LU Decomposition of Banded Matrices and the Solution of Linear Systems on Hypercubes

D. W. Walker, T. Aldcroft, A. Cisneros
G. C. Fox, and W. Furmanski

206-49, California Institute of Technology,
Pasadena, CA 91125, USA

Abstract: We describe the solution of linear systems of equations, Ax = b, on distributed-memory concurrent computers whose interconnect topology contains a two-dimensional mesh. A is assumed to be an $M \times M$ banded matrix. The problem is generalized to the case in which there are n_b distinct right-hand sides, b, and can thus be expressed as AX = B, where X and B are both $M \times n_b$ matrices. The solution is obtained by the LU decomposition method which proceeds in three stages: (1) LU decomposition of the matrix A, (2) forward reduction, (3) back substitution. Since the matrix A is banded a simple rectangular subblock decomposition of the matrices A, X, and B over the nodes of the ensemble results in excessive load imbalance. A scattered decomposition is therefore used to decompose the data. The sequential and concurrent algorithms are described in detail, and models of the performance of the concurrent algorithm are presented for each of the three stages of the algorithm. In order to ensure numerical stability the algorithm is extended to include partial pivoting. Performance models for the pivoting case are also given. Results from a 128-node Caltech/JPL Mark II hypercube are presented, and the performance models are found to be a good agreement with these data. Indexing overhead was found to contribute significantly to the total concurrent overhead.

1. Introduction

The solution of systems of linear equations is of fundamental importance in many fields of science and engineering, and in recent years much work has been done on the efficient solution of such systems on concurrent processors ([Heller 78], [Dongarra 84a], [Ortega 85]). Direct methods, in particular Gaussian elimination and the closely-related LU and Cholesky decompositions, have received much attention. In this paper we describe the solution of narrow-banded, non-symmetric systems of linear equations by means of LU decomposition with partial pivoting, followed by forward reduction and back substitution. The concurrent algorithms that we describe in Sec. 4, 5, and 6 are based on work by [Fox 84], and were designed for use on hypercube multiprocessors, although they will run with very little modification on any distributed memory, MIMD multiprocessor whose interconnect topology contains that of a 2-dimensional grid.

The rest of this section will attempt to convey the scope and fecundity of recent research into the solution of linear systems on multiprocessors by Gaussian elimination and related methods. This is not intended to be a full (or even adequate) review of the current state of research in the field, but should rather provide pointers for the interested reader. Section 2 gives a statement of the problem addressed by this paper. The sequential LU decomposition, forward reduction and back substitution algorithms are outlined in Sec. 3. Sections 4 and 5 discuss the decomposition and communication strategies used in the concurrent implementation, which is described in Sec. 6. In Sec. 7 performance models are developed, and these are compared in Sec. 8 with timings made on the Caltech/JPL 128-node Mark II hypercube. The code used for the timings is written in the C programming language, and the hypercube version makes use of the CrOS III communication system and CUBIX, both of which are described in [Fox 87a]. Conclusions drawn from this work are presented in Sec. 9.

Kung and Leiserson [Kung 80] have described an LU decomposition algorithm for a bi-dimensional systolic array processor (SAP). This method restricts the size of the input matrix for a SAP of a given size. In [Johnsson 81] a Gaussian elimination algo-

rithm for solving banded linear systems on a linear SAP with no size restriction was described. More recently Navarro et al. [Navarro 87] have investigated LU decomposition algorithms for linear SAPs with no size restriction. Thakore and Su [Thakore 87] have described algorithms for performing matrix inversion by Gaussian elimination, and LU decomposition on the SM3 multiprocessor system [Baru 86], and have compared performance models for the SM3 and hypercube multiprocessors. Cholesky factorization of symmetric, positive-definite matrices, and LU factorization with partial pivoting of non-symmetric matrices have been investigated by Geist and Heath [Geist 86]. Chu and George have developed the work of Geist and co-workers further, and have described [Chu 87] an algorithm for Gaussian elimination with partial pivoting. Both Geist and Heath, and Chu and George present timings for the algorithms run on an INTEL iPSC hypercube.

The numerical solution of many types of partial differential equations by both the finite difference and finite element methods leads to banded linear systems of equations (see, for example, Chaps. 7 and 8 of [Fox 87a]). Chan et al. [Chan 86] have considered the solution on hypercubes of banded linear systems generated by the solution of elliptic PDE's, and discuss the Alternating Direction Implicit (ADI) method, Gaussian elimination, and multigrid techniques. The use of the ADI method on multiprocessors to solve parabolic PDE's has recently been investigated by Johnsson et al. [Johnsson 87a, 87b]. Concurrent algorithms for the Cholesky factorization of banded matrices are discussed in [Utku 86]. O'Neil et al. [O'Neil 87] have studied the solution of a three-dimensional elasticity problem by the finite element method, and used a concurrent Gaussian elimination algorithm to solve the resultant linear system on the BBN Butterfly Parallel Processor. Dongarra and Sameh [Dongarra 84b] have considered algorithms for solving narrow-banded, diagonally-dominant linear systems on different concurrent processors, and present timings made on the Cray X/MP-4 and Denelcor HEP multiprocessors. Johnsson has investigated in [Johnsson 85] the communication and calculation complexity of Gaussian elimination and block cyclic reduction for the solution of block tridiagonal systems on multiprocessors with ring, 2-D mesh, and hypercube interconnect topologies. In [Dongarra 87], Dongarra and Johnsson have extended their analysis of Gaussian elimination to bus-based and switch-based shared memory concurrent processors, and give timing measurements for runs made on the Alliant FX/8 and Sequent Balance 21000 multiprocessors.

2. Problem Statement

This paper describes the solution of linear systems of equations of the type:

$$A\mathrm{x}_k = \mathrm{b}_k \tag{1}$$

where the coefficient matrix, A, is an $M \times M$ non-symmetric, narrow-banded matrix (i.e., bandwidth, $w \ll M$), and x_k and b_k ($k = 0, 1, \ldots, n_b - 1$) are vectors. For convenience Eq. (1) will be rewritten in the form:

$$AX = B \tag{2}$$

where X is an $M \times n_b$ matrix the columns of which are simply the vectors x_k, and similarly B is the $M \times n_b$ matrix whose columns are the vectors b_k. The need to solve such systems of equations arises, for example, when solving a partial differential equation for a number of different boundary conditions.

Equation (2) will be solved using LU decomposition, followed by forward reduction and back substitution. Both the non-pivoting and partial pivoting cases will be considered. Timing results obtained on a 128-node Caltech/JPL Mark II hypercube will be presented and compared with performance models.

3. The Sequential Algorithm

The sequential algorithm described here follows that of [Martin 67], and divides the solution of the banded system in Eq. (2) into three stages. In the first stage the LU decomposition of the coefficient matrix A is performed. This factorizes A into the product of a lower triangular matrix, L, and an upper triangular matrix, U:

$$A = LU \tag{3}$$

The matrices L and U both have a band structure similar to that of A, and the factorization is unique if we require that L have unit diagonal elements. Other factorizations, such as the Cholesky factorization in which the diagonal elements of L and U are the same, are also possible. It is convenient to store L and U in the same memory locations as A, with L replacing the lower triangular part of A, and U the remaining part of A. There is no need to explicitly store the unit diagonal of L.

The LU decomposition of A proceeds in M steps (see [Golub 83]). If m is the halfwidth of the band, given by $w = 2m-1$, then at step k ($k = 0, 1, \ldots, M-1$), only those elements of A within an active $m \times m$ "window" centered on the diagonal are affected (see Fig. 1). At step k the first row and column of the window lie on the k^{th} row and column of A, and the

k^{th} row of U and the k^{th} column of L are evaluated according to:

$$L_{k,k} = 1$$
$$L_{k+i,k} = A_{k+i,k}/A_{k,k} \qquad 1 \le i < m \qquad (4)$$
$$U_{k,k+j} = A_{k,k+j} \qquad 0 \le j < m \qquad (5)$$

The elements of A in the window, but not in the first row or column, are then modified as follows:

$$A_{k+i,k+j} = A_{k+i,k+j} - L_{k+i,k}U_{k,k+j} \qquad (6)$$

where $0 < i < m$, $0 < j < m$. As the algorithm proceeds the window moves along the main diagonal of A, having factored those elements over which it has passed (i.e., those to the left and above the window).

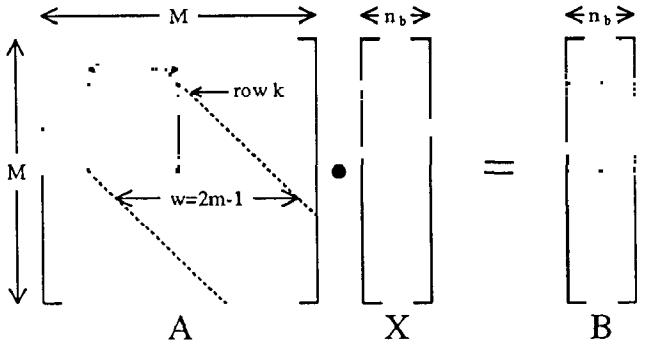

Figure 1. Schematic representation of step k of LU decomposition for an $M \times M$ matrix, A, with bandwidth w. The $m \times m$ window is represented by the dark shaded square. The light shaded part of the band above and to the left of the window has already been factorized, and contains the appropriate elements of L and U. The unshaded part of the band below and to the right of the window has not yet been modified. The shaded region of the matrix B represents the $m \times n_b$ window updated in the forward reduction and back substitution phases.

In the forward reduction stage both sides of Eq. (2) are multiplied by L^{-1} to give:

$$UX = L^{-1}B = B_{FR} \qquad (7)$$

The matrix B_{FR} can be stored in the same memory location as B. As in the LU decomposition algorithm, forward reduction is performed in a series of M steps, with only those elements lying in an active window of B of size $m \times n_b$ elements being modified in each step. In fact, the LU decomposition and forward reduction stages can be merged, thereby reducing the indexing overhead. However, for clarity we will treat LU decomposition and forward reduction as separate stages. In forward reduction the window moves down the matrix B one row at a time. At each step the elements of B lying within the window are updated according to:

$$B_{k+i,j} = B_{k+i,j} - B_{k,j}L_{k+i,k} \qquad (8)$$

where $1 \le i < m$, $0 \le j < n_b$. Thus at step k the elements of B lying above the window have already been modified to B_{FR}. The final back substitution stage evaluates the solution matrix, X, by multiplying Eq. (7) by U^{-1}:

$$X = U^{-1}B_{FR} \qquad (9)$$

The back substitution algorithm again consists of M steps, but in this case we start with the last row and work up the matrix towards the first row. Thus the active window of B begins at the bottom of B and moves one row upwards with each step. The steps are numbered from $k = M - 1$ to 0, and at step k the elements of B within the window are updated as follows:

$$X_{k,j} = B_{k,j}/U_{k,k}$$
$$B_{k-i,j} = B_{k-i,j} - X_{k,j}U_{k-i,k} \qquad (10)$$

where $1 \le i < m$, $0 \le j < n_b$. The solution matrix X can also be placed in the same storage location as B_{FR}. Thus in the forward reduction and back substitution steps the matrix B is first modified to B_{FR}, and then to X.

4. Concurrent Decomposition

We now consider how to decompose the problem on a concurrent processor containing a two-dimensional mesh in its interconnect topology. In the case of a simple algorithm, such as matrix multiplication, the square subblock decomposition has been shown to be optimal ([Fox 85], [Fox 87b]). However, such a decomposition is not appropriate for the problem at hand since at each step of the algorithm only a few of the processors would be involved. In fact, as shown in Fig. 2(a), those processors lying entirely outside the band of matrix A perform no useful work at all.

