University of Leeds

SCHOOL OF COMPUTER STUDIES

RESEARCH REPORT SERIES

Report 93.25

A Two-Way Parallel Partition Method for Solving Tridiagonal Systems

by

C Walshaw & S J Farr

Division of Computer Science

June 1993

Abstract

The Parallel Partition Method for tridiagonal systems is described. It is noted that, in the local reduction phase, the inherent parallelism is not exploited to the full and so a Two-Way Parallel Partition Method is introduced. This new algorithm results in a reduced system of order $P/2-1$ compared to $P-1$ previously and in particular for 4 processors, a much lower arithmetic count. Both versions are tested and the results compared.

1 Introduction

1.1 The Parallel Solution of Tridiagonal Systems

Many nearest neighbour problems (for example those involving spatial finite difference approximations) have at their heart a tridiagonal matrix. Their sparsity pattern suggests that, for large values of n, such systems are ideal candidates for parallelisation. However, the common methods for solving tridiagonal systems, such as Gaussian elimination or matrix decomposition, tend to be inherently sequential in nature and as a result, this topic was one of the earliest subjects investigated in the field of parallel linear algebra. Amongst the first such schemes were those introduced by Stone in the early seventies, [10] & [11] employing a recursive doubling algorithm. Another method, odd-even cyclic reduction, was developed by Golub and stabilised by Buneman, [2] & [3], and since its first introduction for symmetric constant coefficient matrices it has been extended to general non-symmetric tridiagonal systems, [12].

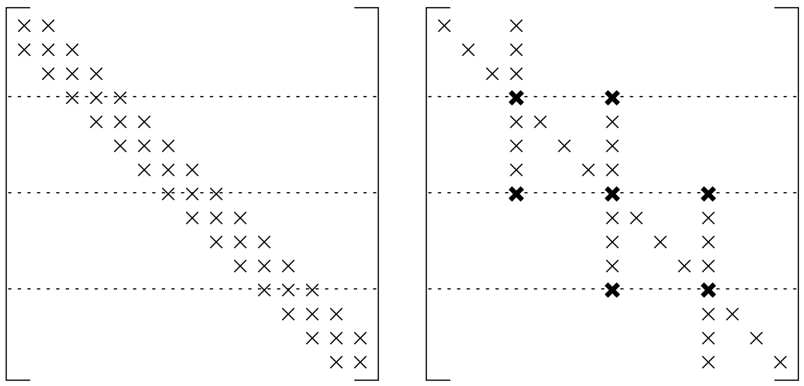

Figure 1: Matrix transformation for the partition algorithm

In 1981 Wang, [17], introduced what he called a new (partition) method. This has also been referred to as the spike algorithm as it proceeds by a completely parallel local Gaussian elimination to transform the system from a tridiagonal one into a diagonal one with spikes or fill-ins at the inter-processor boundaries. With one communication at the end of this local reduction phase, the result is a global order $P-1$ tridiagonal system in terms of the boundary variables (the unknowns at the inter-processor boundaries). Figure 1 shows an example matrix before and after the elimination phase where the dotted lines represent the inter-processor boundaries and the bold $\times$ symbols are the coefficients of the $O(P-1)$ system. This *reduced system* can then be solved globally with, for example, cyclic reduction or two-way Gaussian elimination and finally the internal values calculated each as a linear combination of up to two boundary variables.

A variant, the method of Sameh et al., [9] & [6], uses no communication in the reduction phase but results in a pentadiagonal matrix. However, in a generalisation to narrow banded systems, Johnsson has shown, [4], that the extra communication is valuable and preserves positive definiteness and a form of diagonal dominance (albeit not in the classical sense – see [16] for further discussion).

The majority of these partition algorithms have a common structure and can be split into three distinct phases which shall henceforth be referred to as:–

1. The reduction phase – variables are eliminated to reduce the system to $O(P)$, with an associated matrix usually known as the reduced matrix.
2. The reduced system is solved.
3. The back-substitution phase – the full solution is constructed from the variables of the reduced system.

Of these, phases 1 and 3 can be carried out almost completely in parallel with no inter-processor communication. Phase 2 necessitates solving a system with one unknown per processor and is inherently a sequential operation. Since different algorithms can be used for phases 1 and 2 (phase 3 being determined by these) such schemes are best described as $[L][G]$ algorithms where $[L]$ denotes the *local* algorithm of phase 1 and $[G]$ the *global* phase 2.

The competing methods and their implementation on different architectures have all been comprehensively reviewed by Johnsson in [5]. He shows that a hybrid $GECR$ (Gaussian Elimination locally, Cyclic Reduction globally) algorithm has similar arithmetic complexity to the full cyclic reduction algorithm and that the best method therefore depends on communication considerations.

This report gives a description of the partition method and then goes on to develop a new and more efficient version. Operation counts are established for both to demonstrate the efficiency and finally a comparison of timings for the two methods is given.

1.2 The Algorithms

Three algorithms are presented in this paper, a two-way matrix factorisation algorithm, the Partition Algorithm, and the new Two-Way Partition Algorithm (sections 2-4 respectively). Each will be derived for the solution of the order n tridiagonal system

$$A\mathbf{x} \equiv \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & & & & \\ & \ddots & \ddots & \ddots & \\ & & & & c_{n-1} \\ & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \\ \vdots \\ \\ d_n \end{bmatrix} \equiv \mathbf{d}. \tag{1}$$

For the purposes of this paper it is assumed that A is diagonally dominant and hence that the LU matrix decomposition algorithm can be used to factorise any submatrix of A. The techniques presented should, however, work for A symmetric & positive definite and a Cholesky type decomposition. Either matrix type is very important; for more general matrices a pivoting strategy may be necessary and on a distributed memory network this can add such a large communication overhead as to completely kill any speed-up achieved by parallel solution.

