Yale University
Department of Computer Science

Optimizing Tridiagonal Solvers for Alternating Direction Methods on Boolean Cube Multiprocessors

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-679
January 1989

This work was supported by the Office of Naval Research under contract N00014-86-K-0310. Approved for public release: distribution is unlimited.

† A revised edition of TR-532. To appear in *SIAM Journal on Scientific and Statistical Computing*.

Optimizing Tridiagonal Solvers for Alternating Direction Methods on Boolean Cube Multiprocessors

S. Lennart Johnsson[1] and Ching-Tien Ho
Department of Computer Science
Yale University
New Haven, CT 06520
Johnsson@cs.yale.edu, Johnsson@think.com, Ho@cs.yale.edu

Abstract. Sets of tridiagonal systems occur in many applications. Fast Poisson solvers and Alternate Direction Methods make use of tridiagonal system solvers. Network based multiprocessors provide a cost effective alternative to traditional supercomputer architectures. We investigate the complexity of concurrent algorithms for the solution of multiple tridiagonal systems on Boolean cube configured multiprocessors with distributed memory. Variations of odd-even cyclic reduction, parallel cyclic reduction, and algorithms making use of data transposition with or without substructuring and local elimination, or pipelined elimination are considered. A simple performance model is used for algorithm comparison, and the validity of the model is verified on an Intel iPSC/1. For many combinations of machine and system parameters, pipelined elimination, or equation transposition with or without substructuring is optimum. We present hybrid algorithms that at any stage choose the best algorithm among the considered ones for the remainder of the problem.

It is shown that the optimum partitioning of a set of independent tridiagonal systems among a set of processors yields the embarrassingly parallel case. If the systems originate from a lattice and solutions are computed in alternating directions, then to first order the aspect ratio of a computational lattice shall be the same as that of the lattice forming the base for the equations.

Our experiments demonstrate the importance of combining in the communication system for architectures with a relatively high communications start-up time.

Key words. tridiagonal systems, Boolean cubes, pipelined Gaussian elimination, transposition, substructuring, balanced cyclic redcution

AMS(MOS) subject classifications. 15A06, 65F05, 65F10, 65N05, 68Q25

1 Introduction

Tridiagonal systems of equations occur in many methods used for the solution of partial differential equations. In the Alternating Direction Method (ADM), tridiagonal matrices arise from one-directional central difference approximations of partial derivatives. In so called fast Poisson solvers an FFT in one dimension decouples the system into a number of independent tridiagonal systems. Tridiagonal systems are also used as preconditioners for the conjugate gradient

[1]Also with Dept. of Electrical Engineering, Yale Univ., and Thinking Machines Corp., Cambridge, MA 02142.

method.

In parallel architectures with distributed memory, the communication is a critical issue with respect to performance, in particular, if the number of processors approaches the number of unknowns [24,22,23,3,10,20]. The communication time for a given computation on a given architecture depends on the data allocation to the distributed memory, the choice of numerical algorithm, and the communication algorithms. These issues are the focal points of this work. The target architectures have processors interconnected as a Boolean n-cube, which has $N = 2^n$ nodes. One- and multi-dimensional lattices can be embedded in Boolean cubes with one array node per cube node preserving adjacency by a *binary-reflected* Gray code [21.17]. This embedding uses all cube nodes when the lattice sides are powers of two. For other side lengths other embeddings may be more effective [6,2]. In our analysis the time for the communication of one byte between a pair of processors is denoted t_c, the time for the communication overhead (start-up) is denoted τ, the time for an arithmetic operation t_a, and the maximum packet size imposed by a limited buffer size B_m. Moreover, we assume that communication can take place only on one port per processor at a time, *one-port communication.* The time for the solution of the systems of equations is denoted $T_{alg}(P, Q; N, t_a, t_c, \tau, B_m)$, where *alg* identifies the algorithm. the first argument is the size (P) of a tridiagonal system and the second the number of systems (Q). The third parameter is the number of Boolean cube processors to which the $P \times Q$ equations are allocated. When necessary, other parameters may be added or omitted.

For the solution of a single tridiagonal system we consider:

- Substructuring followed by odd-even cyclic reduction, CR, [1] for the reduced system. as described in [17,12].

- Substructuring followed by parallel cyclic reduction, PCR [8].

- The gathering of the equations to 2^{n-i} processors by i steps of a data transposition algorithm such that each processor holds $\frac{P}{N}2^i$ equations, followed by two-way elimination. and a final phase in which the results are distributed to the set of processors originally holding the equations; algorithm TGET(i) for i-step data gathering by *Transposition*, *Gaussian elimination*, and i-step data scattering by *Transposition.* With substructuring algorithm TGET(i) becomes algorithm SS/TGET(i).

- The broadcasting of equations from every node to every other node within i-dimensional subcubes such that the system is evenly distributed across 2^{n-i} processors and there are 2^i copies of the same system. No distribution of the solution is needed between the $(n-i)$-dimensional subcubes. With substructuring, algorithm BCGE(i) becomes algorithm SS/BCGE(i).

- A set of hybrid algorithms obtained by combining substructuring, odd-even cyclic reduction, and the TGET(i) algorithm.

The first transpose operation in TGET(i) is a gather operation, and the second a scatter operation. We refer to both as transpose operations since each gather, or scatter, operation is equivalent to a vector transpose. Furthermore, in the case of multiple tridiagonal systems the multiple gather, or scatter, operations are transpose operations. In the TGET(i) algorithm only two processors are used for the elimination. The purpose of the transpose operation is to reduce the number of communication start-ups compared to a direct two-way elimination. The reduction is accomplished by using a transpose algorithm that requires i steps in moving the equations to an $(n-i)$-cube. With *one-port communication* a doubling algorithm such as a spanning *binomial tree* algorithm is optimal [16]. The data transfer time increases for every step of the transpose algorithm, and is proportional to $\frac{P}{N}(2^i - 1)$ for P equations and a spanning binomial tree algorithm. In the case of concurrent communication on all ports of every processor, *n-port communication*, the data transfer time can be reduced by a factor of i by using communication according to *balanced trees* [16,7]. Depending upon the ratio of the data transfer time to the start-up time there may exist a non-trivial, optimum value of i.

Cyclic reduction algorithms make use of many processors for the elimination, and yields a lower, parallel, arithmetic complexity for $N \geq 16$ (see Table 1). But the lower bound for the number of start-ups for a vector transposition is approximately half of the minimum number of start-ups for odd-even cyclic reduction with communications restricted to one send *or* one receive operation at a time. A lower bound of n start-ups for the transposition is shown in [14]. Though the number of start-ups for cyclic reduction is higher in the case of unlimited buffer sizes, it has a lower data transfer time than the transpose algorithm, and possibly also fewer start-ups in the case of buffers of limited size. These different characteristics give rise to some interesting optimization problems that we investigate in the context of hybrid algorithms.

For the solution of multiple tridiagonal systems we consider:

- Pipelined two-way Gaussian elimination.
- Gathering of equations and separation of systems into subcubes by equation transposition, solution of systems by pipelined two-way Gaussian elimination, and scattering of the solution. This algorithm is the multiple systems version of algorithms TGET(i). With initial substructuring, it becomes SS/TGET(i).
- Substructuring followed by *Balanced Cyclic Reduction* [13] for the reduced system solver, SS/BCR.
- Combinations of the above three algorithms.

We have verified our analytical complexity models on a medium scale, parallel architecture: the Intel iPSC/1. A binary-reflected Gray code was used for the data mapping. The Intel iPSC/1 effectively only supports one send *or* one receive operation per processor at a time, and most of our complexity estimates are specialized to this case. Because of the high copying time on the Intel iPSC/1 there exists a packet size $B_{cp} < B_m$ above which it is beneficial to send data directly rather than combining several packets into one, in order to minimize the number of start-ups [14]. For the system software available at the time of the experiments $B_{cp} = 256 \approx \tau/t_{cp}$ bytes, where the time for copying one byte is t_{cp}. The copy time is sufficiently large to affect the choice of algorithm. The start-up time for communication is also significant on the Intel iPSC/1, and we demonstrate the effectiveness of message combining in order to reduce the communications overhead.

2 Solving a Single Tridiagonal System on an n-Cube

We assume that the equations are assigned to processors by dividing the system into blocks. or by applying incomplete nested dissection [4] and suitably associating separator nodes with adjacent partitions. The partitioned tridiagonal system of equations has the following form:

$$\left(\begin{array}{cccccccccccc}
x & x & & & & & & & & & & \\
x & x & x & & & & & & & & & \\
 & x & x & x & & & & & & & & \\
\hline
 & & x & x & x & & & & & & & \\
 & & & x & x & x & & & & & & \\
 & & & & x & x & x & & & & & \\
\hline
 & & & & & x & x & x & & & & \\
 & & & & & & x & x & x & & & \\
 & & & & & & & x & x & x & & \\
\hline
 & & & & & & & & x & x & x & \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x
\end{array}\right) X = Y.$$

The horizontal lines indicate the partitioning/substructuring. For each partition we apply Gauss-Jordan elimination as described in [12,25]. One communication is needed for the first and the last equations in each partition. By a suitable allocation of matrix coefficients to processors it is not necessary to transfer an entire equation, and some element transfers [18] can be saved. The execution time is approximately

$$T_{SS}(P,1;N,t_a,t_c,\tau) = 17\left(\left\lceil\frac{P}{N}\right\rceil - 1\right)t_a + 2(16t_c+\tau) + 2(4t_c+\tau), \quad B_m \geq 16, \qquad (1)$$

where it is assumed that a processor can perform one send *or* one receive operation at a time. that the system is real, and that floating-point numbers are represented by 4 bytes (single-precision). The form of the system after the substructured elimination is shown below. Note that the elimination of the last block is done in reversed order such that the reduced system formed by the last equation of each block, but the last, is of order $2^n - 1$.

$$\left(\begin{array}{cccccccccccc}
x & & x & & & & & & & & & \\
 & x & x & & & & & & & & & \\
 & & x & & & x & & & & & & \\
\hline
 & & x & x & & x & & & & & & \\
 & & x & & x & x & & & & & & \\
 & & x & & & x & & & x & & & \\
\hline
 & & & & & x & x & & x & & & \\
 & & & & & x & & x & x & & & \\
 & & & & & x & & & x & & & \\
\hline
 & & & & & & & & x & x & & \\
 & & & & & & & & x & & x & \\
 & & & & & & & & x & & & x
\end{array}\right) X = Y.$$

2.1 Cyclic Reduction (CR)

Odd-even cyclic reduction requires 17 operations per unknown, the same as substructured Gaussian elimination. Substructuring is always preferable with odd-even cyclic reduction, since only four communications are needed for the substructuring with one send *or* one receive at a time.

In [12,13] an *exchange* algorithm, and an *in-place* algorithm are proposed for the solution of tridiagonal systems distributed with one equation per processor in a Boolean n-cube. In the

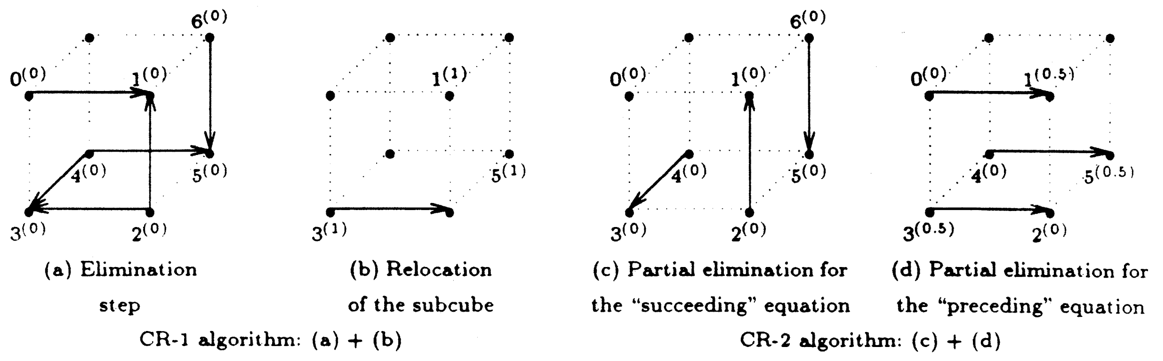


Figure 1: One forward elimination step of CR-1 and CR-2 algorithms on a 3-cube.

exchange algorithm the active set of equations are recursively moved into an easily identified subcube of one less dimension for each elimination step. Communication is needed for both the elimination step (Figure 1-(a)) and the reallocation to the subcubes (Figure 1-(b)). The reallocation can be performed with only one communication, such that with communication restricted to one send *or* receive operation at a time, $3(n-1)$ communications suffice for the elimination *and* the reallocation. The same number of communications is required in the back substitution phase. We refer to this algorithm as SS/CR-1. Note that in Figure 1-(a), the processor holding equation $3^{(0)}$ needs to store the equation received from the preceding processor ($2^{(0)}$) to avoid using $O(n)$ memory per processor. The number of communications per step can be reduced from three to two at the expense of $O(n)$ memory per processor, instead of constant memory. By performing a partial elimination (Figure 1-(c)), then moving the equation subject to elimination to the processor for the next elimination step, and completing the elimination there (Figure 1-(d)), one communication is saved. A total of $4(n-1)$ communications are needed. This algorithm is called SS/CR-2. For the *in-place* algorithm, SS/CR-IR, we simply use the routing software of the Intel iPSC.