In order to minimize the effects of load imbalance the scattered square decomposition, shown in Fig. 2(b), will be used [Fox 84]. This ensures that all processors are involved in computation at each stage of the algorithm in as load-balanced a fashion as possible. In the scattered square decomposition a decomposition template is placed periodically over the data so that adjacent matrix elements lie in different processors. Decomposition templates for concurrent processors with 2-D mesh and hypercube topologies are shown in Fig. 3(a) and (b), respectively. We will discuss here only square templates, which restricts us to the use of hypercubes of even dimension. The extension to rectangular templates is straightforward. In Fig. 3(b) the decomposition template for the hypercube topology is based on a 2-dimensional Grey code ordering ([Gilbert 58], [Salmon 84]), which ensures that adjacent cells in the decomposition template correspond to nearest neighbors in the hypercube topology. Moreover, it should be noted that the

nodes in any row or column constitute a subcube of the hypercube. This feature can be exploited to allow rows and columns of a matrix to be broadcast by means of a *pipe* algorithm, as explained in Sec. 5.

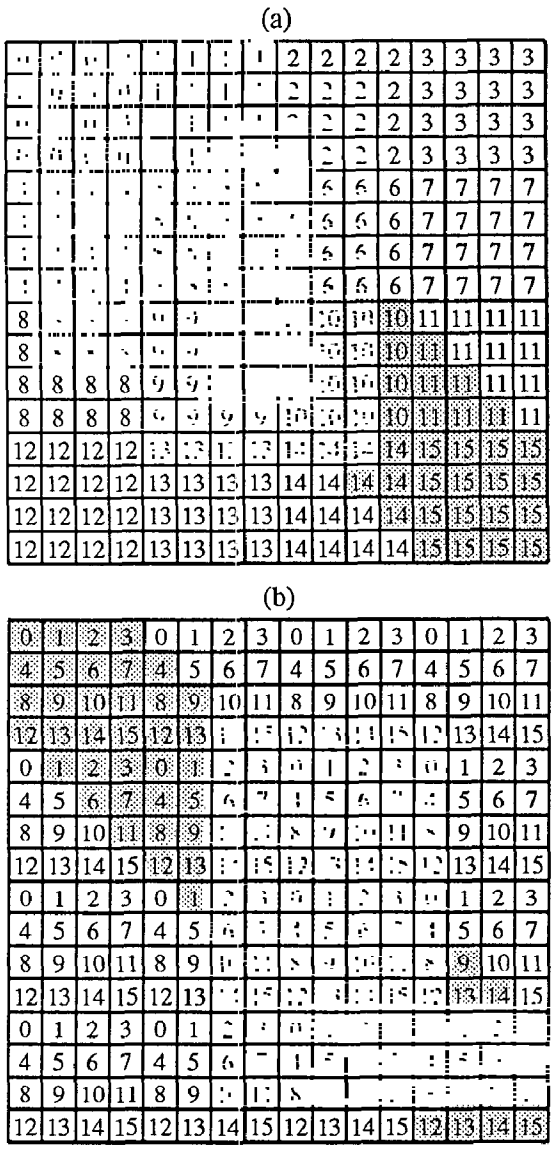

Figure 2. Comparison of (a) local square and (b) scattered square decompositions of a 16×16 banded matrix onto a 16-node concurrent processor. The band is shown shaded, and the zero elements outside the band are not actually stored. Each cell represents a matrix element, and the number in the cell indicates the node in which this element is stored. This figure shows the numbering of nodes appropriate for a 2-dimensional array of processors. The numbering for a hypercube would be based on the 2-dimensional Grey code scheme shown in Fig. 3(b).

A similar scattered decomposition has been used by Saad and Schultz [Saad 85] in their discussion of the Gaussian elimination algorithm on hypercubes. Saad and Schultz, however, allow each cell of the decomposition template to hold a subblock of $\hat{m} \times \hat{m}$ elements (see Eq. (13)), rather than just one element as in our decomposition. A comparison of these two decompositions is given in [Beernaert 87].

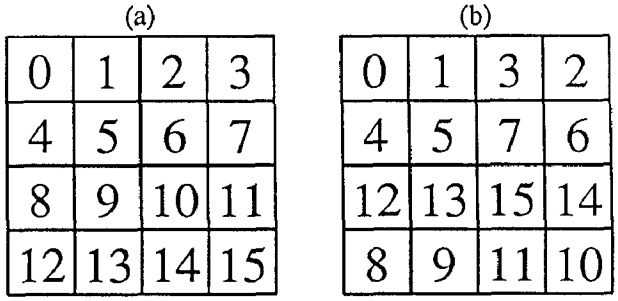

Figure 3. Decomposition templates for (a) 2-D mesh and (b) hypercube topologies.

5. Communication Strategies

In order to update the current window elements of the matrix A according to Eq. (6), the first column of the window ($L_{k+i,k}$, $0 < i < m$) must be sent across the window to the right, and the first row of the window ($U_{k,k+j}$, $0 < j < m$) must be sent down the window. This means that each node to which elements in the first column of the current window are assigned must send their part of the first column to all the nodes in the same row of the decomposition template. Similarly each node to which elements in the first row of the current window are assigned must send their part of the first row to all the nodes in the same column of the decomposition template. After the first row and column of the window have been transmitted each node contains all the information necessary to update the window elements assigned to it.

On a hypercube the nodes lying in a particular row, or column, of the decomposition template constitute a subcube, and the communication of rows and columns across the current window could therefore be performed by means of the tree-based broadcast algorithm described in Chap. 14 of [Fox 87a]. However, we will use a *pipe* broadcast method, which has the advantage of being simple to implement, and which can be used on any multiprocessor whose connection topology contains a ring. In the simplest pipe broadcast, called a *linear pipe*, the communicating nodes are mapped onto a line, with the source node at one end. The source node sends the data one packet at a time to the next node in the line. As this node receives each packet it passes it on to the next node, and so on until all of the nodes have received all of the packets sent by the source node. If t_{comm} is the time taken to read a floating-point number from one node and then write it to the next, then the first number arrives at the last node in the pipe in time $(D-1)t_{comm}$, where D is the number of nodes in the

pipe. Thereafter one number arrives every t_{comm} seconds, so the total time to broadcast r numbers to all nodes in the pipe is:

$$T_{linear} = (r + D - 2)t_{comm} \tag{11}$$

The quantity $(D-2)t_{comm}$ is commonly called the *pipe startup time*.

In the *split pipe* algorithm the communicating nodes are again mapped onto a line, but in this case the source node is placed, as nearly as possible, at the middle of the line, rather than at one end. The source node then sends each packet in turn to the two nodes on either side of it, which in turn pass on the incoming packets to the next nodes. The nodes at each end of the line receive packets, but do not forward any. If the source node is capable of simultaneously sending a packet out on two communication channels in the same time that it takes to send it on one channel, then the split pipe broadcast has the advantage of having only half the pipe startup time of the linear pipe:

$$T_{split} = \left[r + \frac{(D-2)}{2} \right] t_{comm} \tag{12}$$

It should be noted that if there are only two nodes in the pipe, the implementation of the split pipe reduces to that of the linear pipe.

In the concurrent algorithms described in Sec. 6, we will use the split pipe algorithm to broadcast rows and columns across the windows of the matrices A and B.

6. The Concurrent Algorithm

6.1. LU Decomposition With No Pivoting

We now consider the concurrent implementation of the basic LU decomposition algorithm given in Eqs. (4), (5), and (6). We first introduce the quantity:

$$\hat{m} = m/\sqrt{N} \tag{13}$$

where N is the number of nodes in the concurrent processor. At some step k of the LU decomposition each processor holds $\hat{m}^2$ matrix elements in the active window. Since we are using a scattered decomposition, these elements do not comprise a contiguous $\hat{m} \times \hat{m}$ submatrix, but are scattered uniformly throughout the window. By comparing the windows for steps k and $k+1$, we see that row k and column k are active at step k, but not thereafter. However, at step $k+1$ the nodes to which these now inactive elements are assigned obtain a compensating amount of new work from row $k+m$ and column $k+m$, thereby preserving load balance. This useful feature arises from the periodicity of the scattered decomposition.

Each step k of the concurrent LU decomposition algorithm consists of five stages:

(a) Evaluate $1/A_{k,k}$ in the node to which the top left corner element of the current window is assigned.

(b) The top row of the window, which by Eq. (5) is just the nonzero elements of row k of U, is broadcast downwards, except that the top left corner node sends $1/A_{k,k}$, instead of $A_{k,k}$. This broadcast is done using the *split_pipe* algorithm described in Sec. 5.

(c) The nodes assigned to the elements in the first column of the window evaluate the k^{th} column of L according to Eq. (4). The other nodes are idle at this time. The quantity $1/A_{k,k}$ is available to each of the first column of nodes since it was broadcast downwards in step (b). Since the matrix L overwrites the lower triangular part of A, the matrix element $A_{k+i,k}$ is replaced by $A_{k+i,k}/A_{k,k}$, for $0 < i < m$.

(d) The first column in the widow (i.e., the k^{th} column of L), except for the element in the top left corner, is broadcast across the window, so that all nodes in the same row of the decomposition template contain identical subsets of the multipliers $L_{k+i,k}$ $(0 < i < m)$.

(e) The matrix elements of A lying in the window, but not in the first row or column, are updated according to Eq. (6). Following the split pipe broadcasts performed in stages (b) and (d), all the information necessary to do this for a given matrix element is contained in the processor holding that element.

The load imbalance arising in steps (a) and (c) is summarized in Table 1. In the evaluation of $1/A_{k,k}$ in stage (a), only the node assigned to the top left corner of the current window is doing any work. Fortunately this does not constitute a serious sequential bottleneck as only one floating-point operation is involved. Thus although stage (a) is very inefficient, it is irrelevant when compared with the time taken by the dominant stage (e). Stage (c) is more efficient than stage (a), but contributes more to the total concurrent overhead. Stages (a) and (c) are poorly implemented from the concurrent point of view, but do not significantly impact the performance of the concurrent algorithm for sufficiently large grain size problems, since the dominant concurrent stage (e) requires time of order $\hat{m}^2$, while the sequential stages take time of order $\hat{m}$.

6.2. LU Decomposition With Partial Pivoting

The LU decomposition algorithm described in Sec. 6-1 is appropriate when the diagonal elements,

Stage	Time		Local Efficiency	Contribution to Total Concurrent Overhead
	Sequential	Concurrent		
(a) Invert A	O(1)	O(1)	O(1/N)	$O(1/\hat{m}^2)$
(c) Form Multipliers	$\hat{m}\sqrt{N}-1$	$\hat{m}$	$O(1/\sqrt{N})$	$O(1/\hat{m})$
(e) Update Matrix	$(\hat{m}\sqrt{N}-1)^2$	$\hat{m}^2$	$1-O(1/\hat{m}\sqrt{N})$	$O(1/\hat{m}\sqrt{N})$

Table 1 The utilization of the concurrent processor during the three calculation phases of the LU decomposition algorithm. The sequential and concurrent times are given in units of t_{calc}, the time to perform one floating-point operation.