For the global solve of the reduced matrix in phase 2 of the partition algorithm several techniques, such as two-way Gaussian elimination, cyclic reduction and recursive doubling, are possible. However, the characteristics of the machine used for testing (see section 5.1.1 and [15, section 3.1.2] & [5, theorem 5.2]), together with the possibilities for reuse of code suggested the use of two-way matrix factorisation as described in section 2. The operation counts are based on this choice.

Throughout, the assumption is made that the n equations are distributed on a chain of P processors. Secondly it is assumed that $n \gg P$ so that each node has $m = n/P$ unknowns assigned to it. Superscripts are used to denote the processor a particular value is stored on and subscripts the position in a vector.

1.3 Operation Counts

For each algorithm an operation count is given and follows the generally accepted approximations for making theoretical timing estimations, see for example [5]. The machine dependent parameters are defined as:–

$t_a \overset{def}{=}$ time for one floating point operation; $+$, $-$, $\times$ or $\div$

$s_c \overset{def}{=}$ start-up time for each inter-processor communication

$t_c \overset{def}{=}$ time to transmit one number.

For matrices with multiple right hand sides, or which remain constant throughout many iterations, much work can be saved by generating a matrix decomposition once and storing it. In the following sections the square brackets [&] denote operations which, for constant matrices, need only be carried out once, rather than at every iteration. Alternatively, for multiple right hand sides, the costs that are not in square brackets are those required for each right hand side.

2 A Two-Way Matrix Factorisation Algorithm

In general, the classical algorithms for the solution of tridiagonal systems (such as Gaussian elimination and matrix decomposition methods) employ a forward sweep down through the variables followed by a backward sweep up. Whilst apparently sequential in nature, they actually have an inherent parallel degree of two, [8, page 22]. That is to say, two processes can be applied concurrently with no overlap of work.

As an example, the system could be partitioned into a upper and lower half with each assigned to a processor. Then the upper processor can sweep down through the upper partition whilst the lower processor sweeps up through its partition. After an exchange of information, simultaneous back-substitution on each processor commences outwards from the centre. Alternatively consider a tridiagonal matrix distributed over a chain of processors, with each processor holding one variable. In this case the factorisation can sweep in along the chain from both ends and then out from the centre.

This is not a new idea and, for example, two-sided Gaussian elimination was introduced by Babuska, [1, section 5], and a technique referred to as twisted factorisation described in [14]. However since this method is central to the Two-Way Partition Algorithm described below, full details of the two-way factorisation process are given. Henceforth this algorithm will be known as Two-Way (TW) Decomposition and for convenience, these letters will be also used to refer to the two factorisation matrices.

2.1 The Method

Consider the tridiagonal system (1) and let $q = n/2$. Echoing the LU decomposition algorithm, A will be factorised into the matrix product TW, where

$$T = \left[\begin{array}{cccc|cccc}
u_1 & & & & & & & \\
a_2 & u_2 & & & & & & \\
 & \ddots & \ddots & & & & & \\
 & & a_q & u_q & & & & \\
\hline
 & & & & u_{q+1} & c_{q+1} & & \\
 & & & & & \ddots & \ddots & \\
 & & & & & & u_{n-1} & c_{n-1} \\
 & & & & & & & u_n
\end{array}\right]$$

and

$$W = \left[\begin{array}{cccc|cccc}
1 & v_1 & & & & & & \\
 & \ddots & \ddots & & & & & \\
 & & 1 & v_{q-1} & & & & \\
 & & & 1 & v_q & & & \\
\hline
 & & & v_{q+1} & 1 & & & \\
 & & & & v_{q+2} & 1 & & \\
 & & & & & \ddots & \ddots & \\
 & & & & & & v_n & 1
\end{array}\right].$$

Thus, in the upper partition,

$$\begin{aligned}
u_1 &= b_1, & v_1 &= c_1/u_1, & & \\
u_i &= b_i - a_i v_{i-1}, & v_i &= c_i/u_i, & i &= 2, \ldots, q;
\end{aligned}$$

and similarly in the lower partition,

$$\begin{aligned}
u_n &= b_n, & v_n &= a_n/u_n, & & \\
u_i &= b_i - c_i v_{i+1}, & v_i &= a_i/u_i, & i &= n-1, \ldots, q+1.
\end{aligned}$$

Now let

$$W\mathbf{x} = \mathbf{z}, \tag{2}$$

so that

$$T\mathbf{z} = \mathbf{d}.$$

Then in the upper & lower partitions respectively,

$$\begin{aligned}
z_1 &= d_1/u_1, & z_i &= (d_i - a_i z_{i-1})/u_i, & i &= 2, \ldots, q; \\
z_n &= d_n/u_n, & z_i &= (d_i - c_i z_{i+1})/u_i, & i &= n-1, \ldots, q+1.
\end{aligned}$$

From (2) the two central equations are

$$\begin{array}{rcrcl} x_q & + & v_q x_{q+1} & = & z_q, \\ v_{q+1} x_q & + & x_{q+1} & = & z_{q+1}; \end{array}$$

and to solve these the partitions must exchange information; z_q [and v_q for time dependent matrices] must be passed to the lower partition and z_{q+1} [v_{q+1}] passed to the upper partition. Then x_q, x_{q+1} can be calculated from

$$\begin{array}{rcl} (1 - v_q v_{q+1}) x_q & = & z_q - v_q z_{q+1}, \\ (1 - v_q v_{q+1}) x_{q+1} & = & z_{q+1} - v_{q+1} z_q. \end{array} \qquad (3)$$

