Yale University
Department of Computer Science

Multiple Tridiagonal Systems, the Alternating Direction Methods and Boolean Cube Configured Multiprocessors

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-532
June 1987

This work has been supported in part by the Office of Naval Research under contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

Contents

Multiple Tridiagonal Systems, the Alternating Direction Methods and Boolean Cube Configured Multiprocessors

S. Lennart Johnsson[1] and Ching-Tien Ho
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract

In this paper we investigate the complexity of concurrent algorithms for the solution of multiple tridiagonal systems as they appear, for instance in fast Poisson solvers, or in Alternating Direction Methods. We consider divide-and-conquer algorithms in the form of several variations of odd-even cyclic reduction, and algorithms making use of data transposition and local elimination. The processors are assumed to be configured as a Boolean cube. A simple performance model is established, and the validity of the model verified on an Intel iPSC. Depending on machine characteristics, a divide-and-conquer algorithm such as balanced cyclic reduction, or a transpose based algorithm, may be optimum. In general, the former is preferable for sufficiently many processors. Indeed, a hybrid scheme in which the last several steps of the divide-and-conquer algorithm is replaced by a transpose based algorithm, is of lower complexity than either. The transition point depends on the architectural parameters such as communication start-up time, data channel transfer rate, maximum packet size, and the time for an arithmetic operation, but also the number of independent systems to be solved. For the Intel iPSC with its moderate number of processors the transpose algorithms are in general preferable.

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 originates 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.

Some of the transpose based algorithms are novel variations of Gaussian elimination. Our experiments demonstrate the importance of combining in the communication system for architectures with a relatively high communications start-up time.

1 Introduction

Tridiagonal systems of equations occur in many methods used for the solution of partial differential equations. The Alternating Direction Method (ADM) is one such method in which the tridiagonal matrices arise from one-directional central difference approximations

[1]Currently on leave at Thinking Machines Corp.

of partial derivatives. In the two-dimensional case there is one tridiagonal system for each row, or column, of the lattice. Another example where multiple tridiagonal systems occur is in fast Poisson solvers. After an FFT in one dimension the system decouples into a number of independent tridiagonal systems.

The classical methods for solving tridiagonal systems on sequential architectures are Gaussian elimination and odd-even cyclic reduction. The former treats the pivots sequentially and offers very little concurrency. However, in the context of multiple tridiagonal systems the different systems can be solved concurrently. Odd-even cyclic reduction requires about twice as many arithmetic operations per unknown (17 compared to 8), but can be parallelized (and vectorized). It requires $2(p-1)$ steps for a system of $2^p - 1$ equations. A variation of odd-even cyclic reduction is parallel cyclic reduction [13], which can be performed in p steps, but requires $12p$ arithmetic operations per unknown. In parallel architectures, in particular, if the number of processors approaches the number of unknowns, the communication is a critical issue with respect to performance [27,24,25,2,15,22]. Indeed, data movement is a dominating factor in the cost of computing. It is necessary to include the communication requirements in the analysis of the complexity of algorithms, and to use simplified, but realistic, models of the communication cost in concurrent systems. The communication cost (complexity) for a given computation on a given architecture depends on the data mapping on to the architecture, the choice of numerical algorithm, and the communication algorithms. These issues are the focal points of this work. We demonstrate the effectiveness of message combining for architectures with a high communications overhead. The target architectures have processors interconnected as a Boolean n-cube, a versatile communications network that is used in several existing computers. We propose some novel concurrent algorithms, and investigate the effects of the computational speed, the speed of communication, and the overhead in communication, on the choice of algorithm with respect to performance.

A tridiagonal system of equations has the following form:

$$\begin{pmatrix}
x & x & & & & & & & & & & \\
x & x & x & & & & & & & & & \\
 & x & x & x & & & & & & & & \\
 & & x & x & x & & & & & & & \\
 & & & x & x & x & & & & & & \\
 & & & & x & x & x & & & & & \\
 & & & & & x & x & x & & & & \\
 & & & & & & x & x & x & & & \\
 & & & & & & & x & x & x & & \\
 & & & & & & & & x & x & x & \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x
\end{pmatrix} X = Y.$$

The nonzero matrix elements are denoted x, the right hand side(s) Y, and the solution vector(s) X. For the analysis here we assume that pivoting on the diagonal elements is numerically satisfactory. No other assumption is made on the matrix elements x. We denote the number of equations by P, and the number of tridiagonal systems by Q. We will make the additional assumption that $P = 2^p - 1$ and $Q = 2^q - 1$, whenever it simplifies the analysis. The solution methods being considered are parallel versions of Gaussian elimination, two-way elimination, and cyclic reduction, which can be viewed as Gaussian elimination with a different pivoting order.

We have verified our analytical complexity models on a medium scale, parallel architecture, and the Intel iPSC. The processors are interconnected as a Boolean n-cube, which has $N = 2^n$ processors, $n > 1$. For a two processor system two-way elimination is a good choice. One- and multi-dimensional arrays can be embedded in Boolean cubes with one array node per cube node preserving adjacency by a ***binary-reflected*** Gray code [23]. The Boolean cube is a ***host graph*** for the arrays, ***guest graphs.*** For multi-dimensional arrays the *expansion*, i.e., the number of nodes in the *host* graph relative to the number of nodes in the *guest* graph, is one if the number of array nodes in each dimension is a power of two. Hence, for a two-dimensional grid with $P = 2^p - 1$ and $Q = 2^q - 1$ the expansion is close to one. For arbitrary P and Q it may be close to 4, and higher for multi-dimensional arrays [11]. If $p + q \leq n$, then we map one equation to each processor for our analysis. Here we do not investigate algorithms making use of multiple processors per equation. If $p + q > n$ then several equations are mapped into a processor by a substructuring technique [17,18,5], and the processor utilization is increased. The time for the communication of one byte 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 the operating system B_m. Because of the high copying time on the Intel iPSC there exists a packet size $B_{copy} < B_m$ 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 [10]. With the current software $B_{copy} = 256 \approx \tau/t_{copy}$ bytes. The time for copying one byte is t_{copy}. Furthermore, the Intel iPSC 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 for ease of verification of the model by comparison with measured data.

We first consider the solution of a single tridiagonal system of P equations on an n-cube with $N = 2^n \leq P$. The algorithms are most easily described for single systems. We then generalize the algorithms and the results to multiple tridiagonal systems, which are of greater interest. The basic idea for the parallelization of the solution process is to impose a block structure on the tridiagonal matrix, such that each processor receives approximately the same number of equations. For odd-even cyclic reduction (or elimination in the partial order given by nested dissection [6]) interprocessor communication is needed for every step. By applying Gaussian elimination within each block (also known as substructuring), thereby forming a reduced tridiagonal system that has one row for each block in the partition [28], one communication of a row (or part thereof [20]) between adjacent partitions suffices [17,18]. The assignment of block rows to processors is made by a binary-reflected Gray code [19] encoding. The rows of the reduced system inherit this allocation scheme. Having the solution of the reduced system, we perform backsubstitution within the blocks. This phase is also local, except for two nearest neighbor communications. Note that in the substructured elimination 17 operations per unknown is required, compared to 8 for the normal elimination process. The difference is due to fill-in during the substructuring.

The arithmetic complexity of substructured elimination is $17\frac{P}{N} + 8N$ and $8P$ of standard Gaussian elimination. With respect to arithmetic complexity substructuring shall be applied for $N > 2$. It is also necessary to account for the difference in communication requirements, since the system is initially distributed over the entire set of processors, according to our assumptions. The equivalent of a vector transpose is required for a single system (matrix transpose for multiple systems) before the conventional Gaussian elimination algorithm can be applied. Another transpose operation is required for the solution vector. We refer to this algorithm as the *Transpose - Gaussian Elimination - Transpose* (TGET) algorithm. Clearly, the same procedure can also be applied after a substructuring phase. The substructuring reduces the amount of data that needs to be communicated, but increases the total arithmetic complexity. We call this algorithm SS/TGET. We analyze the complexities in detail below. Two-way Gaussian elimination [4,29] can always be used without an increase in the total arithmetic complexity, which means that a recursive transposition [26,3,14,10] can be terminated one step before completion. Only a few elements need to be communicated in the final step after elimination on the parts. Approximately half of the total data transfer of the TGET algorithm is avoided in this T2GET algorithm. Also, the number of parallel arithmetic steps becomes $4P$, instead of $8P$ of the TGET algorithm.

It is also possible to avoid the second transpose operation entirely by replacing the first transpose operation with an ***all-to-all broadcasting*** operation [12]. After such a communication each processor can solve for one variable, independently and concurrently. We call this algorithm BC2GE for ***Broadcasting and Concurrent 2-way Gaussian Elimination.*** As is the case for the TGET algorithm the elimination can be initiated before the last broadcasting step, and thereby reduce the number of element transfers by a factor of two. By performing a forward elimination on $\frac{1}{2}N$ equations, as in two-way elimination, in each processor, prior to the last exchange, and backsubstitution after the final exchange, the arithmetic complexity can be reduced to the same as in two-way elimination (T2GET). We label this algorithm BC2GER.

For a single tridiagonal system we consider:

- Substructuring followed by odd-even cyclic reduction, CR, [1] for the reduced system, as described in [19,17].
- Substructuring followed by parallel cyclic reduction, PCR [13].
- The collection of all equations in one processor that solves for all variables, followed by a distribution of the results, *Transpose, Gaussian Elimination, Transpose*, TGET [18].
- The collection of half of the equations in one processor and the other half in an adjacent processor, elimination in the two halves concurrently, concurrent solution of a 2×2 coupling system, and concurrent backsubstitution in the two halves followed by distribution of the solution variables. This is algorithm T2GET.
- Substructuring followed by algorithm TGET for the reduced system, SS/TGET.
- Substructuring followed by algorithm T2GET for the reduced system, SS/T2GET.
- The broadcasting of equations from every node to every other node such that every processor have all the equations and each solves for precisely one variable by local Gaussian elimination, ***Broadcasting and Concurrent 2-way Gaussian Elimination***, BC2GE and BC2GER.
- Substructuring followed by broadcasting of the reduced set of equations from every node to every other node such that every processor has all the reduced equations and solves for precisely one variable by local Gaussian elimination, SS/BC2GE and SS/BC2GER.
- A hybrid algorithm, SS/H, obtained by combining, substructuring, odd-even cyclic reduction, and the T2GET algorithm.