$A_{k,k}$, are dominant. This is often the case, but in general attention must be given to numerical instability resulting from division by $A_{k,k}$ when its value is small or vanishing. This problem can be avoiding by performing partial pivoting. This is done by searching the first column of the current window for the maximum entry $A_{k+i,k}$, $0 \leq i < m$. If this row corresponds to row number $k + i_{max}$ then the rows of the window corresponding to rows $i = 0$ and $i = i_{max}$ are swapped, and the algorithm continues as before. One complication is that the window must be at least $m + i_{max}$ columns in width to accommodate the elements that are now located in the first row. This possible increase in window size affects the speed of the algorithm, but does not alter in any essential way the steps (a) to (e) outlined above. The variable size of the window encountered here is handled naturally by the scattered decomposition, which ensures that the loads of the different nodes differ by at most one column. This illustrates the utility of this simple decomposition approach. While originally motivated by the need to cope with the banded structure of the matrix, it has proved to be a powerful method for dealing with other sources of imbalance. The added complications arising from the variable window width could be reduced by assuming that the width of the window is always the maximum value of $(2m - 1)$ columns, and padding the unused part of the window with zeros. However, in our implementation this was not done, and the variable size of the window was fully taken into account. This makes it necessary to store the window width at each step of the LU decomposition algorithm so that the correct value can be used in the forward reduction and back substitution stages. Furthermore if, as in our implementation, the LU decomposition and forward reduction are performed in distinct stages, it is also necessary to store the pivot row numbers so the corresponding pivot operations can be performed on the matrix B prior to the forward reduction stage. Clearly, in any production code the LU decomposition and forward reduction stages should be merged.

At step k the concurrent partial pivoting algorithm incorporates the following five additional steps:

(1) Each node containing elements in the first column of the current window determines the local maximum of $|A_{k+i,k}|$ $(0 < i < \hat{m})$ for those elements in the node. This step only involves $\sqrt{N}$ nodes; the other nodes are idle.

(2) The global maximum of the $\sqrt{N}$ local pivot candidates found in step (1) is determined. This could be done by means of the hypercube-specific subcube combine algorithm described in Chap. 20 of [Fox 87a], however we have used a more general algorithm based on the pipe algorithm described in Sec. 5. Each of the $\sqrt{N}$ nodes, except the top node to which the top left corner element in the window has been assigned, passes its candidate pivot element and its row number up to the top node, which finds the maximum of all the candidates and the corresponding number of the pivot row.

(3) The pivot row number is next broadcast to all nodes. In our implementation this was done using a tree broadcast. The location of the pivot row, and the window width are stored for use in the subsequent forward reduction and back substitution steps.

(4) The pivot row is swapped with the top row of the window. If the pivot row and the old top row are both assigned to the same row of the decomposition template this step requires no communication, otherwise the transfer is effected via the shortest connecting linear pipe.

(5) Finally the pivot row is sent to all nodes. This completes the swap, and also implements stage (b) of the non-pivoting algorithm above.

These pivoting steps are followed by those of the non-pivoting algorithm with the obvious changes required by the variable window width. It is apparent that while pivoting is an almost trivial operation on a sequential machine, it can require a fairly complicated algorithm in the concurrent implementation.

6.3. Forward Reduction and Back Substitution

In the k^{th} step of the forward reduction algorithm the elements in the current active window of B must be updated according to Eq. (8). As in the LU decomposition stage, we must first communicate the first row of the window of A (the $L_{k+i,k}$ in Eq. (8)) across the window. Then the top row of the window of B must be communicated downward across the window. Both of these broadcast operations are performed using the split pipe algorithm. Each node then has all the data necessary to update the window elements of B assigned to it.

The k^{th} step of the back substitution algorithm (Eq. (10)) is similar to that of the forward reduction algorithm. The elements in the last column of the current window of A (the $U_{k-i,k}$ in Eq. (10)) are broadcast to the left across the window. Each node to which elements in the bottom row of the window are assigned evaluates the k^{th} row of the solution matrix X by multiplying the bottom row of the window of B by $1/U_{k,k}$. This step involves nodes in only one row of the decomposition template; the other nodes are idle at this time. The bottom row of B is overwritten with the newly determined row of the solution matrix, and is then broadcast upwards across the window. The window elements are then updated according to Eq. (10).

If partial pivoting is performed in the LU decomposition the above description of the forward reduction and back substitution stages are unchanged, expect that the variable window size must be taken into account, and before performing the forward reduction the rows of B must be pivoted to match the pivoting in the LU decomposition stage.

7. Performance Models

Models of the performance of concurrent algorithms are important since they indicate whether the algorithm is efficient on massively parallel systems for a given grain size. Also discrepancies between the predicted and measured performance often point out non-optimal coding.

7.1. The Sequential Algorithm.

The time taken for one step of the sequential LU decomposition algorithm is just the time to evaluate the multipliers (Eq. (4)), plus the time to update the elements of A in the window (Eq. (6)). Thus, for the non-pivoting case, the total time for the M steps of the sequential algorithm is given by:

$$T_1(m) = M[(m-1)t_1 + 2(m-1)^2 t_2] \qquad (14)$$

where M is the order of the matrix. We have assumed that $M \gg m$, and have ignored the time to evaluate $1/A_{k,k}$. The quantity t_1 is the "average" time to evaluate one of the multipliers $L_{k+i,k}$, and is made up of the time t_{calc} to perform one floating-point operation, plus the overhead incurred in each pass through the outer loop. We will ascribe this overhead to the indexing of the matrix elements referred to in the loop, but there are other sources of overhead, such as that arising from setting up and and controlling the inner loop. Similarly, t_2 is the time per floating-point operation spent in each inner loop, and equals t_{calc} plus half the indexing overhead in each pass through the inner loop. In this context the outer and inner loops correspond to loops over rows and columns, respectively. In the absence of indexing overhead $t_1 = t_2 = t_{calc}$.

The time taken for one step of the sequential forward reduction stage is just the time to update the elements in the active window of the matrix B (Eq. (8)).

$$T_1(m, n_b) = M[2(m-1)n_b t_3 + n_b t_{ind}] \qquad (15)$$

where t_3 is the time per floating-point operation for the inner loop, and t_{ind} is the indexing time in each outer loop.

The time taken by the back substitution stage is similar, except the time taken to evaluate the solution vector in the last row of the window must be included.

$$T_1(m, n_b) = M[2(m-1)n_b t_3 + n_b t_4] \qquad (16)$$

where as usual t_4 is the time per floating-point operation for the outer loop.

7.2 The Concurrent Algorithm.

We next consider the concurrent LU decomposition for the case in which there is no pivoting, and in which the window size, m, is an exact multiple of $\sqrt{N}$. In this case, each node is assigned an equal number of matrix elements in the window, so the decomposition is completely balanced.

As in the sequential algorithm, we ignore the time taken to evaluate $1/A_{k,k}$ in the node in the top left corner of the current window. There are then two calculation steps: the evaluation of the multipliers, and the updating of the window elements, the times for which are $\hat{m}t_1$ and $2\hat{m}^2 t_2$, respectively. The routine *split_pipe* is called twice to broadcast the first row and column of the window. In the first phase of the routine the data to be sent by the source nodes are copied to a buffer, taking time $\hat{m}t_{copy}$. Ideally on packetizing machines such as the Mark II hypercube, this step could be omitted, since the data to be sent lie at regularly-spaced storage locations so *split_pipe* could be implemented using the CrOS III routines *vread* and *vwrite* (see Chap. 14 of [Fox 87a]). However, *vread* and *vwrite* have not been optimally im-

plemented on the Mark II hypercube, so the buffering stage is necessary. In the second phase of *split_pipe*, the data is piped to the destination nodes as described in Sec. 5. Thus the total time for the concurrent algorithm is given by:

$$\frac{T_N(\hat{m})}{M} = 2\hat{m}^2 t_2 + \hat{m}t_1 + 2\hat{m}t_{copy} + 2\left[\hat{m} + \frac{\sqrt{N}-2}{2}\right] t_{comm} \quad (17)$$

The concurrent overhead, f, is defined as:

$$f = \frac{1}{\epsilon} - 1 = \frac{NT_N(\hat{m}) - T_1(m)}{T_1(m)} \quad (18)$$

thus from Eqs. (14) and (17) the concurrent overhead of the LU decomposition algorithm when $m = \hat{m}\sqrt{N}$ is:

$$f_{LU} = \frac{\tau_1}{2\hat{m}} + \frac{4-\tau_1}{2\hat{m}\sqrt{N}} + \frac{(\tau_c+\tau)}{\hat{m}} + \frac{(\sqrt{N}-2)}{2\hat{m}^2}\tau \quad (19)$$

where $\tau_1 = t_1/t_2$, $\tau_c = t_{copy}/t_2$, $\tau = t_{comm}/t_2$, and terms of order $1/\hat{m}^2\sqrt{N}$ have been neglected.

If m is not an exact multiple of $\sqrt{N}$ then the nodes in the first row and column of the decomposition template will be assigned more window elements than others, resulting in an imbalanced decomposition. The most imbalanced case occurs when m exceeds an exact multiple of $\sqrt{N}$ by 1, i.e., $m = \hat{m}\sqrt{N} + 1$. Following the analysis given above for the balanced case, we find the following total time for the concurrent algorithm:

$$\frac{T_N(\hat{m})}{M} = 2(\hat{m}+1)^2 t_2 + (\hat{m}+1)t_1 + 2(\hat{m}+1)t_{copy} + 2\left[\hat{m}+1+\frac{\sqrt{N}-2}{2}\right] t_{comm} \quad (20)$$

which leads to a concurrent overhead of:

$$f_{LU} = \frac{(4+\tau_1)}{2\hat{m}} - \frac{\tau_1}{2\hat{m}\sqrt{N}} + \frac{(\tau_c+\tau)}{\hat{m}} + \frac{(2+\tau\sqrt{N}+\tau_1+2\tau_c)}{2\hat{m}^2} \quad (21)$$

In the concurrent forward reduction algorithm the time taken to update the elements in the active window of matrix B is similar to that for the sequential algorithm given in Eq. (14). However, the multipliers $L_{k+i,k}$ in Eq. (8) must be piped across the window. This communication step is also performed in the LU decomposition stage, and the forward reduction and LU decomposition stages could be merged, thereby avoiding the need to communicate the multipliers twice. However, for clarity we present forward reduction and LU decomposition as two distinct algorithms. A second communication step is necessary to pipe the n_b elements in the top row of the window of B (the $B_{k,j}$ in Eq. (8)) down the window. Adding the times for the calculation and communication steps together we obtain the total time for the concurrent forward reduction algorithm:

$$\frac{T_N(\hat{m},\hat{n}_b)}{M} = 2\hat{m}\hat{n}_b t_3 + \hat{n}_b t_{ind} + (\hat{m}+\hat{n}_b)t_{copy} + (\hat{m}+\hat{n}_b+\sqrt{N}-2)t_{comm} \quad (22)$$

where we have assumed a balanced composition, i.e, $m = \hat{m}\sqrt{N}$. From Eqs. (15) and (22), the concurrent overhead for the forward reduction phase is therefore:

$$f_{FR} = \frac{1}{\hat{m}\sqrt{N}} + \frac{(1-1/\sqrt{N})}{2\hat{m}}\tau_i + \left(\frac{1}{\hat{n}_b}+\frac{1}{\hat{m}}\right)\frac{(\tau_c'+\tau')}{2} + \frac{(\sqrt{N}-2)}{2\hat{m}\hat{n}_b}\tau' \quad (23)$$

where $\tau_i = t_{ind}/t_3$, $\tau' = t_{comm}/t_3$, and $\tau_c' = t_{copy}/t_3$.