Finally, back-substitution gives

$$\begin{array}{rcll} x_i & = & z_i - v_i x_{i+1}, & i = q-1, \ldots, 1; \\ x_i & = & z_i - v_i x_{i-1}, & i = q+2, \ldots, n. \end{array}$$

2.2 Stability of the algorithm

Suppose A is strictly diagonally dominant. Then it is well known that A is non-singular and the matrix can be factorised into an LU product, see for example [18, pages 122-124]. In addition this factorisation is strongly stable. The TW algorithm essentially comprises of two LU decompositions and thus is well defined in this event. The only possible problem arises in the division by $(1 - v_q v_{q+1})$ in equation (3). However, a simple check shows that $|v_i| < 1$ for all i and hence the decomposition is stable. A proof of this property is presented in [15, section 3.3].

2.3 Operation Count

The arithmetic costs of this algorithm are:–

- for the decomposition $[3q - 2]$
- for the inwards sweep $3q - 2$
- for the central calculation $3 + [2]$
- for the outwards sweep $2q - 2$

2.3.1 Two Processors

The communication for a two processor algorithm requires 1 communication start-up (assuming the processors can exchange data simultaneously) and the transmission of $1 + [1]$ numbers.

The total cost for two processors is therefore:–

$$\{5q - 1 + [3q]\}t_a + 1s_c + \{1 + [1]\}t_c. \tag{4}$$

If communication costs are small and n large, this is almost a linear speed-up over $\{5n - 4 + [3n - 3]\}t_a$, the cost of the sequential LU algorithm.

2.3.2 n Processors

If instead there are n processors each assigned to one of the variables then the communication cost is modified slightly. There are $n-1$ communication start-ups; $q-1$ each for both inward and outward sweeps plus one extra for the central exchange. For each communication in the inward sweep and central exchange $1 + [1]$ numbers (z_i & $[v_i]$) are transmitted and in the outward sweep 1 number (x_i) is transmitted.

The total cost for n processors is therefore:–

$$\{5q - 1 + [3q]\}t_a + \{2q - 1\}s_c + \{2q - 1 + [q]\}t_c. \tag{5}$$

3 The Partition Algorithm

Consider the solution of (1) on an array of P processors. The system can be distributed consecutively across the processors by partitioning it into P tridiagonal subsystems $\{S^i : i = 1, \ldots, P\}$ of order m interspersed with $P-1$ boundary equations $\{E^i : i = 1, \ldots, P-1\}$. Thus m is defined by the relation $n = mP + P - 1$ and for simplicity it is assumed that n and P are such that m is an integer.

In this way processor i is assigned the (relabelled) tridiagonal system

$$a_1^i \mathbf{e}_1 x^{i-1} + T^i \mathbf{x}^i + c_m^i \mathbf{e}_m x^i = \mathbf{d}^i \qquad (S^i)$$

where T^i is a submatrix of A, $\mathbf{e}_1$ & $\mathbf{e}_m$ denote the standard unit basis vectors with $[e_j]_k = \delta_{jk}$ and $a_1^1 = c_m^P = 0$. The unknowns x^{i-1} and x^i denote the boundary variables and each of the boundary equations

$$a^i x_m^i + b^i x^i + c^i x_1^{i+1} = d^i \qquad (E^i)$$

lies between the subsystems (S^i) and (S^{i+1}). The optimal assignment of (E^i) to a particular processor (to i or $i+1$ for example) depends on the algorithm used in the phase 2 of the method.

3.1 The Method

As described above the method has three phases: a local reduction phase to eliminate off-diagonals in the subsystems (phase 1), the global solution of the reduced matrix for the boundary variables (phase 2), and (phase 3) the local back-substitution for the subsystem variables.

3.1.1 Phase 1

The (S^i) are rearranged to give $\mathbf{x}^i$ as a linear combination of the two bounding variables x^{i-1} and x^i. Thus each processor 'inverts' the matrix T^i (for example via LU decomposition or Gaussian elimination) and uses it in (S^i) to obtain

$$\mathbf{x}^i = \mathbf{w}^i - \mathbf{r}^i x^{i-1} - \mathbf{s}^i x^i \qquad (S^i)'$$

where

$$\begin{aligned}
\mathbf{w}^i &\overset{def}{=} (T^i)^{-1}\mathbf{d}^i, \\
\mathbf{r}^i &\overset{def}{=} a_1^i (T^i)^{-1}\mathbf{e}_1, \\
\mathbf{s}^i &\overset{def}{=} c_m^i (T^i)^{-1}\mathbf{e}_m.
\end{aligned}$$

The values for x_1^i, x_1^{i+1}, x_m^{i-1} and x_m^i given by $(S^i)'$ can now be substituted into the boundary equations to yield

$$-a^i r_m^i x^{i-1} + (b^i - a^i s_m^i - c^i r_1^{i+1})x^i - c^i s_1^{i+1} x^{i+1} = d^i - a^i w_m^i - c^i w_1^{i+1} \qquad (E^i)'$$

and this can then be rewritten as the reduced $O(P-1)$ tridiagonal system

$$R\mathbf{x} \equiv \begin{bmatrix} \tilde{b}^1 & \tilde{c}^1 & & \\ \tilde{a}^2 & & & \\ & \ddots & \ddots & \ddots & \\ & & & \tilde{c}^{P-2} \\ & & \tilde{a}^{P-1} & \tilde{b}^{P-1} \end{bmatrix} \begin{bmatrix} x^1 \\ \\ \vdots \\ \\ x^{P-1} \end{bmatrix} = \begin{bmatrix} \tilde{d}^1 \\ \\ \vdots \\ \\ \tilde{d}^{P-1} \end{bmatrix}.$$