The hybrid algorithm is motivated by the following observation. For the solution of the reduced system, i.e., the system of equations after substructuring, cyclic reduction algorithms have a lower arithmetic complexity than a sequential algorithm for $N \geq 16$ (shown later), 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 [10]. Since two equations are needed for the elimination in the reduction phase, and these two equations reside in distinct processors, at least two communications are needed per step. The data transfer time is lower for cyclic reduction than for the transpose algorithm. These different characteristics give rise to some interesting optimization problems that we investigate in the context of a hybrid algorithm.

Depending on the value of different parameters, one or another of the methods for single systems yields the lowest execution time. No method is clearly inferior for all realistic architectural parameters.

For the solution of multiple tridiagonal systems we consider:

- Substructuring followed by *Balanced Cyclic Reduction* [18], SS/BCR, for the reduced system solver.
- The T2GET algorithm.
- Substructuring followed by the T2GET algorithm, SS/T2GET, for the reduced system solver.
- Substructuring followed by a hybrid algorithm, SS/H, for the reduced system solver.

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 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 use standard Gaussian elimination. The execution time for this substructuring phase is approximately

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

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). In the substructuring phase there is one communication needed for the first and the last equations in a partition, if the algorithm in [28,17] is used. The first equation in a partition is sent to the processor holding the preceding partition, and correspondingly a row is received from the succeeding partition for the last equation. By a suitable allocation of matrix coefficients to processors it is not necessary to transfer an entire equation and save some element transfers [20]. The form of the system after the substructured elimination is shown below. Note that the elimination of the last block is done in the reversed order such that the reduced system formed by the last equation of each block, but the last, is of order $2^n - 1$.

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

The substructuring phase is independent of the method used for the solution of the reduced tridiagonal system. This phase is also completely local, with the exception of the elimination operation on the last equation in each partition.

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.

The solution of tridiagonal systems on Boolean cubes by cyclic reduction is discussed extensively in [17,18]. Two algorithms are suggested for tridiagonal systems distributed with one equation per processor: an *exchange* algorithm, and an *in-place* algorithm. In the *exchange* algorithm the active set of equations are recursively moved into an easily identified subcube of one less dimension for each step. In addition there is communication needed for the communication of data for the reduction. One of the communications in the exchange operation is unnecessary, since that communication is identical to one of the communications for the reduction operation. Hence, with one send *or* receive operation at a time $3(n-1)$ communications instead of $4(n-1)$ suffice for each of the factoring-forward substitution and back substitution phases of odd-even cyclic reduction with one equation per processor. We refer to this algorithm as SS/CR-1. The number of communications can be reduced further to $2(n-1)$ at the expense of $O(n)$ memory instead of constant memory. This further reduction in the number of communications is accomplished by observing that in the exchange after a reduction step, the "exchanged" equation to be used in the next reduction step is moved to the location where one of the equations used for the elimination resides prior to the exchange. Hence, by performing a partial elimination, then moving the equation subject to elimination to its location for the next reduction step, and completing the elimination in this location, one communication is saved. A total of $4(n-1)$ communications are needed. However, some of the nodes keep accumulating partially reduced equations. 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 $P = 2^n - 1$ equations on an n-cube by the SS/CR-2 algorithm [17,18] is given by

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

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

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

The communication time for the SS/CR-1 algorithm is 50% higher than for the SS/CR-2 algorithm. The number of floating-point operations per unknown is 12 per equation and reduction step of CR and 5 per backsubstitution equation and step. One of these operations is a division in the backsubstitution with the diagonal element of the factored matrix. This division can be performed concurrently for all equations after the forward/reduction phase. With real coefficients, one right hand side, and one processor per equation, a time of $3t_a$ is required between the forward and backward substitution. The number of arithmetic operations per reduction/backsubstitution step becomes 16 instead of 17. The last three terms correspond to the solution of a tridiagonal system with 3 equations, and the three arithmetic operations just mentioned. For the solution of a single tridiagonal system on a Boolean cube architecture the arithmetic complexity can be reduced to 11 [18] sequential operations per step instead of 16 operations by dividing the arithmetic operations for the reduction 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 [16]. For medium scale concurrency and the solution of tridiagonal systems in the context of Poisson's equation, or an Alternating Direction Method, there will typically be multiple tridiagonal systems, or pieces thereof, assigned to each processor. The techniques for increased processor utilization and lower parallel arithmetic complexity are not applicable in this case. We use the higher number of arithmetic operations (16) and lower communication complexity in this paper.

The execution times of the CR-2 and CR-IR algorithms on the Intel iPSC are shown in Figure 1. The substructuring time is the same for both methods and is not included. The execution times of the CR-2 and CR-IR algorithms are almost identical. The router performs as well as the carefully coded communications algorithm. Moreover, the CR-IR algorithm has constant storage requirements. Note that the number of reduction steps for

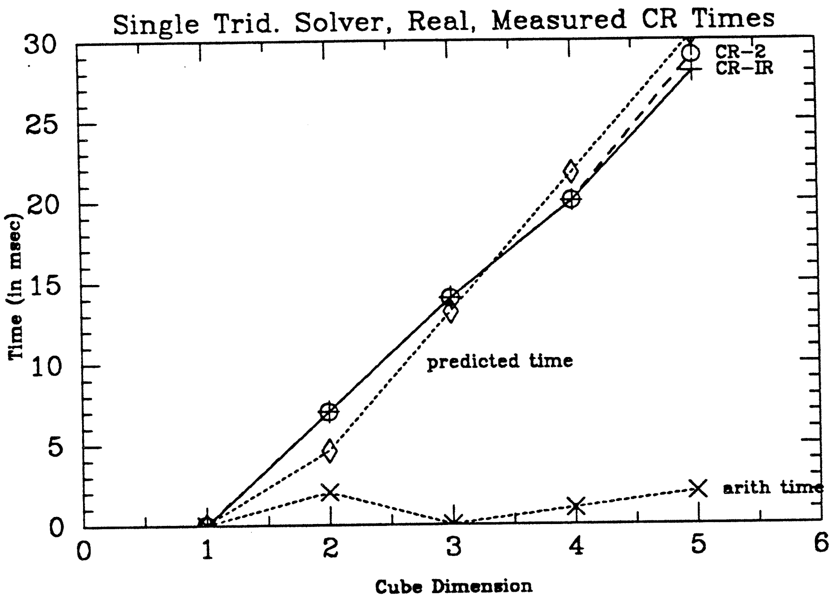


Figure 1: Measured values of $T_{CR-2}(N-1,1;n,t_a,t_c,\tau)$ and $T_{CR-IR}(N-1,1;n,t_a,t_c,\tau)$.

$N = P+1$ is $\lfloor \log_2(2^P - 1) \rfloor$, i.e., $n-1$ for an n-cube. We attribute the difference between the predicted and measured times for algorithm SS/CR-2 to the variability in measured times on the Intel iPSC. Performance measurements are difficult due to non-reproducible timings. Equation (3) also describes the execution times for algorithm CR-IR well.

2.2 Parallel Cyclic Reduction (PCR)

Parallel Cyclic Reduction [13] performs a reduction on every equation in the first step, not only half of the equations. With one equation per processor all processors are active. After this step the problem is reduced to two sets of equations, each with half the number of equations of the original set. After repeating the procedure n times for 2^n equations, each set is reduced to a single equation.

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

No backsubstitution is needed, unlike for odd-even cyclic reduction. The total arithmetic complexity of the PCR algorithm is $12n2^n$, the parallel complexity $12n$. The total number of messages in communication, ignoring the distance, is $2n2^n - 2n + 1$, and the number of parallel communications in sequence $4n$ (two exchanges) with one send *or* one receive at a time. Substructuring is always preferable. The parallel arithmetic complexity for the reduced system is $\frac{3}{4}$ of that of CR (with 16 operations per step and equation), and the number of communications in sequence is $\frac{1}{2}$ of that of CR-2. During step j, $0 \leq j < n$, the i^{th} row of the reduced system exchanges its data with the $(i-2^j)^{th}$ and the $(i+2^j)^{th}$ rows (if $i - 2^j \geq 0$, and $i + 2^j \leq P - 1$). With a binary-reflected Gray code encoding i and $i + 2^j$ are at most a distance 2 apart in the cube [17]. 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 2 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 hypercube 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 reduction 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 exchange algorithm changes the allocation of equations to processors such that the solution variables are encoded in binary code. 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 [14]. 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 2 sends or receives are decomposed into two "nearest-neighbor" communications performed concurrently for all nodes such that there is no edge conflict[14]. Moreover, since the data sent to both distance 2 neighbors in each SS/PCR step are the same, and the two paths to the distance 2 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 naive routing discipline like the one used in the Intel iPSC, routes the dimensions that need to be routed in increasing order. Such a routing order is conflict-free for the routing needed in SS/PCR-IR [17].