The time to solve a single tridiagonal system of $P = 2^n - 1$ real equations on an n-cube by the SS/CR-2 algorithm [12,13] is given by

$$\begin{aligned} T_{CR\text{-}2}(N-1,1;N,t_a,t_c,\tau) &= (\log N - 2)(16t_a + 2(16+4)t_c + 2\times 2\tau) \\ &\quad + (14+3)t_a + 2(16+4)t_c + 2\times 2\tau, \quad B_m \geq 16. \end{aligned} \tag{2}$$

For $P > N$ substructuring is applied and the estimated time is given by

$$T_{SS/CR\text{-}2}(P,1;N,t_a,t_c,\tau) = T_{SS}(P,1;N,t_a,t_c,\tau) + T_{CR\text{-}2}(N-1,1;N,t_a,t_c,\tau). \tag{3}$$

In Equation (2) we assume that the division normally carried out in the back substitution phase is performed concurrently for all equations after the elimination phase. With real coefficients, one right hand side, and one processor per equation, a time of $3t_a$ is required. The arithmetic complexity can be reduced to 11 [13] sequential operations per step (instead of 16) without an increase in the communication complexity by dividing the arithmetic operations for the elimination and backsubstitution phases among pairs of processors. A further reduction to 9 operations is possible at the expense of additional communication. For complex coefficients the number of arithmetic operations per step is 82, 47, and 43, respectively [11]. The techniques for lower parallel arithmetic complexity are not applicable to the case of multiple tridiagonal systems. The complexity estimates in this paper apply to algorithms using 16 arithmetic operations in sequence for each equation, and a compatible communication scheme. The predicted and measured times agree well for the Intel iPSC/1 [15].

2.2 Parallel Cyclic Reduction (PCR)

Parallel Cyclic Reduction [8] performs an elimination on every equation in every step. With one equation per processor all processors are active throughout the computation, which completes in n steps. After each step the problem is transformed such that the number of equations per tridiagonal system is halved, and the number of systems doubled.

$$\begin{pmatrix}
x & x & & & & & & \\
x & x & x & & & & & \\
 & x & x & x & & & & \\
 & & x & x & x & & & \\
 & & & x & x & x & & \\
 & & & & x & x & x & \\
 & & & & & x & x & x \\
 & & & & & & x & x
\end{pmatrix} \Rightarrow
\begin{pmatrix}
x & & x & & & & & \\
 & x & & x & & & & \\
x & & x & & x & & & \\
 & x & & x & & x & & \\
 & & x & & x & & x & \\
 & & & x & & x & & x \\
 & & & & x & & x & \\
 & & & & & x & & x
\end{pmatrix} \Rightarrow
\begin{pmatrix}
x & & & & x & & & \\
 & x & & & & x & & \\
 & & x & & & & x & \\
 & & & x & & & & x \\
x & & & & x & & & \\
 & x & & & & x & & \\
 & & x & & & & x & \\
 & & & x & & & & x
\end{pmatrix}$$

No backsubstitution is needed. The total arithmetic complexity of the PCR algorithm is $12n2^n$, the parallel complexity $12n$. Substructuring is always preferable. The total number of messages is $2n2^n - 2n + 1$, and the number of parallel communications in sequence $4n$ (two exchanges) with one send *or* one receive operation at a time. During step j, $0 \le j < n$, the ith row of the reduced system exchanges its data with the $(i - 2^j)$th and the $(i + 2^j)$th rows (if $i - 2^j \ge 0$, and $i + 2^j \le P - 1$). With a binary-reflected Gray code encoding i and $i + 2^j$ are at most a distance two apart in the cube [12]. In implementing the SS/PCR algorithm on a Boolean cube, we can:

1. Perform an exchange operation after each step of PCR to move each subset of equations into the same subcube, such that communications for the next step remains *nearest-neighbor* for each subset, with no interference between communication for different subsets. This is algorithm SS/PCR-1.

2. Combine exchange communications with communications for elimination, as in the SS/CR-2 algorithm, and arrive at a SS/PCR-2 algorithm.

3. Decompose and combine the two distance-two sends and receives into three exchanges. i.e., six "nearest-neighbor" sends or receives. This is algorithm SS/PCR-3.

4. Use a static allocation and the routing logic, SS/PCR-IR.

In SS/PCR-1, $6n - 2$ start-ups are required. In the SS/PCR-2 algorithm the number of start-ups is reduced to $4n$ by splitting the elimination on the row into two parts; one equation is received and the associated elimination performed, then the partially modified row is sent to the "after-exchanged" processor, and the elimination completed using the row in the new processor. The allocation of equations to processors is changed. The solution variables are allocated according to binary code encoding of the indices. To move the solution of the reduced system back to the corresponding partition a binary code to Gray code conversion is required. Such a conversion requires $2n - 2$ start-ups [9]. The total number of start-ups is $8n - 4$ for SS/PCR-1 and $6n - 2$ for SS/PCR-2. In the SS/PCR-3 algorithm, the distance-two sends or receives are decomposed into two "nearest-neighbor" communications performed concurrently for all nodes such that there is no edge conflict [9]. Moreover, since the data sent to both distance-two neighbors in each SS/PCR step are the same, and the two paths to the distance-two neighbors can be arranged such that they share the first edge, the number of start-ups can be reduced from $8n - 4$ to $6n - 2$, i.e., the same as for the SS/PCR-2 algorithm. Note that the SS/PCR-3 algorithm is an *in-place* algorithm and therefore does not require the binary code to Gray code conversion at the end. In the SS/PCR-IR algorithm the communication time depends entirely on the routing logic. A routing discipline, like the one used in the Intel iPSC, that routes the dimensions that need to be routed in increasing order yields a conflict-free routing for the SS/PCR-IR algorithm [12].

For the analysis we use the following estimated time for PCR-2 (and PCR-3)

$$T_{PCR\text{-}2,3}(N, 1; N, t_a, t_c, \tau) = \log N(12t_a + 6 \times 16t_c + 6\tau) - 2(16t_c + \tau), \quad B_m \geq 16, \tag{4}$$

and for the substructured version

$$T_{SS/PCR\text{-}2,3}(P, 1; N, t_a, t_c, \tau) = T_{SS}(P, 1; N, t_a, t_c, \tau) + T_{PCR\text{-}2,3}(N, 1; N, t_a, t_c, \tau). \tag{5}$$

The predicted times for algorithms PCR-3 and PCR-IR are significantly lower than the measured times, and also significantly higher than the measured times for the odd-even cyclic reduction algorithms [15]. We attribute the difference between the measured and predicted times for the PCR algorithms to synchronization delays. The PCR algorithm is not of interest for multiple systems per processor due to its higher arithmetic complexity compared to odd-even cyclic reduction.

2.3 Equation Transposition and Gaussian Elimination.

Conventional Gaussian elimination only requires 8 operations per unknown, but substructured elimination requires 17 operations per unknown. Two-way elimination allows two processors to be used for the same tridiagonal system (almost) without an increase in total arithmetic complexity. With respect to arithmetic complexity substructuring shall be used if $N > \frac{17}{4}$. Note that the start-up time of TGET$(n - 2)$ is the same as that of TGET$(n - 1)$ in the event of one start-up per communication (unbounded buffers). When the equations are gathered by a doubling algorithm, such as a spanning binomial tree algorithm [5], the number of communication start-ups is reduced compared to a direct two-way elimination for a sufficiently large buffer size. After gathering the equations by an i-step transpose algorithm the equations are allocated in an $(n - i)$-dimensional subcube with $\frac{P}{N}2^i$ equations per processor, if no substructuring was applied initially; otherwise there are 2^i equations per processor. After the i steps of equation gathering the set of equations can either be solved by two-way elimination, or by substructuring followed by the solution of a system with $2^{n-i} - 1$ equations distributed with one equation per processor in an $(n - i)$-cube.

The time for i steps of equation gathering and i steps of scattering of the solution from an $(n-i)$-cube to an n-cube is

$$T_{sbt}(P,1;N,t_c,\tau,B_m,i) \;=\; \sum_{j=0}^{i-1}\left(\left\lceil\frac{16P\times 2^j}{B_mN}\right\rceil+\left\lceil\frac{4P\times 2^j}{B_mN}\right\rceil\right)\tau+20\frac{P}{N}(2^i-1)t_c. \qquad (6)$$

We refer to the algorithm that performs "i steps of gathering, two-way elimination for the solution of the systems of equations, and i-steps of scattering of the solutions" as a TGET(i) algorithm. However, for an optimum combination of substructuring and Gaussian elimination the choice of continuing the gathering of equations, or solving the system by two-way elimination, or performing a substructuring operation should be reevaluated for each step. We consider such algorithms in the section about hybrid algorithms. Let $T_{2GE}(P,1;N,t_a,t_c,\tau)$ be the time for two-way Gaussian elimination performed on P equations distributed across $N>1$ processors. Then,

$$T_{2GE}(P,1;N,t_a,t_c,\tau)=(4P-2)t_a+N\tau+(8N+8)t_c. \qquad (7)$$

In this expression it is assumed that both processors involved in the elimination of the "middle" 2×2 system solve for one variable after an exchange of the coupling equations. It is possible to save two element transfers at the expense of an increased arithmetic complexity of two operations by having one processor send one equation to the other, which upon the solution of a 2×2 system returns one of the solution variables.

For $P>N=4$, $T_{2GE}(P,1;4,t_a,t_c,\tau)<T_{SS}(P,1;4,t_a,t_c,\tau)+T_{2GE}(4,1;4,t_a,t_c,\tau)$. Hence, with P equations spread across four processors, substructuring should never be applied. Two-way Gaussian elimination has a processor efficiency of 50% with 8 floating-point operations per equation, whereas substructuring has 100% efficiency, but requires 17 floating-point operations per equation. Furthermore, two-way Gaussian elimination has the same communication time regardless of the number of equations per processor. Substructuring needs an additional time of $4\tau+40t_c$.

The gathering of equations to a subset of processors, *all-to-one personalized communication* [16] in subcubes, can either be carried out by a send operation using the general router of the Intel iPSC/1, or by a user coded spanning binomial tree algorithm with combining (which is optimal for one-port communication [5]). The router also uses a binomial tree routing, but does not perform message combining. For the scatter operation, *one-to-all personalized communication*, there is the additional possibility of a *one-to-all broadcasting* of the entire solution vector. More data than necessary is communicated, no splitting of packets are required, and many architectures have efficient implementations of data broadcast. In order to evaluate the different algorithms for gathering and scattering equations on the Intel iPSC/1 we implemented:

- *All-to-one personalized communication* using the router (no combining) followed by router broadcast.
- *All-to-one personalized communication* with combining and *one-to-all personalized communication* with splitting, both based on the spanning binomial tree routing [16].

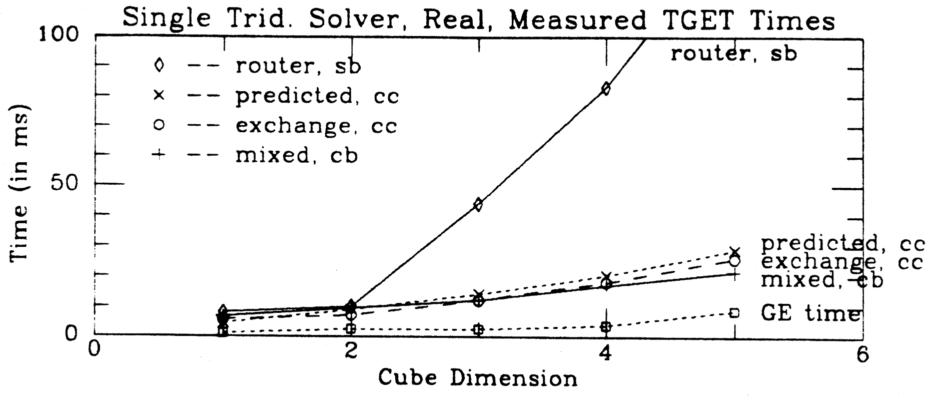


Figure 2: Measured values of $T_{TGET(n)}(N, 1; N, t_a, t_c, \tau)$ using routing with combining (-cc), the routing logic of the Intel iPSC/1 (no combining), and broadcasting (-sb), and combining and broadcasting (-cb).

- *All-to-one personalized communication* with combining followed by router broadcast.

We implemented three different communication algorithms for the SS/TGET(n) algorithm. The measured times compare as shown in Figure 2. The substructuring part is the same for all alternatives and is not included. We refer to the three communication algorithms with the postfix -sb, -cc, and -cb. The measured times for the SS/TGET(n)-sb algorithm increase exponentially due to the lack of message combining. The implementation using combining and broadcasting is more efficient than the implementation using combining for both phases of the transposition. The latter is about 10% slower than the former. The arithmetic times are less than 50% of the total times for cubes of less than 5 dimensions. Since the number of communications in the case of an unlimited buffer size increases linearly with the number of dimensions of the cube. but the data volume and arithmetic time increases exponentially, the time for data transfer and arithmetic will eventually dominate the total time as the number of cube dimensions increases. With a limited package size, the number of start-ups will eventually also increase exponentially in the number of cube dimensions.

