence in the calculation of the elements of *x* (after each iteration the rest of vector b must be updated using equation (7)). In this first part we could use a 1-partition of the mesh, but this would force us to introduce a massive routing stage for changing the partition when the factorization of *A* has ended. In order to avoid this partition change, we maintain the 2-partition associating the elements of vectors *x* and *y* to the rows of the mesh, *X*. With this partition we introduce redundancy in the distribution of *x* and *y* in the local memories of the PEs of the mesh.

The number of PEs assigned to each dimension are Q_0 and Q_1, with the limitations given by $0<Q_0\leq X$ and $0<Q_1\leq Y$. The 2-partition permits the representation of the index *r* of each processing element in the mesh, PE(r), by means of a vector (r_1, r_0), with $r=r_1\cdot Q_0+r_0$ where r_1 and r_0 indicate the column and row in which PE(*r*) is located in the mesh. Thus, r_0 is associated to the rows of *W* and *Z* as well as to vectors *x* and *y*, whereas r_1 is associated to the columns of *W* and *Z*.

The calculation of the determinant, necessary for solving the systems (2×2) of equations (4) and (10), requires carrying out operations with elements of *Z* that are specularly symmetric with respect to the central axis of the matrix. In order to eliminate an excessive information exchange in this operation, it is convenient to store these elements in the same PE. To do this we use folded matrices. The folding consists in transforming matrix $A_{i\,j}$ into matrix $Z_{i\,j}^m$ (the algorithm is *in place* and matrix *Z* initially contains matrix *A*), representing with the third index, 0≤m≤3, those elements which are specularly symmetric in the four quadrants of the original matrix :

$$Z_{ij}^0 = A_{ij} \qquad Z_{ij}^1 = A_{i,N-j-1}$$
$$Z_{ij}^2 = A_{N-i-1,j} \qquad Z_{ij}^3 = A_{N-i-1,N-j-1} \qquad \textbf{(12)}$$

This way we convert matrix *A*(*N*,*N*) into matrix *Z*(*N*/2, *N*/2, 4). The transformation reduces the diffusion time of specularly symmetric data. Matrix *W* will also be folded and converted into three-dimensions, using the same process as for *Z*. We will also transform vector *b*(*N*) into a matrix *y*(*N*/2, 2) so that

$$y_i^0 = b_i \qquad y_i^1 = b_{N-i-1} \qquad \textbf{(13)}$$

obtaining the solution of the system in matrix *x*(*N*/2,2).

As we have stored in the same PE elements which are specularly symmetric with respect to the central axis of matrix A, the loops associated with the third index, m, will be completely executed by each processor without any need for communications.

According to these considerations, each PE(r_1, r_0) will store two local submatrices LZ(r_0, r_1, 4) and LW(r_0, r_1, 4) of dimensions ($n_0 \times n_1 \times 4$) and two local vectors LY(r_0, 2) and LX(r_0, 2) of dimensions ($n_0 \times 2$), where

$$n_0 = \left\lceil \frac{(N+1)/2}{Q_0} \right\rceil = \left\lceil \frac{n+1}{Q_0} \right\rceil \qquad n_1 = \left\lceil \frac{(N+1)/2}{Q_1} \right\rceil = \left\lceil \frac{n+1}{Q_1} \right\rceil \qquad \textbf{(14)}$$

If we analyze the sequential algorithm we observe a high level of locality in the operations to be carried out. For this reason, we will adopt a cyclic scheme for the distribution of the variables and a pure binary indexing for the PEs. The distribution of the elements is as follows:

(a) Elements Z_{ij}^m and W_{ij}^m are stored in position ($\lfloor i/Q_0 \rfloor$, $\lfloor j/Q_1 \rfloor$, m) of the local submatrices LZ and LW, respectively, in those processors with r_0=(i mod Q_0) and r_1=(j mod Q_1).

(b) Elements Y_i^m and X_i^m are stored in position ($\lfloor i/Q_0 \rfloor$, m) of the local subvectors LY and LX respectively, in all the PEs with r_0=(i mod Q_0) and $0 \le r_1 < Q_1$.