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

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

and for the substructured version

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

The arithmetic complexity is lower than that of the SS/CR-1 and SS/CR-2 algorithms, the number of start-ups is 50% higher despite the fact that only n instead of $2(n-1)$ steps

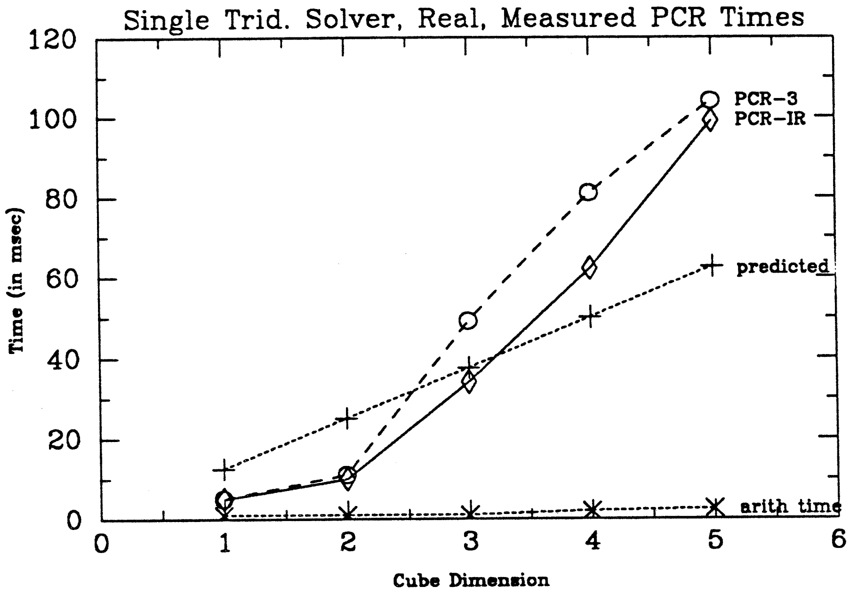


Figure 2: Measured values of $T_{PCR-3}(N,1;n,t_a,t_c,\tau)$ and $T_{PCR-IR}(N,1;n,t_a,t_c,\tau)$.

are carried out. The number of element transfers for the PCR algorithm is a factor of 2.4 higher than that of the CR-2 algorithm, and a factor of 1.6 higher than that of the CR-1 algorithm. Hence, if the communication dominates the CR algorithms are expected to execute faster than the PCR algorithms.

The measured times on the Intel iPSC for algorithms PCR-3 and PCR-IR are shown in Figure 2. As was the case for the CR implementations the PCR-IR algorithm is slightly more efficient than the "hand coded" algorithm. The advantage varies between 5% and 30% in our experiments. The measured times do not agree well with the predictions of equation (5). We attribute the difference to synchronization delays. The measured times for any PCR algorithm is higher than for our CR algorithms. The estimated execution times yields the same relative ordering of the methods as the measurements.

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, Gaussian Elimination, Solution Transposition (TGET)

Conventional Gaussian elimination only requires 8 operations per unknown, but substructured elimination 17 operations per unknown. With respect to arithmetic operations substructuring should be made for $N > \frac{17}{8}$. The communication prior to the solution is *all-to-one personalized communication* for which lower bound algorithms are given in [9,12]. A spanning binomial tree routing is optimal with communication restricted to one port at a time. In a multiprocessor system two-way elimination allows two processors to be used for the same tridiagonal system without an increase in total arithmetic complexity, and a speed-up of two is achieved with respect to the arithmetic complexity. Two-way elimination yields lower communication complexity than one-way elimination. The break-even point between substructured elimination or not is at $N > \frac{17}{4}$, considering arithmetic operations alone. Half of the element transfers in the first transpose of the TGET method takes place in its last step, if the spanning binomial tree routing is used.

We first give the complexity estimates for TGET and T2GET, then the estimates for SS/TGET and SS/T2GET. The T2GET algorithm has lower arithmetic and communication complexity than the TGET algorithm. The interesting comparison is between SS/T2GET and T2GET.

The estimated time of the TGET and T2GET methods are

$$\begin{aligned} T_{TGET}(P,1;n,t_a,t_c,\tau) &= 20(N-1)\lceil\frac{P}{N}\rceil t_c \\ &\quad + \sum_{i=0}^{n-1}(\lceil\frac{16P\times 2^i}{NB_m}\rceil + \lceil\frac{4P\times 2^i}{NB_m}\rceil)\tau + (8P-7)t_a, \end{aligned} \tag{6}$$

$$\begin{aligned} T_{T2GET}(P,1;n,t_a,t_c,\tau) &= (20(\frac{1}{2}N-1)\lceil\frac{P}{N}\rceil + 24)t_c \qquad B_m \geq 12 \\ &\quad + (\sum_{i=0}^{n-2}(\lceil\frac{16P\times 2^i}{NB_m}\rceil + \lceil\frac{4P\times 2^i}{NB_m}\rceil) + 2)\tau + (4P-2)t_a, \end{aligned} \tag{7}$$

and for the substructured versions the times are

$$T_{SS/TGET}(P,1;n,t_a,t_c,\tau) = T_{SS}(P,1;n,t_a,t_c,\tau) + T_{TGET}(N,1;n,t_a,t_c,\tau), \tag{8}$$

$$T_{SS/T2GET}(P,1;n,t_a,t_c,\tau) = T_{SS}(P,1;n,t_a,t_c,\tau) + T_{T2GET}(N,1;n,t_a,t_c,\tau). \tag{9}$$

It is assumed that in the two-way elimination both processors involved in the elimination 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 an equation to the other, which upon the solution of a 2×2 system returns one of the solution variables.

The *all-to-one personalized communication* can either be carried out as a send operation using the router of the Intel iPSC, or by a user coded spanning binomial tree algorithm with combining. The router also uses a binomial tree routing, but does not perform combining of messages. For the reverse operation, *one-to-all personalized communication* the same options apply. In addition, for a moderate number of processors it is possible to carry out this operation as a *one-to-all broadcasting*, or copy-scan [7], of the entire solution vector. More data than necessary is communicated, but many architectures have efficient implementations of data broadcast. 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 [12].
- *All-to-one personalized communication* with combining followed by router broadcast.

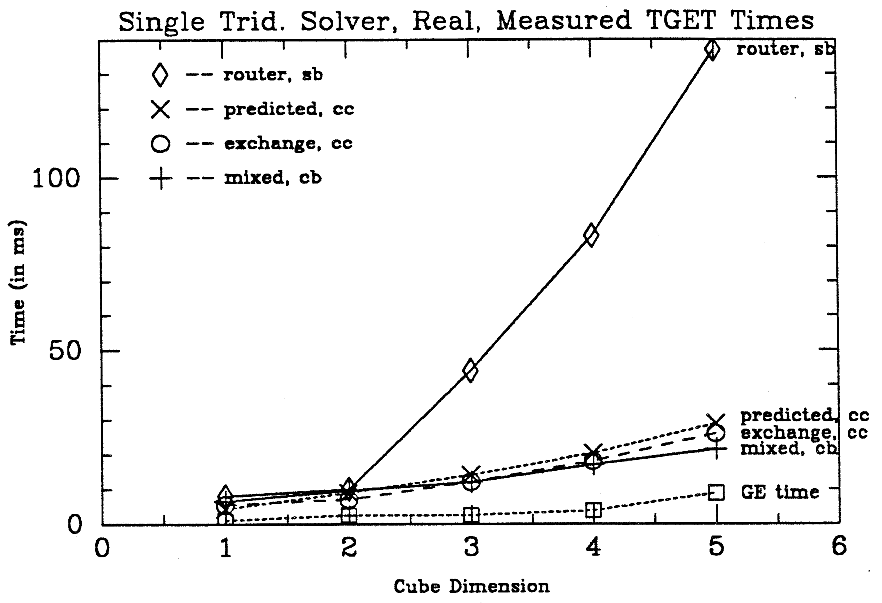


Figure 3: Measured values of $T_{TGET}(N,1;n,t_a,t_c,\tau)$ using routing with combining (-cc), the routing logic of iPSC (no combining) and broadcasting (-sb), and combining and broadcasting (-cb).

We refer to these three implementations with the postfix -sb, -cc, and -cb, respectively. The measured times for the three alternatives for the SS/TGET algorithm compare as shown in Figure 3. Again the substructuring part is the same for all alternatives and is not included. The times for the SS/TGET-sb implementation 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 with unlimited buffer size increases linearly in the dimensions of the cube, while the data volume and arithmetic time increases exponentially, these two components of the total time 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.

The data transfer time for the SS/T2GET algorithm is almost the same as for the SS/TGET algorithm with one dimension less except the one exchange needed for the coupling equations, and so is the total communication time, if the buffer size $B_m \ll N$. If the buffer size is unbounded, and the start-up times dominates, then the communication times are approximately the same. The measurements for up to 5-cubes yields a communication complexity of SS/T2GET, excluding the substructuring that is about half way between SS/TGET of the same dimension and one dimension less, Figure 4.

The predicted times for T2GET is approximately 20% less than those for the TGET algorithm with the parameters that apply for the Intel iPSC. The measurements confirm these estimates. The predicted execution times are about 15% higher than the measured times

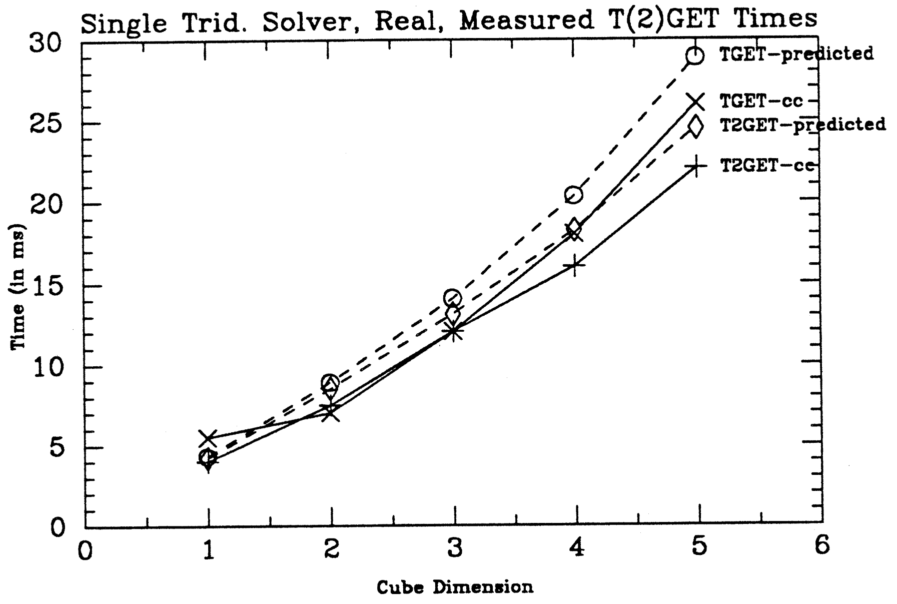


Figure 4: A comparison of predicted and measured (Intel iPSC) execution times for TGET-cc and T2GET-cc, $P = 2^n - 1$.

for both the TGET and T2GET algorithms. The T2GET algorithm is always preferable with respect to execution time.