2.4 Broadcasting and Concurrent Two-way Gaussian Elimination.

In the TGET(i) algorithm, only one ($n - i$)-dimensional subcube is active. It is possible to avoid distributing the results of the two-way elimination to the processors storing the equations. if all processors send their equation(s) to every other processor, *all-to-all broadcasting* [16]. If the broadcasting from every processor is restricted to an i-cube, then the original problem is transformed into solving 2^i identical systems in 2^i independent ($n-i$)-cubes. In each subcube it is sufficient to solve for 2^{n-i} variables. No communication of the components of the solution vector between subcubes is necessary. (If $i = n$ then backsubstitution can be avoided entirely by making the forward elimination terminate at the appropriate equation.) The algorithm can be used with or without a substructuring phase, called SS/BCGE(i) and BCGE(i), respectively. With one send *or* one receive operation at a time, the communication complexity of the broadcasting phase of the BCGE(i) algorithm is twice that of the gathering of equations in algorithm TGET(i). The parallel arithmetic complexity of algorithm BCGE(i) is the same as that of TGET(i). The estimated execution times without any intermediate substructuring and two-way elimination for the equations are

$$\begin{aligned} T_{BCGE(i)}(P,1;N,t_a,t_c,\tau,B_m,i) &= \left(32\frac{P}{N}(2^i-1)+2^{n-i+3}+8\right)t_c \qquad B_m \geq 12 \\ &+2\left(\sum_{j=0}^{i-1}\left\lceil\frac{16P\times 2^j}{NB_m}\right\rceil+2^{n-i-1}\right)\tau+(4P-2)t_a. \end{aligned} \tag{8}$$

$$T_{SS/BCGE(i)}(P,1;N,t_a,t_c,\tau,B_m,i) = T_{SS}(P,1;N,t_a,t_c,\tau)+T_{BCGE(i)}(N,1;N,t_a,t_c,\tau,B_m,i). \tag{9}$$

The measured times are somewhat lower than the predicted times [15] for algorithm BCGE($n-1$) assuming one send *or* receive operation at a time. A 20% overlap between send and receive operations explains the difference between the predicted and measured times. Such an overlap has been observed also in other experiments. Note that if the communication time for one exchange operation is twice that of one send or one receive operation, then algorithm BCGE(i) is always inferior to algorithm TGET(i). However, if send and receive operations can be performed concurrently, then the BCGE(i) algorithm is of lower complexity than the TGET(i) algorithm.

The BCGE(i) algorithm is of sequential complexity $O(2^iP)$. With multiple tridiagonal systems distributed with one equation per processor, and each processor having one equation of every system, the arithmetic complexity of algorithm BCGE(i) makes it clearly inferior to algorithm TGET(i).

2.5 Hybrid Algorithms

The parallel arithmetic complexity of algorithm TGET(i) is higher than that of the CR algorithms, while the former requires less start-ups than the latter algorithm in the case of unbounded communication buffers. The data transfer times for the CR algorithms are lower than the data transfer time for algorithm TGET(i). These characteristics motivate the consideration of hybrid algorithms. Let $T_{trid}(P,1;N,t_a,t_c,\tau,B_m)$ be the total time required to solve a single system of P equations in an N processor cube for given t_a, t_c, τ and B_m. Assume that P is a power of two, $P \geq N \geq 4$ and $B_m \geq 16$. Then, $T_{trid}(P,1;N,t_a,t_c,\tau,B_m) \leq$

$$\begin{cases} T_{2GE}(P,1;4,t_a,t_c,\tau), & \text{if } N=4, \\ \min\{T_{2GE}(N,1;N,t_a,t_c,\tau), & \\ \quad T_{sbt}(N,1;N,t_c,\tau,B_m)+T_{trid}(N,1;\frac{N}{2},t_a,t_c,\tau,B_m), & \\ \quad T_{cr}(t_a,t_c,\tau)+T_{trid}(\frac{N}{2},1;\frac{N}{2},t_a,t_c,\tau,B_m)\}, & \text{if } P=N, \\ \min\{T_{2GE}(P,1;N,t_a,t_c,\tau), & \\ \quad T_{sbt}(P,1;N,t_c,\tau,B_m)+T_{trid}(P,1;\frac{N}{2},t_a,t_c,\tau,B_m), & \\ \quad T_{SS}(P,1;N,t_a,t_c,\tau)+T_{trid}(N,1;N,t_a,t_c,\tau,B_m)\}, & \text{otherwise,} \end{cases}$$

$$\begin{aligned}
\text{where } T_{2GE}(P,1;N,t_a,t_c,\tau) &= (4P-2)t_a + N\tau + (8N+8)t_c, \\
T_{SS}(P,1;N,t_a,t_c,\tau) &= 17(\frac{P}{N}-1)t_a + 4\tau + 40t_c, \\
T_{sbt}(P,1;N,t_c,\tau,B_m) &= \left(\left\lceil\frac{16P}{B_m N}\right\rceil + \left\lceil\frac{4P}{B_m N}\right\rceil\right)\tau + \frac{20P}{N}t_c, \text{ and} \\
T_{cr}(t_a,t_c,\tau) &= 16t_a + 4\tau + 40t_c.
\end{aligned}$$

$T_{sbt}(P,1;N,t_c,\tau,B_m)$ is the time for gathering of $\frac{P}{N}$ equations per processor in N processors to $\frac{2P}{N}$ equations per processor in $\frac{N}{2}$ processors, *and* scattering $\frac{2P}{N}$ components of the solution vector in each of $\frac{N}{2}$ processors to $\frac{P}{N}$ components of the solution vector in each of N processors. $T_{cr}(t_a,t_c,\tau) = T_{CR\text{-}2}(N,1;N,t_a,t_c,\tau) - T_{CR\text{-}2}(\frac{N}{2},1;\frac{N}{2},t_a,t_c,\tau)$ is the time for one-step cyclic reduction using algorithm CR-2. Note that in an N processor cube, only $n-1$ steps is required for SS/CR algorithm to solve for a single system as seen from equations (2) and (3). Since the "one-step saving" does not generalize to the multiple systems case, we ignore the possible one-step saving for CR algorithm. For the solution of P equations on $N = 4$ processors two-way Gaussian elimination yields the lowest complexity of the algorithms we consider. Hence. $T_{trid}(P,1;4,t_a,t_c,\tau,B_m) = T_{2GE}(P,1;4,t_a,t_c,\tau)$. For N equations on $N > 4$ processors we consider:

- Two-way Gaussian elimination,
- One-step gathering of equations followed by the solution of a system of N equations evenly distributed over $\frac{N}{2}$ processors,
- One-step cyclic reduction followed by the solution of a system of $\frac{N}{2}$ equations on $\frac{N}{2}$ processors.

For the solution of $P > N$ equations on $N > 4$ processors we consider:

- Two-way Gaussian elimination,
- One-step gathering of equations followed by the solution of a system of P equations on $\frac{N}{2}$ processors,
- Substructuring followed by the solution of a system of N equations on N processors.

Note that partial substructuring, i.e., reduction to more than one equation per processor, is in general not preferable with respect to performance. The "total" and "partial" substructurings require the same number of communication start-ups, and the only disadvantage of substructuring compared to the TGET(i) algorithm is the start-up time for $N > 4$. (The time for arithmetic operations for substructuring and for the TGET($n-1$) algorithm compares as $17\frac{P}{N}$ to $4P$, approximately.)

Figure 3 shows the calling sequencies. The label on an edge of the graph is the expression for the time required to traverse that edge, with the arguments for the number of systems (which is one) and the parameters t_a, t_c, τ and B_m omitted. The argument of T_{sbt} and T_{SS} in the figure is $\frac{P}{N}$. The direct solution of P equations on N processors is not represented in the figure. The cyclic

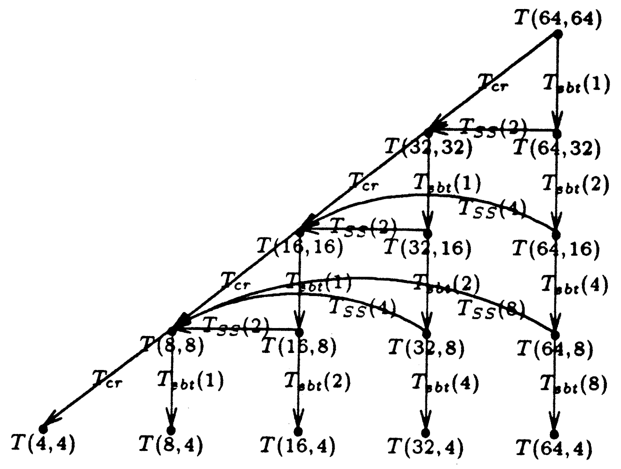


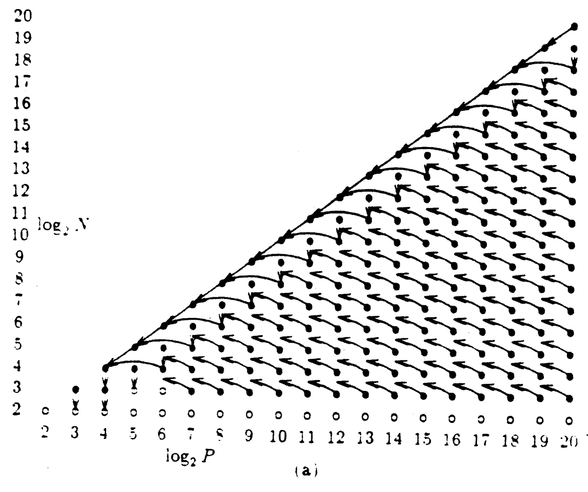
Figure 3: The calling sequencies.

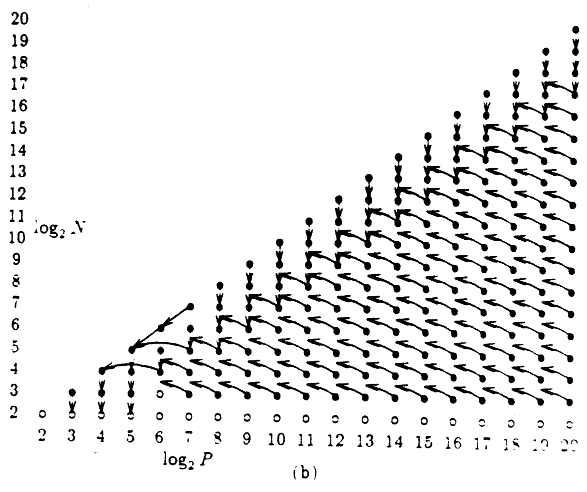
reduction (or parallel cyclic reduction) algorithm is represented by the edges on the diagonal. For each node on the diagonal there is one equation per processor. Applying the TGET(i) algorithm implies a vertical motion to level $\frac{P}{N} = 2^i$. Substructuring implies a horizontal motion to the node on the diagonal. Figures 4 and 5 show the optimal calling sequencies (among the considered algorithms) for various ratios of t_a/t_c and τ/t_c, and B_m (with the maximum required buffer size specified). The "o" sign denotes the two-way Gaussian elimination algorithm in the figures.

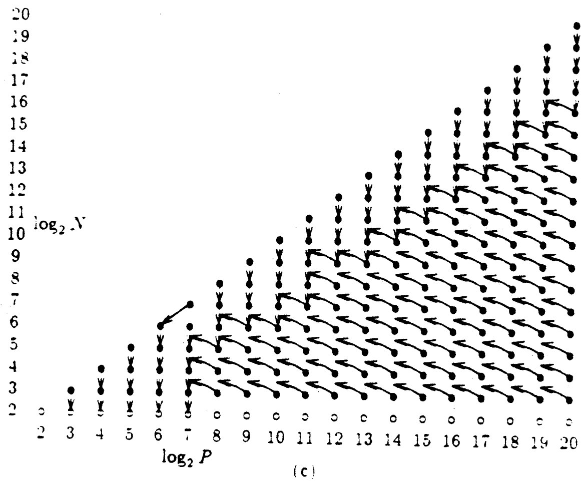
2.6 Summary of Algorithms for the Solution of Single Tridiagonal Systems

Table 1 summarizes the complexities of the individual algorithms. Memory requirements are expressed in units of 4 elements of 4 bytes each (real systems, single precision). Some measured and predicted times for the solution of a single tridiagonal system on an Intel iPSC/1 are given in Figure 6. The measured values of the machine parameters are approximately: t_a = 30 μsec, t_c = 10 μsec and τ = 2 $m$sec. The predicted execution times agree within 20% with the measurements, except for the PCR algorithm. The measured times for this algorithm are considerably higher than the predicted times. For the Intel iPSC/1 the predicted (and by far the measured) times for the PCR algorithm are always higher than those of any of the CR algorithms described here. With respect to performance, odd-even cyclic reduction is always preferable to parallel cyclic reduction on the Intel iPSC/1.

The complexity of algorithm SS/TGET($n-2$) is always lower than that of algorithm SS/TGET($n-1$), if $N \geq 4$. If one exchange operation takes the same time as one *send* or *receive* operation, then algorithm SS/BCGE(i) is always of lower complexity than algorithm SS/TGET(i). With respect to arithmetic time, the break-even point between the algorithms TGET(i) and SS/TGET(i) (for $i < n$) is at $N = \frac{17}{4}$, independent of P; substructuring should always be performed, except for N = 2 or 4. With respect to data transfer time, algorithm SS/TGET($n-2$) is of lower complexity than algorithm TGET($n-2$), if $P > N \geq 16$. If $N = 8$ the break-even point is $P = 24$. For $N = 4$ algorithm TGET($n-2$) is always of lower complexity







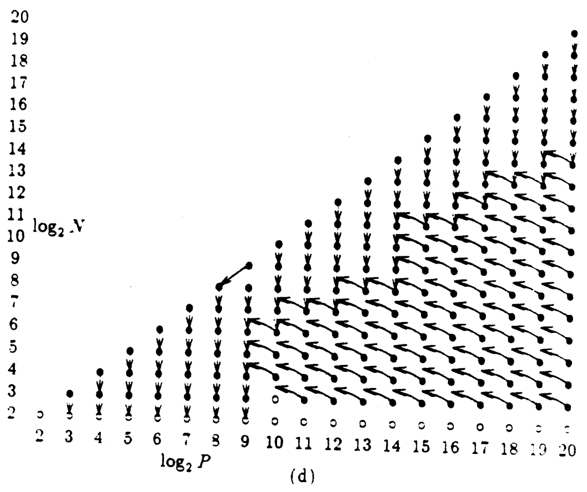


Figure 4: The optimal calling sequencies are shown for $t_a/t_c = 10$, $\tau/t_c = 1000$ and (a) $B_m = 16$ bytes, (b) $B_m = 32$ bytes and (c) $B_m \geq 256$ bytes, respectively. In (d), the optimal calling sequencies for $t_a/t_c = 1$, $\tau/t_c = 1000$ and $B_m \geq 1024$ bytes are shown.

than algorithm SS/TGET$(n-2)$. With respect to start-up time, an algorithm with substructuring requires four more start-ups than the corresponding algorithm without substructuring. assuming unlimited buffer size. However, for limited buffer size, the start-up time asymptotically becomes proportional to the data transfer time.