This data distribution scheme will make interprocessor communications necessary. For this algorithm, the communications needed are the broadcasting of a message stored by a PE to the rest of the column, X *Broadcasting*, to the rest of the row, *Y Broadcasting* , or to the rest of the mesh, *XY Broadcasting*.

2.2 Parallel QIF

2.2.1 Parallel WZ factorization

We will use the SCMD (*Single Code Multiple Data*) programming model. The parallel algorithm executed by each processor for the WZ factorization of matrix A can be written as follows:

```
void WZ()
   {
L1.  for (k=0; k<n; k++)
      {
L2.    Calculation of the local indices corresponding
       to element (k,k): s=k/Q_0, t=k/Q_1.
L3.    Calculation of the determinant:
       det=$LZ_{ij}^0 \cdot LZ_{ij}^3 - LZ_{ij}^1 \cdot LZ_{ij}^2$
L4.    X broadcasting of det and of row k in Z
L5.    for (i=(k+1)/$Q_0$; i<$n_0$; i++)
L6.      Calculation of column $LW^m$[i][t]      eq (4)
L7.    Y broadcasting of column k in W
L8.    for (i=(k+1)/$Q_0$; i<$n_0$; i++)
L9.        for (j=(k+1)/$Q_1$; j<$n_1$; j++)
L10.           { Update $LZ^m$[i][j] }      eq (5)
      }
   }
```

The structure of the sequential algorithm is maintained, but the number of iterations in each loop is given by the dimensions of the local matrices LZ and LW. In each iteration of the non parallelizable loop (line L1) two rows of Z and two columns of W are calculated, and the elements of matrix A, are updated. The calculation of the columns of W, equation (4), is carried out in lines L3-L6. Line L3 calculates the determinant of the system (2×2). After this, in L4, the determinant and the row of Z that was updated in the previous iteration are broadcast in the X direction. These elements will be updated for the calculation of the columns of W in L6, and for updating Z, in L10. In L7, columns k and N-k-1 of W, which have been calculated in this iteration k, are broadcast to the PEs that will update LZ. This update is performed by means of lines L8-L10 (equation (5) is used in L10).

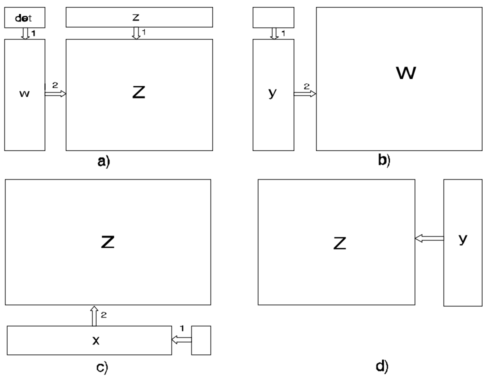

Fig. 1 Parallelism in the different phases of the QIF algorithm: a) WZ; b) $Wy=b$; c) and d) $Zx=y$.

A graphic representation of the parallelism extracted for the WZ algorithm, which is executed in two steps can be seen in figure 1 (a). In the first step (1 in the figure) the determinant and the row of Z are broadcast. After the calculation of the column of W, in the second step (2 in the figure), it is broadcast to the submatrix of Z for updating purposes. This scheme is iterated for the next k in the already updated submatrix of Z.

2.2.2 Parallel solution of linear systems

The code we now present solves the system $Wy=b$.

```
void vector_y()
```

```
    {
L1.   for (k=0; k<n; k++)   {
L2.     Calculation of s and t.
L3.     X broadcast of $LY^m[s]$
L4.     for (i=(k+1)/$Q_0$; i<$n_0$; i++)
L5.             { Update $LY^m[i]$ }                    eq (7)
L6.     Y broadcast of LY[*][*]   }
    }
```

The components *k* and *N*-*k*-1 of vector *y*, are obtained in each iteration *k* of the *n* iterations corresponding to the loop of line L1. In L3, The two components which were updated in the previous iteration are broadcast in the *X direction*. With this information, the rest of vector y is updated in L5 according to equation (7). Finally, in line L6, these modifications are broadcast in the Y direction *to* the PEs of the same row. In figure 1 (b) we present the broadcasting process (step 1) corresponding to line L3 and the one corresponding to L6 (step 2), after updating *y*.