Though each step of cyclic reduction reduces the number of remaining arithmetic operations by a factor of 2, the communications for each step is only higher than that of the T2GET algorithm (or TGET) by a constant factor, a hybrid scheme in which some steps of cyclic reduction is performed, followed by the T2GET algorithm suggests itself.

Before analyzing this hybrid scheme we notice that for a single tridiagonal system one can also broadcast every equation to every node and have each processor solve for the variable corresponding to the partition it stores. This algorithm is called BC2GE.

2.4 Broadcasting and Concurrent 2-way Gaussian Elimination (BC2GE)

It is possible to avoid distributing the results of the local Gaussian elimination in algorithm TGET to the processors storing the equations, if all processors send their equation(s) to every other processor, *all-to-all broadcasting* [12]. Then, each processor can solve for one variable by two-way Gaussian elimination converging to different equations. Backsubstitution is unnecessary. The algorithm can be used with or without a substructuring phase, called SS/BC2GE and BC2GE, respectively. With communication restricted to one port at a time, as on the Intel iPSC, an algorithm performing exchanges in the different dimensions is optimal [12]. Such an algorithm generates N binomial trees. The data set being exchanged doubles for each step of the algorithm, as in the *all-to-one personalized communication*. But, with one send *or* one receive operation at a time, the communication complexity is twice that of the initial transpose of the TGET algorithm. The parallel arithmetic complexity of this algorithm is $5P - 4$ (the total $(5P - 4)N$) for BC2GE and

$17(\frac{P}{N} - 1) + 5N - 4$ (total $17P + 5N^2 - 21N$) for SS/BC2GE. The estimated execution times are

$$T_{BC2GE}(P,1;n,t_a,t_c,\tau) = 2 \times 16(N-1)\lceil\frac{P}{N}\rceil t_c + 2\sum_{i=0}^{n-1}\lceil\frac{16P \times 2^i}{NB_m}\rceil\tau + (5P-4)t_a, \quad (10)$$

$$T_{SS/BC2GE}(P,1;n,t_a,t_c,\tau) = T_{SS}(P,1;n,t_a,t_c,\tau) + T_{BC2GE}(N,1;n,t_a,t_c,\tau). \quad (11)$$

The communication time can be reduced by performing elimination operations before the last step of the communication algorithm. In such a case all processors perform an elimination towards the middle equation, as in the T2GET method. Similarly, there is an exchange step for the computation of the middle variable. Then, each processor performs backsubstitution until it has computed its assigned variable (SS/BC2GER) or variables, if no substructuring is performed (BCGER). The communication complexity until elimination starts is twice that of the T2GET algorithm. The parallel arithmetic complexity is $4P-2$ for BC2GER and $17(\frac{P}{N}-1)+4N-2$ for SS/BC2GER, the same as in T2GET and SS/T2GET. Hence, BC2GER and SS/BC2GER have lower communication complexity as well as lower arithmetic complexity than BC2GE and SS/BC2GE respectively.

$$\begin{aligned} T_{BC2GER}(P,1;n,t_a,t_c,\tau) &= 2 \times (16(\frac{1}{2}N-1)\lceil\frac{P}{N}\rceil + 12)t_c \quad B_m \geq 12 \\ &+ 2(\sum_{i=0}^{n-2}\lceil\frac{16P \times 2^i}{NB_m}\rceil + 1)\tau + (4P-2)t_a. \end{aligned} \quad (12)$$

$$T_{SS/BC2GER}(P,1;n,t_a,t_c,\tau) = T_{SS}(P,1;n,t_a,t_c,\tau) + T_{BC2GER}(N,1;n,t_a,t_c,\tau). \quad (13)$$

The *all-to-all broadcasting* operation can be performed either by using the combining inherent in the exchange algorithm, or by using the router of the Intel iPSC. The running times for algorithm BC2GE are shown in Figure 5. The lack of combining in the router has a serious effect on the performance. The router based implementation for a 5-cube has an execution time that is approximately five times that of the algorithm with combining. The exponential growth rate in the execution time for the algorithm without combining is apparent. The time for arithmetic operations is less than 25% of the total time for cubes of less than 5 dimensions.

The data transfer time of the BC2GER algorithm is a factor of 1.6 higher than for the T2GET algorithm, and the number of start-ups the same or higher. Hence, the execution time of the BC2GER algorithm is expected to be the same or higher than that of the T2GET algorithm, and the execution time of the BC2GE algorithm higher than that of both the T2GET and BC2GER algorithms. For the Intel iPSC the measured times of the T2GET algorithm are about the same as those of the BC2GE algorithm for $P = 2^n - 1$, and real systems, Figure 6. The execution time for the BC2GER algorithm is lower than that of the BC2GE algorithm, as expected, as well as that of the T2GET algorithm. With a complex system, the T2GET algorithm is about 10% faster than the BC2GE algorithm. We attribute the difference between the predicted and observed difference in execution times between the T2GET and BC2GER algorithms to implementation details, difficulties in generating reproducible timings, and an overly pessimistic communications model with no overlapping between sends and receives.

The BC2GE and BC2GER algorithms are novel and interesting, but does not offer any particular advantages over the T2GET algorithm according to our performance model, but

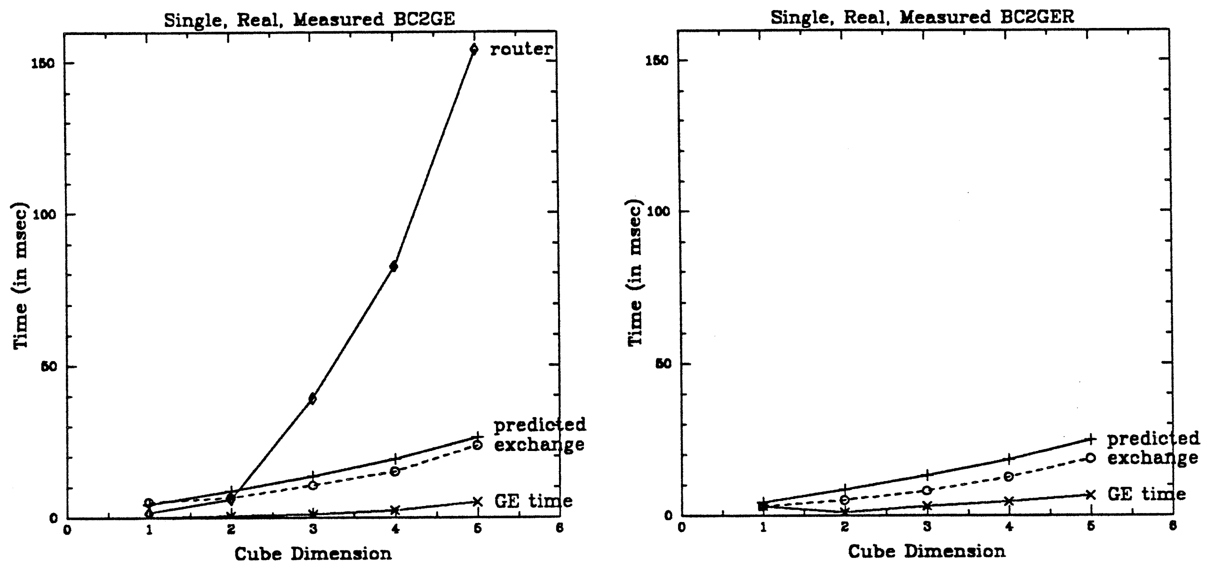


Figure 5: Measured values of $T_{BC2GE}(N, 1; n, t_a, t_c, \tau)$ using *all-to-all broadcasting* through exchanges (combining), and the router of the Intel iPSC (non-combining).

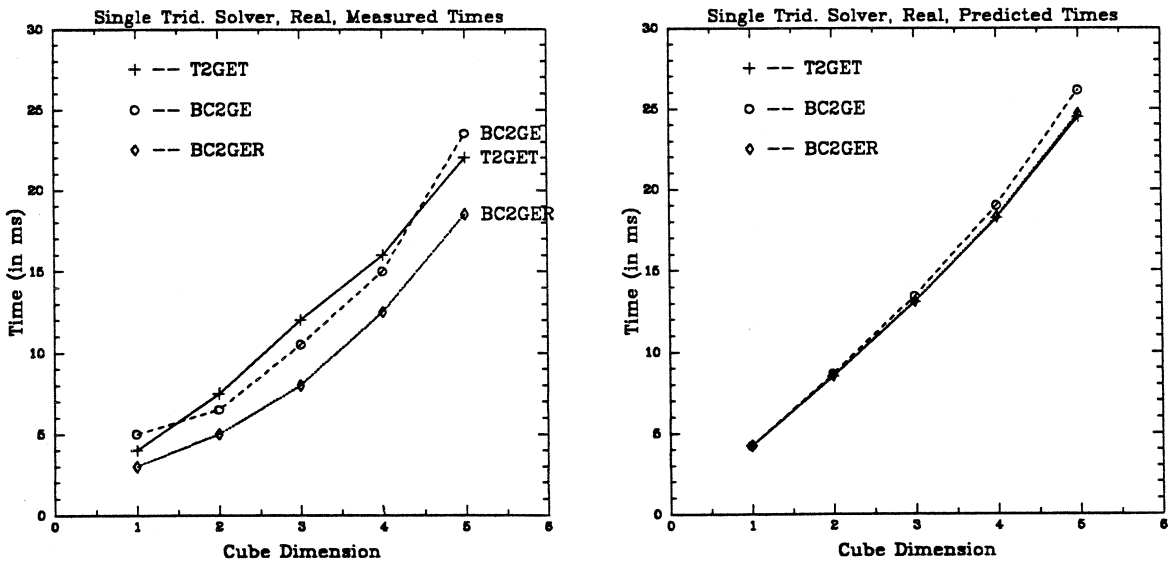


Figure 6: Measured (Intel iPSC) and predicted execution times of BC2GE, BC2GER, and T2GET.

our implementation of the BC2GER algorithm executes approximately 20% faster than our T2GET implementation on the Intel iPSC. If sends and receives could be performed concurrently, then the BC2GER algorithm would be of lower complexity than the T2GET algorithm. A 20% overlap between sends and receives for the Intel iPSC has been observed also in other experiments. The BC2GE algorithm is of sequential complexity $O(PN)$. With multiple tridiagonal systems distributed with one equation per processor, and each processor having one equation of every system, the higher arithmetic complexity of the BC2GE method makes it clearly inferior to the T2GET algorithm.