The back substitution phase involves communication steps similar to those in the forward reduction phase, except that in the former case the right-hand column of the window of A and the bottom row of the window of B are communicated (see Eq. (10)). The time for the concurrent back substitution algorithm in the balanced decomposition case is therefore:

$$\frac{T_N(\hat{m},\hat{n}_b)}{M} = 2\hat{m}\hat{n}_b t_3 + \hat{n}_b t_4 + (\hat{m}+\hat{n}_b)t_{copy} + (\hat{m}+\hat{n}_b+\sqrt{N}-2)t_{comm} \quad (24)$$

From Eqs. (16) and (24), the concurrent overhead for the back substitution phase is therefore:

$$f_{BS} = \frac{1}{\hat{m}\sqrt{N}} + \frac{(1-1/\sqrt{N})}{2\hat{m}}\tau_4 + \left(\frac{1}{\hat{n}_b}+\frac{1}{\hat{m}}\right)\frac{(\tau_c'+\tau')}{2} + \frac{(\sqrt{N}-2)}{2\hat{m}\hat{n}_b}\tau' \quad (25)$$

where $\tau_4 = t_4/t_3$.

7.3 Partial Pivoting

In general, the time taken for the LU decomposition algorithm in the partial pivoting case depends on which rows take part in the pivoting, and this in turn depends on the elements of A. At any given stage of the LU decomposition the width of the window depends on the previous pivoting history. Moreover the

time to send the top row in the window to the position of the pivot row via the shortest geodesic pipe, depends on the location of the pivot row. Thus a time for the general case of LU decomposition with partial pivoting cannot be derived. In Table 2 we present the times for each stage of the sequential and concurrent LU decomposition algorithms. For clarity we assume here that indexing overhead is negligible. At step k of the algorithm the active window of matrix A will, in general, be rectangular, being m rows by m_k columns. In the case of no pivoting $m_k = m$ at each step, however, in the case of partial pivoting $m \leq m_k < 2m$. In Table 2 $\hat{m}_k$ equals $\lceil m_k/\sqrt{N} \rceil$, where $\lceil x \rceil$ is the smallest integer greater than or equal to x. At stage (d) in Table 2 we give the time to send the top row in the window to the position of the pivot row. If the top row and pivot row are assigned to the same row of the decomposition template no communication is necessary, so this stage takes no time (a memory-to-memory copy is performed which we ignore). If the top row and pivot row are not assigned to the same row of the decomposition template then the top row is sent to the pivot row position via the shortest geodesic pipe. In Table 2 we denote the number of nodes involved in this pipe by l_k. If each row of the decomposition template is equally likely to contain the pivot row then the expectation value of the time taken by stage (d) is:

$$E_d = \sum_{l_k=2}^{\sqrt{N}/2+1} P(l_k)(\hat{m}_k + l_k - 2)t_{comm} \qquad (26)$$

where $P(l_k)$ is the probability that there are l_k nodes in the shortest pipe from the top row to the pivot row.

$$P(l_k) = \begin{cases} 2/\sqrt{N} & \text{if } l_k = 2, 3, \ldots, \sqrt{N}/2 \\ 1/\sqrt{N} & \text{if } l_k = \sqrt{N}/2 + 1 \end{cases} \qquad (27)$$

Equations (26) and (27) give the expectation value of stage (d) in the case of "random" pivoting, as:

$$E_d = \left[\frac{(\sqrt{N}-1)}{\sqrt{N}} \hat{m}_k + \frac{(\sqrt{N}-2)^2}{4\sqrt{N}} \right] t_{comm} \qquad (28)$$

If we further assume that for most of the M steps the width of the window is at its maximum value, i.e., $m_k = 2m - 1$, then $\hat{m}_k = 2\hat{m}$ and the overhead for LU decomposition with partial pivoting is:

$$f_{LU} = \frac{1}{\hat{m}\sqrt{N}} + \frac{0.5}{\hat{m}} + \frac{\tau}{\hat{m}} + (1 - 2/\sqrt{N})\frac{\tau}{4\hat{m}} + h(N)\frac{\tau}{\hat{m}^2} \qquad (29)$$

where $h(N)$ is a function of N.

Stage	Time	
	Sequential	Concurrent
(a) Find Pivot	$m\, t_{calc}$	$\hat{m}\, t_{calc}$
(b) Transmit to Corner Node	0	$2(\sqrt{N}-1)\, t_{comm}$
(c) Inform All Nodes of Pivot Parameters	0	$2 \log_2 N\, t_{comm}$
(d) Send Top Row in Window to Position of Pivot Row	0	0 or $(\hat{m}_k + l_k - 2)\, t_{comm}$
(e) Send Pivot Row Everywhere	0	$\left[\hat{m}_k + \left(\frac{\sqrt{N}-2}{2}\right)\right] t_{comm}$
(f) Invert $A_{k,k}$	t_{inv}	t_{inv}
(g) Form Multipliers	$(m-1)\, t_{calc}$	$\hat{m}\, t_{calc}$
(h) Pipe Multipliers Across Window	0	$\left[\hat{m} + \left(\frac{\sqrt{N}-2}{2}\right)\right] t_{comm}$
(i) Correct Matrix	$2(m-1)(m_k-1)\, t_{calc}$	$2\hat{m}\hat{m}_k\, t_{calc}$

Table 2 Expected timings for each stage of the LU decomposition algorithm with partial pivoting for the sequential and concurrent cases. Indexing overhead has been ignored.

Comparing Eq. (29) with Eq. (21), for the non-pivoting case in which $\tau_1 = 1$ and $\tau_c = 0$, we see that the increased width of the window in the pivoting case results in a larger grain size, and so tends to decrease the overhead. However, this decrease is offset by the increase arising from the extra communication necessary to perform the pivoting. Nevertheless, if the grain size is sufficiently large that the $1/\hat{m}^2$ terms in the overhead are negligible, then the pivoting case will have less overhead provided τ is less than about 2.

8. Hypercube Timings

In this Section we present timings made on the 128-node Caltech/JPL Mark II hypercube. Since the decomposition template is restricted to a square grid, only hypercubes of even dimension may be used. We therefore made timings on 0, 2, 4 and 6 dimensional hypercubes. The 1-node timings were used to deduce concurrent overheads for the 4, 16, and 64 node results. In some cases the matrix data would not fit into the memory of a single node, so a model of the 1-node program was used to extrapolate the expected run time. In the LU decomposition algorithm the window size is progressively reduced for steps $k > M - m + 1$ as the window moves off the bottom right corner of the matrix. A similar effect occurs in the forward reduction and back substitution algorithms. These "end effects" have been ignored in the analysis of Sec. 7, and result in slightly lower execution times. We are interested in narrow-banded matrices for which the end effects are negligible, and the timings presented in this section have therefore been corrected for end effects by measuring the time for the first $M - m + 1$ steps and then multiplying this time by $M/(M - m + 1)$.

8.1 LU Decomposition Results

In Table 3(a) we present the 1-node timings for LU decomposition with no pivoting for a matrix of order $M = 150$. As expected from Eq. (14), the plot of $T_1(m)/(m-1)$ against bandwidth, $w = 2m - 1$, shown in Fig. 4, yields a straight line. A least-squares fit to this data gives $t_1 = 219.7$ μsec and $t_2 = 48.8$ μsec, so that $\tau_1 = t_1/t_2 = 4.5$. Equation (14) can therefore be written as:

$$\frac{T_1(m)}{M} = 48.8 \times (m-1)[2(m-1) + 4.5] \quad \mu\text{sec} \quad (30)$$

From Eq. (19), we see that indexing overhead makes a substantial contribution to the total concurrent overhead. In the code that produced the timings in Table 3(a) little effort was made to minimize the indexing overhead. In Table 3(b) we present timings for code a which matrix elements are accessed by means of pointers, rather than be explicitly subscripted arrays. This minor change results in the program running about 15% faster, and the timings for this case are also shown in Fig. 4. For the optimized version of the code $t_1 = 202.3$ μsec and $t_2 = 40.7$ μsec, giving $\tau_1 = 5.0$. The optimized code runs faster, but increases the concurrent overhead. This is because the indexing overhead in the inner loop has been reduced by a greater factor than that in the outer loop. The value of $t_2 = 40.7$ μsec corresponds to a speed of about 25 kflops for one node of the Mark II hypercube.

Window Size, m	Bandwidth $w=2m-1$	$T_1(m)$ (seconds)	$\frac{T_1(m)}{(m-1)}$
16	31	3.79	0.253
18	35	4.78	0.281
20	39	5.90	0.311
22	43	7.14	0.340
24	47	8.49	0.369
26	51	9.96	0.398
28	55	11.56	0.428
30	59	13.27	0.458
32	63	15.08	0.486

Table 3(a) Timings for the sequential LU decomposition algorithm running on a single node of the 128-node Caltech/JPL hypercube. The order of the matrix was 150. No pivoting was performed, and matrix elements were accessed by explicit indexing.

Window Size, m	Bandwidth $w=2m-1$	$T_1(m)$ (seconds)	$\frac{T_1(m)}{(m-1)}$
16	31	3.20	0.214
18	35	4.05	0.238
20	39	4.98	0.262
22	43	6.03	0.287
24	47	7.15	0.311
26	51	8.40	0.336
28	55	9.72	0.360
30	59	11.17	0.385
32	63	12.68	0.409

Table 3(b) Same as for Table 3(a), except that matrix elements were accessed using pointers instead of explicit indexing.

In Tables 4(a), (b) and (c) we present timings, efficiencies, and concurrent overheads for the LU decomposition algorithm for 4, 16, and 64 node hypercubes. No pivoting was performed, and matrix elements were accessed by explicit indexing. The bandwidths considered correspond to the case of a balanced decomposition, i.e., $m = \hat{m}\sqrt{N}$. Tables 5(a), (b), and (c) give results for the case of maximum imbalance, when $m = \hat{m}\sqrt{N} + 1$.

m	$\hat{m}$	$1/\hat{m}$	$T_4(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
40	20	0.0500	18.67	62.75	0.840	0.190
44	22	0.0455	22.24	75.89	0.853	0.172
48	24	0.0417	26.11	90.28	0.864	0.157
52	26	0.0385	30.28	105.92	0.875	0.143
56	28	0.0357	34.79	122.81	0.882	0.133
60	30	0.0333	39.57	140.94	0.890	0.123
64	32	0.0313	44.67	160.33	0.897	0.114
68	34	0.0294	50.10	180.96	0.903	0.107
72	36	0.0278	55.84	202.84	0.908	0.101

Table 4(a) Timings (in seconds), efficiencies, and overheads for the LU decomposition algorithm on a 4-node hypercube. No pivoting was performed. The window widths, m, correspond to the balanced case in which $m = \hat{m}\sqrt{N}$. The matrix order is $M = 400$. The 1-node times, $T_1(m)$, were estimated from Eq. (14), using $t_1 = 219.7$ μsec, and $t_2 = 48.8$ μsec.

m	$\hat{m}$	$1/\hat{m}$	$T_{16}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
80	20	0.0500	24.06	312.93	0.813	0.230
88	22	0.0455	28.61	378.54	0.827	0.209
96	24	0.0417	33.55	450.40	0.839	0.192
104	26	0.0385	38.87	528.50	0.850	0.177
112	28	0.0357	44.61	612.84	0.859	0.165
120	30	0.0333	50.73	703.42	0.867	0.154
128	32	0.0313	57.25	800.24	0.874	0.145
136	34	0.0294	64.13	903.30	0.880	0.136
144	36	0.0278	71.39	1012.60	0.887	0.128

Table 4(b) Same as for Table 4(a), but for a 16-node hypercube, and $M = 500$.