The coefficients are given by:–

$$\begin{aligned}
\tilde{a}^i &= -a^i r_m^i & i &= 2, \ldots, P-1 \\
\tilde{b}^i &= b^i - a^i s_m^i - c^i r_1^{i+1} & i &= 1, \ldots, P-1 \\
\tilde{c}^i &= -c^i s_1^{i+1} & i &= 1, \ldots, P-2 \\
\tilde{d}^i &= d^i - a^i w_m^i - c^i w_1^{i+1} & i &= 1, \ldots, P-1.
\end{aligned}$$

3.1.2 Phase 2

The second phase is the solution of this reduced system for the boundary variables. Again this can be accomplished in a number of ways, but, for the purposes of both the operation count and the results, the two-way matrix factorisation presented in section 2 is employed.

3.1.3 Phase 3

When each processor has the value of the two boundary variables of its local tridiagonal system it can then substitute them into $(S^i)'$ to construct the solution. Thus the third phase can start locally as soon as the sweep has passed by.

3.2 Operation Count

Assuming LU decomposition is used, the arithmetic costs of phases 1 & 3 are as follows:-

	Calculation	Count	Notes
Phase1	$[L^i, U^i]$	$[3(m-1)]$	
	$\mathbf{w}^i$	$5m-4$	
	$[\mathbf{r}^i]$	$[4m-3]$	$(1,2)$
	$[\mathbf{s}^i]$	$[m-1]$	$(1,2)$
Phase3	$\mathbf{x}^i$	$4m$	(2)

Notes:–

(1) Since $T^i\mathbf{r}^i = a^i_1\mathbf{e}_1$ and $T^i\mathbf{s}^i = c^i_m\mathbf{e}_m$, either the forward or the backward substitution (depending on the implementation of the LU decomposition) involves significantly fewer arithmetic operations than for a full right hand side such as $\mathbf{w}^i$.

(2) Processor 1 need not calculate $\mathbf{r}^1$ and processor P need not calculate $\mathbf{s}^P$, so the cost of phase 3 for these two processors is just $2m$. This may result in a saving for the algorithm since, being at either end of the chain for phase 2, they are the last to receive the boundary variables. However the saving is difficult to generalise as it depends upon the size of m and on the relative costs of communication to arithmetic.

Thus the total cost of vector operations in phases 1 and 3 is

$$\{9m - 4 + [8m - 7]\}t_a. \tag{6}$$

The calculation of the reduced matrix R and right hand side $\tilde{\mathbf{d}}$ requires $4 + [6]$ arithmetic operations per processor plus the transmission of $1 + [2]$ numbers. No communication start-ups are required as they can be included in the first step of phase 2. This adds the cost

$$\{4 + [6]\}t_a + \{1 + [2]\}t_c. \tag{7}$$

For phase 2, it can be seen from (5) that, writing $q = (P-1)/2$, the cost can be expressed as

$$\{5q + 2 + [3q]\}t_a + \{2q\}s_c + \{2q + [q]\}t_c. \tag{8}$$

The slight modification from (5) arises from solving $2q$ equations on $2q-1$ processors.

Thus, from (6)-(8), the complete cost, in terms of m and q, is

$$\{9m + 5q + 2 + [8m + 3q - 1]\}t_a + \{2q - 1\}s_c + \{2q + [q + 2]\}t_c. \tag{9}$$

3.3 Summary

The algorithm given in this section shall henceforth be referred to as *LUTW*; local *LU* decomposition, global *TW* decomposition. It is presented both to demonstrate this common strain of solver and to use as a building block to demonstrate the new version. In terms of n and P, the operation count can be summarised as $P - 1$ communications with $17n/P + 4P$ arithmetic operations for time-dependent matrices and $9n/P + 5P/2$ operations for constant ones.

4 The Two-Way Partition Algorithm

This method combines the algorithms of sections 2 & 3 to obtain a more efficient algorithm for solving a tridiagonal system on an array of P processors.

As a starting point consider the *LUTW* algorithm for $P/2$ processors and note that the local *LU* decomposition of phase 1 does not exploit the fullest parallelism possible. As seen in section 2 matrix factorisation of this type has a parallel degree of 2. Thus it is possible to assign each tridiagonal subsystem (S^i) to *two* processors and employ the *TW* algorithm for the local reduction. As a result a boundary equation is only required at *every other* inter-processor boundary and hence the reduced matrix is of order $P/2 - 1$ rather than $P - 1$!

A number of advantages manifest themselves:–

(i) It is immediate that the operation count in (9) is reduced. If $n \gg P$ and P is small the gain is minimal, however, it can be seen that the algorithm *LUTW* is non optimal.

(ii) If implemented on a chain of processors with additional next-nearest neighbour connections, the number of communication start-ups can be reduced to $P/2$. This is possibly a significant saving.

(iii) Algorithms of the form $LUTW$ have often been demonstrated, see for example [5], [13] or [17], with a diagram showing a 4 processor version similar to figure 1 (page 1). These will usually have 3 (or sometimes 4) spikes or boundary fill-ins. However, using the TW algorithm for the first phase results in only one fill-in – see figure 2. This has a consequence of reducing the operation count for phases 1 & 3 from $17n/P$ down to $14n/P$ [$9n/P$ down to $7n/P$ for constant matrices] and is a significant saving for 3 or 4 processor implementations.