For the hybrid algorithms we conclude:

- With increasing maximum buffer size the boundary between the regions for gathering and substructuring moves in a direction implying more steps of gathering instead of substructuring. For a very small buffer size cyclic reduction should be used.
- Increasing the significance of the arithmetic time implies a growing region for substructuring, and cyclic reduction.
- Increasing the data transfer time implies a growing region for substructuring, and cyclic reduction.

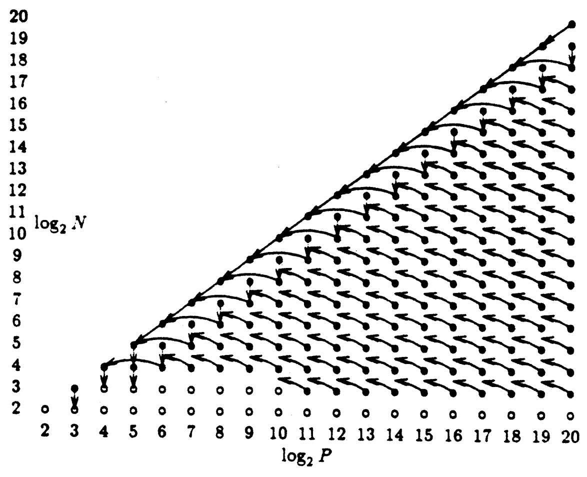


Figure 5: The optimal calling sequencies for $t_a/t_c = 0.01$, $\tau/t_c = 1$ and $B_m \geq 16$ bytes.

For complex systems the number of real arithmetic operations increases faster than the data volume. Hence, the communication contributes a smaller fraction to the total execution time for a complex system than a real system. See [15] for measured and predicted execution times on the Intel iPSC/1.

3 Multiple Tridiagonal Systems

With complete freedom of distributing the systems of equations the obvious choice is to allocate all equations of a single system to the same processor. Then, the systems can be solved locally by Gaussian elimination. No communication is required and the number of arithmetic operations is the minimum of the methods considered here. The solution is trivially parallel.

If the tridiagonal systems arise in connection with the solution of some form of partial differential equation in two or more dimensions, then other considerations may guide the allocation of lattice points to processors. With a one-dimensional partitioning of the lattice all tridiagonal systems may be allocated identically over the entire cube. This allocation of equations is assumed in this section. We will consider two-dimensional partitionings in the next section.

It follows from our analysis (and experiments) of single system solvers that parallel cyclic reduction and broadcasting followed by Gaussian elimination are not of interest for the multiple systems case. Repeated application of odd-even cyclic reduction for each system results in poor load balance, since the processor holding the middle equation is active in all reduction stages. *Balanced Cyclic Reduction* (BCR) [13] balances the load by performing the reduction such that for a set of N systems the reduction process converges to a unique processor for each system.

We consider three methods for solving multiple tridiagonal systems in this section: pipelined Gaussian elimination, combinations of transposition of systems of equations and substructuring, and balanced cyclic reduction. We also determine the optimum combination of these three algorithms as a function of machine parameters, the number of systems, and the size of each

Algorithm	Arithmetic	Byte transfers	Min start-ups	Memory
TGET(n)	$8P-7$	$20P(1-\frac{1}{N})$	$2n$	P
TGET(i)	$4P-2$	$20\frac{P}{N}(2^i-1)+2^{n-i+3}+8$	$2n+2^{n-i}-2(n-i)$	$\frac{P}{N}2^i$
BCGE(i)	$4P-2$	$32\frac{P}{N}(2^i-1)+2^{n-i+3}+8$	$2n+2^{n-i}-2(n-i)$	$\frac{P}{N}2^i$
SS/CR-1	$17\lceil\frac{P}{N}\rceil+16n-32$	$60n-40$	$6n-4$	$\frac{P}{N}$
SS/CR-2	$17\lceil\frac{P}{N}\rceil+16n-32$	$40n$	$4n$	$\frac{P}{N}+\log N$
SS/PCR-2,3	$17\lceil\frac{P}{N}\rceil+12n-17$	$96n+8$	$6n+2$	$\frac{P}{N}$
SS/TGET(n)	$17\lceil\frac{P}{N}\rceil+8N-24$	$20N+20$	$2n+4$	$\frac{P}{N}+N$
SS/TGET(i)	$17\lceil\frac{P}{N}\rceil+4N-19$	$20(2^i-1)+2^{n-i+3}+48$	$2n+2^{n-i}-2(n-i)+4$	$\frac{P}{N}+2^i$
SS/BCGE(i)	$17\lceil\frac{P}{N}\rceil+4N-19$	$32(2^i-1)+2^{n-i+3}+48$	$2n+2^{n-i}-2(n-i)+4$	$\frac{P}{N}+2^i$

Table 1: Complexity comparison of algorithms for single tridiagonal system.

system.

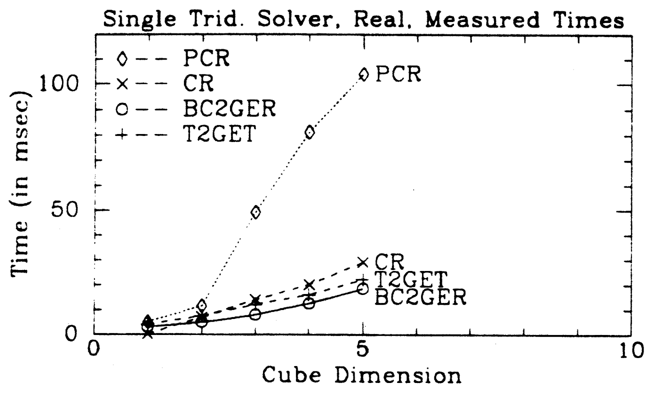
3.1 Pipelined Gaussian Elimination

In the case of two-way pipelined Gaussian elimination a processor performs forward elimination on one system, sends the last equation of that system to the next processor, then continues with the first equation it has of the next system, etc. The two middle processors solve for the appropriate variables, and the backsubstitution proceeds in a fashion similar to the forward elimination. The number of communication start-ups can be reduced by communicating the data for several equations at once, causing a reduction in the arithmetic efficiency of the pipe of processors. (However, blocking of equations may increase the utilization of each processor, if the functional units of a processor are internally pipelined.)

Let $T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,i)$ be the time for pipelined two-way elimination with 2^i systems per packet. Then, the complexity of a pipelined two-way elimination with one system per packet is

$$T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,0) = \qquad (10)$$
$$\left\{\frac{8P}{N}(Q+\frac{N}{2}-1)+3Q-5\right\}t_a+\left\{4(Q+\frac{N}{2}-2)+2\left\lceil\frac{12Q}{B_m}\right\rceil\right\}\tau+(56Q+16N-64)t_c.$$

for $Q>1$. During the forward phase, the number of floating-point operations is $\frac{5P}{N}$ per system per processor (the first and the last processors have 5 less operations). During the backward phase, the number of floating-point operations is $\frac{3P}{N}$ per system per processor. The two middle processors have 3 more operations per system due to the fact that 6 operations are required to solve the two by two system of the two middle processors. The first term in the expression for the number of start-ups contains a factor of four: a factor of two is due to forward and backward



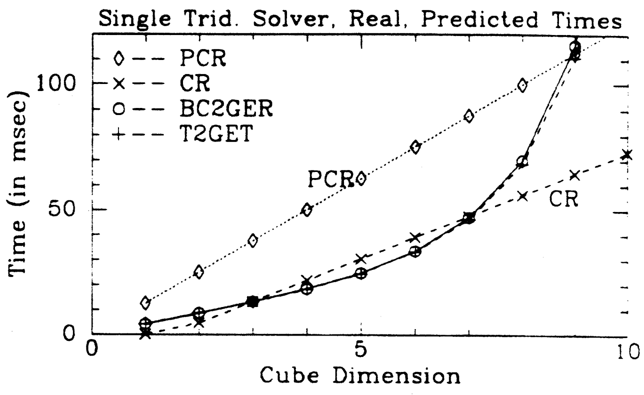


Figure 6: Measured times (on the left) and predicted times (on the right) of the CR-2, PCR-3, TGET($n-1$)-cc and BCGE($n-1$) algorithms on the Intel iPSC/1 for matrices of real elements. $P = N$ ($P = N - 1$ for CR-2).

elimination, and another factor of two is due to the sending *and* receiving of data during the pipelining. Four start-ups are subtracted totally for the first and the last step of the pipelining, for both forward and backward phases. It is assumed that $B_m \geq 12$. The second term in the expression for the number of start-ups accounts for the exchange of the $3Q$ floating-point numbers between the two middle processors. It is assumed that the exchange is done for all the Q systems together. The number of element transfers are

$$2 \cdot 4(3+1)(Q + \frac{N}{2} - 1 - 1) + 2 \cdot 4 \cdot 3Q.$$

For 2^i systems per packet, $0 \leq i < \log_2 Q$, the complexity estimate is

$$\begin{aligned} T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,i) &= \left\{\frac{8P}{N}(Q + 2^i(\frac{N}{2} - 1)) + 3Q - 5 \cdot 2^i\right\} t_a \qquad (11) \\ &+ \left\{2\left(\left\lceil\frac{12 \cdot 2^i}{B_m}\right\rceil + \left\lceil\frac{4 \cdot 2^i}{B_m}\right\rceil\right)(\frac{Q}{2^i} + \frac{N}{2} - 2) + 2\left\lceil\frac{12Q}{B_m}\right\rceil\right\}\tau \\ &+ (56Q + 2^{i+4}N - 2^{i+6})t_c. \end{aligned}$$

The number of element transfers are

$$2 \cdot 4(3+1) \cdot 2^i(\frac{Q}{2^i} + \frac{N}{2} - 1 - 1) + 2 \cdot 4 \cdot 3Q.$$

For $2^i = Q$, the complexity is

$$\begin{aligned} T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,\log Q) &= Q(4P-2)t_a + 8Q(N+1)t_c \qquad (12) \\ &+ \left\{(\frac{N}{2}+1)\left\lceil\frac{12Q}{B_m}\right\rceil + (\frac{N}{2}-1)\left\lceil\frac{4Q}{B_m}\right\rceil\right\}\tau. \end{aligned}$$

Note that the coefficients of τ and t_c in equation (11) does not specialize to the coefficients of τ and t_c in equation (12), because the communication changes from "send *and* receive" to "send *or* receive". Equation (11) specializes to equation (10) when $i = 0$. When $N = 2$ and $2^i = Q$, both equations (11) and (12) degenerate to two-way elimination on two processors without pipelining. The complexity is

$$Q(4P-2)t_a + 2\left\lceil \frac{12Q}{B_m} \right\rceil \tau + 24Qt_c.$$

The optimal value of 2^i is $\approx \sqrt{\frac{Q\tau}{Pt_a+4Nt_c}}$ assuming $B_m \geq 12Q$, and

$$T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,i_{opt}) \approx \frac{8PQ}{N}t_a + 2(N-3)\tau + 56Qt_c + 8\sqrt{Q\tau(Pt_a+4Nt_c)}.$$

3.2 Equation Transposition and Substructured Elimination

If all the systems are identically distributed across the entire cube, then a transpose operation on the data structure results in the trivially parallel case. A transpose operation on the result of the solution to the equations may be required. This procedure is the multiple systems version of algorithm TGET(n) (or TGET($n-1$)). The number of arithmetic operations is the same as in the pipelined case, but there is no inefficiency in the use of arithmetic units. With $Q \bmod N = 0$, all processors perform an equal amount of work in the solution process. Two-way elimination does not offer any reduction in arithmetic complexity, unlike in the single system case.

The transposition of the set of equations gathers the equations of $\frac{Q}{N}$ systems into a single processor (and different processors for different sets). Each step of the transposition doubles the number of equations of a system that are present in a processor (and reduces the number of different systems in a processor by a factor of two). It may be advantageous to perform substructuring after every few transpose steps to reduce the data volume that needs to be communicated.

The transposition can be performed through a sequence of exchanges in the different dimensions. Such an algorithm is optimal with communication restricted to one port at a time. With concurrent communication on all ports of every processor, the communication time can be reduced by a factor that is at most equal to the dimension of the (sub)cube in which the systems to be gathered are allocated [16]. Since each step for $Q \bmod N = 0$ is an exchange step, the transposition in the multiple systems case requires at least twice the number of start-ups of the single system case. Additional start-ups may be required for a limited buffer size B_m, since half of the data set being transposed takes part in every exchange.