2.5 A Hybrid Algorithm: Substructuring with CR and T2GET

Our previous analysis shows that the odd-even cyclic reduction algorithm is never inferior to parallel cyclic reduction for the solution of a single system of equations, if the architecture is constrained by one send *or* one receive at a time. Our measurements on the Intel iPSC indeed shows that odd-even cyclic reduction is significantly faster. The estimated complexities of T2GET and BC2GER are very close, with a slight advantage for T2GET on an architecture that only supports one send *or* one receive at a time. Our measurements show a slight advantage for the BC2GER algorithm. If concurrent sends and receives are supported, then the BC2GER algorithm may have an advantage. But, for the following discussion we only consider odd-even cyclic reduction and the T2GET algorithm. The arithmetic complexity of the odd-even cyclic reduction algorithm is approximately $16n - 15$ compared to $4N - 2$ for the T2GET algorithm, $P = 2^n - 1$. Hence, for $N > 8$ odd-even cyclic reduction is of lower arithmetic complexity. But, it may require a greater communication time ($4(n-1)$ start-ups compared to at least $2n$ for T2GET). Architectures with a large start-up time relative to the time for arithmetic operations, such as is the case for the Intel iPSC, favors the T2GET algorithm over odd-even cyclic reduction for small n.

The data transfer time for odd-even cyclic reduction quickly becomes lower than that of the T2GET algorithm as a function of N. The number of start-ups may also become lower, if the buffer size, B_m is small. For each step of the cyclic reduction algorithm the amount of remaining arithmetic work is reduced by a factor of 2. The number of start-ups per reduction/ backsubstitution step totals 4 for a maximum packet size greater than 4 floating-point numbers for real systems and algorithm CR-2, and one for each step of the *all-to-one personalized communication* and *one-to-all personalized communication* algorithms. The two transpose operations requires a total of 2 start-ups for each step and unlimited packet size. However, the communicated data volume in the cyclic reduction algorithm remains constant throughout the algorithm, but it doubles for every step of the combining *all-to-one personalized communication* algorithm. The observed behavior of the CR-IR algorithm on the Intel iPSC is approximately the same as that of the CR-2 algorithm, and for the Intel iPSC CR-IR should be implemented instead of the CR-2 algorithm due to its simpler implementation. In the hybrid algorithm we first perform ℓ steps of cyclic reduction then solve the reduced set of $N/2^\ell$ equations by the T2GET algorithm. The time of this hybrid scheme of CR-2 plus T2GET is

$$\begin{aligned} & T_{SS/H}(P,1,\ell;n,t_a,t_c,\tau) \\ = \;& T_{SS}(P,1;n,t_a,t_c,\tau) + T_{CR-2}(N-1,1;n,t_a,t_c,\tau) \\ & - T_{CR-2}(\frac{N}{2^\ell}-1,1;n-\ell,t_a,t_c,\tau) + T_{T2GET}(\frac{N}{2^\ell},1;n-\ell,t_a,t_c,\tau). \end{aligned} \quad (14)$$

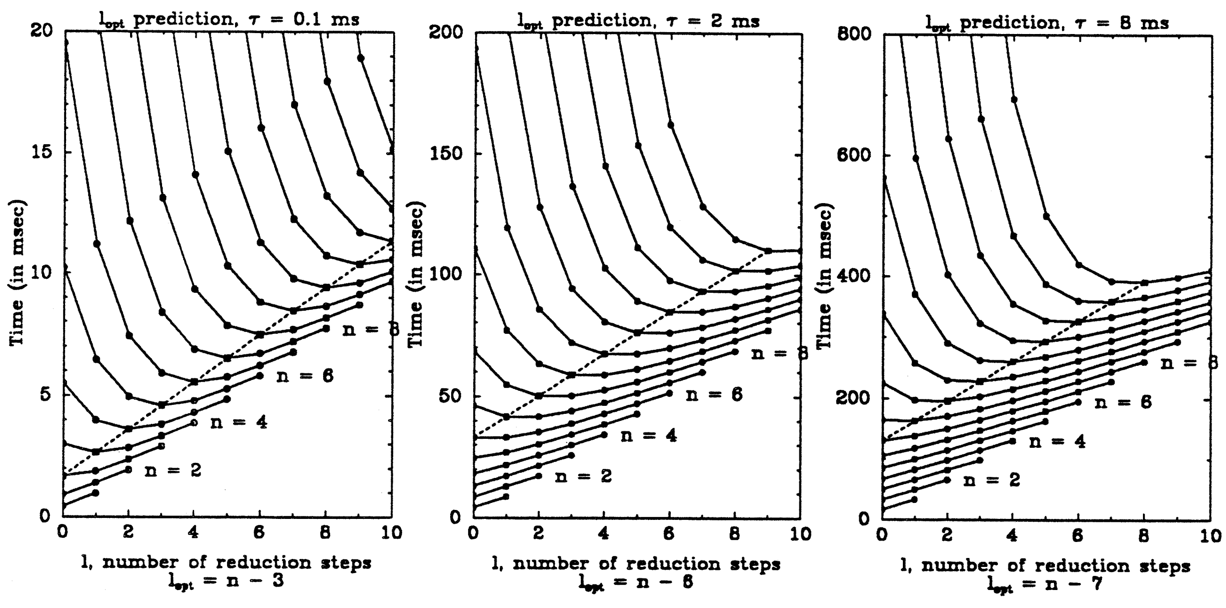


Figure 7: The predicted execution times for the hybrid method for a single tridiagonal system, different cube sizes, and start-up times. The dashed lines join ℓ_{opt} for different cube dimensions.

For the Intel iPSC $t_a \approx 33$ μsec and $t_c \approx 1$ μsec. Start-up times of $\tau \approx 8$ and 2 msec correspond to measured start-up times for different versions of the Intel iPSC communication software. We also include a start-up time of 0.1 msec. The optimum values of ℓ are approximately, $n-7$, $n-6$ and $n-3$. Figure 7 shows the predicted execution times as a function of ℓ for various cube sizes. The dashed line joins ℓ_{opt} for different cube dimensions.

2.6 Comparison

Table 1 summarizes the complexities of the algorithms. The memory figures are in units of 4 elements for real systems. The complexity estimates in the various equations given previously are compiled into Figure 8 for $P = 2^n - 1$. Three different start-up times are considered: 0.1 msec, 2 msec and 8 msec. The curves for CR and PCR increase approximately linearly with n. The arithmetic component, data transfer component, and start-up component are all proportional to n, even for a very limited buffer size for the communication. The curves for T2GET-cc and BC2GER-c have at least two terms increasing exponentially in n: the arithmetic component, and the data transfer component. The start-up component eventually also increases exponentially in n for a limited buffer size B_m. Combining is critical to good performance on the Intel iPSC (for which the router does not perform combining). Measured times are given in the upper plots of Figure 9 for the solution of a single real system (upper left) and a single complex system (upper right). The start-up time is measured to be approximately 2 msec. The corresponding predicted times are shown in the second row of plots.

The predicted execution times agree within 20% with the measurements, except for

Algorithm	Arithmetic	Byte transfers	Min start-ups	Memory
TGET	$8P-7$	$20P(1-\frac{1}{N})$	$2n$	P
T2GET	$4P-2$	$10P(1-\frac{2}{N})+24$	$2n$	$\frac{P}{2}$
BC2GE	$5P-4$	$32P(1-\frac{1}{N})$	$2n$	P
BC2GER	$4P-2$	$16P(1-\frac{2}{N})+24$	$2n$	$\frac{P}{2}$
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	$17\lceil\frac{P}{N}\rceil+8N-24$	$20N+20$	$2n+4$	$\frac{P}{N}+N$
SS/T2GET	$17\lceil\frac{P}{N}\rceil+4N-19$	$10N+44$	$2n+4$	$\frac{P}{N}+\frac{N}{2}$
SS/BC2GE	$17\lceil\frac{P}{N}\rceil+5N-21$	$32N+8$	$2n+4$	$\frac{P}{N}+N$
SS/BC2GER	$17\lceil\frac{P}{N}\rceil+4N-19$	$16N+32$	$2n+4$	$\frac{P}{N}+\frac{N}{2}$

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

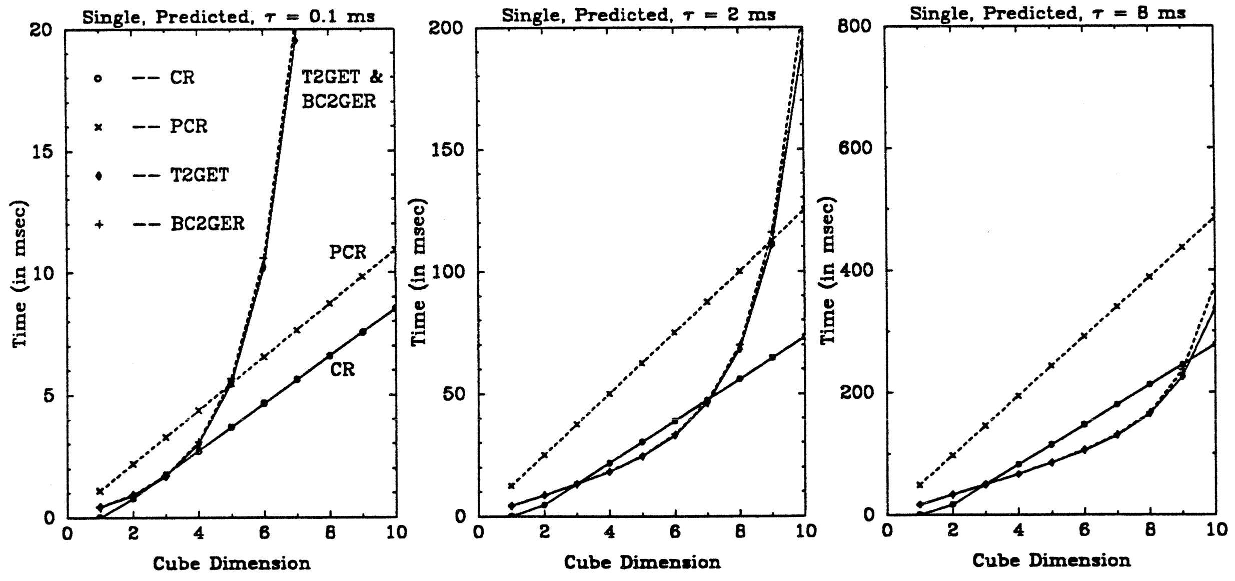


Figure 8: Predicted times of CR-2, PCR-3, T2GET-cc and BC2GER-c schemes with start-up time = 0.1 msec, 2 msec, or 8 msec and $P = 2^n - 1$. The times for CR-2, PCR-3, T2GET-cc and BC2GER-c are marked by circle, x, diamond and plus symbols respectively.