In the solution of the system *Zx*=*y*, two situations can be discriminated, *N* even or *N* odd. If *N* is odd, we first obtain the central element x_n of vector *x*; and then obtain the rest of the elements by pairs in *n* iterations. If *N* is even, all the elements of the vector are obtained by pairs in *n*+1 iterations. The code for calculating the central element of vector *x* is shown below.

```
void Central_element()
    {
L1. Calculation of s and t for position (n, n)
L2. Calculation of LX[s][0]=LX[s][1]=$y_n/z_{nn}$     eq(8)
L3. XY broadcast of LX[s][0]
L4. for (i=0; i<$n_0$; i++)
L5.     { Update $LY^m[i]$ }                          eq(9)
L6. Y broadcast of LY[*][*]
    }
```

In line L1, we determine the local coordinates at which element $z_{n\ n}$, used for the calculation of the central element of *x*, x_n, is stored. The calculation is carried out in L2 according to equation (8). This information is broadcast to the whole mesh in order to update the rest of vector LY in line L5, following equation (9). The updated vector, *y*, is broadcast in the Y direction before going on to the next stage.

Finally, the parallel program for obtaining the rest of vector *x* (or the whole vector if *N* is even) is shown below.

```
void Vector_x(p)
    {
L1.  for (k=p-1; k≥0; k--)  {
L2.     Calculation of s and t.
L3.     Calculation of the determinant:
        det=$LZ_{ij}^0 \cdot LZ_{ij}^3 - LZ_{ij}^1 \cdot LZ_{ij}^2$
L4.     Y broadcast of  det and row k of Z
L5.     Calculation of $LX^m[s]$                      eq (10)
L6.     X broadcast of $LX^m[s]$
L7.     for (i=0; i<$n_0$; i++)
L8.          { Update  $LY^m[i]$ }                    eq (11)
L9.     Y broadcast of LY[*][*]
        }
    }
```

Line L3 calculates the determinant of the linear system (2×2) of equation (10). In L4 this determinant and row *k* of *Z* are broadcast for the calculation of LX according to equation (10) in line L5. This information is broadcast in the X direction, L6, before performing the updating of the rest of vector *y* following equation (11) in line L8. The updated LY vector is broadcast before starting with the next iteration. In figure 1 (c) the broadcast (step 1) corresponds to line L4 and L6 to the broadcast (step 2). In figure 1 (d) the broadcast of the updated LY vector corresponding to L9 can be viewed.

3. Evaluation

The characteristics of the parallel QIF algorithm are a function of the variables which define the size of the problem (number of rows and columns of matrix *A*, *N*) and of the dimensions of the mesh, *X*×*Y*. The complexity corresponding to the WZ factorization is,

$$O\left[4\ \frac{N}{2}\ (n_0 n_1\ +\ n_1 a\ +\ n_0 b)\right] \tag{15}$$

where

$$n_0=\left\lceil\frac{n+1}{Q_0}\right\rceil \quad n_1=\left\lceil\frac{n+1}{Q_1}\right\rceil, \quad a=Q_0-1 \quad b=Q_1-1. \tag{16}$$

The term ($n_0\ n_1$) has to do with the local calculation operations in the PEs and the terms ($n_1\,a$) and ($n_0\,b$), with the operations for the exchange of rows of matrix *Z* and columns of *W*, respectively. The factor *N*/2 is due to the non parallelizable loop, which is associated with the number of iterations *k*, and the factor of 4 is due to the folding of the matrix. The second part of the algorithm, solution of the system for obtaining vector *x*, which includes functions **vector_y**, **vector_x** and **central_element**, has a complexity of:

$$O\left[\frac{N}{2}\ (4n_1\ +\ 2n_0\ +\ 2b(2n_1\ +\ n_0)\ +\ a\right] \tag{17}$$

where we have terms due to local calculations in the

PEs, $(4n_1+n_0)$, terms associated with the communications between PEs, $(2b(2n_1+n_0)+a)$, and a factor of $N/2$ associated with the number of iterations.