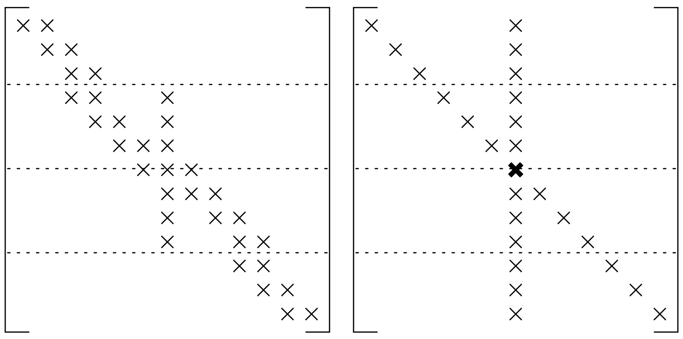

Figure 2: Two-way reduction for phase 1 using 4 processors

It is also perhaps a more natural algorithm than $LUTW$ since its restriction to two processors just results in the TW algorithm with an approximate arithmetic operation count of $4n$, whereas the restriction of $LUTW$ to two processors still results in boundary fill-ins and a count of $7n$. This together with (iii) above makes it especially attractive for 4 or less processors, whilst (i) and in particular (ii) show general improvements.

4.1 The Method

Again consider the solution of the tridiagonal system (1) on an array of P processors. For simplicity assume that P is even and that $q = P/2$. This time the system is distributed by dividing it up into q order $2m$ tridiagonal subsystems $\{S^i : i = 2, 4, 6, \ldots, P\}$ interspersed with $q-1$ boundary equations $\{E^i : i = 2, 4, 6, \ldots, P-2\}$. Here m, the size of the tridiagonal block stored on each processor, is defined by the relation $n = 2mq + q - 1$. The subsystems are each assigned to two processors, so that each pair of processors $i-1$ & i store the system

$$a_1^i \mathbf{e}_1 x^{i-1} + T^i \mathbf{x}^i + c_{2m}^i \mathbf{e}_{2m} x^i = \mathbf{d}^i \qquad (S^i)$$

which is distributed by assigning the first m equations to processor $i-1$ and the remaining m to processor i. Again the unknowns x^{i-1} and x^i denote the boundary variables and the boundary equation

$$a^i x_{2m}^i + b^i x^i + c^i x_1^{i+2} = d^i \qquad (E^i)$$

lies in between the subsystems (S^i) and (S^{i+2}).

Phases 1 & 3 proceed in exactly the same way as in section 3.1, the only difference being that the calculation of $\mathbf{w}^i$, $\mathbf{r}^i$ and $\mathbf{s}^i$ necessitates each pair of processors to use a *TW* factorisation with one communication exchange. Details of the matrix transformations and fill-in patterns of phase 1 are shown in figure 3. This shows the method implemented with two-way Gaussian elimination rather than decomposition but the concept is identical. The first matrix shows the non-zeroes after the inward sweep and just before the communication exchange and the second shows them after the outward sweep. The inter-processor boundaries are marked with dotted lines.

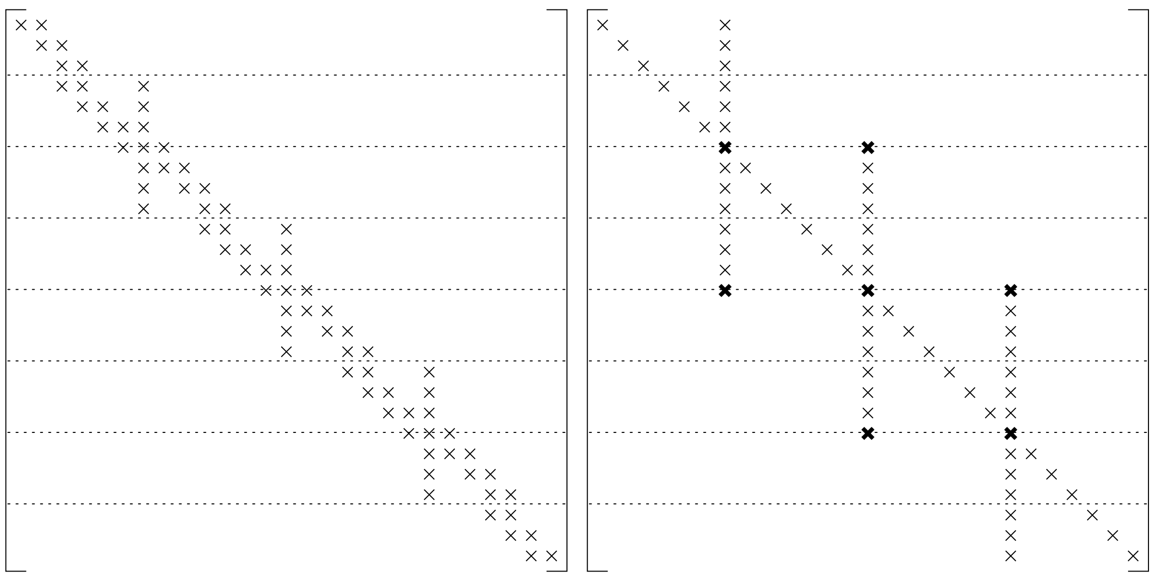

Figure 3: Matrix transformations and fill-ins in phase 1 of algorithm *TWTW*

4.1.1 Phase 2

The reduced system is solved for the boundary variables, again by applying a *TW* decomposition to R. Since the order of the reduced system is $q-1$ $(= P/2-1)$, the sweeps through the processor chain will employ half of the processors solely as communication relays. For machines with fixed communication channels this may be the only possible implementation, but, if direct communication with next nearest neighbouring processors is possible, a further enhancement can be added. In this case, if each boundary equation is stored on *both* sides of the inter-processor boundaries, the network can be divided into two independent parts, the even numbered processors, $\{2,4,\ldots,P-2\}$, and the odd numbered processors, $\{3,5,\ldots,P-1\}$, each with their own copy of the reduced matrix. Utilising the next nearest neighbour connections, both odd and even systems can then be solved simultaneously (with a TW decomposition) for a cost of approximately q communications start-ups, a considerable saving. Note that, as the outward sweeps finish, the values of x^2 and x^{P-2} must be transmitted to processors 1 and P respectively to enable them to commence phase 3.