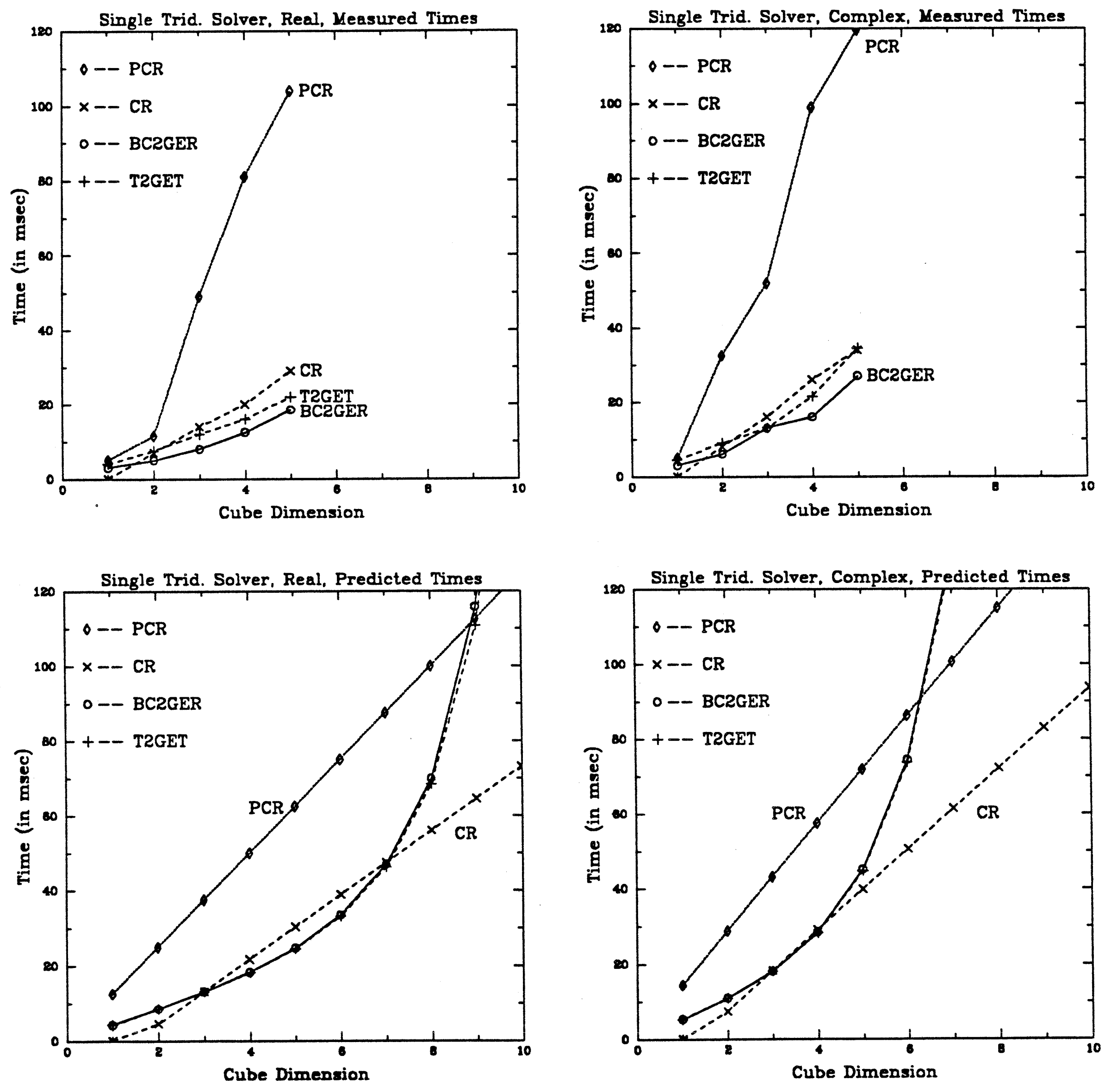


Figure 9: Measured times of the CR-2, PCR-3, T2GET-cc and BC2GER-c algorithms on the Intel iPSC for matrices of real elements (upper left) and complex elements (upper right), $P = N$ ($P = N - 1$ for CR-2). The predicted times are shown below the measured times.

	$\tau = 0.1$ msec	$\tau = 2$ msec	$\tau = 8$ msec
real	$n-3$	$n-6$	$n-7$
complex	$n-2$	$n-4$	$n-5$

Table 2: The optimum number of reduction steps in the hybrid method.

the PCR algorithm. The measured PCR times are considerably higher than the predicted times. The predicted, and by far the measured times for the PCR algorithm, are always higher than those of any of our CR algorithms for the Intel iPSC. CR is always preferable to PCR on the Intel iPSC. In the PCR algorithm, all processors are active throughout the computation. The communication involves all processors throughout the n steps. Any processor delay will tempt to delay all other processors.

The T2GET, BC2GE and BC2GER algorithms have qualitatively the same behavior, and estimated execution times that are very close. The T2GET algorithm has a slight advantage over BC2GER for architectures that only support one send *or* one receive operation at a time. With concurrent sends and receives the BC2GER algorithm is of lower complexity. The difference is entirely due to differences in communication complexity. BC2GE is inferior to BC2GER.

The choice of algorithm for solving the reduced system on an architecture that only supports one send *or* one receive operation, is between the T2GET and CR-IR algorithms. Since a decreasing start-up time favors the CR algorithms more than the T2GET algorithm, the break-even point moves towards smaller cube dimensions for smaller start-up times. The Hybrid algorithm combines the two algorithms such that the T2GET algorithm is used when the reduction process has progressed sufficiently far. The optimum values for the number of reduction steps in the hybrid method for real and complex systems are summarized in Table 2. The dependence of the optimum number of reduction steps on architectural parameters is illustrated in Figure 10.

We have given complexity estimates only for real systems. In the case of 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. This property is reflected both in our measurements for a complex system, and the predictions given in Figures 9 and 10.

What remains to be compared is the T2GET algorithm and the SS/Hybrid algorithm. First we notice that for a real system and for the parameters of the Intel iPSC, a few steps of the CR-IR algorithm shall be made for $n > 6$ and $P = 2^n - 1$. T2GET shall be used when the reduced system fits in a 6-cube. If $P > N$, and $n > 6$ substructuring shall always be used. For $n \leq 6$ and $P = 2^n - 1$ the algorithm of choice is T2GET, and for $P > N$ there is a choice between SS/T2GET or T2GET. The comparison between the SS/Hybrid, SS/T2GET, and T2GET algorithms is made in Figure 11. The comparison is based on the estimated complexities and for various values of t_a, t_c, τ, and B_m. Recall that the SS/Hybrid algorithm with $\ell = 0$ is the SS/T2GET algorithm.

The comparison between SS/T2GET and T2GET for the particular parameters of the Intel iPSC is made in Figure 12. For a 2-cube substructuring shall be used for $P > 1023$

