August 1989

UILU-ENG-89-2223
DAC-15

COORDINATED SCIENCE LABORATORY
College of Engineering

A MULTILEVEL PARALLEL SOLVER FOR BLOCK TRIDIAGONAL AND BANDED LINEAR SYSTEMS

Ibrahim N. Hajj
Stig Skelboe

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) (DAC-15) UILU-ENG-89-2223 (DAC-15)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217 Santa Clara, CA
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program & INTEL 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-84-C-0149
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217 Santa Clara, CA	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
"A Multilevel Parallel Solver for Block Tridiagonal and Banded Linear Systems"

12. PERSONAL AUTHOR(S)
Hajj, Ibrahim N. and Skelboe, Stig

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1989 August 25	31

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Parallel solver, banded linear systems, block tridiagonal, cyclic reduction, multilevel decomposition, hypercube, multiprocessing.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

over

This paper describes an efficient algorithm for the parallel solution of systems of linear equations with a block tridiagonal coefficient matrix. The algorithm comprises a multilevel LU-factorization based on block cyclic reduction and a corresponding solution algorithm.

The paper includes a general presentation of the parallel multilevel LU-factorization and solution algorithms, but the main emphasis is on implementation principles for a message passing computer with hypercube topology. Problem partitioning, processor allocation and communication requirements are discussed for the general block tridiagonal algorithm.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19. Abstract (continued)

Band matrices can be cast into block tridiagonal form, and this special but important problem is dealt with in detail. It is demonstrated how the efficiency of the general block tridiagonal multilevel algorithm can be improved by introducing the equivalent of two–way Gaussian elimination for the first and the last partitioning and by carefully balancing the load of the processors. The presentation of the multilevel band solver is accompanied by detailed complexity analyses.

The properties of the parallel band solver were evaluated by implementing the algorithm on an Intel iPSC hypercube parallel computer and solving a larger number of banded linear equations using 2 to 32 processors. The results of the evaluation include speed–up over a sequential processor, and the measured values are in good agreement with the theoretical values resulting from complexity analysis. It is found that the maximum asymptotic speed–up of the multilevel LU–factorization using p processors and load balancing is approximated well by the expression (p+6)/4.

Finally, the multilevel parallel solver is compared with solvers based on row interleaved organization and with other block solvers.

A MULTILEVEL PARALLEL SOLVER FOR BLOCK TRIDIAGONAL AND BANDED LINEAR SYSTEMS

Ibrahim N. Hajj *
Stig Skelboe †

*Coordinated Science Laboratory and the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801, USA

†Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK–2100 Copenhagen, Denmark

Abstract

This paper describes an efficient algorithm for the parallel solution of systems of linear equations with a block tridiagonal coefficient matrix. The algorithm comprises a multilevel LU–factorization based on block cyclic reduction and a corresponding solution algorithm.

The paper includes a general presentation of the parallel multilevel LU–factorization and solution algorithms, but the main emphasis is on implementation principles for a message passing computer with hypercube topology. Problem partitioning, processor allocation and communication requirements are discussed for the general block tridiagonal algorithm.

Band matrices can be cast into block tridiagonal form, and this special but important problem is dealt with in detail. It is demonstrated how the efficiency of the general block tridiagonal multilevel algorithm can be improved by introducing the equivalent of two–way Gaussian elimination for the first and the last partitioning and by carefully balancing the load of the processors. The presentation of the multilevel band solver is accompanied by detailed complexity analyses.

The properties of the parallel band solver were evaluated by implementing the algorithm on an Intel iPSC hypercube parallel computer and solving a larger number of banded linear equations using 2 to 32 processors. The results of the evaluation include speed–up over a sequential processor, and the measured values are in good agreement with the theoretical values resulting from complexity analysis. It is found that the maximum asymptotic speed–up of the multilevel LU–factorization using p processors and load balancing is approximated well by the expression (p+6)/4.

Finally, the multilevel parallel solver is compared with solvers based on row interleaved organization and with other block solvers.

1 Introduction

Many technical and scientific problems involve the solution of linear systems of equations

$$A\,x = b \tag{1.1}$$

where A can be structured as a block tridiagonal matrix. The discretization of boundary value problems for both ordinary and partial differential equations may lead to band matrices which can be structured into block tridiagonal matrices. Likewise, the analysis of linearly connected substructures (mechanical, electrical,...) often leads to block tridiagonal matrices. These matrices may be numerically symmetric, symmetric in structure only, or nonsymmetric but still block tridiagonal in structure.

This paper presents an algorithm for the efficient solution of (1.1) on a parallel computer with medium to large number of processors. A version of the algorithm for solving structurally symmetric band systems has been implemented for the Intel iPSC hypercube, and the performance of the implementation of the algorithm is evaluated in great detail.

The multilevel parallel solver is based on block cyclic reduction [1] which permits separate LU–factorization and solution stages. The formulation of the block cyclic reduction which we have used is related to nested dissection [2,3], and it is an LU–factorization and solution of a block reordered system. The communication required by the band matrix implementation of the parallel solver is shown to be negligible.

Our implementation of the parallel solver introduces some enhancements to standard block cyclic reduction which may also be used for a general block tridiagonal matrix. The first enhancement is *two–way Gaussian elimination* for the first and the last block. This eliminates fill–ins in the two blocks and permits the second enhancement, an efficient *load balancing* which improves the efficiency with the equivalent of 6 extra processors. The third enhancement is a *block parallel* organization which improves processor utilization at the lower levels of the algorithm, with a modest communication penalty.

The performance of the implementation of the multilevel parallel band solver on the Intel iPSC with 32 processors is evaluated carefully, and the measured execution times are compared with predictions derived from complexity analysis.

Direct parallel solvers can be classified into two different groups: block methods (as the one described here and [4, 5, 6, 7]) and row interleaved methods [8]. The block methods pay a heavy penalty in terms of fill–ins while communication cost is negligible. The block methods can exploit many processors, limited only by the dimension/bandwidth ratio.

The row interleaved algorithm is computationally identical to a sequential Gaussian elimination. The pivot row is broadcasted and the processors perform the elimination in parallel. Ideal speed–up is prevented by communication which is fairly expensive on medium grain parallel processors such as the Intel iPSC [9]. The row interleaved algorithm can only exploit a number of processors corresponding to the half bandwidth.

The block methods are therefore advantageous for narrow band problems and medium to large number of processing elements. For wide band problems and few processors, the row interleaved algorithm is the better. It is also worth mentioning at this point that almost all reported implementations of parallel solvers of banded systems are done on

shared–memory vector machines, such as the Alliant and the Cray computers, while our implementation is done on a distributed–memory system with a limited number of processors and no vectorization, namely, the Intel iPSC.

This paper is organized as follows. In section 2 block tridiagonal matrices are briefly described. Sections 3 and 4 explain the multilevel LU–factorization and solution steps. Section 5 describes the general principles for implementing the multilevel algorithm on a hypercube, including the communication scheme among the processors. The details of the implementation on the hypercube are given in section 6. Section 6.1 describes the partitioning and allocation of tasks to different processors and the ordering of the first and the last partitions to reduce fill–ins by using 2–way Gaussian elimination. The performance of the multilevel algorithm is estimated in section 6.2 using complexity analysis. The important problem of load balancing is addressed in section 6.3. As a result of the complexity analysis it becomes clear that uniform partitioning of the band matrix will lead to poor load balance for the LU–factorization. Balance equations for selecting the sizes of the partitions are then derived. The performance of the parallel band matrix solver is presented in section 7. This includes the executing time graph model and actual numerical results. A comparison between the multilevel approach and the row interleaved factorization and solution approach is given in section 7.3.

2 Block tridiagonal matrices

Consider the system of linear equations (1.1) where A, x and b are partitioned as follows:

$$A = \begin{bmatrix} A_1 & B_1 & & & & \\ C_2 & A_2 & B_2 & & & \\ & C_3 & A_3 & B_3 & & \\ & & \cdots & \cdots & \cdots & \\ & & & C_N & A_N \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_N \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_N \end{bmatrix} \tag{2.1}$$

N is odd by assumption and

$$A_r \epsilon R^{n_r \times n_r} \text{ and } x_r, b_r \epsilon R^{n_r} \text{ for } r = 1, 2, .., N.$$

$$B_r \epsilon R^{n_r \times n_{r+1}} \text{ for } r = 1, 2, .., N-1 \text{ and}$$

$$C_r \epsilon R^{n_{r-1} \times n_r} \text{ for } r = 2, 3, .., N.$$

The entries of A are zero outside the tridiagonal band of matrices.

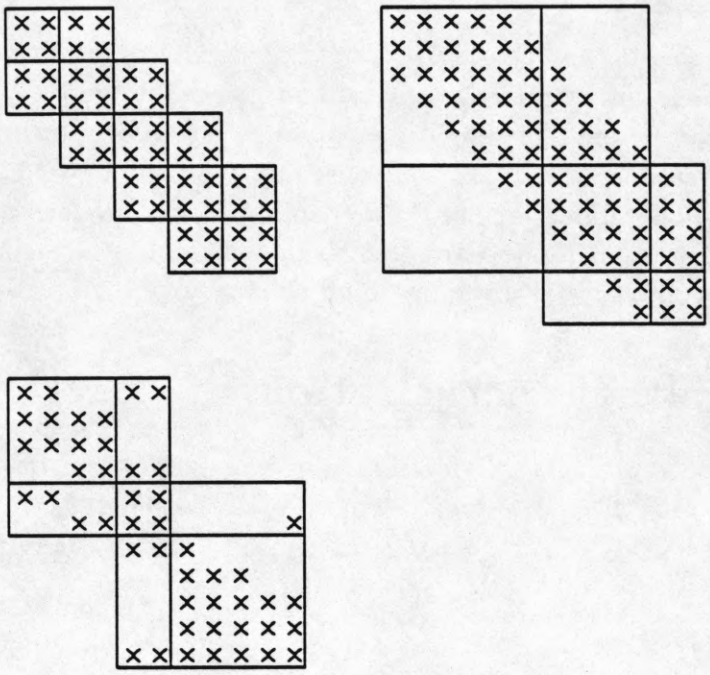

Figure 2.1.: Examples of typical block tridiagonal matrices.

Figure 2.1 shows three different examples of typical block tridiagonal matrices. The first matrix occurs in the lower levels of the multilevel algorithm applied to a structurally symmetric band matrix. The second matrix is a nonsymmetric band matrix with a block tridiagonal structure superimposed on it. The third example illustrates two connected arbitrary nonsymmetric sub–structures grouped into a block tridiagonal form.