The total complexity of the algorithm is the sum of the complexities associated with the WZ factorization and solving the linear system. But the most influential term, specially when the dimension N of the matrix is large, is the one introduced by the WZ factorization. This is why the dimensions of the mesh, $X \times Y$, which lead to the shortest execution times for the algorithm are those verifying $X=Y$, conclusion obtained from the minimization of the complexity in equation (15). If the complexity dominating the algorithm was the second part, equation (17), when we minimized it we would reach the conclusion that we would be interested in $X>Y$. In fact, if we have 8 PEs, the algorithm is faster on a 4×2 mesh then on a 2×4 mesh.

Another interesting feature that can be inferred from equation (15) is that the algorithmic complexity of the parallel algorithm becomes the algorithmic complexity of the sequential algorithm ($O[N^3/2]$) when we consider a 1×1 ($Q_0=1$ and $Q_1=1$ and therefore $a=b=0$) mesh.

With respect to the data distribution after folding the matrix, we have used a cyclic distribution. This distribution balances the load better, but it penalizes communications in the sense that the broadcasts must be global to the whole column. If we had implemented a consecutive distribution scheme we would have unbalanced the load, but we would have also minimized communications as the broadcasts would be partial. For example, the broadcast to the whole column would not be necessary, this broadcast would be in one direction, up or down, but not in both. This last distribution scheme dynamically frees PEs during execution, and for this reason can be interesting in multi user environments.

Another one of the methods used for solving linear systems is Gaussian Elimination which is based on the LU factorization of the system matrix. Zapata et al. [RDBZP90] parallelized the LU factorization for hypercube computers. The complexities of the LU and WZ parallel algorithms are compared in [GMBZ90], where it is concluded that for large N the ratio between the WZ/LU complexities is the same for parallel algorithms and their sequential counterparts, 1/2.

In table I we present the execution times (in seconds) of the QIF algorithm proposed in this work for different matrix sizes, N, and different mesh sizes, $X \times Y$. In figure 2 we show the efficiency calculated for these times. It can be observed that the efficiency increase as the size of the problem (N) increases with respect to the dimensions of the mesh ($X \times Y$). This behavior is due to the fact that the number of communication operations grows less rapidly with the size of the problem than the number of local computations. It can be observed that the efficiency comes close to its optimum value, 1, when $N \gg X \cdot Y$ that is, when the local operations predominate over the communication instructions.

Finally, in figure 3 we can see a comparison between the execution times of the parallel algorithms for the two topologies, hypercube and mesh. We have considered the following partitions ({(hypercube),mesh}): {(2,2),4×4}, {(2,1),4×2} y {(1,1),2×2}. It can be observed how the execution time in the mesh topology is approximately the same (slightly shorter) as in the hypercube. This is justified because the broadcasting time in the mesh is longer than in the hypercube, but the indexing time in the mesh is shorter. However, for large sizes of the mesh, the broadcasting time will grow significantly, whereas the indexing time will decrease, so we can assume that the execution time in the mesh will be longer than in the hypercube. When this happens we can use a toroidal topology so that the broadcasting does not penalize the process so much. As a conclusion we can say that computers with mesh topology are advantageous with respect to hypercube computers as they represent lower hardware cost and a similar parallel performance.

Table I Execution times of QIF algorithm (sizes *N*) on several meshes (PEs).