The complexity of the initial substructuring is

$$T_{SS}(P,Q;N,t_a,t_c,\tau,B_m) = 17Q(\left\lceil \frac{P}{N} \right\rceil - 1)t_a + 2(16+4)Qt_c + 2(\left\lceil \frac{16Q}{B_m} \right\rceil + \left\lceil \frac{4Q}{B_m} \right\rceil)\tau, \quad (13)$$

and the complexity of one gather and one scatter step of the transpose algorithm is

$$T_{sbt}(P,Q;N,t_c,\tau,B_m) = \frac{20QP}{N}t_c \quad (14)$$
$$+ \left(\left\lceil \left\lceil \frac{Q}{2} \right\rceil \frac{16P}{NB_m} \right\rceil + \left\lceil \left\lfloor \frac{Q}{2} \right\rfloor \frac{16P}{NB_m} \right\rceil + \left\lceil \left\lceil \frac{Q}{2} \right\rceil \frac{4P}{NB_m} \right\rceil + \left\lceil \left\lfloor \frac{Q}{2} \right\rfloor \frac{4P}{NB_m} \right\rceil \right) \tau.$$

For i steps of the transpose operation starting with P equations followed by substructuring (assuming $Q \bmod 2^i = 0$) we have

$$T_{sbt}(P,Q;N,t_c,\tau,B_m,i) + T_{SS}(P,Q;\frac{N}{2^i},t_a,t_c,\tau,B_m) = 2i\left(\left\lceil\frac{8QP}{NB_m}\right\rceil + \left\lceil\frac{2QP}{NB_m}\right\rceil\right)\tau + \frac{20iQP}{N}t_c$$
$$+ 17Q\frac{P}{N}(1-\frac{1}{2^i})t_a + 2\left(\left\lceil\frac{16Q}{2^iB_m}\right\rceil + \left\lceil\frac{4Q}{2^iB_m}\right\rceil\right)\tau + \frac{40QP}{2^iN}t_c,$$

where the first row is the transpose time, and the second row the substructuring time. If the procedure is repeated recursively, then $\frac{P}{N} = 1$ for all applications (since the substructuring always reduces the number of remaining equations per processor to one per system), except possibly for the first. In the last application it may also be the case that substructuring is replaced by (two-way) Gaussian elimination (if N is 1 or 2).

In the above complexity expressions the time for local data movement is ignored. For some computers, like the Intel iPSC/1 this approximation is very poor [14]. The transpose operation implies a recursive partitioning of data sets into smaller and smaller blocks. With a high start-up time for communication it may be desirable to form messages of several blocks to fill a communication buffer. Accounting for the local data motion time, or many start-ups because of many messages being smaller than B_m, can significantly alter the complexity of the algorithm. With a copy time of t_{cp} for one byte, the copy time can be accounted for by using the following expression.

$$\begin{aligned} T_{sbt}(P,Q;N,t_c,\tau,B_m,t_{cp},i) &= 2\sum_{j=1}^{i}\left(\min\left(2^{j-1}\left\lceil\frac{16PQ}{2^jNB_m}\right\rceil\tau, \left\lceil\frac{8PQ}{NB_m}\right\rceil + T_{cp\text{-}f}\right)\right. \\ &\quad \left. + \min\left(2^{j-1}\left\lceil\frac{4PQ}{2^jNB_m}\right\rceil\tau, \left\lceil\frac{8PQ}{NB_m}\right\rceil + T_{cp\text{-}b}\right)\right) + \frac{20iQP}{N}t_c \end{aligned} \tag{15}$$

where $T_{cp\text{-}f} = \frac{8PQ}{N}t_{cp}$ and $T_{cp\text{-}b} = \frac{2PQ}{N}t_{cp}$ are the copy times for half of the local array for the forward and backward phases, respectively. Note that if $\frac{16PQ}{2^iN} \geq B_m$, then no extra overhead is required due to local data motion during transposition. For the Intel iPSC/1 the copy time for the communications buffer of size $1k$ bytes is approximately equivalent to 4τ. Including the copy time yields an expression of the form $(1+2+4+4+\cdots+4)\tau$ compared to $(1+1+1+1+\cdots+1)\tau$ if the copy time is ignored, assuming $\frac{8PQ}{N} \leq B_m$. This difference is large enough to affect the choice of solution method for multiple tridiagonal systems.

An alternative to the optimized transpose algorithm used for the estimates above is to use the router. However, on the Intel iPSC/1 matrix transposition by the router software is inferior by approximately a factor of 5 for large (relative to the packet size) matrices, and by two orders of magnitude for small matrices, Table 2.

For the arithmetic and communication parameters of the Intel iPSC/1, the complexity estimates of an $n-1$ step transpose followed by two-way elimination and the same algorithm with substructuring preceding the transpose are compared in Figure 7 for up to 5-cubes. Measured times are given in the same figure. We conclude that substructuring is not advantageous for the Intel iPSC/1. Note the importance of including the copy time for a good agreement between measured and predicted times.

P, Q	cube dim.	Algorithm	
		optimum	router
1024	4	1.8 sec	8.7 sec
1024	5	1.3 sec	6.9 sec
128	4	0.058 sec	4.4 sec
128	5	0.075 sec	9.0 sec

Table 2: Matrix transposition on the Intel iPSC/1 for one-dimensional partitioning.

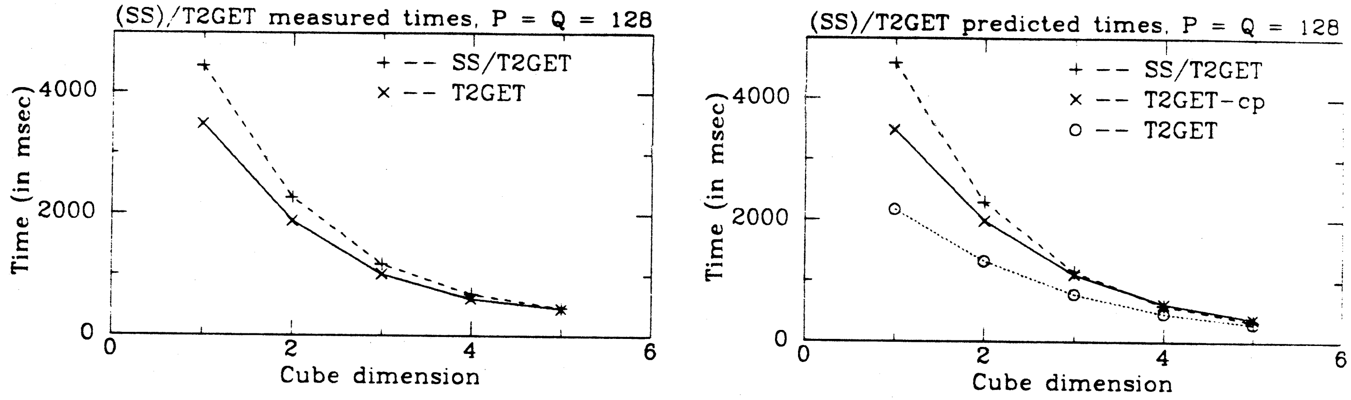


Figure 7: The measured and predicted times for the T2GET and SS/T2GET algorithms. T2GET-cp includes the copy time.

In the substructured as well as the non-substructured estimates above we assume that all the data for an equation is moved in the transposition. But, in some instances it is possible to store the system matrix pre-transposed, and only transpose the right hand side and the solution vector.

3.3 Balanced Cyclic Reduction

The arithmetic complexity of the substructured elimination is the same as that of cyclic reduction. but cyclic reduction reduces the communication requirements. After the substructured elimination each processor has one equation of each of the Q systems. In the BCR algorithm. the reduction process for each set of $\frac{Q}{N}$ systems "converges" to a distinct processor, thereby keeping the load on the processors balanced. For each step of the reduction phase, the Q systems are partitioned into successively smaller subsets by virtue of the equations on which the reduction is performed. For instance, in the first step the set of equations is partitioned in half: in one half elimination is performed on even equations, in the other on odd equations. By numbering equations and systems from 0 and letting the elimination on even equations be performed for even systems, the lowest order bit in the binary encoding of the equation and system indices can be used to control the communication and elimination operations. Another obvious choice is to perform the elimination on even equations for the first half of the systems, in which case the highest order bit in the system index and the lowest order bit in the equation index are used for the control in the first step. By performing the operation recursively, the control proceeds toward higher/lower order bits.

The BCR algorithm can be implemented as an *exchange* algorithm with exchanges of equations between adjacent processors between elimination steps in order to keep all equations subject to further elimination in easily identifiable subcubes for each system. Such an exchange algorithm requires 3 exchanges for each step in the reduction phase, and 3 in the backsubstitution phase. Hence, with one send *or* one receive at a time, a total of approximately $12n$ start-ups is required, if the buffer size $B_m \geq 8Q$ bytes for real systems. The scheme is then applied to both subcubes recursively. Notice that 4 of these 12 communications can be saved, if the computations at each step are split into two parts — those depending on the preceding and succeeding row respectively (as described for the CR-2 algorithm). Figure 8-(a) and (b) show the four communication steps for one step of the BCR algorithm in the forward phase; (c) and (d) are another alternative of the same complexity.

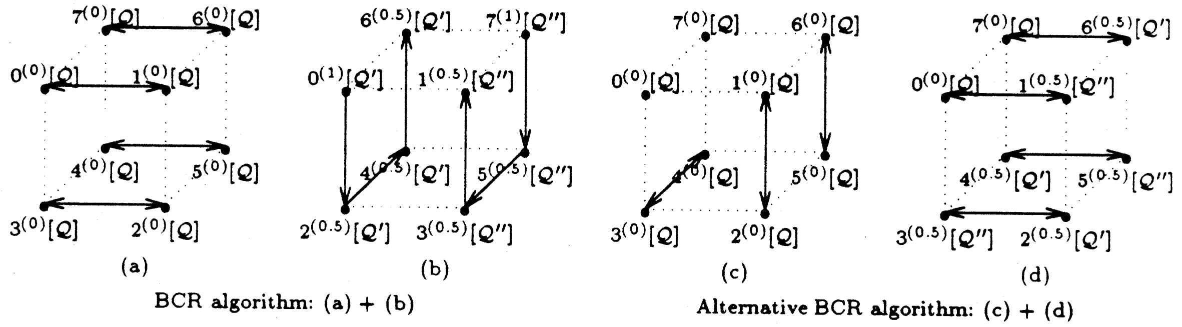


Figure 8: The four communication steps for one step of BCR algorithm in the forward phase. $Q = Q' \cup Q''$ is the set of Q systems. In (a) the left subcube sends its equations of the set Q'' and receives its neighbor's equations of the set Q'. In (b) equations $1^{(0)}[Q']$, $3^{(0)}[Q']$ and $5^{(0)}[Q']$ are sent to equations $2^{(0.5)}[Q']$, $4^{(0.5)}[Q']$ and $6^{(0.5)}[Q']$, respectively; the communication in the right subcube is similar.

The estimated times for the *exchange* version of the BCR and SS/BCR algorithms for Q mod $N = 0$ are

$$T_{BCR}(N,Q;N,t_a,t_c,\tau,B_m) = 17Q(1-\frac{1}{N})t_a + \{4(16+4)Q(1-\frac{2}{N}) + 2(16+4)\frac{Q}{N}\}t_c \quad (16)$$
$$+\left\{4\sum_{i=1}^{\log N-1}\left(\left\lceil\frac{16Q}{2^iB_m}\right\rceil + \left\lceil\frac{4Q}{2^iB_m}\right\rceil\right) + 2\left(\left\lceil\frac{16Q}{NB_m}\right\rceil + \left\lceil\frac{4Q}{NB_m}\right\rceil\right)\right\}\tau.$$

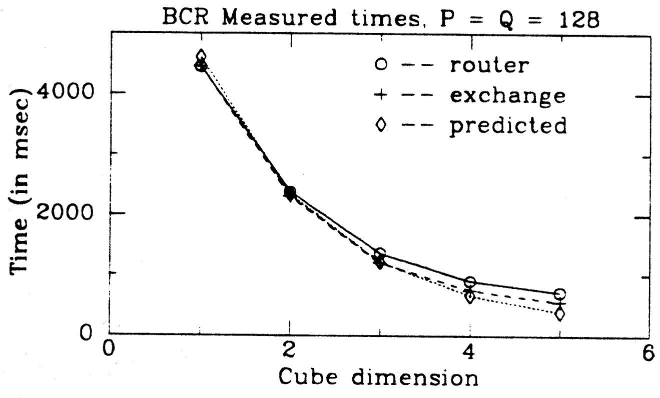
$$T_{SS/BCR}(P,Q;N,t_a,t_c,\tau,B_m) = T_{SS}(P,Q;N,t_a,t_c,\tau,B_m) + T_{BCR}(N,Q;N,t_a,t_c,\tau,B_m). \quad (17)$$

For artitrary Q and $P = N$, the complexity of one-step BCR is

$$T_{bcr}(Q;t_a,t_c,\tau,B_m) \;=\; 17\left\lceil\frac{Q}{2}\right\rceil t_a + 40Qt_c \quad (18)$$

$$+2\left(\left\lceil\left\lceil\frac{Q}{2}\right\rceil\frac{16}{B_m}\right\rceil+\left\lceil\left\lfloor\frac{Q}{2}\right\rfloor\frac{16}{B_m}\right\rceil+\left\lceil\left\lceil\frac{Q}{2}\right\rceil\frac{4}{B_m}\right\rceil+\left\lceil\left\lfloor\frac{Q}{2}\right\rfloor\frac{4}{B_m}\right\rceil\right)\tau.$$

Hence, the complexity of i-steps of BCR for arbitrary Q can be derived by summing the complexity of each step with the first parameter being Q, $\lceil Q/2\rceil$, $\lceil\lceil Q/2\rceil/2\rceil$, etc. An alternative to the exchange based BCR algorithm is to use an *in-place* algorithm. For such an algorithm we choose to use the router of the Intel iPSC/1. The result of implementing the two versions of BCR is shown in Figure 9. The algorithm using the router requires 50 – 100% more time for solving the reduced systems on the Intel iPSC/1. The total time for the router based *in-place* algorithm is about 30% higher than that of the algorithm using exchanges for each step in the case of a 5-cube. For a fixed size problem the difference in the total time increases as the number of cube dimensions increases, since the substructuring part reduces in significance.