The first level of the multilevel algorithm (block cyclic reduction) is based on the reordering of A given in (2.2).

$$\tilde{A} = \left[\begin{array}{cccc:cccc} A_1 & & & & B_1 & & & \\ & A_3 & & & C_3 & B_3 & & \\ & & A_5 & & & C_5 & B_5 & \\ & & & \cdots & & & \cdots & \cdots \\ & & & A_N & & & & C_N \\ \hdashline C_2 & B_2 & & & A_2 & & & \\ & C_4 & B_4 & & & A_4 & & \\ & & C_6 & \cdots & & & A_6 & \\ & & & \cdots & & & & \cdots \\ & & & \cdots \; B_{N-1} & & & & A_{N-1} \end{array}\right] \tag{2.2}$$

The sets $\{C_r, A_r, B_r, B_{r-1}, C_{r+1}\}$ for r=2, 4, .., N-1 are called separators since they separate the matrix A into independent blocks $A_r, r = 1, 3, .., N$. The reordering of A into $\tilde{A}$ therefore involves a symmetric row column reordering where the separators are moved to the last rows and columns. If A is diagonally dominant, so is $\tilde{A}$ since the symmetric

reordering preserves diagonal dominance. Obviously, any symmetry of A will also be preserved.

N is assumed to be odd, and this is merely a convenient assumption. If N is originally even, either two block rows and columns can be merged to reduce N by one or the reordering of (2.2) can be modified slightly to account for an even value of N.

In practical applications, the dimensions of separators, n_r for r=2,4,..,N-1, are usually small compared with the dimensions of the remaining blocks, n_r for r=1,3,...,N, but this is not a necessary condition for the application of the multilevel algorithm although it may be a necessary condition for obtaining high efficiency.

3 Multilevel LU–factorization

The first level of the multilevel LU–factorization is a standard LU–factorization of $\tilde{A}$ defined in (2.2) stopping after $n_1 + n_3 \ldots + n_N$ pivot rows. This leaves the lower right–hand block of dimension $n_2 + n_4 \ldots + n_{N-1}$ unfactored. The partially factored matrix is called $\bar{A}$ and is defined in (3.1)

$$\bar{A} = \left[\begin{array}{ccccccc|ccccccc} L_1U_1 & & & & & & & \bar{B}_1 & & & & & & \\ & L_3U_3 & & & & & & \bar{C}_3 & \bar{B}_3 & & & & & \\ & & L_5U_5 & & & & & & \bar{C}_5 & \bar{B}_5 & & & & \\ & & & \ddots & & & & & & \ddots & \ddots & \ddots & & \\ & & & & L_NU_N & & & & & & & & \bar{C}_N & \\ \hline \bar{C}_2 & \bar{B}_2 & & & & & & \bar{A}_2 & \bar{D}_2 & & & & & \\ & \bar{C}_4 & \bar{B}_4 & & & & & \bar{E}_4 & \bar{A}_4 & \bar{D}_4 & & & & \\ & & \bar{C}_6 & \ddots & & & & & \bar{E}_6 & \bar{A}_6 & \ddots & & & \\ & & & \ddots & & & & & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \bar{B}_{N-1} & & & & & & \ddots & \bar{A}_{N-1} & & \end{array}\right] \tag{3.1}$$

The submatrices in (3.1) are related to the ones in (2.2) as follows

$$L_rU_r = A_r, r = 1, 3, .., N \tag{3.2a}$$

where L_r and U_r are lower and upper triangular matrices, respectively.

$$\bar{B}_r = L_r^{-1}B_r,\ r = 1, 3, .., N-2. \tag{3.2b}$$

$$\bar{C}_r = L_r^{-1}\ C_r,\ r = 3, 5, .., N. \tag{3.2c}$$

$$\bar{C}_{r+1} = C_{r+1}U_r^{-1},\ r = 1, 3, .., N-2. \tag{3.2d}$$

$$\bar{B}_{r-1} = B_{r-1}U_r^{-1},\ r = 3, 5, .., N. \tag{3.2e}$$

$$\bar{A}_s = A_s - \bar{C}_s\bar{B}_{s-1} - \bar{B}_s\bar{C}_{s+1},\ s = 2, 4, .., N-1. \tag{3.2f}$$

$$\bar{D}_s = D_s - \bar{B}_s\bar{B}_{s+1},\ s = 2,4,..,N-3. \tag{3.2g}$$

$$\bar{E}_s = E_s - \bar{C}_s\bar{C}_{s-1},\ s = 4,6,..,N-1. \tag{3.2h}$$

The tableau in (3.1) and the relations (3.2) reveal the block parallelism in computing $\bar{A}$. The factorization of a diagonal block A_r (r is odd) and the associated row operations leading to $\bar{C}_r, \bar{B}_r, \bar{B}_{r-1}, \bar{C}_{r+1}, \bar{D}_{r-1}$ and $\bar{E}_{r+1}$ can be performed independenly of the corresponding computations for different r-values. The only overlap is in the computation of $\bar{A}_s$ as specified in (3.2f). At the end of this section, a convenient organization for the parallel computation of $\bar{A}$ is presented.

Notice that $D_s = 0$ $(s = 2,4,..,N-3)$ and $E_s = 0$ $(s = 4,6,..,N-1)$ for the block tridiagonal matrix A defined in (2.1). However, the multilevel algorithm handles at no extra cost the case where the separators are extended with $D_s \neq 0$ and $E_s \neq 0$. In this case the even numbered rows and columns of A (the separators) will contain five matrix blocks: $E_s, C_s, A_s, B_s, D_s$and$D_{s-2}, B_{s-1}, A_s, C_{s+1}, E_{s+2}$, respectively for s = 4, 6, .., N-3. For s = 2, D_0 and E_2 will be absent while D_{N-1} and E_{N+1} will be absent for s = N-1.

The D_s and E_s matrices have the following dimensions:

$$D_s \epsilon R^{n_s \times n_{s+2}} \text{ for } s = 2,4,..,N-3$$

$$E_s \epsilon R^{n_s \times n_{s-2}} \text{ for } s = 4,6,..,N-1.$$

In the tridiagonal case (2.1), the matrices $\bar{D}_s$ and $\bar{E}_s$ are fill–ins. The LU–factorization will in general also create fill–ins in the original blocks of A unless they are full from the outset.

The four blocks of $\bar{A}$ seperated by the dashed lines can be expressed in a compact form as:

$$\bar{A} = \begin{bmatrix} LU & V \\ W & A_t \end{bmatrix} \tag{3.3a}$$

where

$$\tilde{A} = \begin{bmatrix} L & 0 \\ W & I \end{bmatrix} \begin{bmatrix} U & V \\ 0 & A_t \end{bmatrix} \tag{3.3b}$$

A_t is the Schur complement and L and U are lower and upper block triangular matrices, respectively, composed of L_r and U_r, r=1, 3, .., N.

The multilevel algorithm (block cyclic reduction) is now based on the fact that A_t is a block tridiagonal matrix which can be reordered into a form similar to $\tilde{A}$ in (2.2), partially LU–factored like $\bar{A}$ leaving a block tridiagonal partially factored lower right–hand block etc.

If $N = 2^{d+1} - 1$, the process will terminate after d levels with a Schur complement consisting of just one block which is then factored. If N is composed differently, some

of the intermediate block tridiagonal matrices will have an even block dimension, but as mentioned in the previous section, this is only a minor complication.

The partial factorization of $\tilde{A}$ leading to $\bar{A}$ can be performed conveniently in parallel by partitioning the original block tridiagonal matrix A as follows:

$$Q_1 = \begin{bmatrix} A_1 & B_1 \\ C_2 & A_2 \end{bmatrix} \text{ and } Q_N = \begin{bmatrix} A_N & C_N \\ B_{N-1} & 0 \end{bmatrix} \tag{3.4a, b}$$

$$Q_r = \begin{bmatrix} A_r & B_r & C_r \\ C_{r+1} & A_{r+1} & E_{r+1} \\ B_{r-1} & D_{r-1} & 0 \end{bmatrix} \text{ for } r = 3, 5, .., N-2 \tag{3.4c}$$

$D_{r-1} = 0$ and $E_{r+1} = 0$ when A is block tridiagonal, and $D_{r-1} \neq 0$ and $E_{r+1} \neq 0$ only when A has extended separators as discussed priviously. The Q–matrices can be considered slightly reordered samples of A which are straightforward to establish.

The Q–matrices can be partially factored in parallel leading to the $\bar{Q}$–matrices defined in (3.5).

$$\bar{Q}_1 = \begin{bmatrix} L_1 U_1 & \bar{B}_1 \\ \bar{C}_2 & \hat{A}_2 \end{bmatrix} \text{ and } \bar{Q}_N = \begin{bmatrix} L_N U_N & \bar{C}_N \\ \bar{B}_{N-1} & \tilde{A}_{N-1} \end{bmatrix} \tag{3.5a, b}$$

$$\bar{Q}_r = \begin{bmatrix} L_r U_r & \bar{B}_r & \bar{C}_r \\ \bar{C}_{r+1} & \hat{A}_{r+1} & \bar{E}_{r+1} \\ \bar{B}_{r-1} & \bar{D}_{r-1} & \tilde{A}_{r-1} \end{bmatrix} \text{ for } r = 3, 5, .., N-2. \tag{3.5c}$$

The $\bar{Q}$–matrices can be computed conveniently by applying standard LU–factorization to the Q–matrices and stopping when A_r, for r being odd, is completely factored.

The entries of the $\bar{Q}$–matrices are as defined in (3.2) except for $\hat{A}_s$ and $\tilde{A}_s$:

$$\hat{A}_s = A_s - \bar{C}_s \bar{B}_{s-1} \text{ and}$$

$$\tilde{A}_s = -\bar{B}_s \bar{C}_{s+1} \text{ leading to}$$

$$\bar{A}_s = \hat{A}_s + \tilde{A}_s \text{ for } s = 2, 4, .., N-1$$

Pivoting has not been considered so far since the possibilities are limited in the multi-level algorithm. From the partial LU–factorizations leading to the $\bar{Q}$–matrices, however,it is obvious that pivoting can be done as usual during the factorization of A_r for r being odd as long as the search for a pivot element is limited to A_r. If A_r is singular, the algorithm requires fundamental modifications to work properly.

