Optimum Data Distributions for Parallel Partitioned LU Decomposition

Julius Dichter[1], Ausif Mahmood[1], and Howard Sholl[2]

[1] Department of Computer Science and Engineering, University of Bridgeport, Bridgeport CT, 06601
[2] Department of Computer Science and Engineering University of Connecticut, Storrs CT, 06269

Abstract

Abstract: The solution of a system of linear equations is required in many applications. The maximum concurrency available in LU decomposition is well understood [1], but it is not trivial to determine the proper assignment of matrix blocks to processors so as to maximize concurrency and minimize communication. We carry out a detailed analysis of the communication requirements in parallel dense matrix LU decomposition and develop optimal assignment schemes for varying degrees of granularity. We prove and experimentally validate that the overall communication cost obtainable from this scheme is within 5% off the optimal case, if such a case existed.

1. Introduction

The solution of a linear system of equations is ultimately encountered in many engineering and scientific problems such as PDE and ODE solvers [1-2], finite element analysis [3] and VLSI circuit simulation [4-5]. These programs often use implicit methods for numerically solving systems of differential equations which transform the problem to a system of nonlinear equations. The non-linear equations are further converted to a system of linear equations by employing a Newton-type process. The resulting matrix equation is of the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $m \times m$ matrix and $\boldsymbol{x}$ and $\boldsymbol{b}$ are $m \times 1$ vectors. Although Gaussian elimination can be used to solve this matrix equation for the unknown x's, it is not efficient in cases where the coefficient matrix $\boldsymbol{A}$ stays the same but the $\boldsymbol{b}$ vector changes many times. A variation of the Gaussian elimination is LU decomposition (time complexity $O(n^3)$ for an $n \times n$ matrix) in which the $\boldsymbol{A}$ matrix is transformed into a product of lower and upper triangular matrices. After $\boldsymbol{A}$ has been decomposed into its $\boldsymbol{L}$ and $\boldsymbol{U}$ factors, two triangular equations $\boldsymbol{Ly} = \boldsymbol{b}$ and $\boldsymbol{Ux} = \boldsymbol{y}$ are solved to find the unknown x's. The solution of a triangular equation has a time complexity of $O(n^2)$. The maximum available parallelism in LU decomposition for dense matrices has been previously identified [6,7]. The schemes presented in [6] and [7] are equivalent in terms of overall parallelism, but differing in execution sequence. We use the decomposition execution sequence developed by Hwang and Cheng in [6] (see Figure 1). In [6], a given matrix $\boldsymbol{A} = (a_{ij})$ is partitioned into n^2 submatrices (or blocks) each of size $m \times m$. The parallel LU decomposition algorithm first computes the LU decomposition using a serial LU decomposition algorithm in the matrix block $A_{0,0}$. Then after computing $L_{0,0}^{-1}$ and $U_{0,0}^{-1}$, in step 2, all remaining blocks in the first row and first column can be computed in parallel for their U and L computations respectively. Similarly step 3 computes the LU decomposition in the diagonal block $A_{1,1}$ followed by computation of $L_{1,1}^{-1}$, $U_{1,1}^{-1}$, and then step 4 concurrently computes the U factors in row 1, and L factors in column 1. The computation proceeds in a similar manner until the LU decomposition is performed in the last diagonal block in step $2n+1$, where n is the number of blocks in a row or column. As an example, $U_{3,4}$ is computed by first determining the A-prime $(A_{3,4})' = A_{3,4} - L_{3,0}\,U_{0,4} + L_{3,1}\,U_{1,4} + L_{3,2}\,U_{2,4}$. Then, $U_{3,4} = (L_{3,3})^{-1}\,(A_{3,4})'$. The existing parallel LU decomposition algorithms do not identify how the different blocks should be assigned to available processors so as to minimize the overall parallel execution time including communication costs involved. This is often referred to as the data distribution problem and has drawn recent research interest [8-11]. The work reported in [8] has been directed at finding the optimal data distribution scheme for LU decomposition. However, their approach targets the case when a serial LU decomposition algorithm is parallelized by the conventional elimination by rows technique. Another recent implementation of parallel LU decomposition on a Transputer array [11] has proposed a somewhat efficient data distribution scheme. However, neither achieves least communication cost and maximum concurrency.