4.2 Stability of the algorithm

Suppose A is strictly diagonally dominant. Then, since the TW algorithm is known to be stable, section 2.2, the only question is whether the reduced system inherits diagonal dominance from the full system. For the *classical* partition method, it is shown in [16] that this is indeed the case, regardless of the local reduction technique. Therefore, since a system reduced by local TW factorisation on P processors is identical to the reduced system after local LU factorisation on $P/2$ processors, it is readily seen that diagonal dominance is retained.

4.3 Odd Numbers of Processors

For $P \neq 2Q$ (integer valued Q) it is, of course, not possible to pair up all processors for local TW factorisation. In this case one of the processors must employ LU decomposition whilst the rest use the two-way algorithm. This results in a reduced system of size $(P-1)/2$. Implementation is not difficult as the TW code can be easily modified to do LU decomposition.

4.4 Operation Count

The arithmetic costs of phases 1 & 3 are as follows:-

	Calculation	Count	Notes
Phase1	$[T^i, W^i]$	$[3m]$	
	$\mathbf{w}^i$	$5m-1$	
	$[\mathbf{r}^i]$	$[4m-2]$	(1)
	$[\mathbf{s}^i]$	$[m]$	(1)
Phase3	$\mathbf{x}^i$	$4m$	(2)

Notes:–

(1) For processor $i-1$ the costs of $\mathbf{r}^i$ and $\mathbf{s}^i$ are $4m-2$ and m respectively. For processor i the costs are reversed.

(2) Again for processors 1 and P the cost of phase 3 is just $2m$.

In addition in phase 1 there is the cost of one communication exchange of $1+[2]$ numbers ($1+[1]$ for the calculation of $\mathbf{w}^i$ plus an extra [1] for one of the fill-ins). Thus the total cost of phases 1 and 3 is

$$\{9m-1+[8m-2]\}t_a+1s_c+\{1+[2]\}t_c. \tag{10}$$

The calculation of the reduced matrix R and right hand side $\tilde{\mathbf{d}}$ again requires $4+[6]$ arithmetic operations per processor plus the transmission of $1+[2]$ numbers. A total of

$$\{4+[6]\}t_a+1s_c+\{1+[2]\}t_c. \tag{11}$$

For convenience in cost estimation for phase 2 assume that q is odd, so that the reduced system order $q-1$ is even. Then from (5) and writing $l=(q-1)/2=(P-2)/4$ the cost can be expressed as

$$\{5l-1+[3l]\}t_a+\{2l-1\}s_c+\{2l-1+[l]\}t_c. \tag{12}$$

Finally at the end of phase 2, x^2 must be transmitted to processor 1 and x^{P-2} to processor P, a cost of

$$1s_c+1t_c. \tag{13}$$

Thus, from sums (10)-(13), the complete cost, in terms of m and l, is

$$\{9m+5l+2+[8m+3l+4]\}t_a+\{2l+2\}s_c+\{2l+2+[l+4]\}t_c. \tag{14}$$

4.5 Summary

Henceforth, using familiar taxonomy, this algorithm shall be referred to as *TWTW* – Two-Way decomposition locally, Two-Way decomposition globally. The operation counts can be summarised as $P/2+1$ communications with $17n/P+2P$ arithmetic operations for time-dependent matrices or $9n/P+5P/4$ operations for constant ones. For 3 and 4 processor implementations this reduces to approximately $14n/P$ and $7n/P$ respectively, a substantial saving on the *LUTW* algorithm.

5 Results and Conclusions

Both algorithms have been implemented and timed extensively. Figure 4a (page 19) shows the cost in seconds for the solution of a system of size $n = 50,000$ on varying numbers of processors. The timings presented here are for 4 to 32 processors in increments of 4. As a comparison, the cost of solving the same system using LU decomposition on a single processor is 1.886976 seconds which shows a maximum speed-up of 14.52 for $TWTW$ on 32 processors. Since the arithmetic costs for $TWTW$ (or any other partition method) are of the order $17n/P$ compared to $8n$ for LU, the maximum possible speed-up, neglecting any communication overhead, is about $8P/17$ or 15.06 for 32 processors.

5.1 The Results

The following results can be subdivided into different cases dependent on P the number of processors. For 2 processors it is clear that the algorithm of choice is simply the TW decomposition, section 2, with no boundary variables. Since this does not fall into the category of an $[L][G]$ (local-global) method no results are included here. The 4 processor case does have boundary variables but is somewhat different to those with $P > 4$, section 4.5 and so these results are treated separately.

5.1.1 Hardware: The Meiko Computing Surface

The experiments were carried out on a Meiko Computing Surface consisting of 32 T800 transputers. This is a distributed memory MIMD machine with a configurable network. The code was written in Meiko C with the message passing implemented using CSTools, [7]. Each processor has an on-board memory of 4MB and this limited the size of system that could be solved to approximately 23,500 equations per processor (i.e. 750,000 variables for 32 processors). The sequential LU code has much smaller memory requirements and runs with up to 62,000 equations. This latter memory limit made true speed-up figures impossible for systems of a larger size and so the results have been extrapolated to get approximate speed-up figures.