The next level of the multilevel algorithm involves the partial LU–factorization of A_t defined in (3.3). The block representation of A_t as given in (3.1) is

$$A_t = \begin{bmatrix} \bar{A}_2 & \bar{D}_2 & & & \\ \bar{E}_4 & \bar{A}_4 & \bar{D}_4 & & \\ & \bar{E}_6 & \bar{A}_6 & \dots & \\ & & \dots & \dots & \dots \\ & & & \dots & \bar{A}_{N-1} \end{bmatrix}$$

The A_t–matrix is sampled similarly to A to create Q_2, Q_6,.., Q_{N-1} defined in (3.6). From (3.5) it is seen that Q_2, Q_6,.., Q_{N-1} can also be obtained by sampling $\bar{Q}_r$ for r being odd. The relationships are listed in (3.7) where $\left(\bar{Q}_{s-1},\ \bar{Q}_{s+1}, \bar{Q}_{s+3}\right) \rightarrow Q_s$ specifies that Q_s is composed of samples from $\bar{Q}_{s-1}, \bar{Q}_{s+1}$ and $\bar{Q}_{s+3}$ etc.

$$Q_2 = \begin{bmatrix} \bar{A}_2 & \bar{D}_2 \\ \bar{E}_4 & \bar{A}_4 \end{bmatrix} \text{ and } Q_{N-1} = \begin{bmatrix} \bar{A}_{N-1} & \bar{E}_{N-1} \\ \bar{D}_{N-3} & 0 \end{bmatrix} \tag{3.6a, b}$$

$$Q_s = \begin{bmatrix} \bar{A}_s & \bar{D}_s & \bar{E}_s \\ \bar{E}_{s+2} & \bar{A}_{s+2} & 0 \\ \bar{D}_{s-2} & 0 & 0 \end{bmatrix} \text{ for } s = 6, 10, .., N-5. \tag{3.6c}$$

$$\left(\bar{Q}_{s-1},\ \bar{Q}_{s+1},\ \bar{Q}_{s+3}\right) \rightarrow Q_s \text{ for s} = 2, 6, 10, .., \text{N} - 5. \tag{3.7a}$$

$$\left(\bar{Q}_{N-2},\ \bar{Q}_N\right) \rightarrow Q_{N-1} \tag{3.7b}$$

The partial LU–factorization of A_t can now be performed in parallel by partially factoring Q_2, Q_6,.., Q_{N-1} which are resampled to create Q_4, Q_8,.., Q_{N-3} etc. until the LU–factorization is completed by factoring just one block.

Each partial LU–factorization of a Q–matrix completes the factorization of the first block row and column leaving the lower right–hand 2×2 block unfactored and the subject of the next level of factorization.

So far it has been implied that only one processor is to be used for the partial factorization of a Q–matrix. Since each level of the multilevel algorithm deals with only half the number of Q–matrices as the previous level, one might consider using more than one processor for each Q–matrix to try to keep all processors busy.

In the block parallel organization described in [10], one processor is assigned to each Q–matrix in the top level, two processors in the next level etc. up to 4 processors for the 2×2 Q–matrices and 8 processors for the 3×3 Q–matrices. The natural partitioning of the Q–matrices into blocks is used to allocate one or several blocks to each processor. Since the LU–factorization is partial, this approach results in good load balance and moderate communication overhead.

4 Multilevel solution

The purpose of the solution step is to compute x of (1.1). The solution is expressed symbolically as

$$x = A^{-1}\, b \tag{4.1}$$

The reordering of A into $\tilde{A}$ is expressed by the permutation matrix H_1 such that $H_1\, A\, H_1^T = \tilde{A}$. This leads to the equation

$$\tilde{A}\, \tilde{x} = \tilde{b} \tag{4.2}$$

where $\tilde{x} = H_1\, x$ $\left(\text{and } \mathrm{x} = \mathrm{H}_1^{\mathrm{T}}\, \tilde{\mathrm{x}}\right)$ and $\tilde{b} = H_1\, b$. Equation (4.2) can now be expressed by the partial factorization in (3.3b),

$$\begin{bmatrix} L & 0 \\ W & I \end{bmatrix} \begin{bmatrix} U & V \\ 0 & A_t \end{bmatrix} \begin{bmatrix} x_u \\ x_t \end{bmatrix} = \begin{bmatrix} b_u \\ b_t \end{bmatrix} \tag{4.3}$$

where (x_u, x_t) and (b_u, b_t) are partitionings of $\tilde{x}$ and $\tilde{b}$, respectively, corresponding to the block factorization of $\tilde{A}$.

Equation (4.3) is solved by the standard approach,

$$y_u = L^{-1}\, b_u \ , \ \bar{b}_t = b_t - W y_u \tag{4.4a, b}$$

$$x_t = A_t^{-1}\, \bar{b}_t \tag{4.4c}$$

$$x_u = U^{-1}\, (y_u - V x_t) \tag{4.4d}$$

The computation of x_t in (4.4c) expresses symbolically the solution of a block tridiagonal system of equations similar to (1.1), and (4.4c) is therefore analogous to (4.1). The algorithm outlined by the equations (4.1), (4.2), (4.3) and (4.4) is therefore applied recursively until all components of the solution are eventually computed.

When the complete multilevel LU–factorization is available, the multilevel solution is obtained by a recursive application of the relations (4.4a,b,d) until (4.4c) involves just one LU–factored block. In order to describe the multilevel solution algorithm the vectors x_u, x_t , b_u and b_t are defined from the partitioning in (2.1) and the permutation H_1.

$$x_u = \begin{bmatrix} x_1 \\ x_3 \\ \cdots \\ x_N \end{bmatrix} \ , \ x_t = \begin{bmatrix} x_2 \\ x_4 \\ \cdots \\ x_{N-1} \end{bmatrix} \ , \ b_u = \begin{bmatrix} b_1 \\ b_3 \\ \cdots \\ b_N \end{bmatrix} \ , \ b_t = \begin{bmatrix} b_2 \\ b_4 \\ \cdots \\ b_{N-1} \end{bmatrix}$$

Furthermore partitioned vectors y_u and $\bar{b}_t$ similar to x_u and b_t, respectively, are defined. The detailed block relations corresponding to (4.4 a,b,d) are:

$$y_r = L_r^{-1}\, b_r \text{ for } \mathrm{r} = 1, 3, .., \mathrm{N}. \tag{4.5a}$$

$$\bar{b}_{r+1} = b_{r+1} - \bar{C}_{r+1}\, y_r - \bar{B}_{r+1}\, y_{r+2} \text{ for } \mathrm{r} = 1, 3, .., \mathrm{N} - 2. \tag{4.5b}$$

$$x_1 = U_1^{-1}\left(y_1 - \bar{B}_1 x_2\right)\ ,\ x_N = U_N^{-1}\left(y_N - \bar{C}_N x_{N-1}\right)\ ,$$
$$x_r = U_r^{-1}\left(y_r - \bar{C}_r\, x_{r-1} - \bar{B}_r x_{r+1}\right) \text{ for r} = 3,5,..,\text{N}-2 \tag{4.5d}$$

Each of the three sets of relations (4.5a), (4.5b) and (4.5d) can be computed in parallel using (N+1)/2 processors. The computation of y_r and x_r requires factorized matrix blocks from $\bar{Q}_r$ defined in (3.5) while $\bar{b}_s$ requires blocks from both $\bar{Q}_{s-1}$ and $\bar{Q}_{s+1}$.

5 Hypercube implementation

This section describes the general principles for the implementation of the multilevel algorithm on a hypercube parallel computer of dimension d. The number of processors is $p = 2^d$, and it will be assumed that all processors are used for the top level of the multilevel algorithm which means that p is related to the block dimension N through the following relation,

$$p = 2^d = (N+1)/2 \tag{5.1}$$

If N is determined by the problem (which is not the case for a band matrix) and violates relation (5.1), the multilevel algorithm may require minor modifications. If $N > 2p-1$, more processors may be simulated by running several processes on each processor. If $N < 2p-1$, some processors may be left idle or more than one processor may be used for some or all Q–matrices (3.4).

It will be assumed in the rest of this section that (5.1) is satisfied. The Q–matrices of the top level, level 1, are allocated to the processors as follows. Matrix Q_{2i+1} is allocated to processors P_i for i=0,1,..,p-1. This is expressed formally as

$$Q_{2i+1} \longrightarrow P_i \text{ for i} = 0,1,..,\text{p}-1 \tag{5.2a}$$

The allocation relation for level 2 is

$$Q_{4i+2} \longrightarrow P_{2i} \text{ for i} = 0,1,..,(\text{p}/2)-1 \tag{5.2b}$$

The general allocation relation for level ℓ is

$$Q_{2^\ell i+2^{\ell-1}} \longrightarrow P_{2^{(\ell-1)}i} \text{ for i} = 0,1,..,\left(\text{p}/2^{(\ell-1)}\right)-1 \tag{5.3}$$

The allocation principle is illustrated in Fig. 5.1 for p=8 and consequently N=15. The processors are labeled 0,1,..,7 expressed as binary numbers. Processors in a hypercube can be labeled such that processors whose labels differ in only one bit position are

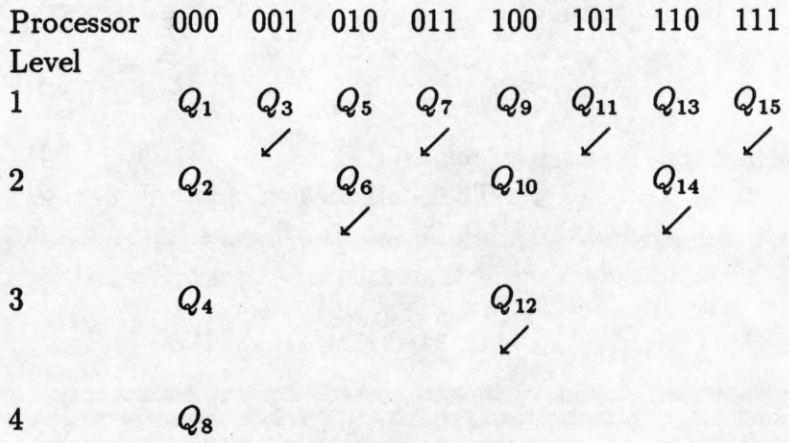

Figure 5.1: Processor allocation and communication structure for a hypercube implementation of the multilevel algorithm.

neighbors. The allocation relation in (5.3) is constructed to permit multilevel factorization and solution using only neighbor to neighbor communication.

The multilevel LU–factorization proceeds as follows. At level 1, Q_1, Q_3,.., Q_N are partially LU–factored in parallel. Then the unfactored parts of $\bar{Q}_3$, $\bar{Q}_7$,.., $\bar{Q}_N$ are passed on to the processors storing $\bar{Q}_1$, $\bar{Q}_5$,.., $\bar{Q}_{N-2}$ (see Fig. 5.1). This involves only neighbor to neighbor communication, and it can be done completely in parallel. However, according to (3.7a) the construction of Q_2, Q_6,.., Q_{N-5} requires contributions from three $\bar{Q}$–matrices from the previous level, and thus more communication and a more elaborate communication scheme to limit communication distance are required.