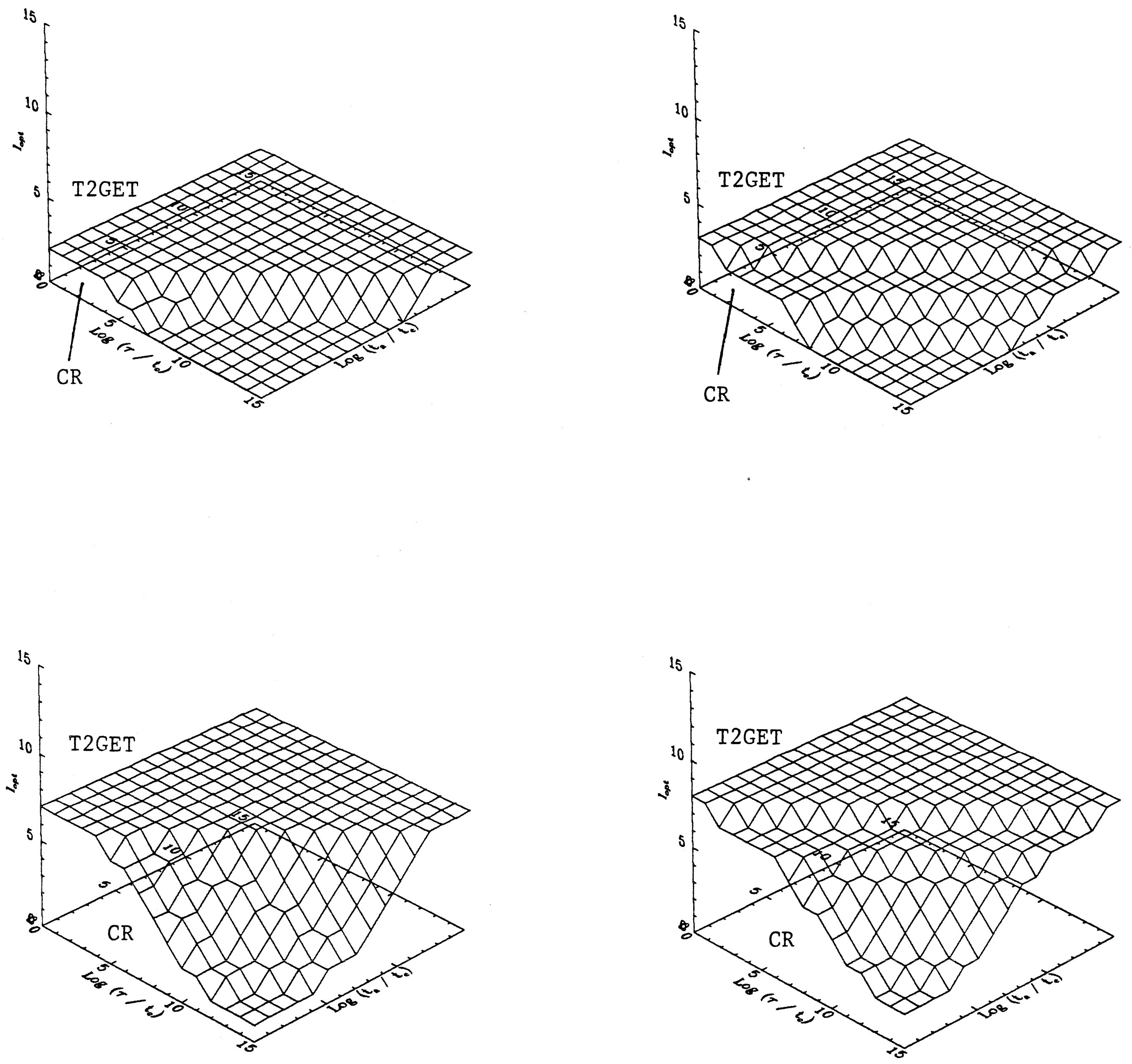


Figure 10: The optimum number of reduction steps, ℓ_{opt}, of the hybrid method, CR/T2GET, as a function of $\log \frac{\tau}{t_c}$ and $\log \frac{t_a}{t_c}$. $N = 32$ in the upper plots and 1024 in the lower plots. The left plots are for real systems and the right plots for complex systems.

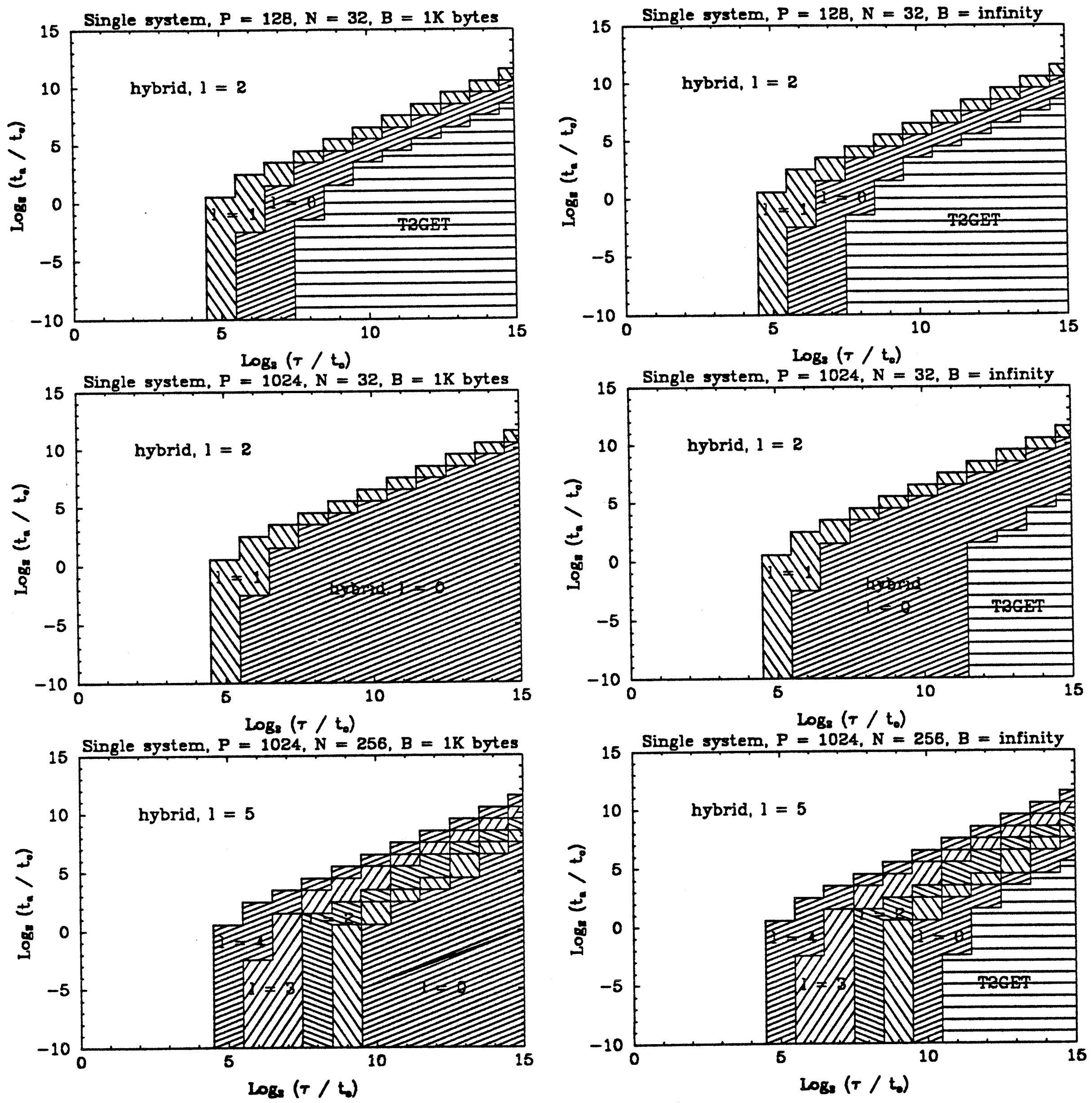
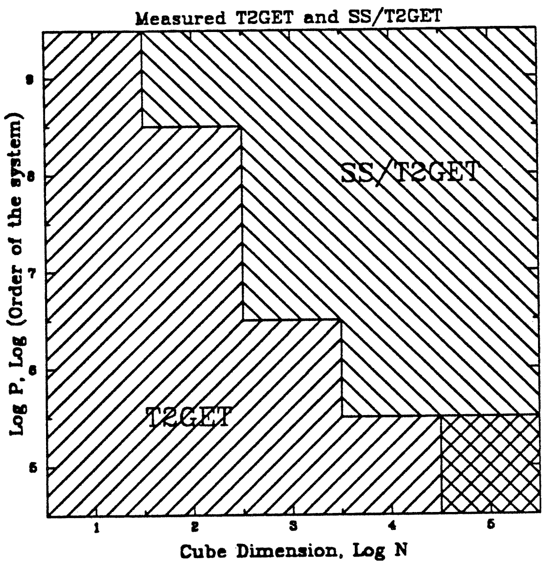


Figure 11: Lowest complexity algorithm of the SS/Hybrid and the T2GET as a function of $\log \frac{\tau}{t_c}$ and $\log \frac{t_a}{t_c}$. B_m is 1K bytes for the left plots and ∞ for the right plots.



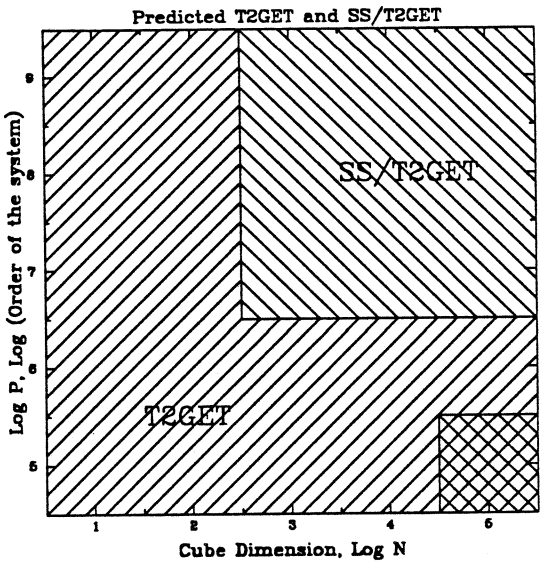


Figure 12: Regions of minimum complexity for T2GET and SS/T2GET on the Intel iPSC.

according to our predictions, and for $P > 255$ according to our measurements. For a 3-cube the predictions and measurements both give substructuring for $P > 63$. For a 4-cube predictions indicate $P > 63$ and measurements $P > 31$ for substructuring. The qualitative behavior is the same, and the agreement good.

3 Multiple Tridiagonal Systems

We now turn to the solution of Q tridiagonal systems each of P equations, $P \geq N$. For $Q \bmod N = 0$ and complete freedom of distributing the systems the obvious choice is to allocate $\frac{Q}{N}$ systems to each processor and solve the systems 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, 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.

If all the systems are identically distributed across the entire cube using the binary-reflected Gray code, 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. With substructuring we get a SS/TGET algorithm for multiple systems. The complexity of substructuring in the multiple systems case is

$$T_{SS}(P,Q;n,t_a,t_c,\tau) = 17Q(\lceil \frac{P}{2^n} \rceil - 1)t_a + 2(16+4)Qt_c + 2(\lceil \frac{16Q}{B_m} \rceil + \lceil \frac{4Q}{B_m} \rceil)\tau. \qquad (15)$$

With $Q \bmod N = 0$ systems of equations, 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 arithmetic complexity of TGET is $(8P-7)\frac{Q}{N}$ without substructuring and $17Q(\lceil \frac{P}{N} \rceil - 1) + (8N-7)\frac{Q}{N}$ with substructuring. The use of two-way elimination does reduce the data being communicated in the last transpose step. The transposition is performed through a sequence of exchanges in the different dimensions. With one send *or* one receive operation at a time, such a transpose algorithm is optimal [10,8]. The transposition 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.