2. Communication in Parallel LUD

In the parallel LU decomposition algorithm of Hwang and Cheng [6], the matrix is divided into blocks which define the logical processing units as well as the computational granularity. A 1000×1000 matrix could be divided into 10^2 blocks with each block being a submatrix of size 100×100. Dividing the matrix into 20^2 blocks (each block of size 50×50) would reduce the granularity and result in more parallelism but also more. The parallel LU decomposition algorithm follows a wavefront type of execution sequence as was indicated by Figure 1. In each iteration of the algorithm the diagonal block computation is followed by computation of blocks to the right of and below the diagonal. Figure 2 shows the computation sequence for a matrix in terms of the iterations ($\boldsymbol{I}_i$'s) of the parallel LU decomposition algorithm. As the algorithm progresses, the blocks in $\boldsymbol{I}_0$ execute, followed by blocks in $\boldsymbol{I}_1$, and so on until the last $\boldsymbol{I}_n$ finishes. The diagonal blocks are marked with asterisks. $\boldsymbol{I}_i$* is the current iteration, which is shown shaded for clarity.

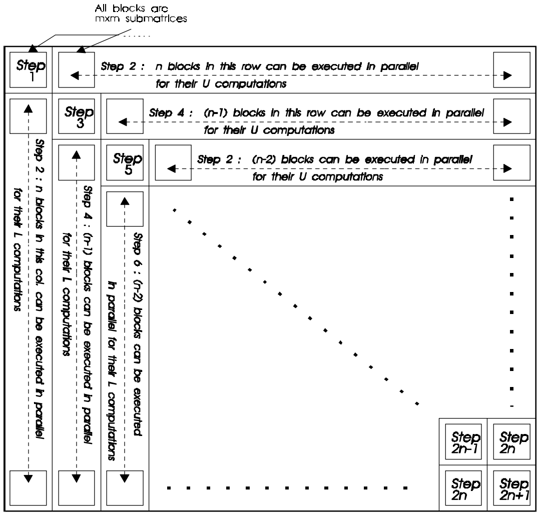

Figure 1. Sequence and Parallelism in LUD

Parallel Partitioned LU Decomposition Algorithm

PARALLEL_PARTITIONED_LUD(***A***)
for $q = 0$ to n /* $n+1$ = number of blocks in a row/col */
 if ($q > 0$) then compute_A_prime(A,q,q)
  Decompose $A_{q,q}$ into $L_{q,q}.U_{q,q}$
 if ($q < n$) then /* not the last iteration */
  Compute inverse matrices $L_{q,q}^{-1}$ and $U_{q,q}^{-1}$
   /* S1 and S2 are computed in parallel */
  S1: for all $p = (q+1)$ to n do
    if ($q > 0$) then compute_A_prime(A,p,q)
     compute $L_{p,q} = A_{p,q} \,.\, U_{q,q}^{-1}$
S2: for all $p = (q+1)$ to n do
    if ($q > 0$) then compute_A_prime(A,q,p)
     compute $U_{q,p} = L_{q,q}^{-1} \,.\, A_{q,p}$
end PARALLEL_PARTITIONED_LUD

compute_A_prime(A,p,q)
 $A' = 0$; $r = \min(p,q)$
 for $s = 0$ to r-1 do
   $A' = A' + L_{p,s} \,.\, U_{s,q}$
 $A_{p,q} = A_{p,q} - A'$
end *compute_A_prime*