5.1.2 4 Processors

The operation counts show, sections 3.3 & 4.5, that the 4 processor $TWTW$ algorithm has a lower arithmetic cost than $LUTW$ ($14n/P$ compared to $17n/P$) and this is clearly borne out by the actual timings. Figure 4b (page 19) shows the time for one solve for a number of different sizes of system and it is clear that the n dependence is smaller for $TWTW$. Figures 5a & 5b show the percentage reduction in time for $TWTW$ over $LUTW$ (i.e. $100 \times (t_{LUTW} - t_{TWTW})/t_{LUTW}$) and speed-up over the 1 processor LU algorithm respectively and for greater clarity these are drawn with n on a $\log_{10}$ scale.

5.1.3 More than 4 Processors

For $P > 4$ the arithmetic costs are approximately equivalent and it is the communication costs that are reduced by the $TWTW$ algorithm. Figure 6a (page 19) shows the percentage reduction in time for $TWTW$ over $LUTW$ for 32 processors and varying sizes of system (again n is drawn on a $\log_{10}$ scale). Of course as the size of system increases, the arithmetical costs start to dominate and swamp any performance increase gained by communication. As a result, in all the tests, the percentage reduction in time appeared to reach an asymptotic limit of between 2-4% as the system size approached the maximum of 23,500 unknowns per processor.

For smaller systems the gain in using $TWTW$ over $LUTW$ can be quite substantial, but this must be moderated by the overall efficiency of the method. Figure 6b (page 19) show the corresponding speed-ups for the percentage differences shown in figure 6a. For example, if $n = 10,000$ ($\log_{10} n = 4$) $TWTW$ is 17.7% faster than $LUTW$. However, for this system size $TWTW$ is only 11.7 times faster than the sequential LU algorithm.

5.2 Conclusions

The testing has revealed the $TWTW$ algorithm to be generally more efficient than $LUTW$. This is particularly noticeable for 4 or fewer processors where the algorithm is clearly seen to reduce the n dependence in the operation count. It is no more difficult to implement than $LUTW$ and the two experimental versions tested here used much of the same code. For large numbers of processors, the scalability issues of the algorithm lie in the global solve of the reduced system and hence the two-way partition method can be no worse than the original partition method.

Future work in this area might involve the construction of an analogous algorithm for symmetric, positive-definite systems with a Cholesky type matrix decomposition. In addition, further testing on different machines is desirable and possibly the introduction of a different algorithm such as Cyclic Reduction for the global solve.

Acknowledgements

This algorithm was designed at the Mathematics Department, Heriot-Watt University, Edinburgh and benefited greatly from the help of Dr. D. B. Duncan.

References

[1] I. Babuska. Numerical stability in problems of linear algebra. *SIAM J. Numer. Anal.*, 9:53–77, 1972.

[2] O. Buneman. A compact non-iterative Poisson solver. Tech. Rep. 294, Stanford University Institute for Plasma Research, Stanford, CA, 1969.

[3] B. Buzbee, G. Golub, and C. Neilson. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.*, 7:627–656, 1970.

[4] S. Lennart Johnsson. Solving Narrow Banded Systems on Ensemble Architectures. *ACM Trans. Math. Soft.*, 11:271–288, 1985.

[5] S. Lennart Johnsson. Solving Tridiagonal Systems on Ensemble Architectures. *SIAM J. Sci. Stat. Comput.*, 8:354–392, 1987.

[6] D. H. Lawrie and A. H. Sameh. The Computation and Communication Complexity of a Parallel Banded System Solver. *ACM Trans. Math. Soft.*, 10:185–195, 1984.

[7] Meiko Ltd., Bristol. *Meiko CS Tools Manual*, 1989.

[8] James M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[9] A. H. Sameh and D. J. Kuck. On Stable Parallel Linear System Solvers. *J. ACM*, 25:81–91, 1978.

[10] Harold S. Stone. An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations. *J. ACM*, 20:27–38, 1973.

[11] Harold S. Stone. Parallel Tridiagonal Equation Solvers. *ACM Trans. Math. Soft.*, 1:289–307, 1975.

[12] Paul N. Swarztrauber. A Parallel Algorithm for Solving General Tridiagonal Equations. *Math. Comp.*, 33:185–199, 1979.

[13] Paul N. Swarztrauber and Roland A. Sweet. Vector and Parallel Methods for the Solution of Poisson's Equation. *J. Comput. Appl. Math.*, 27:241–263, 1989.

[14] H. A. van der Vorst. Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. *Parallel Comput.*, 5:45–54, 1987.

[15] C. H. Walshaw. *Parallel Algorithms for Large Sparse Systems of Differential Equations.* PhD thesis, Heriot-Watt University, 1991.

[16] C. H. Walshaw. Diagonal Dominance in the Parallel Partition Method for Tridiagonal Systems. (submitted for publication), 1993.

[17] H. H. Wang. A Parallel Method for Tridiagonal Equations. *ACM Trans. Math. Soft.*, 7:170–183, 1981.

[18] Burton Wendroff. *Theoretical Numerical Analysis.* Academic Press, New York, 1966.