In order to circumvent this problem and use the simple allocation and communication scheme examplified in Fig. 5.1, the Q–matrices of levels other than the first one are redefined slightly.

$$Q_2' = \begin{bmatrix} \bar{A}_2 & \bar{D}_2 \\ \bar{E}_4 & \hat{A}_4 \end{bmatrix}, \quad Q_{N-1}' = \begin{bmatrix} \bar{A}_{N-1} & \bar{E}_{N-1} \\ \bar{D}_{N-3} & \tilde{A}_{N-3} \end{bmatrix} \tag{5.4a, b}$$

$$Q_s' = \begin{bmatrix} \bar{A}_s & \bar{D}_s & \bar{E}_s \\ \bar{E}_{s+2} & \hat{A}_{s+2} & 0 \\ \bar{D}_{s-2} & 0 & \tilde{A}_{s-2} \end{bmatrix}, \text{ for s} = 6, 10, .., \text{N} - 5 \tag{5.4c}$$

The definition in (5.4) will supersede (3.6) in the following, and the prime symbol will be left out from the Q–matrices defined by (5.4). Likewise, the construction rule (3.7) is superseded by

$$\left(\bar{Q}_{s-1}, \bar{Q}_{s+1}\right) \rightarrow Q_s \text{ for s} = 2, 6, .., \text{N} - 1 \tag{5.5}$$

The construction of the general matrix Q_s in (5.4c) is easily verified by writing $\bar{Q}_{s-1}$ and $\bar{Q}_{s+1}$ in detail, based on (3.5c):

$$\bar{Q}_{s-1} = \begin{bmatrix} L_{s-1}U_{s-1} & \bar{B}_{s-1} & \bar{C}_{s-1} \\ \bar{C}_s & \hat{A}_s & \bar{E}_s \\ \bar{B}_{s-2} & \bar{D}_{s-2} & \tilde{A}_{s-2} \end{bmatrix}, \quad \bar{Q}_{s+1} = \begin{bmatrix} L_{s+1}U_{s+1} & \bar{B}_{s+1} & \bar{C}_{s+1} \\ \bar{C}_{s+2} & \hat{A}_{s+2} & \bar{E}_{s+2} \\ \bar{B}_s & \bar{D}_s & \tilde{A}_s \end{bmatrix}$$

Q_s is composed from the lower right-hand 2×2 blocks and $\bar{A}_s$ is computed as $\bar{A}_s = \hat{A}_s + \tilde{A}_s$.

The observation leading to the redefinition of the Q-matrices for levels greater than one is that the partial factorization of a Q-matrix leaves the first block row and column completely factored and the rest unfactored. This means that the first block row and column (and especially A_s, s=2,6,..,N-1) must hold the final values before the factorization while intermediate values (like $\hat{A}_s$ and $\tilde{A}_s$, s=4,8,..,N-3) suffice for the remaining entries of the Q-matrix.

After the partial LU-factorization, the $\bar{Q}$-matrices of level 2 will have the same structure as the $\bar{Q}$-matrices of level 1 defined in (3.5). The allocation algorithm ensures that the Q-matrices on each level are allocated to processors that are pairwize neighbors. This means that the partial LU-factorization of the Q-matrices of level ℓ and neighbor to neighbor communication to compose the Q-matrices of level ℓ+1 can be continued until the last level, d+1. At the last level $Q_{(N+1)/2}$, which is only one block, is computed by adding the lower right-hand blocks of $\bar{Q}_{(N+1)/4}$ and $\bar{Q}_{3(N+1)/4}$, and finally $Q_{(N+1)/2}$ is fully LU-factored.

The blocks of the b-vector defined in (2.1) are allocated with the corresponding A-blocks (diagonal blocks) which means that

$$(b_{2i+1}, b_{2i+2}) \longrightarrow P_i \text{ for } \mathrm{i} = 0, 1, .., \mathrm{p} - 2 \tag{5.6a}$$

$$b_N \longrightarrow P_{p-1} \tag{5.6b}$$

With this allocation, the first level of the solution algorithm defined in (4.5a,b) is performed as follows

$$P_0 : \; y_1 = L_1^{-1} b_1 \; , \; \hat{b}_2 = b_2 - \bar{C}_2 y_1 \tag{5.7a, b}$$

$$P_i : \; y_r = L_r^{-1} b_r \; , \; \hat{b}_{r+1} = b_{r+1} - \bar{C}_{r+1} y_r \tag{5.7c, d}$$

$$\tilde{b}_{r-1} = -\bar{B}_{r-1} y_r \tag{5.7e}$$

$$\text{for } \mathrm{r} = 2\mathrm{i} + 1 \text{ and } \mathrm{i} = 1, 2, .., \mathrm{p} - 2$$

$$P_{p-1} : \; y_N = L_N^{-1} b_N \; , \; \tilde{b}_{N-1} = -\bar{B}_{N-1} y_N \tag{5.7f, g}$$

The partial $\bar{b}$-vectors $\hat{b}_s$ and $\tilde{b}_s$ are associated with the unfactored diagonal blocks $\hat{A}_s$ and $\tilde{A}_s$. This implies that they are communicated along the same routes and the calculation $\bar{b}_s = \hat{b}_s + \tilde{b}_s$ takes place in the same processor and at the same level as the calculation $\bar{A}_s = \hat{A}_s + \tilde{A}_s$.

The first communication step then involves

$$\text{Send } \left(\tilde{b}_{4i+2}\,,\ \hat{b}_{4i+4}\right) \text{ to } P_{2i} \text{ for } i = 0, 1, .., (p/2) - 2.$$

$$\text{Send } \tilde{b}_{N-1} \text{ to } P_{p-2}$$

At level 2 the computation of $\bar{b}_2$, $\bar{b}_6$, .., $\bar{b}_{N-1}$ are completed according to (4.5b),

$$\bar{b}_{4i+2} = \hat{b}_{4i+2} + \tilde{b}_{4i+2} \text{ for } i = 0, 1, .., (p/2) - 1 \tag{5.8}$$

The remaining $\bar{b}$-vectors, $\bar{b}_4$, $\bar{b}_8$, .., $\bar{b}_{N-3}$ are not computed at level 2 since they are not needed at this level. Besides, their computation would require communication among non–neighbors.

The computation algorithm and communication scheme as outlined can be continued down to the bottom level and up again according to (4.5d). Instead of giving a formal algorithm which will come out rather complicated, the informal description of the multilevel solution algorithm will be supplemented with an example for N=7 and p=4 processors. The example shown in Fig. 5.2 also includes the $\bar{Q}$–matrices and a specification of the level where they were computed.

The communication pattern of the solution phase can be deduced from the data dependencies among the different levels of computations shown in Fig. 5.2.

Proc. Level	00	01	10	11
1	$\begin{bmatrix} L_1U_1 & \bar{B}_1 \\ \bar{C}_2 & \hat{A}_2 \end{bmatrix}$	$\begin{bmatrix} L_3U_3 & \bar{B}_3 & \bar{C}_3 \\ \bar{C}_4 & \hat{A}_4 & \bar{E}_4 \\ \bar{B}_2 & \bar{D}_2 & \tilde{A}_2 \end{bmatrix}$	$\begin{bmatrix} L_5U_5 & \bar{B}_5 & \bar{C}_5 \\ \bar{C}_6 & \hat{A}_6 & \bar{E}_6 \\ \bar{B}_4 & \bar{D}_4 & \tilde{A}_4 \end{bmatrix}$	$\begin{bmatrix} L_7U_7 & \bar{C}_7 \\ \bar{B}_6 & \tilde{A}_6 \end{bmatrix}$
2	$\begin{bmatrix} L_2U_2 & \bar{\bar{D}}_2 \\ \bar{\bar{E}}_4 & \hat{\hat{A}}_4 \end{bmatrix}$		$\begin{bmatrix} L_6U_6 & \bar{\bar{E}}_6 \\ \bar{\bar{D}}_4 & \tilde{\tilde{A}}_4 \end{bmatrix}$	
3	L_4U_4			
1	$y_1 = L_1^{-1}b_1$ $\hat{b}_2 = b_2 - \bar{C}_2y_1$	$y_3 = L_3^{-1}b_3$ $\hat{b}_4 = b_4 - \bar{C}_4y_3$ $\tilde{b}_2 = -\bar{B}_2y_3$	$y_5 = L_5^{-1}b_5$ $\hat{b}_6 = b_6 - \bar{C}_6y_5$ $\tilde{b}_4 = -\bar{B}_4y_5$	$y_7 = L_7^{-1}b_7$ $\tilde{b}_6 = -\bar{B}_6y_7$
2	$\bar{b}_2 = \hat{b}_2 + \tilde{b}_2$ $y_2 = L_2^{-1}\bar{b}_2$ $\hat{\hat{b}}_4 = \hat{b}_4 - \bar{\bar{E}}_4y_2$		$\bar{b}_6 = \hat{b}_6 + \tilde{b}_6$ $y_6 = L_6^{-1}\bar{b}_6$ $\tilde{\tilde{b}}_4 = \tilde{b}_4 - \bar{\bar{D}}_4y_6$	
3	$\bar{\bar{b}}_4 = \hat{\hat{b}}_4 + \tilde{\tilde{b}}_4$ $y_4 = L_4^{-1}\bar{\bar{b}}_4$ $x_4 = U_4^{-1}y_4$			
2	$x_2 = U_2^{-1}$ $\left(y_2 - \bar{\bar{D}}_2x_4\right)$		$x_6 = U_6^{-1}$ $\left(y_6 - \bar{\bar{E}}_6x_4\right)$	
1	$x_1 = U_1^{-1}$ $\left(y_1 - \bar{B}_1x_2\right)$	$x_3 = U_3^{-1}$ $\left(y_3 - \bar{B}_3x_4 - \bar{C}_3x_2\right)$	$x_5 = U_5^{-1}$ $\left(y_5 - \bar{B}_5x_6 - \bar{C}_5x_4\right)$	$x_7 = U_7^{-1}$ $\left(y_7 - \bar{C}_7x_6\right)$

Figure 5.2: Parallel solution algorithm for N=7 based on multilevel LU–factorization.

6 Parallel band matrix solver for hypercube

6.1 *Partitioning and allocation*

The principles of a multilevel parallel solver described in the previous sections were used for an implementation on the Intel iPSC hypercube. The implementation is restricted to linear systems with structurally symmetric band matrices. This restriction is exploited in the implementation to improve the basic implementation principles described in the previous section. The improvement involves the equivalent of 2–way Gaussian elimination for the partial factorization of Q_1 and Q_N to permit an efficient load distribution. Programs and documentation are presented in [11].

The band matrix is charaterized by dimension n and by bandwidth w=1+2m where m is the number of upper or lower off–diagonal elements.

The block tridiagonal structure of (2.1) is imposed on the band matrix by choosing N and $n_1, n_2.., n_N$ properly. There exist the following constraints:

$$\sum_{r=1}^{N} n_r = n \text{ and } n_r \geq m \text{ for } r = 1, 2, .., N$$

N is chosen to match the actual hypercube (or sub–cube) of dimension d which means that relation (5.1) is fulfilled.

The dimension of the separators is chosen to the minimum dimension,

$$n_s = m \ , \ s = 2, 4, .., N-1$$

in order to minimize the amount of work involved in the lower levels of the algorithm.