m	$\hat{m}$	$1/\hat{m}$	$T_{64}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
160	20	0.0500	38.83	1999.86	0.805	0.243
176	22	0.0455	46.04	2419.51	0.821	0.218
192	24	0.0417	53.85	2879.09	0.835	0.197
208	26	0.0385	62.40	3378.60	0.846	0.182
224	28	0.0357	71.60	3918.05	0.855	0.170
240	30	0.0333	81.40	4497.44	0.863	0.158
256	32	0.0313	91.90	5116.77	0.870	0.149
272	34	0.0294	102.81	5776.03	0.878	0.139
288	36	0.0278	114.60	6475.22	0.883	0.133

Table 4(c) Same as for Table 4(a), but for a 64-node hypercube, and $M = 800$.

m	$\hat{m}$	$1/\hat{m}$	$T_4(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
41	20	0.0500	20.42	65.91	0.807	0.239
45	22	0.0455	24.24	79.37	0.819	0.222
49	24	0.0417	28.17	94.07	0.835	0.198
53	26	0.0385	32.51	110.03	0.846	0.182
57	28	0.0357	37.14	127.22	0.856	0.168
61	30	0.0333	42.10	145.67	0.865	0.156
65	32	0.0313	47.36	165.37	0.873	0.146
69	34	0.0294	52.94	186.31	0.880	0.137
73	36	0.0278	58.83	208.50	0.886	0.129

Table 5(a) Timings (in seconds), efficiencies, and overheads for the LU decomposition algorithm on a 4-node hypercube. No pivoting was performed, and the matrix order, $M = 400$. The window widths, m, correspond to the imbalanced case in which $m = \hat{m}\sqrt{N} + 1$. The 1-node times, $T_1(m)$, were estimated from Eq. (14), using $t_1 = 219.7$ μsec, and $t_2 = 48.8$ μsec.

m	$\hat{m}$	$1/\hat{m}$	$T_{16}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
81	20	0.0500	26.29	320.79	0.763	0.311
89	22	0.0455	31.02	387.19	0.780	0.282
97	24	0.0417	36.17	459.82	0.795	0.259
105	26	0.0385	41.69	538.70	0.808	0.238
113	28	0.0357	47.62	623.82	0.819	0.221
121	30	0.0333	53.93	715.18	0.829	0.207
129	32	0.0313	60.65	812.78	0.838	0.194
137	34	0.0294	67.72	916.62	0.846	0.182
145	36	0.0278	75.18	1026.70	0.854	0.172

Table 5(b) Same as for Table 5(a), but for a 16-node hypercube, and $M = 500$.

m	$\hat{m}$	$1/\hat{m}$	$T_{64}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
161	20	0.0500	42.31	2024.92	0.748	0.337
177	22	0.0455	49.82	2447.06	0.767	0.303
193	24	0.0417	58.10	2909.14	0.782	0.278
209	26	0.0385	66.90	3411.15	0.797	0.255
225	28	0.0357	76.40	3953.10	0.808	0.237
241	30	0.0333	86.50	4534.98	0.819	0.221
257	32	0.0313	97.41	5156.80	0.827	0.209
273	34	0.0294	108.91	5818.56	0.835	0.198
289	36	0.0278	121.19	6520.25	0.841	0.190

Table 5(c) Same as for Table 5(a), but for a 64-node hypercube, and $M = 800$.

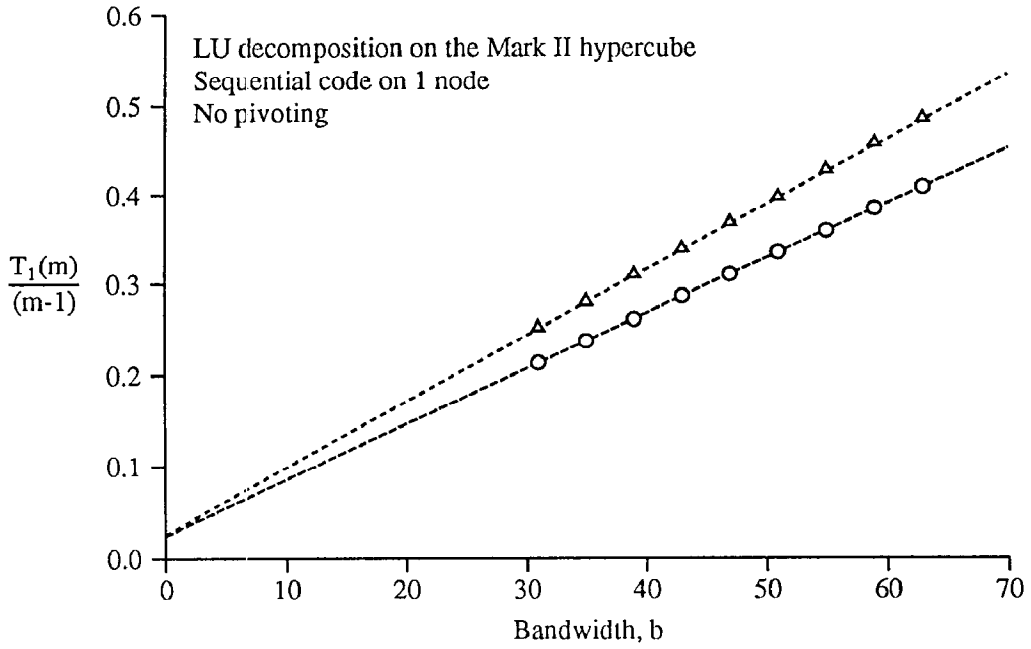

Figure 4 Timing results for the sequential LU decomposition algorithm running on one node of the hypercube. The triangles correspond to the data in Table 3(a), and the circles to the data in Table 3(b). The dashed lines give the least-squares best fit to the data, and give $t_1 = 219.7$ μsec, $t_2 = 48.8$ μsec (upper line) and $t_1 = 202.3$ μsec, $t_2 = 40.7$ μsec (lower line).

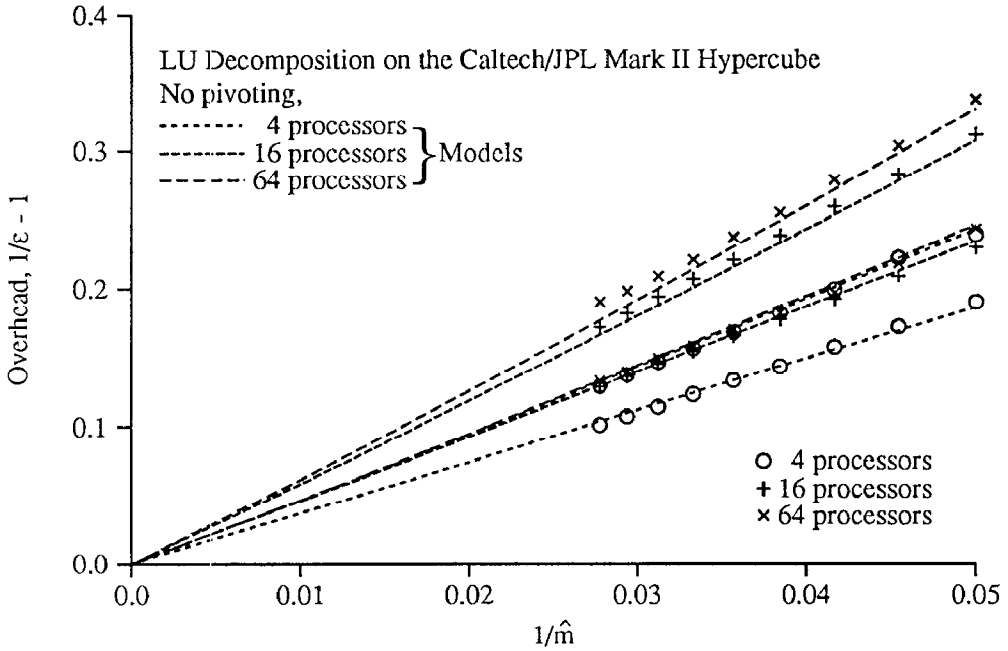

Figure 5 Concurrent overhead for the LU decomposition algorithm with no pivoting, for 4, 16 and 64 node hypercubes. The lines correspond to the performance models of Eqs. (19) and (21) and are discussed in the text.

In Fig. 5 we plot concurrent overhead, f_{LU}, as a function of $1/\hat{m}$ for the both the balanced and imbalanced cases. Equations (19) and (21) indicate that f_{LU} should depend almost linearly on $1/\hat{m}$ for large values of $\hat{m}$. In Fig. 5 we also plot the dependence of f_{LU} upon $1/\hat{m}$ predicted by the performance models discussed in Sec. 4. The upper set of curves correspond to the imbalanced case, and the lower set to the balanced case. A value of $\tau_1 = 4.5$, deduced from the 1-node results in Table 3(a), was used, and τ_c was taken to be 0.3. The results for the 16 and 64 node hypercubes were found to be well fitted by value of $\tau = 1.8$, however the results for the 4-node hypercube lie below the curve predicted by this value on the model line corresponding to $\tau = 1.2$. The reason for this apparent discrepancy lies in the implementation of the CrOS III routine *cwrite*, which sends data out from one node on a set of communication channels specified in the argument list. The time to send a packet out on N_{chan} communication channels is:

$$T_{cwrite} = \alpha + \beta N_{chan} \qquad (31)$$

where α is the overhead incurred in initializing the transmission, and β is the asymptotic time to send a packet over one channel. In the split pipe routine described in Sec. 5, the source node must send each packet out on two communication channels. Thus the value of t_{comm} in Eq. (12) has the form of Eq. (31) with $N_{chan} = 2$. However, if there are only 2 nodes in the pipe, as is the case for a 2-dimensional hypercube mapped onto a 2×2 decomposition template, the source node writes on only one channel, so $N_{chan} = 1$. Thus, when using the split pipe algorithm we would expect t_{comm} to be lower for a 2-dimensional hypercube than for hypercubes of higher dimension. Although the split pipe algorithm has a smaller pipe startup time than the linear pipe, the implementation of *cwrite* on the Mark II hardware makes it slower if many packets are transmitted. The values of τ given above are consistent with values found in other work on the Mark II hypercube. In the almost linear regime plotted in Fig. 5 it is difficult to distinguish the effects due to changes in τ_c and τ. As may be seen in Eq. (19), the important quantity is $\tau_c + \tau$. The close similarity between the observed and predicted results shown in Fig 5. indicates that the performance models of Sec. 4 correctly represent the behavior of the algorithm. For large grain sizes the overhead tends asymptotically to zero, and provided $\hat{m}^2 > 2\tau\sqrt{N}$ the algorithm scales well as the number of nodes increases. In the case of the Mark II hypercube, overheads of less than about 20% were obtained in the balanced case for all hypercube dimensions considered for $\hat{m} > 25$.

Results for the sequential LU decomposition algorithm with partial pivoting are given in Table 6. The performance model outlined in Table 2 predicts that $T_1(m)/(m-1)$ should be proportional to the window width, $\bar{m}$, averaged over the M steps of the algorithm. The value of $\bar{m}$ is in turn proportional m. In Fig. 6 $T_1(m)/(m-1)$ is plotted against m, and as expected the measured data are fitted well by a straight line. The non-zero y intercept of this line indicates that there are additional sources of overhead, such as indexing overhead, which are not accounted for in the simple model in Table 2. Some of this overhead can doubtless be removed by more careful coding, however, as for the non-pivoting case, some overhead will remain. Thus the model presented in Table 2 can be expected to give only a qualitative description of the performance of a real program. A straight line fit to the data plotted in Fig. 6 gives the following model for the performance of the sequential LU decomposition algorithm with partial pivoting:

$$\frac{T_1(m)}{M} = 45.9 \times (m-1)(23.2 + 4m) \quad \mu\text{sec} \qquad (32)$$

Equation (32) has been written so that if $\bar{m}$ equals the maximum possible value of $2m-1$, then the time to perform a single float-point operation in the inner loop is 45.9 μsec. This time also includes any overhead associated with the inner loop. The value of 45.9 μsec is close to the corresponding value of 48.8 μsec found in the non-pivoting case, in fact the ratio of the two numbers can be interpreted as $\bar{m}/2m$, giving $\bar{m} = 1.89m$. This result indicates that for most of the algorithm, the window width is close to the maximum value.