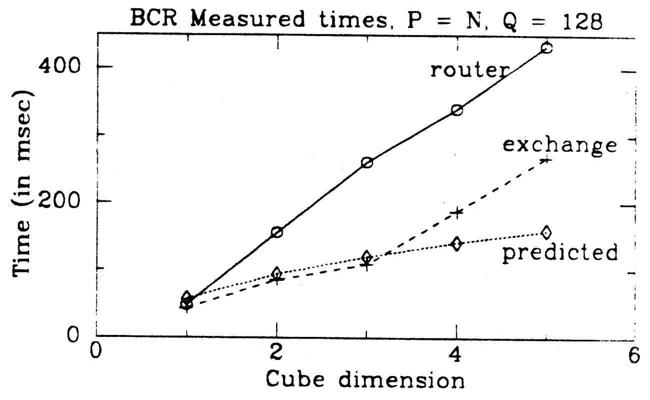


Figure 9: Measured and predicted times of balanced cyclic reduction (BCR) using the *exchange* and the *in-place* algorithms.

The predicted times for solving the reduced systems is lower than the measured times by approximately 30 – 70 % for the 4- and 5-cubes mainly due to the synchronization delay. In fact, an algorithm with each communication step involving all the processors tends to have a more significant delay for a larger cube. This characteristic is often the reason why the difference between the measured time and the predicted time increases with the cube size, Figure 9.

With concurrent communication on all ports of every processor the number of start-ups can be reduced by a factor of two. Hence, each BCR step in the forward phase requires two communication steps, Figure 8. This number of start-ups is minimal because in each step the processors that need to communicate are a distance two apart. However, it is possible to reduce the data transfer time by a factor of up to four, Figure 8, by pipelining (a) and (b), or by overlapping the two alternatives ((a) + (b) and (c) + (d)) with each working on half of the systems. The overlapping algorithm does not cause any increase in the number of start-ups, and therefore is better than the pipelined algorithm.

3.4 Hybrid Algorithms

The optimum choice of algorithm depends on machine parameters, the number of equations per system P, and the number of systems Q. In the balanced cyclic reduction method, and in the transpose and substructuring method, a problem is recursively changed into a number of problems on independent, smaller subcubes. The choice of algorithm should be reevaluated during the recursion process. The recursion can be stopped at any point by pipelined Gaussian elimination. The optimum method is determined by the following set of equations

$$T_{trid}(P,Q;N,t_a,t_c,\tau,B_m) \leq \begin{cases} T_{2GE}(P,Q;2,t_a,t_c,\tau,B_m), & \text{if } N=2, \\ \min\{T_{2GE}(N,Q;N,t_a,t_c,\tau,B_m), & \\ \quad T_{sbt}(N,Q;N,t_c,\tau,B_m)+T_{trid}\left(N,\left\lceil\frac{Q}{2}\right\rceil;\frac{N}{2},t_a,t_c,\tau,B_m\right), & \\ \quad T_{bcr}(Q;t_a,t_c,\tau,B_m)+T_{trid}\left(\frac{N}{2},\left\lceil\frac{Q}{2}\right\rceil;\frac{N}{2},t_a,t_c,\tau,B_m\right)\}, & \text{if } P=N, \\ \min\{T_{2GE}(P,Q;N,t_a,t_c,\tau,B_m), & \\ \quad T_{sbt}(P,Q;N,t_c,\tau,B_m)+T_{trid}\left(P,\left\lceil\frac{Q}{2}\right\rceil;\frac{N}{2},t_a,t_c,\tau,B_m\right), & \\ \quad T_{SS}(P,Q;N,t_a,t_c,\tau,B_m)+T_{trid}(N,Q;N,t_a,t_c,\tau,B_m)\}, & \text{otherwise,} \end{cases}$$

$$\begin{aligned} \text{where } \quad T_{2GE}(P,Q;N,t_a,t_c,\tau,B_m) &= \min_{0\leq i\leq \log_2 Q}\{T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,i)\}, \\ T_{SS}(P,Q;N,t_a,t_c,\tau,B_m) &= 17Q(\frac{P}{N}-1)t_a+2\left(\left\lceil\frac{16Q}{B_m}\right\rceil+\left\lceil\frac{4Q}{B_m}\right\rceil\right)\tau+40Qt_c, \\ T_{sbt}(P,Q;N,t_c,\tau,B_m) &= 2\left(\left\lceil\frac{8PQ}{NB_m}\right\rceil+\left\lceil\frac{2PQ}{NB_m}\right\rceil\right)\tau+\frac{20QP}{N}t_c, \\ T_{bcr}(Q;t_a,t_c,\tau,B_m) &= 17\left\lceil\frac{Q}{2}\right\rceil t_a+4\left(\left\lceil\frac{8Q}{B_m}\right\rceil+\left\lceil\frac{2Q}{B_m}\right\rceil\right)\tau+40Qt_c. \end{aligned}$$

It is assumed that $Q \bmod 2 = 0$. If this is not true, then T_{sbt} becomes Equation (14) and T_{bcr} becomes Equation (18). $T_{2GE\text{-}p}(P,Q;N,t_a,t_c,\tau,B_m,i)$ is the time for pipelined two-way elimination with 2^i systems per packet, equation (11). T_{2GE} is the time with optimal packet size 2^i.

Figure 10 shows the general calling sequencies for $Q = kN$. The label on an edge of the graph is defined as in Figure 3. The two arguments of T_{SS}, and T_{sbt}, are $\frac{P}{N}$ and Q. The arithmetic time of T_{SS}, and the data transfer time of T_{sbt}, are proportional to the product of its two arguments. Figure 11-(a) through (c) shows how the calling sequencies depend on the maximum packet size for $t_a = 0$, $\tau = 1$, and $t_c = 0$. In the figure pipelined two-way Gaussian elimination with a block size of Q equations and one equation are represented by "o" and "$*$" signs, respectively. When B_m is increased to ∞, part of the region for substructuring is replaced by transposition. Figure 11-(d) through (f) gives the optimum calling sequencies if both arithmetic time and data transfer times are accounted for, and the communications overhead is high. Figure 12 shows the calling sequencies for a set of parameter values for which the value of B_m is irrelevant.

In the figures showing the optimum choice of algorithm assuming $t_{cp} = 0$ the BCR algorithm is never used, whereas with a large value of t_{cp}, the BCR algorithm may be preferable, as seen

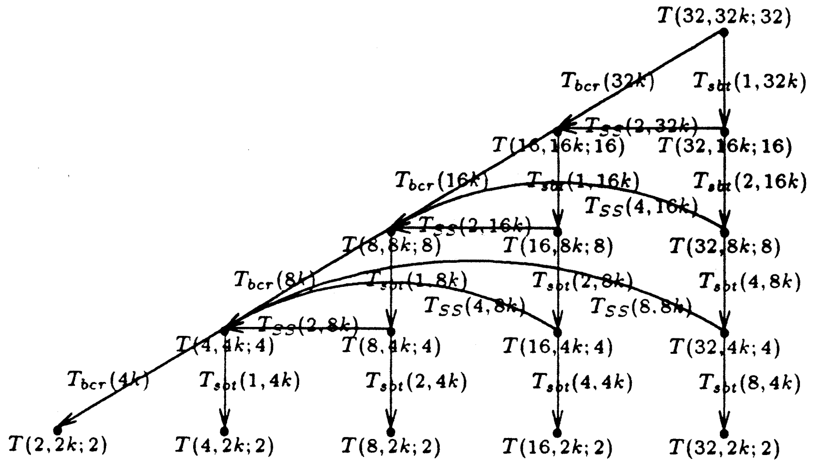


Figure 10: The general calling sequencies for $32 \geq P \geq N$ and $Q = kN$.

from Figure 11-(e) $t_{cp} = 0$ and Figure 13 $t_{cp} = 4$ (the value for the Intel iPSC/1). As B_m increases, the region for BCR extends to larger N along the $P = N$ diagonal line.

The time for i steps of the BCR algorithm is

$$\sum_{j=0}^{i-1} T_{bcr}(\frac{Q}{2^j}; t_a, t_c, \tau, B_m) = 17Q(1-\frac{1}{2^i})t_a + 4\sum_{j=0}^{i-1}\left(\left\lceil \frac{8Q}{2^j B_m}\right\rceil + \left\lceil \frac{2Q}{2^j B_m}\right\rceil\right)\tau + 80Q(1-\frac{1}{2^i})t_c, \quad (19)$$

assuming $P = N$ and $Q \bmod 2^i = 0$. If the time for local data motion can be ignored then i-steps of the transpose algorithm followed by substructuring is

$$\sum_{j=0}^{i-1} T_{sbt}(N, \frac{Q}{2^j}; \frac{N}{2^j}, t_a, t_c, \tau, B_m) + T_{SS}(P, \frac{Q}{2^i}; \frac{N}{2^i}, t_a, t_c, \tau, B_m) = \quad (20)$$

$$17Q(1-\frac{1}{2^i})t_a + \left\{2i\left(\left\lceil \frac{8Q}{B_m}\right\rceil + \left\lceil \frac{2Q}{B_m}\right\rceil\right) + 2\left(\left\lceil \frac{16Q}{2^i B_m}\right\rceil + \left\lceil \frac{4Q}{2^i B_m}\right\rceil\right)\right\}\tau + (20iQ + \frac{40Q}{2^i})t_c.$$

In order to account for the local data motion time the first term in the coefficients for both τ and t_c need to be changed according to Equation (15).

Note that for $i = 1$ Equations (19) and (20) yield the same complexity. For $i > 1$ Equations (19) and (20) have the same arithmetic complexity. For $i > 1$ and $B_m \geq 8Q$, the start-up time of (19) is always higher than that of (20). The ratio is two asymptotically. For a limited buffer size, the number of start-ups eventually compares as the data transfer times. The data transfer time of Equation (19) is smaller than that of Equation (20), except if $i = 2$ and 3. Hence, with $t_{cp} = 0$ the complexity of one step of BCR is the same as that of one step transposition followed by substructuring. The complexity of two steps (three steps) of BCR is always higher than that of two-step (three-step) transposition followed by substructuring, independent of the maximum buffer size. With $t_{cp} = 0$, BCR is never preferable over transposition and substructuring with respect to computational complexity for Q being a multiple of N. The optimum number of transposition steps for each substructuring depends on the values of t_a, τ, t_c, B_m and Q. With local data motion requiring a significant time the BCR algorithm becomes competitive, especially for relatively large maximum packet size.

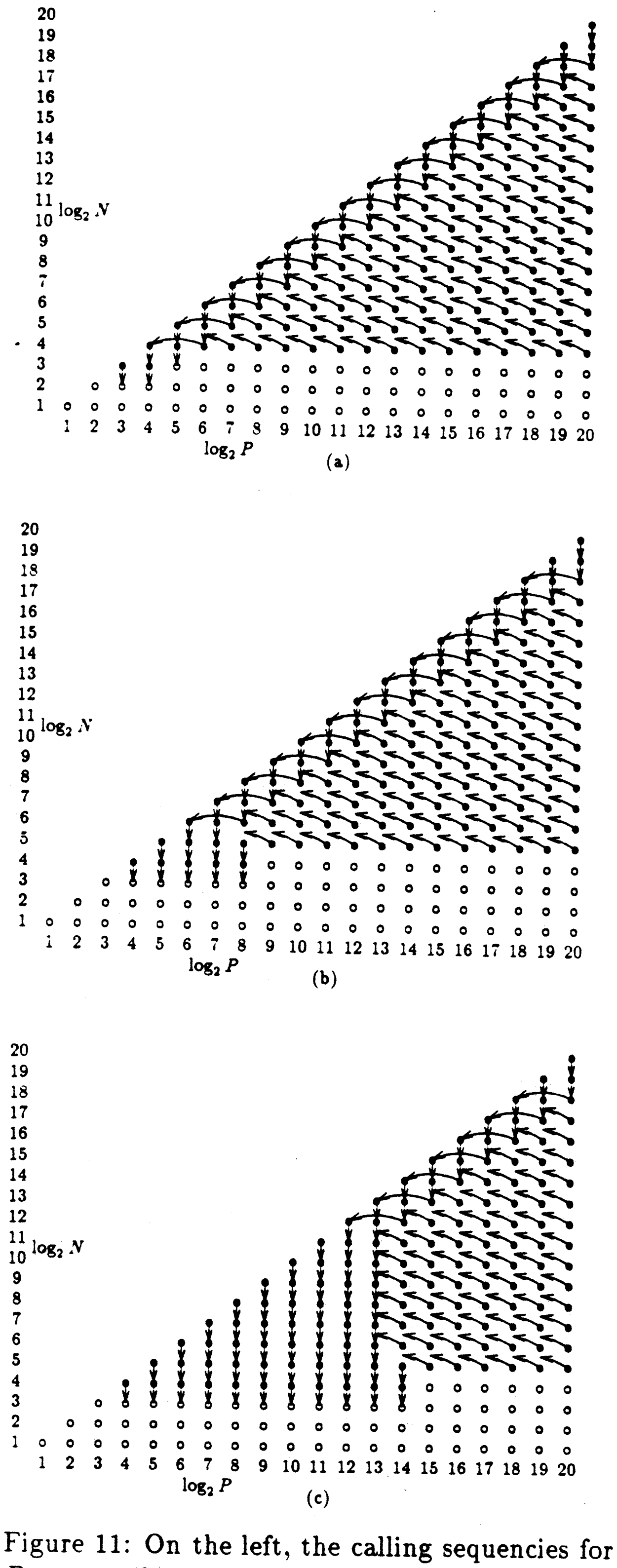
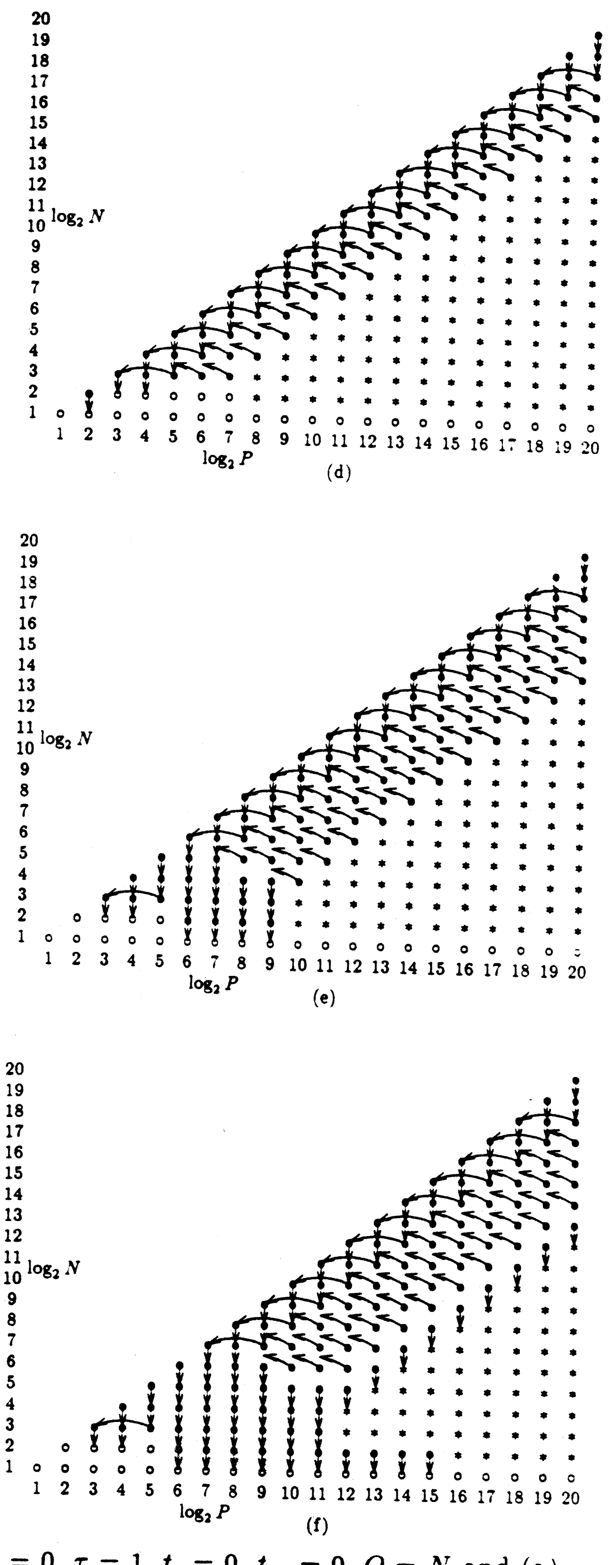