N//PE	**4×4**	**4×2**	**2×2**	**2×1**	**1×1**
100	0,608	0,795	1,275	2,134	3,967
200	3,002	4,760	8,535	15,484	29,843
300	8,847	14,957	27,042	50,565	99,197
400	18,734	32,244	60,163	118,190	234,782
500	35,476	61,865	115,136	220,738	457,067
600	56,721	103,680	203,771	393,301	787,107
700	88,642	163,667	322,451	625,067	1246,92
800	133,938	245,787	479,279	920,418	1857,21
900	192,274	350,050	679,823	1318,254	2545,067
1000	245,170	474,768	921,190	1796,703	3506,754

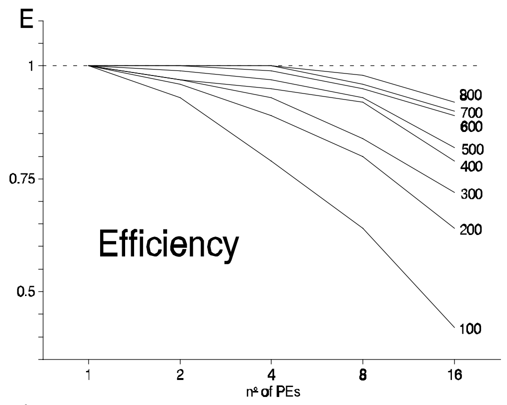

Fig 2. Efficiency (*E*) versus number of PEs for different size of the matrix.

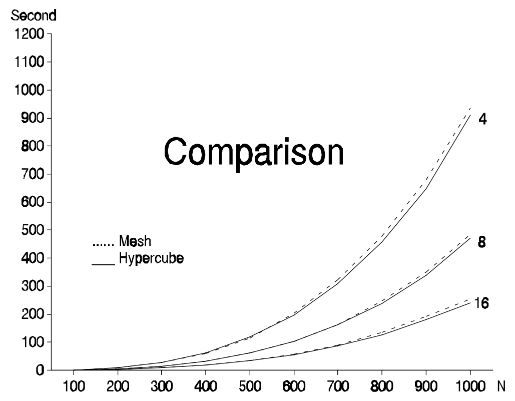

Fig. 3 Processing times of mesh and hypercube.

References

[Evan82] D.J. Evans, Parallel numerical algorithms for linear systems, in: *Parallel Processing Systems*, D.J. Evans, Ed. (Cambridge University Press, 1982) 357-383.

[EvHa79] D.J. Evans and M. Hatzopoulos, A parallel linear system solver, *Internat. J. Comput. Math. Sect. B* 7 (1979) 227-238.

[EvHa80] D.J. Evans and A. Hadjidimos, A modification of the Quadrant Interlocking Factorization Parallel Method, *Internat. J. Comput. Math. Sect. B* 8 (1980) 149-166.

[EHN81] D.J. Evans, A. Hadjidimos and D. Noutsos, The parallel solution of banded linear equation by the new Quadrant Interlocking Factorization (Q.I.F.) Method, *Internat. J. Comput. Math. Sect. B* 9 (1981) 151-161.

[GMBZ90] I. García, J.J. Merelo, J.D. Brugera and E.L. Zapata, Parallel quadrant interlocking factorization on hypercube computers, *J. Parallel Comput.* 15 (1990) 87-110.

[GuBa92] M. Gupta and P. Banerjee, Demostration of Automatic Data Partitionig Techniques fo Parallelizing Compilers on Multicomputers, *IEEE Trans. on Parallel and Distr. Systems*, 3 (2) (1992) 179-193.

[HaEv88] M. Hatzopoulos and D.J. Evans, Comments on the paper "A short proof for the existence of the WZ-factorization, *J. Parallel Comput.* 6 (1988) 259.

[Hatz82] M. Hatzopoulos, Parallel linear solvers for tridiagonal systems, in: *Parallel Processing Systems*, D.J. Evans, Ed. (Cambirdge University Press, 1982) 384-393.

[KaSc87] M. Kaps and M. Schlegl, A short proof for the existence of the WZ-factorization, *J. Parallel Comput.* 4 (1987) 229-232.

[Rive90] F.F. Rivera, Partición y proyección de algoritmos en computadores hipercubo: Reconocimiento de formas, Ph.D. Thesis (in Spanish), Fac. of Phisics, Univ. of Santiago de compostela, Spain, 1990.

[ZRP90] E.L. Zapata, F.F. Rivera and O.G. Plata, On the partition of algorithms into hypercubes, in: Advances on Parallel Computing, D.J. Evans, Ed. (JAI Press, 1990) 149-171.