Window Size, m	Bandwidth $w=2m-1$	$T_1(m)$ (seconds)	$\frac{T_1(m)}{(m-1)}$
10	19	5.24	0.582
11	21	6.20	0.620
12	23	7.14	0.649
13	25	8.29	0.691
14	27	9.39	0.722
15	29	10.72	0.766
16	31	11.99	0.799
17	33	13.45	0.841

Table 6 Timings for the sequential LU decomposition algorithm running on a single node of the 128-node Caltech/JPL hypercube. The order of the matrix was 200. Partial pivoting was performed, and matrix elements were accessed by explicit indexing.

Results for the concurrent LU decomposition algorithm with partial pivoting are given in Tables 7(a), (b) and (c) for 4, 16, and 64 nodes, respectively. In

m	$\hat{m}$	$1/\hat{m}$	$T_4(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
20	10	0.1000	6.30	18.00	0.714	0.400
22	11	0.0909	7.31	21.44	0.733	0.364
24	12	0.0833	8.44	25.18	0.746	0.341
26	13	0.0769	9.59	29.20	0.761	0.314
28	14	0.0714	10.87	33.52	0.771	0.297
30	15	0.0667	12.17	38.14	0.783	0.276
32	16	0.0625	13.60	43.05	0.791	0.264
34	17	0.0588	15.08	48.25	0.800	0.250

Table 7(a) Timings, efficiencies, and overheads for the LU decomposition algorithm on a 4-node hypercube. Partial pivoting was performed and the matrix order, $M = 200$. The 1-node times, $T_1(m)$, were estimated from Eq. (32).

m	$\hat{m}$	$1/\hat{m}$	$T_{16}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
40	10	0.1000	13.53	131.24	0.606	0.650
44	11	0.0909	15.61	157.34	0.630	0.587
48	12	0.0833	18.15	185.80	0.640	0.563
52	13	0.0769	19.51	216.60	0.694	0.441
56	14	0.0714	21.82	249.76	0.715	0.398
60	15	0.0667	25.50	285.27	0.699	0.430
64	16	0.0625	28.30	323.13	0.714	0.401
68	17	0.0588	31.42	363.35	0.723	0.384

Table 7(b) Same as for Table 7(a), but for a 16-node hypercube, and $M = 400$.

m	$\hat{m}$	$1/\hat{m}$	$T_{64}(\hat{m})$	$T_1(m)$	ϵ_{LU}	f_{LU}
80	10	0.1000	27.37	996.19	0.569	0.758
88	11	0.0909	32.56	1199.38	0.578	0.737
96	12	0.0833	38.27	1421.38	0.580	0.723
104	13	0.0769	41.83	1662.20	0.621	0.611
112	14	0.0714	48.48	1921.83	0.619	0.614
120	15	0.0667	52.40	2200.28	0.656	0.524
128	16	0.0625	58.30	2497.54	0.669	0.494
136	17	0.0588	64.37	2813.62	0.683	0.464

Table 7(c) Same as for Table 7(a), but for a 64-node hypercube, and $M = 800$.

Fig. 7, these data are plotted as concurrent overhead against $1/\hat{m}$. The sequential algorithm results, modeled by Eq. (32), show that there are additional sources of overhead present not included in the simple model in Table 2. If it is assumed that the inner and outer loops are subject to differing overheads then the predicted form of the concurrent overhead, given by Eq. (29), becomes:

$$f_{LU} = \frac{0.5\tau_1'}{\hat{m}} + \frac{3-\tau_1'}{2\hat{m}\sqrt{N}} + (5 - 2/\sqrt{N})\frac{(\tau_c' + \tau')}{\hat{m}} + h(N)\frac{\tau'}{4\hat{m}^2} \quad (33)$$

where $\tau_1' = t_1'/t_2'$, $\tau_c' = t_{copy}/t_2'$, and $\tau' = t_{comm}/t_2'$. The quantities t_1' and t_2' are the times per floating-point operation spent in the outer and inner loops, respectively. The 1-node results give $\tau_1' = 12.55$. The dashed curves in Fig. 7 show the concurrent overhead predicted by Eq. (28) with $\tau_1' = 12.55$. The values of τ_c' and τ' were taken to be 0.3 and 1.8 (or 1.2 for the 4-node case), respectively, as in the case of no pivoting. The predicted curves lie above the data, suggesting that a lower value of τ_1' might be more appropriate. Part of the discrepancy between the measured and predicted results plotted in Fig. 7 might be due to the fact that in Eq. (33) we have assumed that the window width is at the maximum value throughout the algorithm. The scatter in the data points is attributable to the dependence of the algorithm's performance on the values of the elements of A.

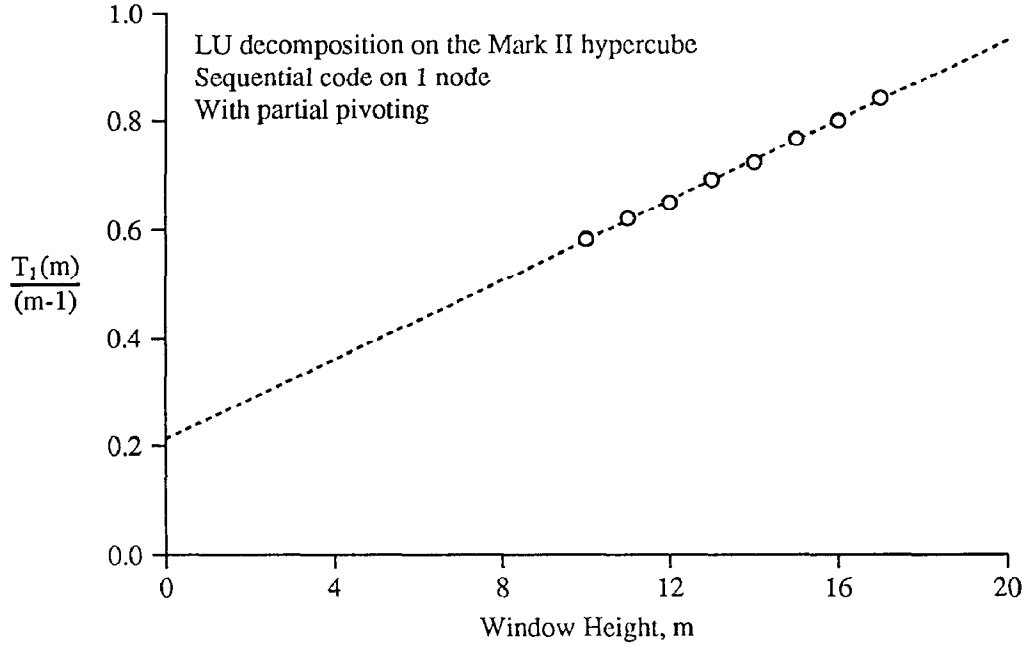

Figure 6 Timings for the sequential LU decomposition algorithm with partial pivoting running on a single node of the 128-node Mark II hypercube. The matrix order was 200.

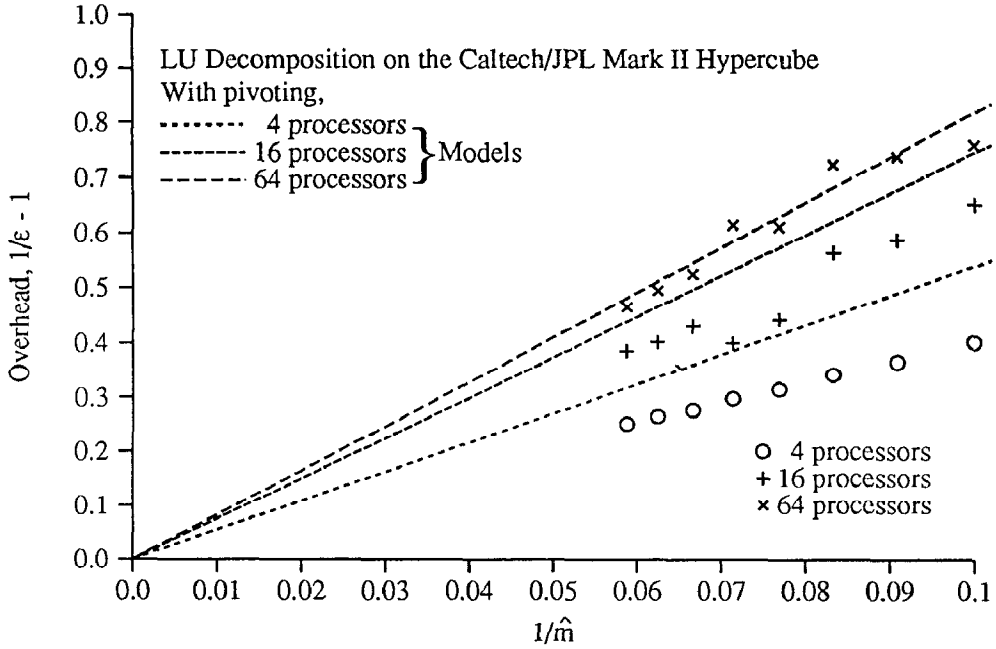

Figure 7 Concurrent overhead, f_{LU}, as a function of $1/\hat{m}$ for LU decomposition with pivoting.

8.2 Forward Reduction Results

In Table 8 we present timings for the sequential forward reduction and back substitution algorithms, run on a single node of the hypercube. Equation (12) indicates that at fixed n_b, the time depends linearly on $(m-1)$. We therefore plot in Fig. 8 $T_1(n_b, m)$ as a function of $(m-1)$. The data values are fitted very well by a straight line, the least-squares best-fit to which gives:

$$\frac{T_1^{FR}}{n_b M} = 49.5 \times [2(m-1) + 0.98] \quad \mu\text{sec} \qquad (34)$$

In Fig. 8 the intercept on the y axis is close to zero, indicating that the overhead associated with the outer loop is small. Equation (34) shows that each floating-point operation in the inner loop takes 49.5 μsec, which is consistent with the value of 48.8 μsec found for the LU decomposition algorithm.

Tables 9(a), (b), and (c) give the results for the concurrent forward reduction algorithm on 4, 16, and 64 nodes. In all cases the value of $\hat{m}$ is held fixed at 20. The corresponding concurrent overheads are plotted in Fig. 9 as a function of $1/\hat{n}_b$. The dashed lines in Fig. 9 are the concurrent overheads predicted by Eq. (23), for $\tau_c' = 0.3$, and $\tau' = 1.2$ and 1.8 for the $N = 4$ and $N > 4$ cases, respectively. The value of τ_i was taken to be 0.98 from the 1-node results. The agreement between the measured and predicted overheads is reasonably good, although it appears from Fig. 9 that a larger value of τ_i would give a better fit. A larger value of τ_i might be attributable to additional overhead in the outer loop that is present in the concurrent algorithm but not in the sequential algorithm.

Window Size, m	Bandwidth $w = 2m-1$	$T_1^{FR}(n_b, m)$ (seconds)	$T_1^{BS}(n_b, m)$ (seconds)
11	21	0.62	0.67
13	25	0.75	0.79
16	31	0.91	0.96
18	35	1.04	1.07
21	41	1.22	1.25
23	45	1.33	1.38
26	51	1.51	1.56
28	55	1.65	1.65
31	61	1.80	1.85

Table 8 Timings for the sequential forward reduction and back substitution algorithms running on a single node of the 128-node Caltech/JPL hypercube. The order of the matrix was 150 and the number of right-hand sides, n_b, was 4. No pivoting was performed, and matrix elements were accessed by explicit indexing.