Figure 11: On the left, the calling sequencies for $t_a = 0$, $\tau = 1$, $t_c = 0$, $t_{cp} = 0$, $Q = N$ and (a) $B_m = 16$, (b) $B_m = 1024$ and (c) $B_m = 2^{16}$. On the right, the calling sequencies for $t_a/t_c = 10$, $\tau/t_c = 1000$, $t_{cp} = 0$, $Q = N$ and (d) $B_m = 16$, (e) $B_m = 1024$ and (f) $B_m = 2^{16}$.

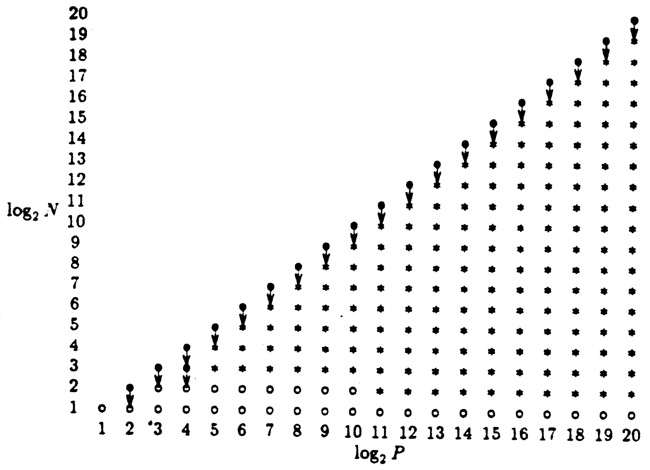


Figure 12: The calling sequencies for $t_a/t_c = 0.01$, $\tau/t_c = 1$, $B_m \geq 16$, $t_{cp} = 0$ and $Q = N$.

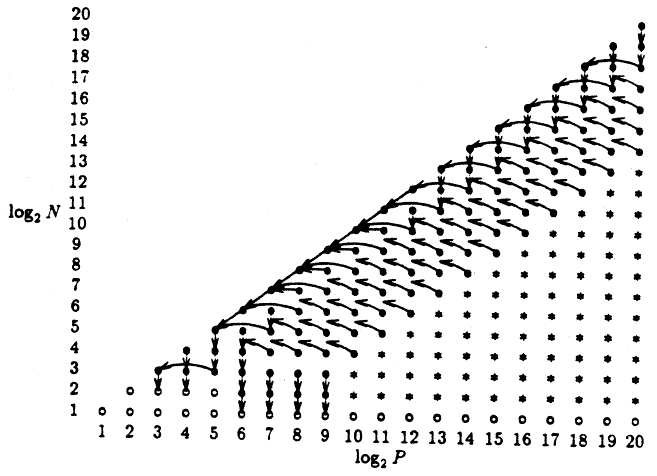


Figure 13: The calling sequencies for $t_a/t_c = 10$, $\tau/t_c = 1000$, $B_m = 1024$, $t_{cp}/t_c = 4$ and $Q = N$.

3.5 Summary of Algorithms for Solution of Multiple Systems.

From the above analysis, and the recursion formulas for optimum choice of solution method we conclude that for multiple systems:

- Relatively high arithmetic time favors pipelined Gaussian elimination and transposition without substructuring.
- Relatively high data transfer time favors pipelined Gaussian elimination instead of transposition.
- Relatively high communication start-up time favors transposition instead of pipelined Gaussian elimination.
- The block size for pipelined Gaussian elimination tends toward all equations when the arithmetic time is insignificant, otherwise the block size tends to be small (one equation) to maximize the use of the processor pipeline.
- A smaller communications buffer tends to favor substructuring instead of transposition.
- For a fixed N, doubling the number of equations almost doubles the arithmetic time for pipelined Gaussian elimination and leaves the communication time unchanged. With substructuring the total arithmetic time approximately doubles, but increases relatively somewhat more than for Gaussian elimination. The communication time is unchanged. Increasing P for fixed N and Q favors pipelined Gaussian elimination.
- Doubling both N and Q for fixed $P > 2N$ keeps the total work per processor constant. The start-up time and data transfer times for pipelined Gaussian elimination approximately doubles. For transposition and substructuring the data transfer time also approximately doubles, but there are only four additional start-ups. Hence, as N increases substructuring is favored.
- With insignificant time for local data motion the BCR algorithm is never competitive with the transpose/substructuring algorithm with respect to computational complexity. With a high cost for local data motion the conclusion may become the opposite because in the BCR algorithm Q/2 active systems out of Q can be selected to be continuous independent of the step. Increasing the maximum packet size favors the BCR algorithm more than the transpose/substructuring algorithm as far as copy time is concerned.
- The communication time of the transpose/substructuring algorithm can be sped up by at most a factor equal to the dimension of the subcube from which data are gathered (i for i-step transposition) with concurrent communication on all ports of every processor. A similar speed-up is not possible for the BCR algorithm, because of communication conflicts.

4 Two-Dimensional Decomposition

In the one-dimensional decomposition for the solution of multiple tridiagonal systems the choice of algorithm is either trivial (embarrassingly parallel), or the analysis above should guide the choice of algorithm. The non-trivial case is of particular interest for two- or higher dimensional problems, and solution methods such as fast Poisson solvers and the Alternating Direction Method.

4.1 One Directional Solution

The one-dimensional analysis is easily generalized to the two-dimensional case by assuming that Q/N_2 systems are distributed to each of N_2 separate subcubes consisting of N_1 processors each, where $N_1 \times N_2 = N$. The allocation of partitions to processors within each subcube is made by a binary-reflected Gray code. The time for the solution of the systems of equations is given by $T_{trid}(P, \frac{Q}{N_2}; N_1, t_a, t_c, \tau, B_m)$, where $N = N_1 \times N_2$. It is fairly easy to verify that the solution time is minimized if N_2 is maximized. The one-dimensional partitioning yielding the "embarrassingly" parallel case is optimum.

4.2 Alternating Direction Methods (ADM)

We consider a grid of $P \times Q$ internal points, and embed it in an $N_1 \times N_2$ processor mesh, that in turn is embedded in a Boolean n-cube by a two-dimensional binary-reflected Gray code. Thus, each processor is assigned $\frac{PQ}{N}$ grid points. The two-dimensional Gray code ensures that each row and column of the processor mesh is itself a subcube, and that adjacency is preserved for each row and column. One ADM step consists of two half steps, each of which implies a number of tridiagonal matrix-vector multiplications and the solution of an equal number of tridiagonal systems. Each of the vectors in the matrix-vector multiplication represents the solution variables along a row (column) of the computational grid, and the tridiagonal matrix the approximation of derivatives along the same row (column). Similarly, tridiagonal systems are solved for each row (column). The second half step is the complement of the first — one forms the matrix vector products along the grid rows and solves tridiagonal systems along columns.

One ADM step consists of two half steps,

$$\begin{aligned}(I - \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}} &= (I + \frac{1}{2}\Delta t B_y)u^i,\\ (I - \frac{1}{2}\Delta t B_y)u^{i+1} &= (I + \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}}.\end{aligned}$$

In each half step computations are performed independently on grid rows or columns, i.e., on independent subcubes [18] . The matrix vector product takes a time of $5\frac{PQ}{N}t_a$ for the arithmetic, and requires exchanging $\frac{Q}{N_2}$ floating-point numbers with nearest (north and south) neighbors for one of the half step. This gives a total time of

$$T_{mpy}(P, \frac{Q}{N_2}; N_1, t_a, t_c, \tau, B_m) = 5\frac{P}{N_1}\frac{Q}{N_2}t_a + 16\frac{Q}{N_2}t_c + \left\lceil \frac{4Q}{N_2 B_m} \right\rceil 4\tau, \qquad (21)$$

for the matrix vector products of one half step. Then $\frac{Q}{N_2}$ tridiagonal systems of order P each are solved on subcubes of N_1 processors.

Let $T_{ADM}(P,Q;N,t_a,t_c,\tau,B_m)$ be the time for one step of ADM. Then,

$$T_{ADM}(P,Q;N,t_a,t_c,\tau,B_m) \leq \tag{22}$$
$$\min_{0\leq\log N_1\leq\log N}\left\{T_{trid}(P,\frac{Q}{N_2};N_1,t_a,t_c,\tau,B_m)+T_{trid}(Q,\frac{P}{N_1};N_2,t_a,t_c,\tau,B_m)\right.$$
$$\left.+T_{mpy}(P,\frac{Q}{N_2};N_1,t_a,t_c,\tau,B_m)+T_{mpy}(Q,\frac{P}{N_1};N_2,t_a,t_c,\tau,B_m)\right\}$$

For matrix-vector multiplication and pipelined Gaussian elimination with one system per block, the leading term of the arithmetic complexity is independent of the shape of the processor array. A lower order term in the arithmetic complexity for the pipelined Gaussian elimination is optimized for $N_1 = \sqrt{\frac{PN}{Q}}$. The number of start-ups for the matrix-vector multiplication as well as one of the two major terms in the expression for the number of start-ups in the pipelined Gaussian elimination are minimized at $N_1 = \sqrt{\frac{PN}{Q}}$. Another major term contributing to the number of start-ups in the pipelined Gaussian elimination algorithm is minimized for $N_1 = \sqrt{N}$. The optimum choice of N_1 for the data transfer time is similar to that of start-up time. For a transpose algorithm followed by local Gaussian elimination, the complexity is independent of N_1 (for any B_m). For the SS/BCR algorithm, the arithmetic time is independent of N_1; the start-up time is independent of N_1 for large B_m, and minimized at $N_1 = \sqrt{\frac{PN}{Q}}$ for small B_m; the data transfer time is minimized at $N_1 = \sqrt{\frac{PN}{Q}}$. With this value of N_1, $\frac{P}{Q} = \frac{N_1}{N_2}$, i.e., the aspect ratio of the processor grid is the same as that of the physical grid.

Let $\check{N}_1 = 2^{\lfloor\frac{\log N+\log P-\log Q}{2}\rfloor}$ and $\hat{N}_1 = 2^{\lceil\frac{\log N+\log P-\log Q}{2}\rceil}$ be the two (or one) power-of-two numbers closest to $N_1 = \sqrt{\frac{PN}{Q}}$. Let $\tilde{N}_1$ be the choice of $\check{N}_1$ and $\hat{N}_1$ that yields the smallest execution time, and let $N_{1_{opt}}$ be the optimal value of N_1 for Equation (22) with T_{mpy} ignored (most of the time is spent on the tridiagonal part). Also let $\tilde{N}_2 = N/\tilde{N}_1$ and $N_{2_{opt}} = N/N_{1_{opt}}$. Using $\tilde{N}_1$ instead of $N_{1_{opt}}$ results in an execution time that is at most 20% higher than optimum for the set of machine parameters investigated earlier and $P = Q$. (The time for $\check{N}_1$ is the same as the time for $\hat{N}_1$ due to symmetricity.) The number of (P,N) pairs for which $\tilde{N}_1 \neq N_{1_{opt}}$ is a small fraction of the considered region of (P,N) pairs.