2.1 Analysis of Communication Costs

The communication cost required in the parallel LU decomposition algorithm is composed of three components: the *distribution*, *UL*, and *diagonal* costs. The distribution cost is any diagonal block (i,i) sending its $L_{i,i}^{-1}$ or $U_{i,i}^{-1}$ to blocks $(i, i+1:n)$ and $(i+1:n, i)$ i.e., sending the inverse to blocks in the same row and to blocks in the same column. The *UL* cost refers to communication involved in computation of U and L values in the off-diagonal blocks (i,j) where $i \neq j$. Blocks (i,j) where $i < j$ (U cost), receive communication from blocks $(0{:}i\text{-}1,j)$ and $(i,0{:}i\text{-}1)$. Blocks (i,j) where $i > j$ (L cost), receive communication from blocks $(0{:}j\text{-}1,j)$ and $(i,0{:}j\text{-}1)$. Diagonal costs refer to diagonal blocks (i,i) which receive communication from blocks $(0{:}i\text{-}1,i)$ and $(i,0{:}i\text{-}1)$. As an example, from the parallel LU decomposition algorithm of Section 1, $U_{3,4}$ is computed by first determining the $(A_{3,4})'$, as in $(A_{3,4})' = A_{3,4} - L_{3,0}\,U_{0,4} + L_{3,1}\,U_{1,4} + L_{3,2}\,U_{2,4}$.

I_0*	I_0	I_0	I_0	I_0	I_0
I_0	...*	…	…	…	…
I_0	…	I_i*	I_i	I_i	I_i
I_0	…	I_i	…	…	…
I_0	…	I_i	…	I_{n-1}*	I_{n-1}
I_0	…	I_i	…	I_{n-1}	I_n*

Figure 2. Computation Sequence in terms of Iterations of the Parallel LUD Algorithm

Then, $U_{3,4}=(L_{3,3})^{-1}(A_{3,4})'$. Thus the block(3,4) expects communication from blocks (0:2,4) and blocks (3,0:2) i.e., U values need to be received from blocks (0,4), (1,4) and (2,4); and L values from blocks (3,0), (3,1), and (3,2), resulting in a total U cost of 6 communication units for block(3,4). Similarly the *distribution* and *diagonal* communication costs can be followed from the computation requirements in the parallel LU decomposition algorithm of Section 1. Figure 3 depicts some examples of the different communication costs involved. For a matrix divided into $(n+1)\times(n+1)$ blocks, the *worst case communication cost* (assuming every block is assigned to a different processor) in terms of communication units is given by (1)

$$2\sum_{i=1}^{n} i + 4\sum_{i=1}^{n-1}\sum_{j=i}^{n-1} i + 2\sum_{i=1}^{n} i \tag{1}$$

which simplifies to

$$\frac{2n^3 + 6n^2 + 4n}{3} \tag{2}$$

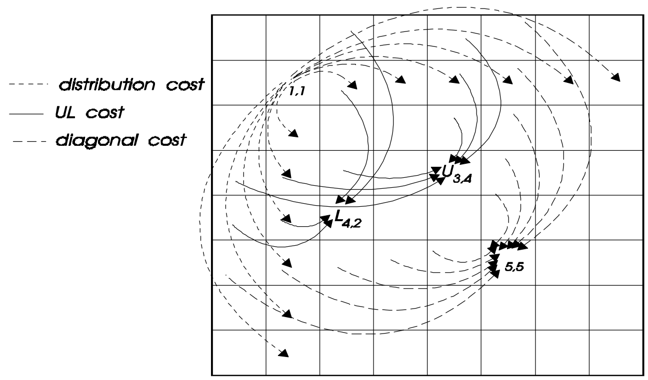

Figure 3. *Distribution* Cost for block(1,1), *U* Cost for block(3,4), *L* Cost for block(4,2) and *Diagonal* Cost for block(5,5)

3. Fixed Granularity Case

In the case of fixed granularity, if there are n available processors, then the matrix is partitioned into $(n/2+1)\times(n/2+1)$ blocks. This way n processors can operate on n number of blocks in a row and column of the matrix besides the diagonal block, yielding a parallel execution time of one time unit per iteration. During the first iteration (I_0), half the processors i.e., $n/2$ processors are executing to the right of the diagonal block and the other $n/2$ below the diagonal. In the next iteration I_1, $n/2$-1 processors are executing to the right and another $n/2$-1 below the diagonal, and so on. The following four lemmas are proved in [10]. They describe the processor-to-block assignments which absolutely minimize communication in this case.