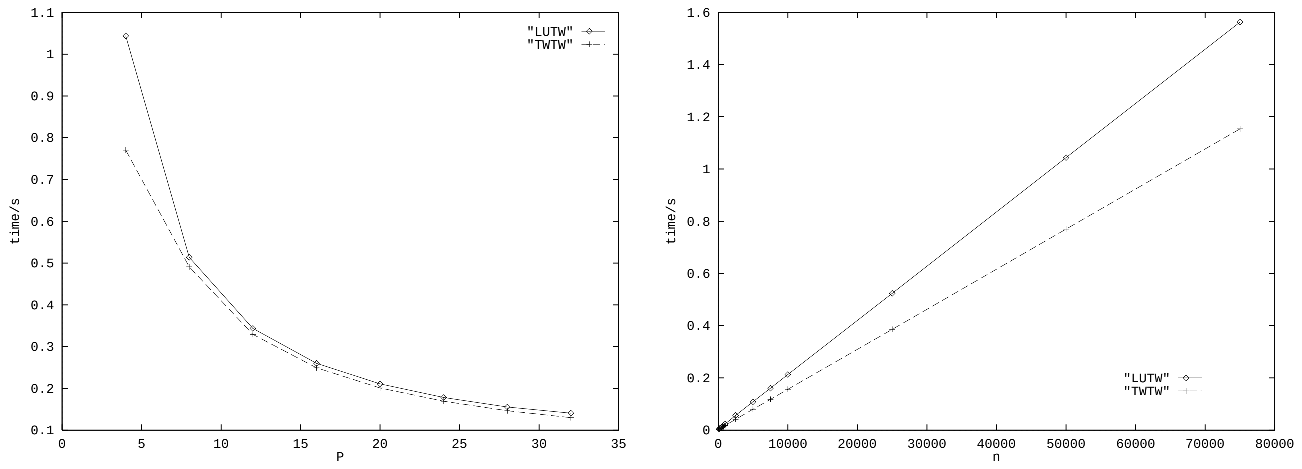

Figure 4: Timings results for (a) n=50,000 & (b) P=4

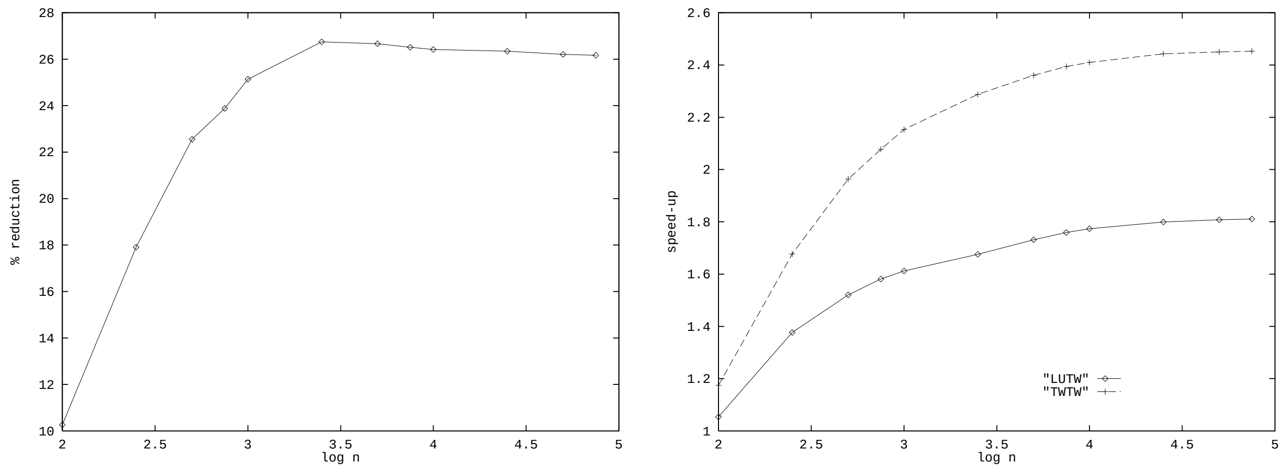

Figure 5: Results for 4 processors

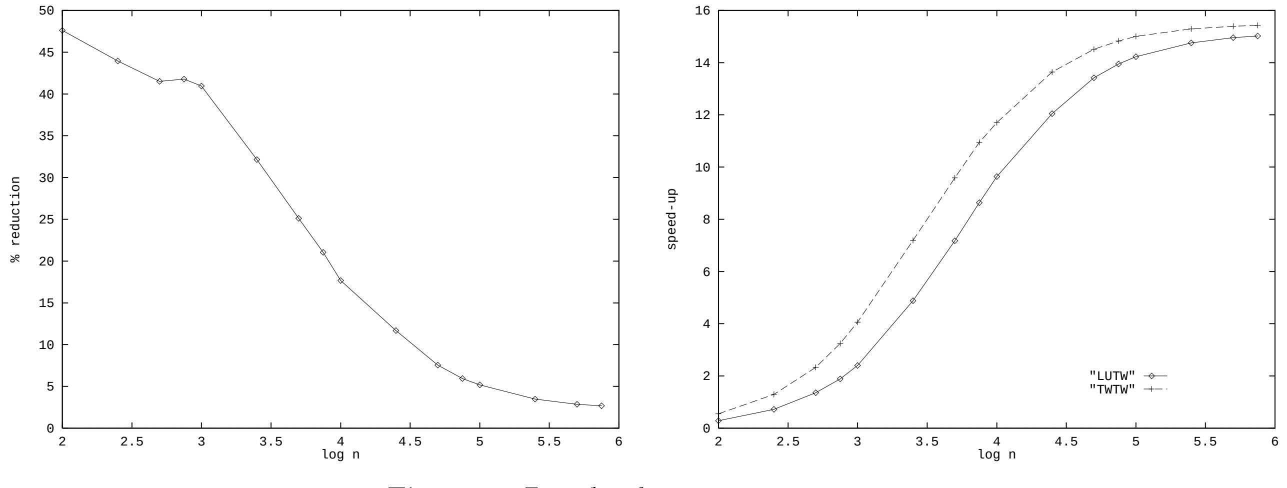

Figure 6: Results for 32 processors