The dimensions of the odd numbered blocks are in general determined by the load distribution algorithm which will be described later. However, a special case should be mentioned here, namely the minimum dimension problem where $n_r = m$, $r = 1, 2, .., N$. This is the case where the blocks of the block tridiagonal matrix in (2.1) are chosen as small as possible. This could be the case if the algorithm is excuted by a fine grain parallel computer where the maximum number of blocks for a particular band matrix are required in order to exploit as many processors as possible.

Besides, the lower right–hand block tridiagonal matrix of (3.1) (called A_t in (3.3)) is a minimum dimension matrix with all blocks of dimension m when $\bar{A}$ originates from a block tridiagonal matrix with seperators of dimension m. The minimum dimension property also applies to the lower levels.

The structure of $\bar{Q}_r$ for r=3, 5, .., N-2 is given in Fig. 6.1. The densely dotted areas correspond to the non–zero entries of Q_r with $D_{r-1} = 0, E_{r+1} = 0$ and the lightly dotted areas correspond to fill–ins created by the partial factorization. $\bar{Q}_1$ and $\bar{Q}_N$ are depicted similarly in Fig. 6.2 and 6.3.

The partial factorization of Q_r leading to $\bar{Q}_r$ for r=3,5,..,N results in rather severe amounts of fill–ins, not just in terms of zero blocks being filled, like $\tilde{A}_{r-1}$, $\bar{D}_{r-1}$ and $\bar{E}_{r+1}$ but also fill–ins inside B_{r-1} and C_r . Q_1 is partially factored completely without fill–ins while Q_N is similar to the general Q_r–matrix in this respect.

However, it is possible to make the partial LU–factorization of level 1 symmetric by modifying the factorization of Q_N. The ordering of the band matrix A_N of Q_N is reversed by a symmetric row column reordering. The column reordering also includes B_{N-1} and the row reordering includes C_N. After this reordering, the partial factorization of Q_N leads to a matrix $\bar{Q}_N$ with a structure identical to $\bar{Q}_1$ shown in Fig. 6.2.

The reordering of Q_N results in a considerable saving in the operations count of the partial LU–factorization. The large differences in operations count in partially factoring Q_1 and Q_N on one side and $Q_3, Q_5, .., Q_{N-2}$ on the other side are exploited in the load distribution algorithm.

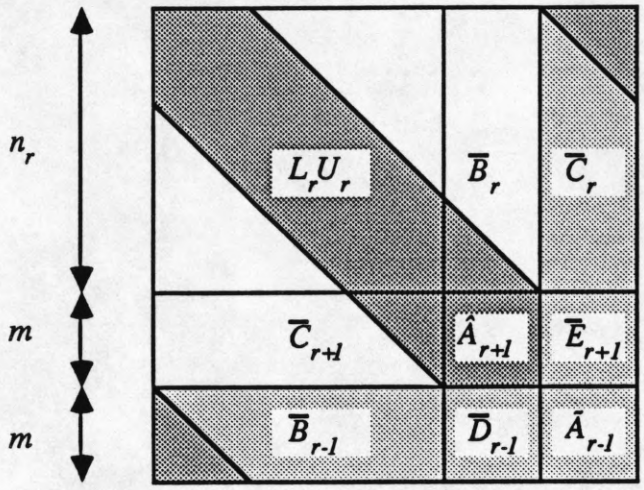

Figure 6.1. The structure of $\overline{Q}_r$ for r=3,5,...,N-2

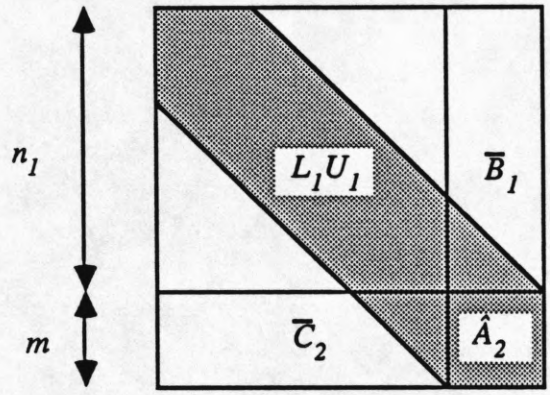

Figure 6.2. The structure of $\overline{Q}_1$

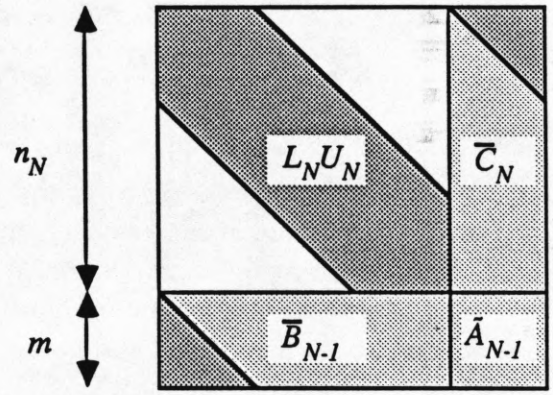

Figure 6.3. The structure of $\overline{Q}_N$

The Q–matrices of the lower levels are all block matrices with full m×m blocks. The extreme matrices, corresponding to Q_1 and Q_N, are 2×2 block matrices while the interior matrices have block dimension 3. The matrix at the lowest level $Q_{(N+1)/2}$ is just one block of dimension m.

6.2 *Complexity analysis*

The performance of the multilevel algorithm can be estimated on the basis of complexity analysis. Let F_r denote the number of floating point operations required for the partial factorization of Q_r. We then have the following operations counts.

Level 1

$$F_r(n_r) = n_r m(2m+1) \ , \ r = 1, N. \tag{6.1a}$$

$$F_r(n_r) = 2n_r m(4m+1) \ , \ r = 3,5,..,N-2. \tag{6.1b}$$

Level ℓ $(2 \leq \ell \leq d)$

$$F_{2(\ell-1)} = F_{N+1-2^{\ell-1}} = \frac{14}{3}m^3 - \frac{3}{2}m^2 - \frac{1}{6}m \tag{6.1c}$$

$$F_{2^\ell i+2^{\ell-1}} = \frac{38}{3}m^3 - \frac{5}{2}m^2 - \frac{1}{6}m \tag{6.1d}$$

$$\text{for } i = 1,2,..,\left(p/2^{\ell-1}\right) - 2$$

Level d+1

$$F_{2^d} = \frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m \tag{6.1e}$$

The operations count of a standard band LU–factorization, called F_{BLU} is

$$F_{BLU} = nm(2m+1) - \frac{4}{3}m^3 - \frac{3}{2}m^2 - \frac{1}{6}m \tag{6.2}$$

A rough estimate of the total complexity of the multilevel algorithm is $F_{MLU} \approx 2\ n\ m(4m+1)\ for\ n \gg m\ and\ N \gg 1$. This implies that $F_{MLU}/F_{BLU} \approx 4$ which is the penalty for being able to do the LU–factorization in parallel. According to this, the maximum speed–up from the multilevel algorithm using p processors is expected to be p/4. A more accurate speed–up calculation including the effect of the load balancing will be given in the next section.

The parallel complexity of the multilevel LU–factorization called F_{PLU} can be computed as the sum of the dominating complexities at each level,

$$F_{PLU} = F_{2^d-1}(n_{2^d-1}) + (d-2)F_{2^d-2} + F_{2(d-1)} + F_{2^d}$$

$$= 2n_{2^d-1}m(4m+1) + (d-2)\left(\frac{38}{3}m^3 - \frac{5}{2}m^2 - \frac{1}{6}m\right) + \frac{16}{3}m^3 - 2m^2 - \frac{1}{3}m \tag{6.3}$$

F_{2^d-1} and F_{2^d-2} represent complexities at top and intermediate levels, respectively. The expression in (6.3) is correct under certain assumptions about the partitioning, e.g. $n_1 = n_3 = ..n_N$ or the partitioning defined by the load balance relations (6.10). F_{PLU} is valid for $d \geq 2$, and using $p = 2^d$ processors, F_{PLU} gives the execution time for an LU–factorization when multiplied by the average floating point excution time called T_F.

The communication time model is based on the model [9]

$$t = T_0 + M/B \tag{6.4}$$

where the latency $T_0 = 1.2msec$ and bandwidth B=1 MByte/sec are measured values for neighbor to neighbor communication on the Intel iPSC. The model in (6.4) is valid for messages of length M bytes fulfilling $0 \leq$ M ≤ 1024 bytes.

The multilevel LU–factorization algorithm involves the communication of 2m×2m matrices from one level to a lower, except to the last level where only an m×m matrix is

transferred. The 2m×2m matrices of double precision numbers fit into 1KByte for m≤5. Under this assumption, the communication model corresponding to (6.3) is

$$T_{CLU} = (d-1)\left(T_0 + 32m^2/B\right) + T_0 + 8m^2/B \tag{6.5}$$

Table 6.1 shows $T_{CLU}/\left(T_F\, F_{PLU}\right)$ for the minimum dimension problem (F_{PLU} is computed from (6.3) with n_{2^d-1} = m). $T_F = 50\mu$sec is an approximate value of the gross floating point execution time measured for the multilevel algorithm. Communication is $O(m^2)$ while computation is $O(m^3)$.

d\m	1	2	3	4	5	6
2	3.75	0.48	0.153	0.070	0.041	0.034
3	3.68	0.39	0.125	0.058	0.033	0.030
4	3.63	0.36	0.115	0.054	0.031	0.029
5	3.61	0.34	0.109	0.051	0.030	0.028

Table 6.1 Communication to computation ratio, $T_{CLU}/(T_F F_{PLU})$, for the multilevel LU–factorization algorithm applied to the minimum dimension problem.

This is clearly reflected by Table 6.1 which shows that communication dominates for m=1 but loses significance very quickly for increasing values of m. The model (6.4) and therefore also (6.5) is only valid for m≤5. The model was extended also to include m=6 which is shown in Table 6.1. This demonstrates that it is sufficient to take the communication cost into account for m≤5.

The communication to computation ratios of Table 6.1 can be considered worst–case values for the multilevel algorithm. The operations count F_{PLU} in (6.3) is $O(n_{2^d-1}\, m^2)$ while T_{CLU} in (6.5) is $O(m^2)$. This means that communication becomes negligible for $n_r \gg m$, r=1, 3,.., N, even for m=1.

The speed–up of the parallel multilevel LU–factorization algorithm is defined as the execution time of a standard band LU–factorization on a single processor divided by the execution time of the parallel algorithm executed on p processors.

The speed–up is computed as $S_{LU} = T_F F_{BLU}/\left(T_F F_{PLU} + T_{CLU}\right)$ where the complexities are given in (6.2), (6.3) and (6.5). Table 7.1 shows speed–up values computed for the minimum dimension problem ($n_{2^d-1} = m$) for selected values of m and d. Theoretical values of S_{LU} are given in parantheses for comparison with experimentally determined values.

The multilevel solution algorithm consists of a forward sweep (levels 1 to d+1) and a backward sweep (levels d+1 to 1), cf. Fig. 5.2. Analogously to the LU–factorization, only the computations of level 1 depend on the total dimension n of the problem while the lower levels only depend on m.