Lemma 1
To guarantee a minimum parallel execution time of one time unit for any iteration i blocks $(i, i+1{:}n/2)$ and blocks $(i+1{:}n/2, i)$ must be assigned to distinct processors.
Lemma 2
For block (i,j) where $i < j$ and $i > 1$, at most half of the U communication cost can be saved without affecting the minimum parallel execution time of an iteration i.
Lemma 3
If block(i,i) is assigned to processor P_m, then to minimize the diagonal cost in an iteration i, for $i > 1$, assign processor P_m to one of the blocks in each of the following pairs:

$$\{(1,i), (i,1)\},\ \{(2,i), (i,2)\},\ \ldots\ldots,\ \{(i-1,i), (i,i-1)\} \tag{3}$$

Lemma 4
To minimize all the communication costs (*distribution*, *diagonal* and *UL*), we assign the n processors to the $n/2+1$ 2 blocks according to Figure 4.
The communication cost for the optimal assignment shown in Figure 4 is given by (4),

$$2\sum_{i=1}^{n/2} i - n + 2\sum_{i=1}^{n/2-1}\sum_{j=i}^{n/2-1} i + \sum_{i=1}^{n/2} i \tag{4}$$

which simplifies to (5).

$$\frac{2n^3 + 18n^2 + 4n}{48} \tag{5}$$

$F(n)$, the ratio of optimal to worst case communication for fixed granularity is obtained from both, equations (2) and (5), modified for an $(n/2+1)\times(n/2+1)$ size matrix, is,

$$F(n) = \frac{2n^3 + 18n^2 + 4n}{4n^3 + 24n^2 + 32n} \tag{6}$$

4. Variable Granularity Case

In the variable granularity case, a processor can be assigned to more than one off-diagonal matrix block in an

X	P_1	P_2	P_3	P_4	..	$P_{n/2-3}$	$P_{n/2-2}$	$P_{n/2-1}$	$P_{n/2}$
$P_{n/2+1}$	**X**	P_2	P_3	P_4	..	$P_{n/2-3}$	$P_{n/2-2}$	$P_{n/2-1}$	$P_{n/2}$
$P_{n/2+2}$	$P_{n/2+2}$	**X**	P_3	P_4	..	$P_{n/2-3}$	$P_{n/2-2}$	$P_{n/2-1}$	$P_{n/2}$
$P_{n/2+3}$	$P_{n/2+3}$	$P_{n/2+3}$	X	P_4	..	$P_{n/2-3}$	$P_{n/2-2}$	$P_{n/2-1}$	$P_{n/2}$
$P_{n/2+4}$	$P_{n/2+4}$	$P_{n/2+4}$	$P_{n/2+4}$	**X**	..	$P_{n/2-3}$	$P_{n/2-2}$	$P_{n/2-1}$	$P_{n/2}$
...	...				..	...		...	...
...	...				..	...		...	...
P_{n-2}	P_{n-2}	P_{n-2}	P_{n-2}	P_{n-2}	..	P_{n-2}	**X**	$P_{n/2-1}$	$P_{n/2}$
P_{n-1}	P_{n-1}	P_{n-1}	P_{n-1}	P_{n-1}	..	P_{n-1}	P_{n-1}	**X**	$P_{n/2}$
P_n	P_n	P_n	P_n	P_n	..	P_n	P_n	P_n	**X**

Figure 4.
Optimal Assignment of Matrix Blocks to Processors

iteration. Thus the block size and the number of blocks in which the matrix is to be partitioned can be adjusted to match the communication performance of the underlying system. We first analyze a base case for granularity in which for the n available processors, there are exactly $2n$ blocks in the I_0 iteration exclusive of the diagonal block, i.e., a processor executes at most two blocks in an iteration. We extend the ideas to develop a scheme for variable granularity without any restriction to the number of blocks a processor executes in an iteration.