We also evaluated the non-optimality of $\tilde{N}_1$ for $1 \leq \log N \leq \log P = \log Q - 4 \leq 20$. In this case the largest deviation in execution time from the optimal value was 50%. Again, the region where the simplified choice results in measurable increase in execution time compared to the optimum choice is small.

The values of $\log N_{2_{opt}} - \log N_{1_{opt}} - \log \tilde{N}_2 + \log \tilde{N}_1$ for a few cases are shown in Figure 14. The closer this number is to zero, the closer $N_{1_{opt}}$ is to $\tilde{N}_1$. The figure also shows the consequencies of selecting an array aspect ratio as $\tilde{N}_1 \times \tilde{N}_2$ instead of $N_{1_{opt}} \times N_{2_{opt}}$ by giving the ratio of $r = T_{N_{1_{opt}}}/T_{\tilde{N}_1}$ where $T_{N_{1_{opt}}}$ and $T_{\tilde{N}_1}$ are the sum of the times for solving tridiagonal systems along the two axis with $N_1 = N_{1_{opt}}$ and $N_1 = \tilde{N}_1$, respectively. For Figures 14-(a) and (b), $r \geq 0.86$ for the considered domain. If B_m in Figures 14-(a) through (d) is reduced to 16, then $N_{1_{opt}} = \tilde{N}_1$ for the considered domain.

Based on our examples, decreasing the maximum packet size, or increasing the data transfer rate tend to yield $N_{1_{opt}} = \tilde{N}_1$.

20																			0
19																		1	1
18																	0	0	0
17																1	1	1	1
16															10	0	0	0	0
15														9	9	1	1	1	1
14													8	8	8	0	0	0	0
13												1	7	7	7	1	1	1	1
12											0	6	6	6	6	0	0	0	0
11										1	1	5	5	5	5	1	1	1	1
10									0	0	4	4	4	4	4	0	0	0	0
9								1	1	1	3	3	3	3	3	1	1	1	1
8							0	0	0	0	2	2	2	2	2	0	0	0	0
7						1	1	1	1	1	1	1	1	1	1	1	1	1	1
6					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(a)

20																			6
19																		5	1
18																	6	6	0
17																5	5	1	1
16															6	6	0	0	0
15														1	5	1	1	1	1
14													0	6	6	0	0	0	0
13												1	5	5	1	1	1	1	1
12											0	0	4	0	0	0	0	0	0
11										1	1	5	1	1	1	1	1	1	1
10									0	0	4	0	0	0	0	0	0	0	0
9								1	1	5	3	1	1	1	1	1	1	1	1
8							0	0	0	4	0	0	0	0	0	0	0	0	0
7						3	3	3	5	1	1	1	1	1	1	1	1	1	1
6					0	2	2	4	0	0	0	0	0	0	0	0	0	0	0
5				1	1	1	3	1	1	1	1	1	1	1	1	1	1	1	1
4			0	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(b)

20																			0
19																		-1	-1
18																	0	0	0
17																1	-1	-1	-1
16															6	0	0	0	0
15														5	5	-1	-1	-1	1
14													4	4	4	0	0	0	0
13												3	3	3	3	-1	-1	1	-1
12											2	2	2	2	2	0	0	0	0
11										1	1	1	1	1	1	-1	1	-1	-1
10									0	0	0	0	0	0	0	0	0	0	0
9								1	1	1	-1	-1	-1	1	-1	-1	-1	-1	-1
8							0	0	0	0	0	0	0	0	0	0	0	0	0
7						1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
6					-6	-6	-4	-4	-4	-4	-2	0	0	0	0	0	0	0	0
5				-5	-5	-5	-5	-5	-5	-3	-3	-3	-1	-1	-1	-1	-1	-1	-1
4			-4	-4	-4	-4	-4	-4	-4	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
3		-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(c)

20																			-2
19																		-3	-3
18																	-2	-2	-2
17																-3	-3	-3	-3
16															-4	-4	-2	-2	-2
15														5	-3	-3	-3	-3	-3
14													4	-4	-2	-2	-2	-2	-2
13												3	-3	-3	-3	-3	-3	-3	-3
12											4	-4	-4	-4	-4	-4	-4	-4	-4
11										5	5	-3	-3	-3	-3	-3	-3	-3	-3
10									4	4	-2	-2	-2	-2	-2	-2	-2	-2	-2
9								3	3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
8							2	2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
7						1	1	1	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
6					0	0	0	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
5				-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4			-4	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
3		-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(d)

20																			1
19																		1	1
18																	1	1	1
17																1	1	1	1
16															.97	1	1	1	1
15														.93	.97	1	1	1	1
14													.93	.93	.96	1	1	1	1
13												.92	.92	.92	.96	1	1	1	1
12											.92	.92	.92	.92	.96	1	1	1	1
11										1	1	1	1	1	1	1	1	1	1
10									1	1	1	1	1	1	1	1	1	1	1
9								1	1	1	1	1	1	1	1	1	1	1	1
8							1	1	1	1	1	1	1	1	1	1	1	1	1
7						1	1	1	1	1	1	1	1	1	1	1	1	1	1
6					.89	.89	.73	.73	.73	.73	.91	1	1	1	1	1	1	1	1
5				.86	.86	.86	.67	.67	.67	.67	.83	.90	1	1	1	1	1	1	1
4			.80	.80	.80	.80	.80	.80	.80	1	1	1	1	1	1	1	1	1	1
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(e)

20																			.99
19																		.99	1
18																	.96	.98	.99
17																.99	1	1	1
16															.96	.97	.98	.99	1
15														.98	.97	.99	.99	1	1
14													.96	.92	.96	.98	.99	.99	1
13												.97	.95	.97	.99	.99	1	1	1
12											.96	.92	.95	.97	.99	.99	1	1	1
11										.97	.98	.98	.99	.99	1	1	1	1	1
10									.92	.95	.93	.96	.98	.99	.99	1	1	1	1
9								.94	.96	.95	.98	.99	.99	1	1	1	1	1	1
8							.92	.93	.92	.96	.98	.99	.99	1	1	1	1	1	1
7						1	1	1	.98	.99	1	1	1	1	1	1	1	1	1
6					1	1	1	.97	.99	.99	1	1	1	1	1	1	1	1	1
5				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4			.87	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\log_2 N$ / $\log_2 P$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(f)

Figure 14: The values of $\log N_{2_{opt}} - \log N_{1_{opt}} - \log \tilde{N}_2 + \log \tilde{N}_1$ for (a) $t_a = 0$, $\tau = 1$, $t_c = 0$, $Q/P = 1$, (b) $t_a/t_c = 1$, $\tau/t_c = 1000$, $Q/P = 1$, (c) $t_a = 0$, $\tau = 1$, $t_c = 0$, $Q/P = 16$, and (d) $t_a/t_c = 1$, $\tau/t_c = 1000$, $Q/P = 16$; all with $B_m = 2^{16}$ and $t_{cp} = 0$. The ratio of $\frac{T_{N_{1_{opt}}}}{T_{\tilde{N}_1}}$ for the same sets of parameters as (c) and (d) are shown in (e) and (f), respectively.

5 Conclusions

The execution time for multiprocessors with a packet switched communication systems and nodes without pipelined arithmetic units can be accurately modeled by the start-up time for a communication, the data transfer time, maximum packet size, and the time for arithmetic operations. It may also be necessary to account for the local data motion time, as is the case for the Intel iPSC/1.

For a single tridiagonal system, the analysis as well as the experiments on the Intel iPSC/1 show that odd-even cyclic reduction can be competitive with parallel cyclic reduction with respect to arithmetic time, and in particular with respect to communication time. Odd-even cyclic reduction performed considerably better than parallel cyclic reduction on the Intel iPSC/1. A similar result has also been observed on the AMETEK system S14 [19].

For multiple tridiagonal systems of equations the domain in the parameter space in which pipelined Gaussian elimination requires less time than the other considered algorithms is increasing at the expense of substructuring with increasing time for an arithmetic operation. The region for pipelined Gaussian elimination is increasing at the expense of transposition with or without substructuring with an increased time for the transfer of an element between processors. A high communication start-up time favors transpose based algorithms instead of pipelined Gaussian elimination. Reducing the size of the communications buffers favors substructuring and pipelined Gaussian elimination instead of transposition. The choice between substructuring and pipelined Gaussian elimination is generally a function of $\frac{P}{N}$, except for relatively small P and N for which pipelined Gaussian elimination is of choice. Increasing the size of individual systems for a fixed size cube and a fixed number of systems favors pipelined Gaussian elimination. Increasing the cube size and the number of systems for systems of fixed order favors substructuring. Balanced cyclic reduction is only of interest with a high cost for local data motion.

For multiprocessors with the performance characteristics of the Intel iPSC/1 and with at least 1024 processors (the Intel iPSC/1 is limited to at most 128 processors) pipelined Gaussian elimination would be the method of choice if $\frac{P}{N} \geq 128$, transposition without substructuring the choice for $\frac{P}{N} \leq 4$, and substructuring the prefered method otherwise, except for $P = N$ for multiprocessors with 32 - 4096 processors. In the last case, balanced cyclic reduction is the preferred method with respect to execution time. For cubes with at most 512 processors the choice of method is very case dependent, Figure 13.

For Alternating Direction Methods choosing the aspect ratio of the processing array as close as possible to the aspect ratio of the physical domain is optimal for a large range of parameter values, and close to optimal for many other values.

Combining in the routing system is important for the performance of several of the algorithms. The required combining is of the merge/split type.

Acknowledgement

We thank Eileen Daily for her editorial efforts, and the generous support by the Office of Naval Research under Contract N00014-86-K-0310.

References

[1] Billy L. Buzbee, Gene H. Golub, and C W. Nielson. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.*, 7(4):627–656, December 1970.

[2] M.Y. Chan. *Dilation-2 Embeddings of Grids into Hypercubes.* Technical Report UTDCS 1-88, Computer Science Dept., University of Texas at Dallas, 1988.

[3] William J. Dally. *Wire-Efficient VLSI Multiprocessor Communication Networks.* Technical Report , MIT, Artificial Intelligence Laboratory, September 1986.

[4] A. George. Nested dissection of a regular finite element mesh. *SIAM J. on Numer. Anal.*, 10:345–363, 1973.

[5] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *1986 International Conf. on Parallel Processing*, pages 640–648, IEEE Computer Society, 1986. Tech. report YALEU/DCS/RR-483, May 1986.

[6] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *1987 International Conf. on Parallel Processing*, pages 188–191, IEEE Computer Society, 1987. Report YALEU/DCS/RR-576, April 1987.

[7] Ching-Tien Ho and S. Lennart Johnsson. *Spanning balanced trees in Boolean cubes.* Technical Report YALEU/DCS/RR-508, Dept. of Computer Science, Yale Univ., New Haven, CT, January 1987. To appear in SIAM J. Sci. Statist. Comput.

[8] Roger W. Hockney and C.R. Jesshope. *Parallel Computers.* Adam Hilger, 1981.

[9] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987. (Tech. Rep. YALEU/DCS/RR-361, Yale Univ., New Haven, CT, January 1985).

[10] S. Lennart Johnsson. *Data permutations and basic linear algebra computations on ensemble architectures.* Technical Report YALEU/DCS/RR-367, Dept. of Computer Science, Yale Univ., New Haven, CT, February 1985.

[11] S. Lennart Johnsson. Fast pde solvers on fine and medium grain architectures. In *Advances in Computer Methods for Partial Differential Equations - VI*, pages 405–410, IMACS, 1987. YALEU/DCS/RR-583, April 1987.

[12] S. Lennart Johnsson. *Odd-even cyclic reduction on ensemble architectures and the solution tridiagonal systems of equations.* Technical Report YALE/DCS/RR-339, Dept. of Computer Science, Yale Univ., October 1984.

[13] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Statist. Comput.*, 8(3):354–392, May 1987. (Tech. Rep. YALEU/DCS/RR-436, Yale Univ., New Haven, CT, November 1985).

[14] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988. YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

[15] S. Lennart Johnsson and Ching-Tien Ho. *Multiple tridiagonal systems, the alternating direction method, and Boolean cube configured multiprocessors.* Technical Report YALEU/DCS/RR-532, Dept. of Computer Science, Yale Univ., New Haven, CT, June 1987.

[16] S. Lennart Johnsson and Ching-Tien Ho. *Spanning graphs for optimum broadcasting and personalized communication in hypercubes.* Technical Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986. Revised November 1987, YALEU/DCS/RR-610. To appear in IEEE Trans. Computers.

[17] S. Lennart Johnsson and Peggy Li. *Solutionset for AMA/CS 146.* Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[18] S. Lennart Johnsson, Yousef Saad, and Martin H. Schultz. Alternating direction methods on multiprocessors. *SIAM J. Sci. Statist. Comput.*, 8(5):, 1987. (Yale Univ., Dept. of Computer Science, August, 1985, YALEU/DCS/RR-382).

[19] David S. Lim and Rex V. Thanakij. A survey of alternating direction implicit (adi) method implementations on hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[20] Abhiram Ranade and S. Lennart Johnsson. The communication efficiency of meshes, Boolean cubes, and cube connected cycles for wafer scale integration. In *1987 International Conf. on Parallel Processing*, pages 479–482, IEEE Computer Society, 1987. Report YALEU/DCS/RR-579, April 1987.

[21] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.

[22] Charles L. Seitz. Concurrent VLSI architectures. *IEEE Trans. Comp.*, 33(12):1247–1265, 1984.

[23] Charles L. Seitz. Experiments with VLSI ensemble machines. *J. VLSI Comput. Syst.*, 1(3), 1984.

[24] Ivan E. Sutherland and Carver A. Mead. Microelectronics and computer science. *Scientific American*, 210–228, September 1977.

[25] H.H. Wang. A parallel method for tridiagonal equations. *ACM TOMS*, 7(2):170–183, 1981.