The complexity of the solution algorithm is stated separately for the forward and the backward sweep. Let $\bar{F}_r$ denote the complexity of the computational step during the forward sweep involving the partially factored matrix $\bar{Q}_r$, e.g. the complexity of

$$y_r = L_r^{-1}\, b_r,\ \hat{b}_{r+1} = b_{r+1} - \bar{C}_{r+1}\, y_r\ ,\ \tilde{b}_{r-1} = -\bar{B}_{r-1}\, y_r$$

Similarly $\tilde{F}_r$ denotes the complexity of a computational step involving $\bar{Q}_r$ during the backward sweep. We then have the following complexities:

Level 1

$$\bar{F}_1\,(n_1) = 2n_1 m\ ,\ \bar{F}_N\,(n_N) = 2n_N m - m \tag{6.6a, b}$$

$$\bar{F}_r\,(n_r) = 4n_r m - m\ ,\ r = 3, 5, .., N-2. \tag{6.6c}$$

$$\tilde{F}_1\,(n_1) = n_1\,(2m+1)\ ,\ \tilde{F}_N\,(n_N) = n_N\ (2m+1) \tag{6.6d, e}$$

$$\tilde{F}_r\,(n_r) = n_r\,(4m+1)\ ,\ r = 3, 5, .., N-2. \tag{6.6f}$$

Level ℓ $(2 \leq \ell \leq d)$

$$\bar{F}_{2^{(\ell-1)}} = \bar{F}_{N+1-2^{(\ell-1)}} = \tilde{F}_{2^{(\ell-1)}} = \tilde{F}_{N+1-2^{(\ell-1)}} = 3m^2 \tag{6.6g}$$

$$\bar{F}_{2^\ell i+2^{(\ell-1)}} = 5m^2 - m,\ \tilde{F}_{2^\ell i+2^{(\ell-1)}} = 5m^2\ for\ i = 1, 2, ..., \left(p/2^{(\ell-1)}\right) - 2 \tag{6.6h, i}$$

Level d+1

$$\bar{F}_{(N+1)/2} = \tilde{F}_{(N+1)/2} = m^2 \tag{6.6j}$$

The parallel complexity of the solution algorithm F_{PS} can now be computed as the sum of the dominating complexities at each level. The expression (6.7) is valid under the same assumptions as (6.3).

$$F_{PS} = (d-2)\left(10m^2 - m\right) + n_{2^d-1}\,(8m+1) + 8m^2 - m \tag{6.7}$$

The communication model for the multilevel solution algorithm is

$$T_{CS}\,(m) = 2\left[(d-1)\,(T_0 + 16m/B) + T_0 + 8m/B\right] \tag{6.8}$$

The parameters T_0 and B are given in connection with (6.5), and (6.8) is valid for m$\leq$64

d\m	1	2	5	10	15	20
2	5.36	1.43	0.24	0.062	0.028	0.016
3	5.36	1.38	0.23	0.058	0.027	0.015
4	5.36	1.35	0.22	0.056	0.026	0.015
5	5.36	1.34	0.22	0.055	0.025	0.015

Table 6.2. Communication to computation ratio, $T_{CS}/\,(T_F F_{PS})$, for the multilevel solution algorithm applied to minimum dimension problems.

Table 6.2 shows how communication affects the efficiency of the multilevel solution algorithm applied to minimum dimension problems, i.e. the computational complexity F_{PS} is computed from (6.7) for $n_{2^d-1} = m$. Comparing with Table 6.1, it is seen that the solution algorithm is communication bounded to a larger extent than the LU–factorization algorithm.

Communication cost only depends on m and d (6.8) while the computational complexity is $O(n_{2^d-1}m)$. This means that Table 6.3 shows a worst–case situation, and that communication cost even for m=1 becomes negligible for $n_{2^d-1}/m \gg 1$.

The operations count of a standard forward–backward solution based on an LU–factored band matrix is

$$F_{BS} = 4nm + n - 2m^2 - 2m \tag{6.9}$$

The speed–up of the multilevel solution algorithm can now be computed for the minimum dimension problem as $S_S = T_F F_{BS} / (T_F F_{PS} + T_{CS})$ where the complexities are given in (6.7) (for $n_{2^d-1} = m$), (6.8) and (6.9). Table 7.3 in section 7 below shows speed-up values computed from S_S for selected values of m and d. The speed–up values estimated by S_S are given in parantheses for comparison with experimentally determined values.

6.3 *Load balancing*

From (6.1 a,b) it is clear that a uniform partitioning of the band matrix, $n_1 = n_3 = .. = n_N$, will lead to poor load balance for the LU–factorization. A first attempt to improve the load balance would be to choose $n_3 = n_5 = .. = n_{N-2}$ and $n_1 = n_N = 4n_3\frac{4m+1}{4m+2}$. This choice results in $F_1 = F_3 = .. = F_N$ and thus load balance in level 1. However, some imbalance still remains in the lower levels as specified by (6.1 c,d).

The lower levels could be load balanced after the same principle followed in level 1 by chosing some of $n_2, n_4, .., n_{N-1}$ greater than m, but this is undesirable since it would increase the computational load of the lower levels where parallelism is harder to exploit.

The load balancing scheme chosen for the band matrix can be explained by referring to the complexity relations (6.1) and to Fig. 5.1. The factorization of Q_1 and Q_3 finish at the same time when $n_1 = 4n_3\frac{4m+1}{4m+2}$ permitting processor P_0 to start the factorization of Q_2 without idle time. Likewise, the factorization of Q_5 and Q_7 complete at the same time when $n_5 = n_7$. The ratio n_5/n_3 is now adjusted such that the factorization of Q_2 and Q_6 finish at the same time. This would complete the load balancing of Fig. 5.1 since the matrix partitioning is symmetric $n_1 = n_N, n_3 = n_{N-2}, \ldots$

The balancing principle is easily generalized and (6.10) gives a general set of balance equations based on (6.1). For a given set of values n, m, and d, the load balance equations determine $n_1, n_3, .., n_N$. The equations are easily solved by expressing $n_3, n_7, ..$ by n_1 and substituting these expressions into the last equation and solving it for n_1.

$$n_r = n_{N+1-r}, \quad r = 1, 3, .., (N-1)/2 \quad (symmetry) \tag{6.10a}$$

$$n_5 = n_7 \tag{6.10b}$$

$$n_9 = n_{11} = n_{13} = n_{15} \tag{6.10c}$$

...

$$n_{2^{(d-1)}+1} = n_{2^{(d-1)}+3} = .. = n_{2^d-1} \tag{6.10d}$$

$$F_1(n_1) = F_3(n_3) \tag{6.10e}$$

$$F_1(n_1) + F_2 = F_7(n_7) + F_6 \tag{6.10f}$$

$$F_1(n_1) + 2F_2 = C_{15}(n_{15}) + 2F_{14} \tag{6.10g}$$

...

$$F_1(n_1) + (d-2)F_2 = F_{2^d-1}(n_{2^d-1}) + (d-2)F_{2^d-2} \tag{6.10h}$$

$$2\left(n_1 + n_3 + 2n_7 + 4n_{15}.. + 2^{d-2}n_{2^d-1}\right) + \left(2^d - 1\right)m = n \tag{6.10i}$$

The solution of (6.10) will in general not lead to integer values of $n_1, n_3, ..$, but the resulting values are rounded to satisfy the last equation which states that the total number of equations is n. When n is too small, an effective load balance is not obtained. The balance equations (6.10) yield n_r–values smaller than m which is not permitted by the present implementation. This situation is dealt with by increasing the n_r–values smaller than m to become m, and by reducing the n_r–values greater than m correspondingly.

Communication cost has not been included into the load balance equations since communication cost is such a small fraction that it can only be taken properly into consideration for m$\leq$2. This can be seen from the following argument.

Equation (6.10e) is modified to include communication cost:

$$F_1(n_1) = F_3(n_3) + \left(T_0 + 32m^2/B\right)/T_F$$

Since $F_3(n_3) = 2n_3m(4m+1)$, communication is only going to affect n_3 after rounding if

$$2\left(T_0 + 32m^2/B\right)/T_F \geq 2m(4m+1)$$

The break–even value for m is m=2.52 which means that communication cost is too small in a relative sense to be included in the load balance if $m > 2$. This is in good agreement with Table 6.1.

It is obvious from (6.1) that the execution time for the multilevel LU–factorization is proportional to n when $n_1 = n_3 = \ldots = n_N$. When the partitioning is based on the load balance equations (6.10), the complexity F_{PLU} defined in (6.3) and thus the execution time is still proportional to n since n_{2^d-1} in (6.3) depends linearly on n through the load balance equations (6.10). The derivative obtained by solving (6.10) is:

$$\partial n_{2^d-1}/\partial n = \left[4(4m+1)/(2m+1) + 2 + 4 + 8 + \ldots + 2^{d-1}\right]^{-1} \tag{6.11}$$

The multilevel solution algorithm can be load balanced using the equations (6.10) where $\bar{F}_r$ functions from (6.6) are substituted for F_r. This entails some approximations besides ignoring communication cost. First, (6.10) has $n_1 = n_N$ while $\bar{F}_1(n_1) \neq \bar{F}_N(n_1)$;

and secondly, $\bar{F}_r \neq \tilde{F}_r$ which means that balance is only obtained for the forward sweep. The discrepancies, however, are either O(m) or O(n_r) and do not lead to serious imbalance.

A crude approximation to the solution algorithm load balance can be based on the equation $\bar{F}_1(n_1) = \bar{F}_3(n_3)$, $n_1 = n_N$ and $n_3 = n_5 = .. = n_{N-2}$. This leads to $n_1 \approx 2n_3$ which should be compared with the corresponding relation $n_1 \approx 4n_3$ for the LU–factorization. This implies that optimum load balance requires different partitionings of the band matrix for LU–factorization and solution. Since the solution must be based on the result of a factorization and the load distribution chosen for the factorization, load distribution should in practice be chosen to minimize total execution time of LU–factorization and solution(s).

The minimization of the execution time of a multilevel LU–factorization followed by a sequence of solution steps (e.g. for pseudo–Newton iteration) is a complicated problem and will not be addressed. If only one solution per LU–factorization is needed, the forward sweep of the solution algorithm can be merged with LU–factorization leading to a multilevel Gaussian elimination, and this part can easily be load balanced by solving (6.10) with $F_r + \bar{F}_r$ substituted for F_r. The backward sweep will not be well balanced, but execution time can only be reduced by a different load distribution for a very large number of processors and very small value of m.

7 Performance of the parallel band matrix solver

7.1 *Execution time model*

Figure 7.1 shows an execution time model for sequential band LU–factorization (marked T_{BLU}). Execution time for the sequential algorithm is proportional to F_{BLU} defined in (6.2).