4.1 Base Case for Variable Granularity

The general form of our optimized assignment scheme for the base case in variable granularity with n processors requiring a matrix to be divided into $(n+1)\times(n+1)$ blocks is shown in Figure 5. It is developed from an extension of the ideas for the fixed granularity case as described in Section 3 (Figure 4). The communication cost for the base granularity case, $B(n)$, as shown by the assignment in Figure 5 is given by (7). Its simplification is given in (8).

$$2\sum_{i-1}^{n-1}\sum_{j=i}^{n-1} i - 2\sum_{i=1}^{n/2} i + [\frac{n}{2}-1] + \sum_{i-1}^{n} i + \sum_{i=1}^{n}[2i-1] - \frac{n}{2} \tag{7}$$

$$B(n) = \frac{4n^3 + 15n^2 - 4n - 12}{12} \tag{8}$$

To gauge the quality of the assignment scheme of Figure 5, we need to compute a *lower bound*. The lower bound is developed by assuming that the communication cost of an iteration i depends only on the individual assignments in iterations 1 to i-1, i.e., the cost with respect to each prior iteration is treated individually (see [10] for details). The lower bound communication cost, $L(n)$, for the base granularity case is given by (9).

$$L(n) = \frac{4n^3 + 9n^2 - 10n}{12} \tag{9}$$

$\mathbf{P_n}$	P_n	$P_{n\text{-}1}$	$P_{n\text{-}2}$	$P_{n\text{-}3}$	..	P_4	P_3	P_2	P_1
P_1	$\mathbf{P_1}$	P_1	$P_{n\text{-}2}$	$P_{n\text{-}3}$	..	P_4	P_3	P_2	P_1
P_2	P_2	$\mathbf{P_2}$	P_2	$P_{n\text{-}3}$	..	P_4	P_3	P_2	P_1
P_3	P_3	P_3	$\mathbf{P_3}$	P_3	..	P_4	P_3	P_2	P_1
...	...				..	...		...	...
...	...				..	...		...	...
$P_{n\text{-}2}$	$P_{n\text{-}2}$	$P_{n\text{-}2}$	$P_{n\text{-}2}$	$P_{n\text{-}2}$	..	$P_{n\text{-}2}$	$\mathbf{P_{n\text{-}2}}$	$P_{n\text{-}2}$	P_1
$P_{n\text{-}1}$	$P_{n\text{-}1}$	$P_{n\text{-}1}$	$P_{n\text{-}1}$	$P_{n\text{-}1}$	..	$P_{n\text{-}1}$	$P_{n\text{-}1}$	$\mathbf{P_{n\text{-}1}}$	$P_{n\text{-}1}$
P_n	P_n	P_n	P_n	P_n	..	P_n	P_n	P_n	$\mathbf{P_n}$

Figure 5. Processor to Matrix Block Assignment Scheme for the Base Variable Granularity Case of n Processors and an $(n+1)\times(n+1)$ Block Matrix

The quality of our assignment scheme can be found by taking the ratio of the communication cost of our scheme to the lower bound communication cost, (10).

$$R(n) = \frac{B(n)}{L(n)} = \frac{4n^3 + 15n^2 - 4n - 12}{4n^3 + 9n^2 - 10n} \qquad (10)$$

The limit $R(n)$, as $n \to \infty$ becomes unity. It shows that our assignment scheme for the base variable granularity case is very good for large number of processors. Figure 6 shows how $R(n)$ converges as a function of the number of processors.