8.3 Back Substitution Results

Equation (16) predicts that the time for the sequential back substitution algorithm in the no pivoting case depends linearly upon $(m-1)$. We therefore plot in Fig. 10 the sequential back substitution times on one node of the hypercube given in Table 8 against $(m-1)$. The resulting plot can be fitted with

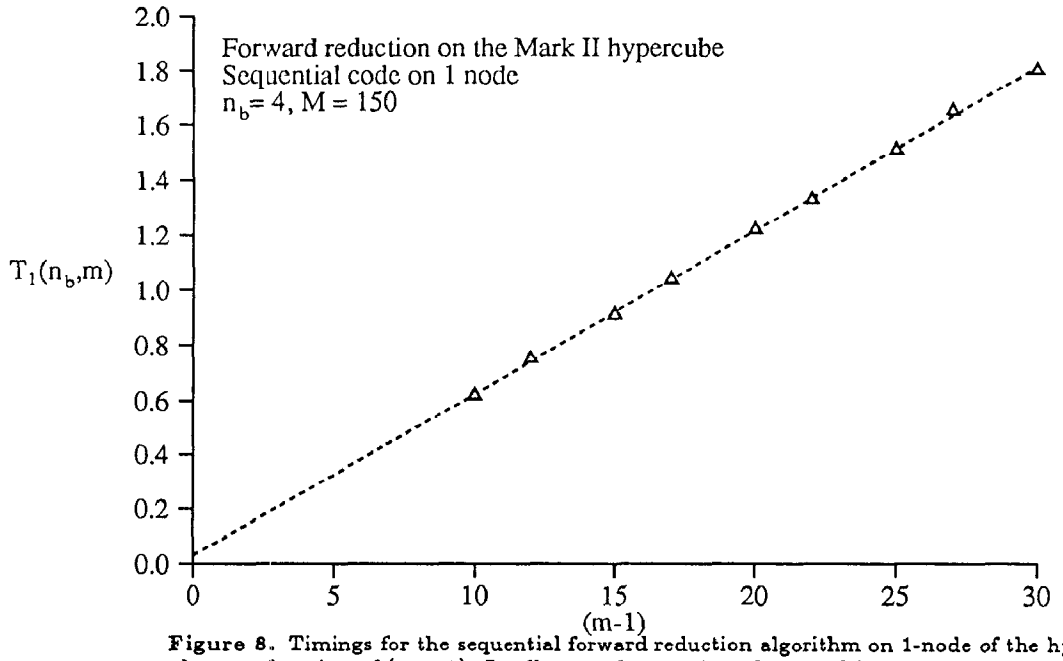

Figure 8. Timings for the sequential forward reduction algorithm on 1-node of the hypercube as a function of $(m-1)$. In all cases the matrix order was $M = 150$, and the number of right-hand sides was $n_b = 4$.

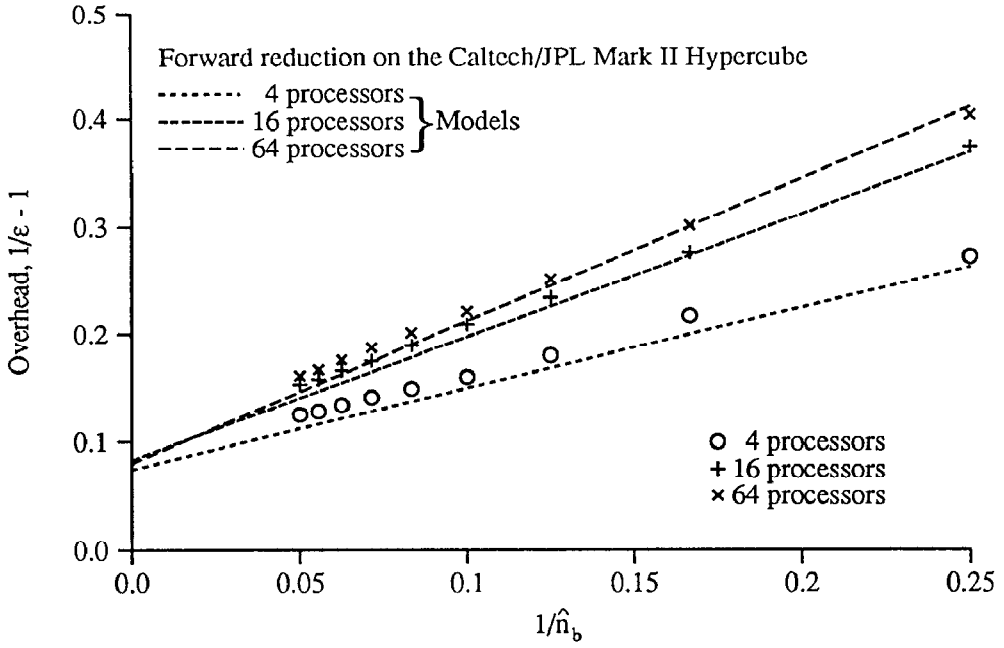

Figure 9. Concurrent overhead as a function of $1/\hat{n}_b$ for the forward reduction algorithm. In all cases the value of $\hat{m}$ equals 20. The dashed curves show the results predicted by Eq. (23), as discussed in the text.

a straight line as follows:

$$\frac{T_1^{BS}}{n_b M} = 49.0 \times [2(m-1) + 2.73] \quad \mu\text{sec} \qquad (35)$$

The value of 49 μsec for each pass through the inner loop is in good agreement with that found in the forward reduction (49.5 μsec) and LU decomposition algorithms (48.8 μsec).

In Tables 10(a), (b), and (c) we present timings, efficiencies and overheads for the back substitution algorithm on the Mark II hypercube for dimensions 2, 4, and 6. As in the forward reduction case, the value of $\hat{m}$ is held fixed at 20, and $\hat{n}_b$ is varied. The overheads are plotted in Fig. 11 as a function of $1/\hat{n}_b$. The dashed lines in Fig. 11 are the model predictions of Eq. (25). The value of τ_4 was taken from the 1-node results to be 2.73. The other model parameters were the same as in Fig. 9 for the forward reduction algorithm, i.e., $\tau_c' = 0.3$, and $\tau' = 1.2$ for $N = 4$, and $\tau' = 1.8$ for $N > 4$. The agreement between the measured and predicted overheads is excellent.

9. Conclusions

In general the overheads measured on the Mark II hypercube are in very good agreement with the performance models developed in Sec. 7, and we can therefore be confident that these models can be used to predict the performance for larger grain sizes, and for higher dimensional hypercubes.

This work has identified the following key issues in the implementation of LU decomposition, forward reduction, and back substitution algorithms on hypercubes such as the Mark II.

(1) The scattered square subblock decomposition is important in reducing load imbalance.

(2) For sufficiently large grain sizes, n, the communication overhead is proportional to $1/\sqrt{n}$, as in the matrix multiplication algorithm.

(3) Indexing overhead makes an important contribution to the concurrent overhead.

(4) Although many nodes are idle during certain stages of the algorithms, the overhead is still small for sufficiently large grain size problems. The algorithms scale well at fixed grain size to higher dimensional hypercubes.

(5) In the case of a balanced decomposition the efficiency of the LU decomposition with no pivoting exceeds 80% for hypercubes of dimension less than or equal to 6, for grain sizes, $n = \hat{m}^2 > 400$. In the case of partial pivoting the corresponding grain size is about 1000. For forward reduction and back substitution, if $\hat{m} = 20$ for number of righthand sides, $\hat{n}_b$, must exceed about 10 for the efficiency to exceed 80%

Acknowledgements: It is a pleasure to acknowledge the helpful and informative comments of Professor S. Lennart Johnsson of Yale University. This work is partially funded by the Department of Energy under grant number DE-FG03-85ER25009.

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_4(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
40	8	0.2500	1.79	5.63	0.786	0.272
40	12	0.1667	2.57	8.44	0.821	0.218
40	16	0.1250	3.32	11.25	0.847	0.180
40	20	0.1000	4.08	14.07	0.862	0.160
40	24	0.0833	4.85	16.89	0.870	0.149
40	28	0.0714	5.62	19.70	0.876	0.141
40	32	0.0625	6.38	22.51	0.882	0.134
40	36	0.0556	7.14	25.32	0.887	0.128
40	40	0.0500	7.91	28.14	0.889	0.125

Table 9(a) Timings, efficiencies, and overheads for the forward reduction algorithm on a 4-node hypercube for a matrix of order $M = 180$. The 1-node times, $T_1(n_b, m)$, were estimated from Eq. (34).

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_{16}(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
80	8	0.2500	3.89	45.31	0.728	0.373
80	12	0.1667	5.42	67.96	0.784	0.276
80	16	0.1250	6.99	90.61	0.810	0.234
80	20	0.1000	8.56	113.27	0.827	0.209
80	24	0.0833	10.10	135.92	0.841	0.189
80	28	0.0714	11.65	158.57	0.851	0.175
80	32	0.0625	13.21	181.23	0.857	0.166
80	36	0.0556	14.75	203.88	0.864	0.158
80	40	0.0500	16.32	226.53	0.868	0.153

Table 9(b) Same as for Table 9(a), but for a 16-node hypercube and a matrix of order $M = 360$.

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_{64}(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
160	8	0.2500	7.97	363.62	0.713	0.403
160	12	0.1667	11.09	545.42	0.768	0.301
160	16	0.1250	14.22	727.23	0.799	0.251
160	20	0.1000	17.34	909.34	0.819	0.221
160	24	0.0833	20.47	1090.85	0.833	0.201
160	28	0.0714	23.60	1272.66	0.843	0.187
160	32	0.0625	26.72	1454.46	0.851	0.176
160	36	0.0556	29.84	1636.27	0.857	0.167
160	40	0.0500	32.97	1818.08	0.862	0.161

Table 9(c) Same as for Table 9(a), but for a 64-node hypercube and a matrix of order $M = 720$.

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_4(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
40	8	0.2500	1.83	5.69	0.778	0.286
40	12	0.1667	2.60	8.54	0.821	0.218
40	16	0.1250	3.39	11.38	0.839	0.191
40	20	0.1000	4.16	14.23	0.855	0.169
40	24	0.0833	4.94	17.08	0.865	0.157
40	28	0.0714	5.71	19.92	0.873	0.146
40	32	0.0625	6.50	22.77	0.876	0.142
40	36	0.0556	7.28	25.61	0.880	0.136
40	40	0.0500	8.05	28.46	0.884	0.131

Table 10(a) Timings, efficiencies, and overheads for the back substitution algorithm on a 4-node hypercube for a matrix of order $M = 180$. The 1-node times, $T_1(n_b, m)$, were estimated from Eq. (35).