The execution time graph for the parallel multilevel algorithm has more complicated features. For a given number of processors, p=(N+1)/2, the smallest problem that can be solved has dimension n=m N. Therefore the execution time graph starts at n=m N.

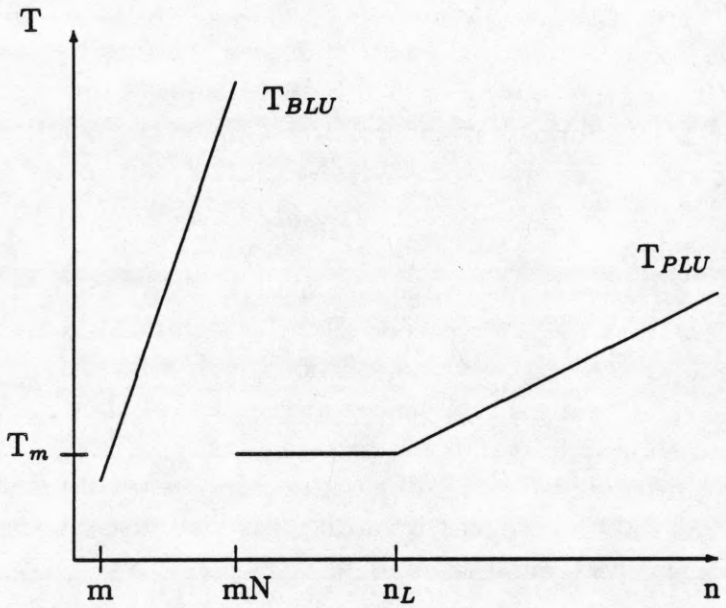

Figure 7.1.: Executing time T as a function of problem dimension n for a given set of m and N($=2^{d+1}$-1).

The load balance equations (6.10) lead to the following relations:

$$n_1 > n_3 > n_7 > n_{15} > \ldots > n_{2^d-1}$$

This means that the smallest value of n for which load balance is effective (n=n_L) is defined by n_{2^d-1}=m. This value is substituted into (6.10h) from which n_1 can be computed. The remaining n_r–values can now be computed from equations (6.10e,f,g,..) and n_L is computed from (6.10i) as

$$n_L = 2\left(n_1 + n_3 + 2n_7 + 4n_{15} + .. + 2^{d-2}m\right) + \left(2^d - 1\right)m$$

Execution time is constant ($T_{PLU} = T_m$) for m N $\leq$ n $\leq n_L$.

For n $> n_L$, the execution time increases linearly with n, according to (6.3) and (6.11). The speed–up of the parallel multilevel algorithm over the sequential algorithm for a given value of n is derived from Fig. 7.1 as $S_{LU} = T_{BLU}/T_{PLU}$. The speed–up is a nonlinear function of n.

Two speed–up values are of particular interest, namely the speed–up of the minimum dimension problem,

$$S_{LU}^{min} = T_{BLU}/T_{PLU}|_{n=mN} \tag{7.1}$$

and the maximum asymptotic speed–up computed for $n > n_L$:

$$S_{LU}^{\infty} = (\partial T_{BLU}/\partial n) \,/\, (\partial T_{PLU}/\partial n) \tag{7.2}$$

The speed–up of the minimum dimension problem S_{LU}^{min} is a worst–case value. It is characterized by only two parameters, m and N, and it does not involve load balancing.

The maximum asymptotic speed-up cannot be attained since

$$S_{LU}^{\infty} = \lim_{n \to \infty} S_{LU}$$

and it corresponds to the speed–up obtained by neglecting the computation involved in the lower levels of the multilevel algorithm. For a given value of n, the speed–up, S_{LU}, is bounded as follows:

$$S_{LU}^{min} \leq S_{LU} < S_{LU}^{\infty}$$

The discussion so far has only been concerned with LU–factorization. The solution algorithms have the same qualitative properties and S_S^{min} and S_S^{∞} are defined analogously to S_{LU}^{min} and S_{LU}^{∞} in (7.1) and (7.2).

7.2 *Numerical results*

Figure 7.2 shows an example of the problems which were solved by the parallel multilevel LU–factorization in order to verify the properties of the algorithm experimentally. The graphs of Fig. 7.2 are of the types presented in Fig. 7.1. The dots indicate measured values. All results in this section are based on measurements reported in [11].

An interesting feature of Fig. 7.2 is that the graphs intersect. This has as a consequence that certain problems are solved more efficiently by fewer processors than the maximum number. The phenomenon originates from the load balance algorithm and from the fact that a doubling of the number of processors increases the depth of the multilevel algorithm by one.

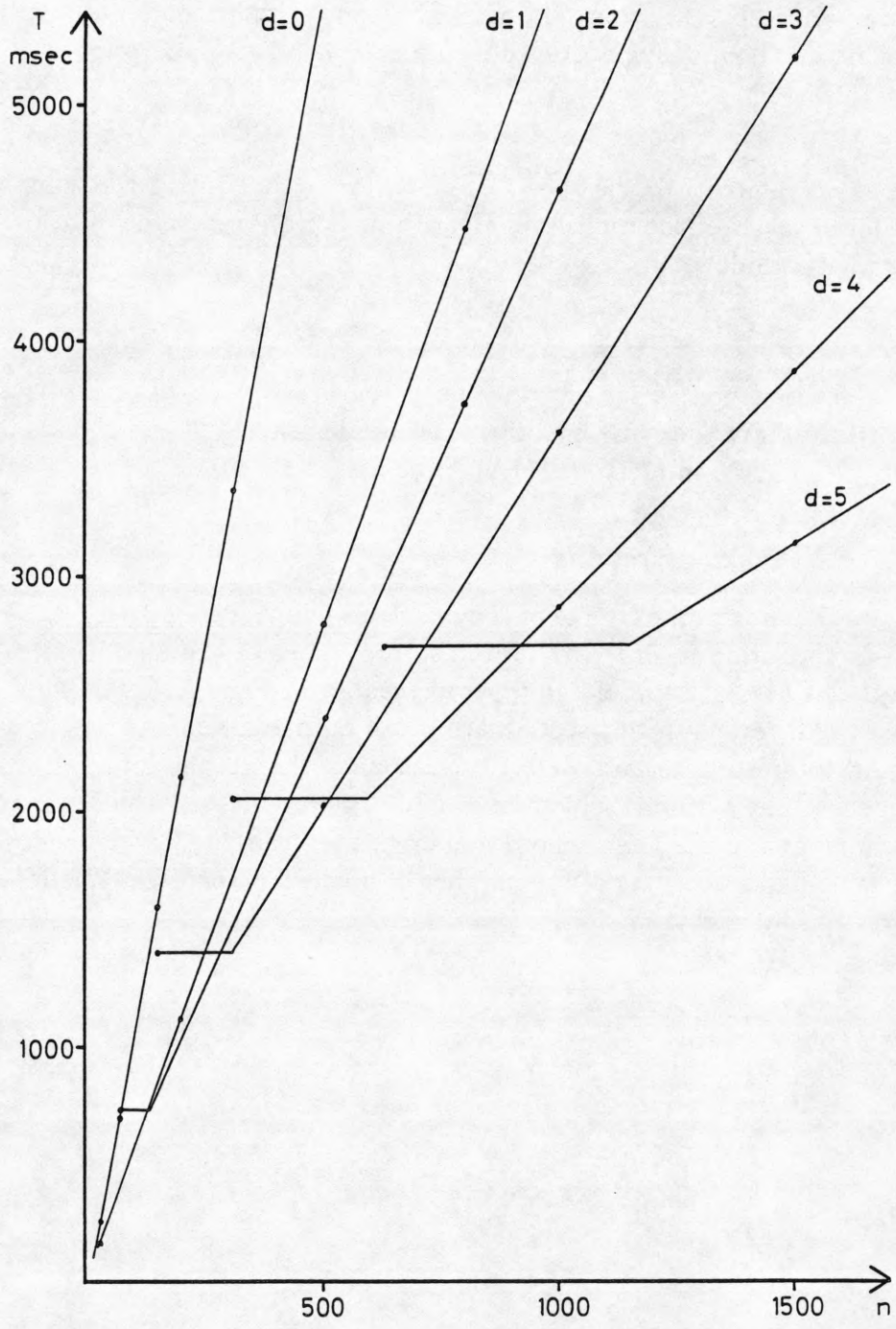

Figure 7.2.: Execution time versus problem size for sequential (d=0) and multilevel parallel LU–factorization (d≥1). Bandwidth w=21 corresponding to m=10.

m\p	2		4		8		16		32	
2	-	(1.06)	0.7	(0.78)	0.8	(1.03)	1.14	(1.54)	2.13	(2.44)
5	1.4	(1.56)	0.95	(0.99)	1.10	(1.19)	1.60	(1.72)	2.57	(2.67)
10	1.5	(1.69)	0.96	(0.99)	1.14	(1.17)	1.64	(1.67)	2.57	(2.59)
15	1.6	(1.71)	0.96	(0.95)	1.14	(1.10)	1.65	(1.57)	2.56	(2.43)
20	1.6	(1.72)	0.97	(0.95)	1.15	(1.10)	1.65	(1.57)	2.53	(2.43)

Table 7.1. Measured and predicted (in paranthesis) values of speed–up of the multilevel parallel LU–factorization over a sequential band factorization. The speed–up, S_{LU}^{min}, applies to the minimum dimension problem, N=2p-1 and n=m N.

Table 7.1 shows measured and predicted (in parantheses) values of speed-up for the parallel multilevel LU–factorization algorithm applied to the minimum dimension problem. All execution times are measured with a resolution of 5msec. This means that execution time for m=2 and p=2 cannot be measured (2-3msec) and that execution time for m=5 and p=2 is not very accurate (20–25msec). The measurements only include LU–factorization and corresponding communication for the parallel algorithm. Downloading of programs, set–up of problem etc. are excluded from the execution time measurements.

The predicted speed–up values are computed as explained in Section 6.1. The expression for F_{PLU} given in (6.3) does not include p=2 (d=1). A special formula, which is easily derived, was used for this column in Table 7.1.

There is good agreement between measured and predicted speed–up values except for m=2 where overhead like procedure calls and initialization leads to smaller speed-up than expected from the model which only includes floating point operations.

For m=15 and m=20 the observed speed–up is slightly greater than the predicted speed–up for p$\geq$4. This phenomenon could be explained by the fact that the block structure of the Q–matrices in the multilevel algorithm leads to the equivalent of unrolling of the loops of the factorization algorithm. The sequential band matrix factorization is programmed in a straightforward style.

Table 7.2 shows measured and predicted (in paranthesis) values of the maximum asymptotic speed-up for the parallel multilevel LU–factorization. Since these speed–up values correspond to neglecting the computational expense of the lower levels, communication is also neglected in the model. The predicted speed–up values are computed from (6.2), (6.3), (6.11) and (7.2). There is very close agreement between measured and predicted values since the top level of the multilevel algorithm is programmed very similarly to the sequential band matrix factorization. This means that the measured speed–up is very close to the ratio of floating point operations counts.