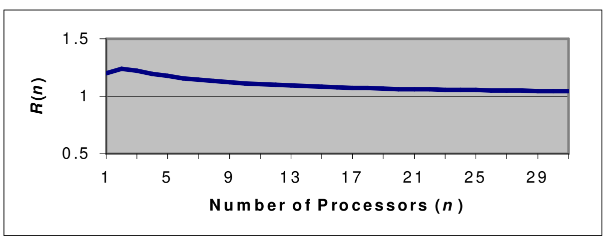

Figure 6. Ratio of the Communication Costs of Our Optimized Assignment Scheme to the Lower Bound

4.2 Generalized Variable Granularity

Now, we extend the ideas of the base case to general variable granularity where a processor can be assigned to many blocks in an iteration. If we allow a variable k to represent the number of times a processor appears in iteration I_0 excluding the diagonal block, then the fixed case is one where k = 1, and the base granularity case as described in Section 4.1 corresponds to k=2. The total number of blocks a matrix gets divided into is a function of the product of n and k. In [10], the theoretical lower bound, $L(n,k)$ and the best assignment $B(n,k)$ costs are computed.

$$L(n,k) = \frac{4n^3k^3 - 8n^2k^3 + 12n^2k^2 + 5n^2k + 6n - 10nk}{12}$$

$$B(n,k) = \frac{4n^3k^3 - 4n^2k^3 + 18n^2k^2 + n^2k - 18nk^2 + 14nk - 12k}{12}$$

Figure 7 shows an example of our optimized processor to block assignment scheme for the case where we have k = 3 repeated groups of n = 4 processors. The assignment is a generalization of the base case for variable granularity shown in Figure 5. Whereas processor sequence n:1 appears in blocks (0,1:n) in Figure 7, it appears repeatedly in blocks (0,1:n), (0,n+1:2n), ..,(0,n(k-1)+1:nk) in the variable granularity case. Similarly in the first column, the n processors are repeated k times in the column blocks. Figure 8 shows that with n=32 processors our communication ratio is about 5% worse than the theoretical lower bound, $L(n,k)$.

4	4	3	2	1	4	3	2	1	4	3	2	1
1	1	1	2	1	4	3	2	1	4	3	2	1
2	2	2	2	1	4	3	2	1	4	3	2	1
3	3	3	3	3	4	3	2	1	4	3	2	1
4	4	4	4	4	4	3	2	1	4	3	2	1
1	1	1	1	1	1	1	2	1	4	3	2	1
2	2	2	2	2	2	2	2	1	4	3	2	1
3	3	3	3	3	3	3	3	3	4	3	2	1
4	4	4	4	4	4	4	4	4	4	3	2	1
1	1	1	1	1	1	1	1	1	1	1	2	1
2	2	2	2	2	2	2	2	2	2	2	2	1
3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4

Figure 7. Optimized Assignment of Processors to Matrix Blocks for, n = 4, k = 3

5. Comparisons of PVM Implementation

In order to verify the effectiveness of our newly developed processor to matrix block assignment, and to compare its performance against a recently reported scheme, we carry out an implementation of the parallel LUD algorithm on a network of workstations using PVM. Figure 9 shows a comparison of execution times of the assignment scheme developed in Section 4.2 to an existing scheme presented in [11]. All experiments used identical SUN-5 workstations connected by 10 Mbit/sec Ethernet processing large (1000-2000 element) square matrices. Figure 9 shows that the assignment scheme developed in this paper improves the execution time over the existing scheme of [11]. The difference between the two schemes decreases as number of workstations are increased, due to packet collisions of the bus topology. If newer networking technologies such as fast and switched Ethernet or ATM were employed, then our assignment scheme would continue to show better advantage.