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_{16}(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
80	8	0.2500	3.96	45.33	0.716	0.396
80	12	0.1667	5.53	67.99	0.769	0.300
80	16	0.1250	7.10	90.66	0.798	0.253
80	20	0.1000	8.68	113.32	0.816	0.225
80	24	0.0833	10.26	135.98	0.829	0.207
80	28	0.0714	11.38	158.65	0.834	0.193
80	32	0.0625	13.40	181.31	0.846	0.182
80	36	0.0556	14.98	203.97	0.851	0.175
80	40	0.0500	16.55	226.64	0.856	0.168

Table 10(b) Same as for Table 10(a), but for a 16-node hypercube and a matrix of order $M = 360$.

m	$\hat{n}_b$	$1/\hat{n}_b$	$T_{64}(\hat{n}_b, \hat{m})$	$T_1(n_b, m)$	ϵ_{FR}	f_{FR}
160	8	0.2500	8.09	361.80	0.699	0.432
160	12	0.1667	11.25	542.69	0.753	0.327
160	16	0.1250	14.42	723.59	0.784	0.275
160	20	0.1000	17.57	904.49	0.804	0.244
160	24	0.0833	20.73	1085.39	0.818	0.223
160	28	0.0714	23.90	1266.28	0.828	0.208
160	32	0.0625	27.06	1447.18	0.836	0.197
160	36	0.0556	30.23	1628.08	0.842	0.188
160	40	0.0500	33.39	1808.98	0.846	0.181

Table 10(c) Same as for Table 10(a), but for a 64-node hypercube and a matrix of order $M = 720$.

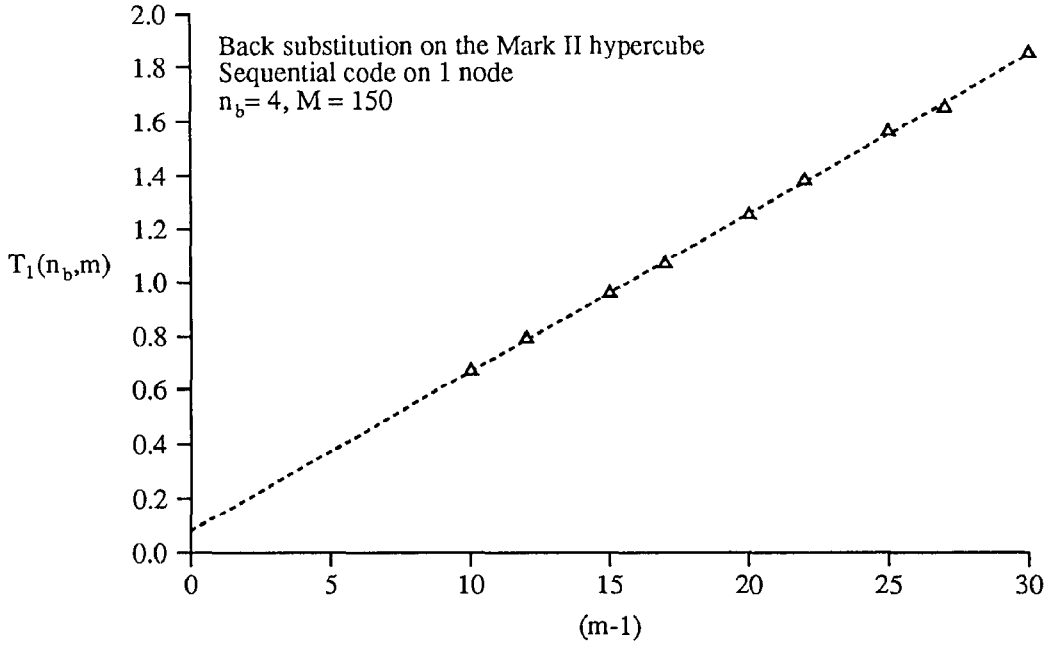

Figure 10. Timings for the sequential back substitution algorithm on 1-node of the hypercube as a function of $(m-1)$. In all cases the matrix order was $M = 150$, and the number of right-hand sides was $n_b = 4$.

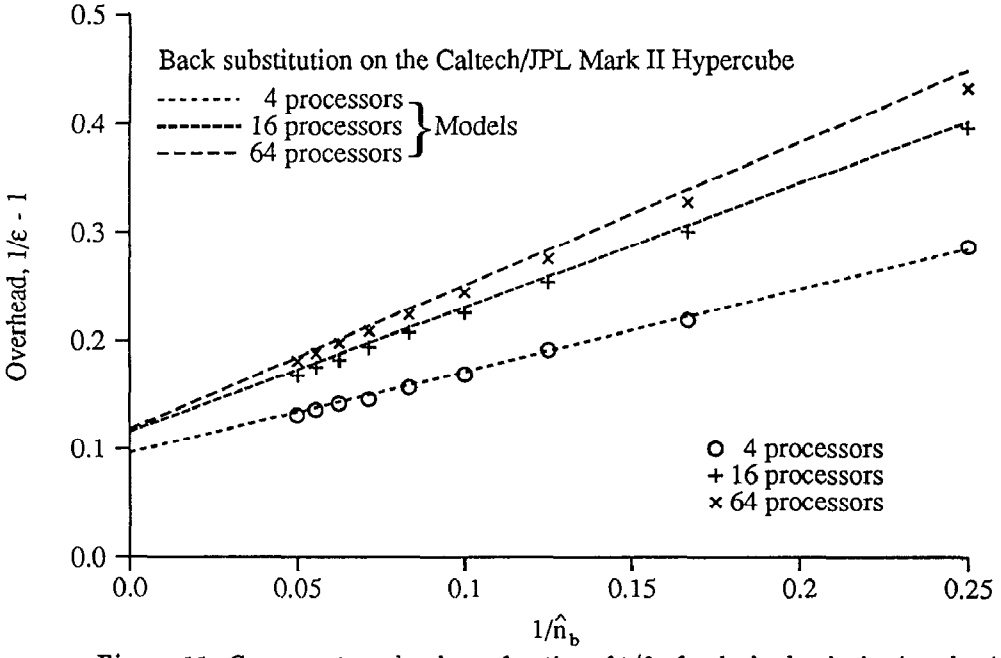

Figure 11. Concurrent overhead as a function of $1/\hat{n}_b$ for the back substitution algorithm. In all cases the value of $\hat{m}$ equals 20. The dashed curves show the results expected from the performance model (Eq. (25)), as discussed in the text.

References

[Baru 86] Baru, C. K., Su, S. Y. W., *The Architecture of SM3: A Dynamically Partitioned Multicomputer System*, IEEE Trans. on Computers, C35:790, 1986.

[Beernaert] Beernaert, L., and Roose, D., *Gaussian Elimination of banded Linear Systems With Pivoting on a Hypercube*, preprint from Dept. Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3030 Heverlee, Belgium, 1987.

[Chan 86] Chan, T., Saad, Y.,and Schultz, M., *Solving Elliptic Partial Differential Equations on the Hypercube Multiprocessor*, Supercomputer, 35:36, May 1986.

[Chu 87] Chu, E., George, A., *Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor*, Parallel Computing, 5:65, 1987.

[Dongarra 84a] Dongarra, J. J., Gustavson, F. G., and Karp, A., *Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine*, SIAM Review, 26:91, 1984.

[Dongarra 84b] Dongarra, J. J., Sameh, A. H., *On Some Parallel Banded System Solvers*, Parallel Computing, 1:223, 1984.

[Dongarra 87] Dongarra, J. J., Johnsson, L., *Solving Banded Systems on a Parallel Processor*, Parallel Computing, 5:219, 1987.

[Fox 84] Fox, G. C., *LU Decomposition for Banded Matrices*, Caltech Report C^3P-99, 1984.

[Fox 85] Fox, G. C., Hey, A. J. G., and Otto, S., *Matrix Algorithms on the Hypercube, I: Matrix Multiplication*, Caltech Report C^3P-206, 1985.

[Fox 87a] Fox, G. C., Johnson, M. J., Lyzenga, G. A., Otto, S. W., Salmon, J. K., Walker, D. W., *Solving Problems on Concurrent Processors*, pub. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987.

[Fox 87b] Fox, G. C., Hey, A. J. G., and Otto, S., *Matrix Algorithms on the Hypercube, I: Matrix Multiplication*, Parallel Computing, 4:17, 1987.

[Geist 86] Geist, G. A., Heath, M. T., *Matrix Factorization on a Hypercube Multiprocessor*, Hypercube Multiprocessors 1986, ed. Michael T. Heath, pub. SIAM, Philadelphia, 1986.

[Gilbert 58] Gilbert, E. N., *Grey Codes and Paths on the n-Cube*, Bell System Technical Journal, 37:815, May 1958.

[Golub 83] Golub, G. H., Van Loan, C. F., *Matrix Computation*, pub. John Hopkins Univ. Press, Baltimore, Maryland, 1983.

[Heller 78] Heller, D., *A Survey of Parallel Algorithms in Numerical Linear Algebra*, SIAM Review, 20:740, 1978.

[Johnsson 81] Johnsson, S. L., *Computational Arrays for Band Matrix Equations*, Technical Report 4287:TR:81, Dept. Computer Science, California Institute of Technology, May 1981.

[Johnsson 85] Johnsson, S. L., *Solving Narrow Banded Systems on Ensemble Architectures*, ACM Trans. Math. Software, 11:271, 1985.

[Johnsson 87a] Johnsson, S. L., Saad, Y., and Schultz, M. H., *Alternating Direct Methods on Multiprocessors*, SIAM J. Sci. Stat. Comput. 8:686, 1987.

[Johnsson 87b] Johnsson, S. L., and Ho, C-T., *Multiple Tridiagonal Systems, the Alternating Direction Method, and Boolean Cube Configured Multiprocessors*, Report YALEU/DCS/RR-532, Yale University, June 1987.

[Kung 80] Kung, H. T., and Leiserson, C., *Algorithms for VLSI Processor Arrays*, Section 8.3 of *Introduction to VLSI Systems*, Mead, C., and Conway, L., pub. Addison-Wesley, Reading, Mass., 1980.

[Martin 67] Martin, R. S., and Wilkinson, J. H., *Solution of Symmetric and Unsymmetric Band Equations and the Calculation of Eigenvectors of Band Matrices*, Numerische mathematik, 9:279, 1967.

[Navarro 87] Navarro, J. J., Llabería, J. M., Núñez, F. J., Valero, M., *LU Decomposition With No Size Restriction Using a One Dimensional Systolic Array Processor*, Vol. 3, pp 218-226, Proc. Second Int. Conf. on Supercomputing, ed., L. P. Kartashev and S. I. Kartashev, pub. International Supercomputing Inst. Inc., St. Petersburg, Florida, 1987.

[O'Neil 87] O'Neil, E., Allik, H., Moore, S., and Tenenbaum, E., *Finite Element Analysis on the BBN Butterfly Multiprocessor*, Vol. 1, pp 366-374, Proc. Second Int. Conf. on Supercomputing,

ed., L. P. Kartashev and S. I. Kartashev, pub. International Supercomputing Inst. Inc., St. Petersburg, Florida, 1987.

[Ortega 85] Ortega, J. M., and Voigt, R. G., *Solution of Partial Differential Equations on Vector and Parallel Computers*, SIAM Review, 27:149, 1985.

[Saad 85] Saad, Y., and Schultz, M. H., *Parallel Direct Methods for Solving Banded Linear Systems*, Yale Research Report YALEU/DCS/RR-387, 1985.

[Salmon 84] Salmon, J., *Binary Grey Codes and the Mapping of a Physical Lattice into a Hypercube*, Caltech report C^3P-51, 1984.

[Thakore 87] Thakore, A. K., and Su, S. Y. W., *Matrix Inversion and LU Decomposition on a Multicomputer System with Dynamic Control*, Vol. 1, pp 291-300, Proc. Second Int. Conf. on Supercomputing, ed., L. P. Kartashev and S. I. Kartashev, pub. International Supercomputing Inst. Inc., St. Petersburg, Florida, 1987.

[Utku 86] Utku, S., Melosh, R., and Salama, M., *Concurrent Cholesky Factorisation of Positive Definite Banded Hermitian Matrices*, Int. J. Numerical Methods in Eng., 23:2137, 1986.