The maximum asymptotic speed–up of the parallel LU–factorization using p processors and load balancing has the following limit which is easily derived from (6.1a,b) and (6.2):

$$S_{LU}^{\infty} \rightarrow (p+6)/4 \quad for \quad m \rightarrow \infty \tag{7.3}$$

Put into words, the performance of the multilevel LU–factorization algorithm with p processor and load balancing according to (6.10) is identical to the performance with p+6 processors and no load balancing ($n_1 = n_3 = \ldots = n_N$).

m\p	2		4		8		16		32	
2	1.99	(2)	2.55	(2.56)	3.66	(3.67)	5.90	(5.89)	10.4	(10.3)
5	1.99	(2)	2.51	(2.52)	3.56	(3.57)	5.65	(5.67)	9.84	(9.86)
10	1.99	(2)	2.50	(2.51)	3.52	(3.54)	5.57	(5.59)	9.69	(9.68)
15	1.99	(2)	2.50	(2.51)	3.51	(3.52)	5.50	(5.56)	9.66	(9.62)
20	1.99	(2)	2.49	(2.51)	3.49	(3.52)	5.48	(5.54)	9.72	(9.59)

Table 7.2. Measured and predicted (in paranthesis) values of maximum asymptotic speed–up, S_{LU}^{∞}, of the multilevel parallel LU–factorization over a sequential band factorization.

m\p	2		4		8		16		32	
2	-	(0.51)	1.3	(0.73)	1.8	(1.06)	2.0	(1.66)	2.7	(2.69)
5	1.5	(1.2)	1.25	(1.36)	2.0	(1.91)	3.21	(2.94)	5.11	(4.73)
10	1.5	(1.5)	1.67	(1.56)	2.41	(2.16)	3.88	(3.30)	6.17	(5.31)
15	1.45	(1.58)	1.77	(1.60)	2.58	(2.21)	4.07	(3.37)	6.60	(5.41)
20	1.59	(1.61)	1.78	(1.61)	2.65	(2.23)	4.11	(3.39)	6.67	(5.44)

Table 7.3. Measured and predicted (in paranthesis) values of speed–up of the multilevel parallel solution algorithm over a sequential forward–backward band substitution algorithm. The speed–up S_S^{min} applies to the minimum dimension problem, N=2p-1 and n=mN.

In Table 7.3 the speed–up values of the parallel solution algorithm are given for the minimum dimension problem. The solution algorithm examplified by Fig. 5.2 is applied to a multilevel LU–factorization produced by the parallel multilevel LU–factorization algorithm.

The measured speed–up values in Table 7.3 for m=2 and m=5 with $p \leq 8$ are rather inaccurate because of the resolution of 5 msec in the execution time measurements.

Disregarding the inaccurate speed–up measurements, the observed speed–up is consistently greater than the predicted speed–up for p>2. This somewhat surprising result was traced to an inadvertent exploitation of processor parallelism at the lower levels of the parallel solution algorithm. Floating point computation on the 80287 processor and index computation on the 80286 processor were to a certain degree overlapped in the parallel algorithm and not in the sequential solution algorithm.

There was no attempt to optimize either the sequential or the parallel implementation, and very careful optimization could probably improve the speed by 25%–50%. However communication would still be insignificant for problems large enough to justify the use of a parallel computer. The experimental implementation therefore fulfils its purpose of demonstrating the feasibility of the multilevel algorithm.

The predicted asymptotic speed–up S_{LU}^{∞} involves the computation of $\partial T_{PLU}/\partial n$ which is computed as

$$\partial T_{PLU}/\partial n = (\partial F_{2^d-1}/\partial n_{2^d-1})\,(\partial n_{2^d-1}/\partial n)\,T_F$$

Because of the nature of the load balance algorithm, we have the following result for n> n_L :

$$\partial T_{PLU}/\partial n = (\partial F_r/\partial n_r)\,(\partial n_r/\partial n)\,T_F \tag{7.4}$$

for r= 1, 3, .., N.

S_{LU}^{∞} can now be expressed as

$$S_{LU}^{\infty} = (\partial T_{BLU}/\partial n) \,/\, [(\partial F_1/\partial n_1)\,(\partial n_1/\partial n)\,T_F] = (\partial n_1/\partial n)^{-1}$$

The predicted asymptotic speed–up for the multilevel solution algorithm with load distribution for optimum *LU–factorization* speed can be expressed as

$$S_S^{\infty} = (\partial T_{BS}/\partial n) \,/\, \left[\left(\partial\left(\bar{F}_1 + \tilde{F}_1\right)/\partial n_1\right)(\partial n_1/\partial n)\,T_F\right] = (\partial n_1/\partial n)^{-1}$$

which is equal to S_{LU}^{∞}.

Since load balance is not for optimum solution algorithm speed, a relation similar to (7.4) is not valid. The partitioning n_1 is approximately twice as large as the optimum value. Therefore we have $\partial F_{PS}/\partial n = \partial\left(\bar{F}_1 + \tilde{F}_1\right)/\partial n$.

A table analogous to Table 7.2 with measured and predicted values of S_S^{∞} for load distribution for optimum LU–factorization was constructed. As expected, it was essentially identical to Table 7.2, and it was therefore omitted.

Load distribution is chosen to be optimum for LU–factorization since this is close to minimum execution time for one LU–factorization followed by one solution step.

When one LU–factorization is computed followed by a large number of solution steps (e.g. Newton iteration) it may be advantageous to load balance for optimum solution speed. In this case we have:

$$S_S^{\infty} \;\rightarrow\; (p+2)/2 \;\; for \;\; m \rightarrow \infty$$

This speed–up is almost twice as large as the corresponding limit value when load distribution is with respect to LU–factorization as stated in (7.3).

7.3 *Row interleaved factorization and solution*

An alternative parallel LU–factorization and solution approach is the row interleaved algorithm [8]. Contrary to the multilevel algorithm, the row interleaved algorithm does not pay any computational penalty for the parallelization, only communication penalty. Each pivot row must be broadcast to all processors to permit parallel factorization. The execution time model for the row interleaved LU–factorization is as follows:

$$T_{RLU} = n\left[m\,(2m+1)\,T_F/2^d + d\,(T_0 + 8m/B)\right]$$

The row interleaved LU–factorization can be compared with the multilevel LU–factorization by comparing asymptotic speed–ups. For small values of m we have $S_{LU}^{\infty} > S_{RLU}^{\infty}$ where S_{RLU}^{∞} is defined analogously to S_{LU}^{∞} :

$$S^{\infty}_{RLU} = (\partial T_{BLU}/\partial n) \,/\, (\partial T_{RLU}/\partial n)$$

Table 7.4 gives the integer m–values for which the algorithms break even, $S^{\infty}_{LU} \approx S^{\infty}_{RLU}$.

p	4	8	16	32
m	13	15	21	31

Table 7.4. Integer values of m for which row interleaved and multilevel LU–factorizations break even in asymptotic speed–up.

The entries of Table 7.5 are computed from the equation

$$\partial T_{RLU}/\partial n = \partial T_{PLU}/\partial n$$

where

$$\partial T_{PLU}/\partial n \approx 4m\,(2m+1)\,T_F/\left(2^d+6\right)$$

This value is a good approximation assuming load distribution and "large" m.

For m–values larger than those given in Table 7.4, the row interleaved LU–factorization will be superior to the multilevel algorithm. For p=2, the multilevel algorithm is always superior.

The values of table 7.4 were compared with measurements of an implementation of the row interleaved algorithm and measurements of the multilevel algorithm. The measurements resulted in m=14 and m=20 for p=8 and p=16, respectively. This is in good agreement with the predictions of the model.

The row interleaved solution algorithm performs very poorly since the number og broadcasts is the same as for the LU–factorization while computation is O(m) for each broadcast for the solution algorithm compared with $O(m^2)$ for the LU–factorization.

Concluding the comparison of multilevel and row interleaved algorithms, the former is superiour for narrow band problems while the latter takes over for wide band problems. The LU–factorizations break even for the m–values given in Table 7.5. For one LU–factorization and one solution step, the m–values corresponding to break even will increase.

It is obvious that the m–values of Table 7.4 are sensitive to the communication and computation performance of the parallel computer as modeled by T_0, B and T_F. However, there is no trend in parallel computer technology towards a substantial shift of the break–even values of m.

Finally, the multilevel solution method and the implementation techniques on the hypercube discribed in this paper should also be applicable to the other members of the family of permutations for parallel solution of block tridiagonal matrices proposed in [7].

Acknowledgement

The careful implementation of the multilevel algorithms together with the numerical experiments were done by Per Ulkjær Andersen who also contributed to the principles behind load balancing, processor allocation and communication for the multilevel algorithm.

The work of I.N.Hajj was supported in part by the U.S.Joint Services Electronics Program and by Intel Corporation.

References

[1] D. Heller, "Some aspects of the cyclic reduction algorithm for block tridiagonal systems", SIAM J. Numer.Anal., Vol. 13, no. 4, pp. 484–496, 1976.

[2] G. Birkhoff and A. George, "Elimination by nested dissection", in Complexity of Sequential and Numerical Algorithms edited by J. F. Traub, pp. 221–269, Academic Press, 1973.

[3] I. N. Hajj, "Sparsity considerations in network solution by tearing, IEEE Trans. Circuit and Systems, Vol. CAS–27, no. 5, pp. 357–366, 1980.

[4] J. J. Dongarra and A. H. Sameh, "One some parallel banded system solvers", Parallel Computing 1 (1984) 223–235.

[5] S. L. Johnsson, "Solving narrow banded systems on emsemble architectures", ACM Trans. Math. Software, Vol. 11, no. 3, pp. 271–288, 1985.

[6] U. Meier, "A parallel partition method for solving banded systems of linear equations", Parallel Computing 2 (1985) 33–43.

[7] S. Utku, M. Salama and R. J. Melosh, "A family of permutations for concurrent factorization of block tridiagonal matrices", IEEE Trans. on Computers, Vol. 38, No. 6, pp. 812–824, June 1989.

[8] Y. Saad and H. M. Schultz, "Parallel direct methods for solving banded linear systems", Linear Algebra and its Applications, Vol. 88/89, pp. 623–650, 1987.

[9] D. K. Bradley, "First and second generation hypercube performance", Report No. UIUCDCS–R–88–1455, Dept. Computer Science, University of Illinois, Urbana Champaign, USA, 1988.

[10] I. N. Hajj and S. Skelboe, "Multilevel parallel solver for banded linear systems", in Aspects of Computation on Asynchronous Parallel Processors, edited by M. H. Wright, IFIP 1989, pp. 69–78.

[11] P. Ulfkjær Andersen, "Implementation of a two–level a multilevel and a block parallel multilevel parallel solver for banded linear systems", Report, Department of Computer Science, University of Copenhagen, 1988.