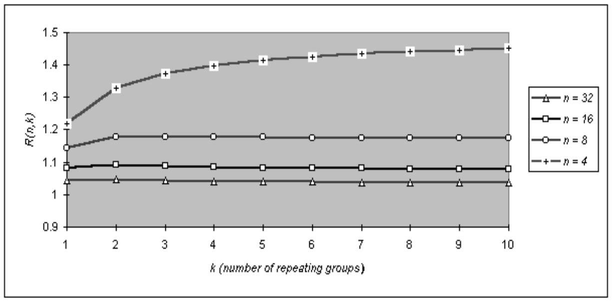

Figure 8.
Plot of $R(n,k)$ as a function of k for various values of n

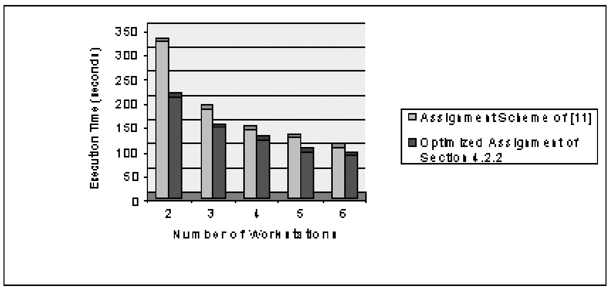

Figure 9.
Comparisons of Processor to Matrix Block Assignment Schemes Under PVM

6. Conclusions

Our paper has addressed the parallel LU decomposition problem from the perspective of data communication. We have developed matrix block to processor assignment schemes for block partitioned parallel LUD algorithm which maintain maximum concurrency while minimizing communication. We have developed an optimum assignment scheme for fixed and variable granularity cases. The quality of this assignment in terms of communication cost was measured against a theoretical lower bound. Despite the fact that the theoretical lower bound on the communication cost is not practically attainable, our assignment generated only 5 percent more communication with n = 32 processors. We feel that our assignment is, in fact, optimal, but do not offer a proof for it. Comparisons of PVM implementations of our matrix block to processor assignment against a recently reported assignment scheme confirm higher gains achievable from our approach. Our future work will focus on developing an efficient assignment scheme for sparse the LU decomposition problem as well as building an LUD performance model, and optimizing n (processor count) and k (granularity).

7. References

[1] Mu, M., Rice, J. R., "Performance of PDE Solvers on Hypercubes," *Proceedings of the Conference on Unstructured Scientific Computation on Hypercubes*, 1992, pp. 345-70.

[2] De Swart, J. J. B.; Blom, J. G., "Experiences with Sparse Matrix Solvers in Parallel ODE Software,"| *Computers & Mathematics with Applications*, vol. 31, no. 9, pp. 43-55.

[3] Kapadia, N.; Lichtenberg, B.; Fortes, J. A. B.; Gray, J. L.; Siegel, H. J.; Webb, K. J., "Parallel Solution of Unstructured Sparse Finite Element Equations," *IEEE Antennas and Propagation Society International Symposium*, 1995.

[4] Trotter, J. A., and P. Agrawal, "Circuit Simulation Algorithms on a Distributed Memory Multiprocessor System," *Proc. of the ICCAD-90*, pp. 438-441, 1990.

[5] Cox, P., R. Burch, et al., "Direct Circuit Simulation Algorithms for Parallel Processing," *IEEE Trans. on CAD of ISCAS*, vol. 10, no. 6, June 1991, pp. 714-725.

[6] Hwang, K. and Y. H. Cheng, "Partitioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Trans. on Computers*, C-31, no. 12, Dec. 1982, pp. 1215-1224.

[7] Navarro, J. J, "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors," *IEEE Computer*, July 1987, pp.77-89.

[8] Rauber, T. and G. Runger, "Optimal Data Distributions for LU Decomposition," *Proc. Of the First International EURO-PAR Conference*, pp. 391-402, 1995.

[9] Desprez, F., P. Fraigniand, A. Mignotte and Y. Robert, "Optimization of the ScaLAPACK LU factorization routine using Communication/Computation overlap," *Proc. Of the Second International EURO-PAR Conference*, pp. 3-10, 1996.

[10] Dichter, J., "Optimizing Performance of Parallel Computation from a Data Communications Perspective", *Doctoral Dissertation*, University of Connecticut, Storrs, May, 1997.

[11] Y. Chu, A. Mahmood, and Donald J. Lynch, "Parallel SOLVE for Direct Circuit Simulation on a Transputer Array," *Proceedings of the 1996 International Conference on High Performance Computing (HiPC),* Dec. 1996, pp. 27-32