It follows from our analysis and experiments on single systems that parallel cyclic reduction is not of interest with the data allocation we assume. 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) [18] balances the load by performing the reduction such that for a set of N equations the reduction process converges to a unique processor for each equation. For $Q \bmod N = 0$ the arithmetic complexity for real systems is $17Q(\lceil \frac{P}{N} \rceil - 1) + 17Q(1 - \frac{1}{N})$. The arithmetic complexity of the SS/BCR algorithm is always higher than that of the SS/TGET (or SS/T2GET) algorithm. In the single system case the arithmetic complexity of odd-even cyclic reduction is lower than that of T2GET for $N > 8$. Hence, cyclic reduction based methods for $Q \bmod N = 0$ are only of interest if the communication complexity is lower than that of a transpose based algorithm.

We will now compare the computational complexity of transpose and cyclic reduction based methods in detail. We first assume that $Q \bmod N = 0$, then consider the case $1 < Q < N$.

3.1 Balanced Cyclic Reduction

The arithmetic complexity of the substructured elimination is the same as that of cyclic reduction, but it 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). The estimated times for the *exchange* version of the BCR and SS/BCR are

$$\begin{aligned} &T_{BCR}(N,Q;n,t_a,t_c,\tau) \qquad (16)\\ &= 17Q(1-\frac{1}{2^n})t_a + \{4(16+4)Q(1-\frac{1}{2^{n-1}}) + 2(16+4)\frac{Q}{2^n}\}t_c \\ &+ \{4\sum_{i=1}^{n-1}(\lceil\frac{16Q}{2^iB_m}\rceil + \lceil\frac{4Q}{2^iB_m}\rceil) + 2(\lceil\frac{16Q}{2^nB_m}\rceil + \lceil\frac{4Q}{2^nB_m}\rceil)\}\tau, \end{aligned}$$

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

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. The result of implementing the two versions of BCR is shown in Figure 13. The algorithm using the router requires 50 - 100% higher time for the reduced systems on the Intel iPSC. 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. The difference in total time becomes more significant for increasing cube dimensions for a fixed size problem, since the substructuring part reduces in significance.

The predicted times for 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, for an algorithm with each communication step involving all the processors tempts to have a more significant delay for a larger cube. This often explains why the difference between the measured time and the predicted times increases as the cube size increases such as the PCR in Figure 2 and the BCR in Figure 13.

3.2 Transpose, Gaussian Elimination, Transpose

We first consider the TGET algorithm with or without substructuring. With respect to the arithmetic complexity substructuring shall not be used. The savings is $9Q(\frac{P}{N}-1)$ operations with no substructuring. Substructuring reduces the communication volume, but may require 4 additional start-ups.

The data movement in the data transposition is analogous to transposing a rectangular matrix stored by a one-dimensional partitioning and Q columns (or rows) per processor. Each element of a column corresponds to 4 data items before the solution and 1 thereafter. An exchange algorithm similar to the one used in the single system case is optimum for communication restricted to one port at a time [12]. The difference is that in the case of Q mod $N = 0$ the data volume remains constant throughout the exchange sequence, which is $\frac{2PQ}{N}$ data elements without substructuring and $2Q$ elements with substructuring.

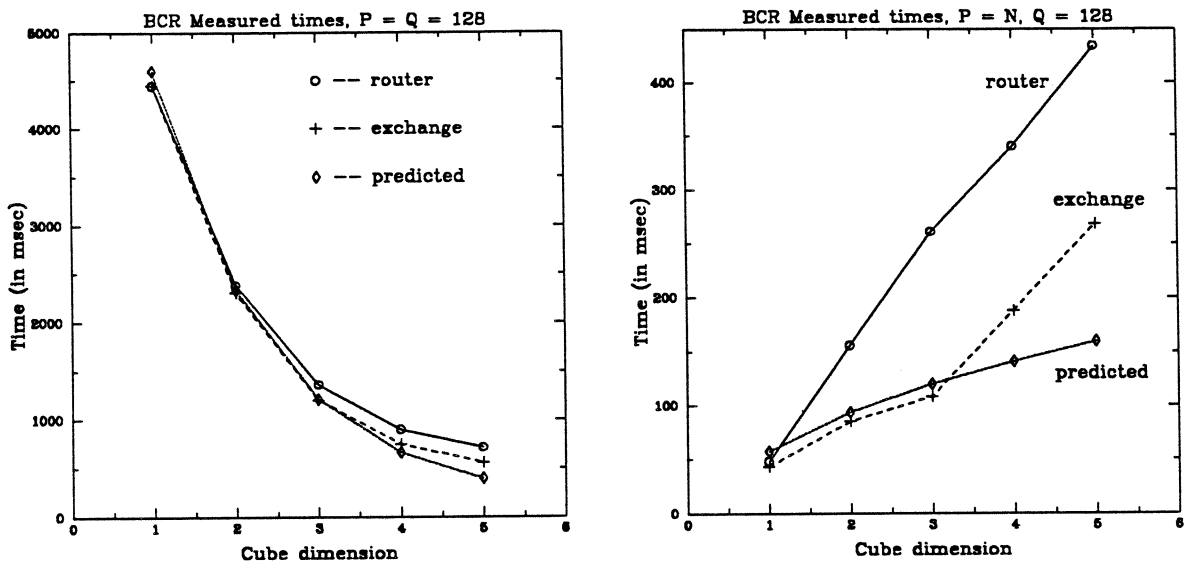


Figure 13: Measured and predicted times of balanced cyclic reduction (BCR) using both the *exchange* and the *in-place* algorithms.

In the T2GET algorithm an elimination is performed on half systems one step before the completion of the transpose operation. Only one equation for each of the $\frac{2Q}{N}$ systems needs to be exchanged in the last transpose step of the T2GET algorithm. Each processor solves half of the equations of $\frac{2Q}{N}$ systems. The arithmetic complexities of the TGET and T2GET algorithms are practically the same ($3Q$ additional operations for the T2GET method as implemented and described here) with $Q \bmod N = 0$. However, for $Q \leq \frac{N}{2}$, the arithmetic complexity of the T2GET algorithm is half of the TGET algorithm.

In carrying out the transpose some copying is necessary. With an optimum copying strategy the transpose expression is complicated [10]. The timing estimates for the TGET and T2GET algorithms without substructuring are

$$\begin{aligned} T_{TGET}(P,Q;n,t_a,t_c,\tau) \approx\ & \frac{Q}{2^n}(8P-7)t_a + 40n\frac{QP}{2\cdot 2^n}t_c \\ & + \{\frac{16QP}{2^n}\max(0,n-\log\lceil\frac{16QP}{B_{copy}2^n}\rceil) + \frac{4QP}{2^n}\max(0,n-\log\lceil\frac{4QP}{B_{copy}2^n}\rceil)\}t_{copy} \\ & + \{\min(n,\log\lceil\frac{16QP}{B_m2^n}\rceil)\lceil\frac{16QP}{2B_m2^n}\rceil + \min(2^n,\frac{16QP}{B_{copy}2^n}) - \min(2^n,\frac{16QP}{B_m2^n}) \\ & + \lceil\frac{16QP}{2B_m2^n}\rceil\max(0,n-\log\lceil\frac{16QP}{B_{copy}2^n}\rceil) \\ & + \min(n,\log\lceil\frac{4QP}{B_m2^n}\rceil)\lceil\frac{4QP}{2B_m2^n}\rceil + \min(2^n,\frac{4QP}{B_{copy}2^n}) - \min(2^n,\frac{4QP}{B_m2^n}) \\ & + \lceil\frac{4QP}{2B_m2^n}\rceil\max(0,n-\log\lceil\frac{4QP}{B_{copy}2^n}\rceil)\}2\tau, \end{aligned} \tag{18}$$

$$T_{T2GET}(P,Q;n,t_a,t_c,\tau) \approx \frac{Q}{2^n}(8P-4)t_a + (40(n-1)\frac{QP}{2\cdot 2^n} + 48\frac{Q}{2^n})t_c \tag{19}$$

	cube	Algorithm	
P, Q	dim.	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 3: Matrix transposition on the Intel iPSC for one-dimensional partitioning.

$$
\begin{aligned}
&+\{\frac{16QP}{2^n}\max(0,n-1-\log\lceil\frac{16QP}{B_{copy}2^n}\rceil)+\frac{4QP}{2^n}\max(0,n-1-\log\lceil\frac{4QP}{B_{copy}2^n}\rceil)\}t_{copy}\\
&+\{\min(n-1,\log\lceil\frac{16QP}{B_m2^n}\rceil)\lceil\frac{16QP}{2B_m2^n}\rceil+\min(\frac{2^n}{2},\frac{16QP}{B_{copy}2^n})-\min(\frac{2^n}{2},\frac{16QP}{B_m2^n})\\
&+\lceil\frac{16QP}{2B_m2^n}\rceil\max(0,n-1-\log\lceil\frac{16QP}{B_{copy}2^n}\rceil)\\
&+\min(n-1,\log\lceil\frac{4QP}{B_m2^n}\rceil)\lceil\frac{4QP}{2B_m2^n}\rceil+\min(\frac{2^n}{2},\frac{4QP}{B_{copy}2^n})-\min(\frac{2^n}{2},\frac{4QP}{B_m2^n})\\
&+\lceil\frac{4QP}{2B_m2^n}\rceil\max(0,n-1-\log\lceil\frac{4QP}{B_{copy}2^n}\rceil)+\lceil\frac{24Q}{2^nB_m}\rceil\}2\tau.
\end{aligned}
$$

The first line is the local Gaussian elimination time (first term) and the data transfer time for the transposition (second term). The second line is the time for copying the data from/to the buffer. The following four lines are the number of start-ups in the forward and backward transpositions respectively. In the T2GET algorithm, a processor sends $\frac{6Q}{N}$ elements and receives the same number of elements in an exchange operation required for the concurrent solution of the coupling 2×2 systems, compared to a total of $\frac{5PQ}{2N}$ elements for the last step of the forward and the first step of the backward transpose.

The estimated execution times for the substructured versions are

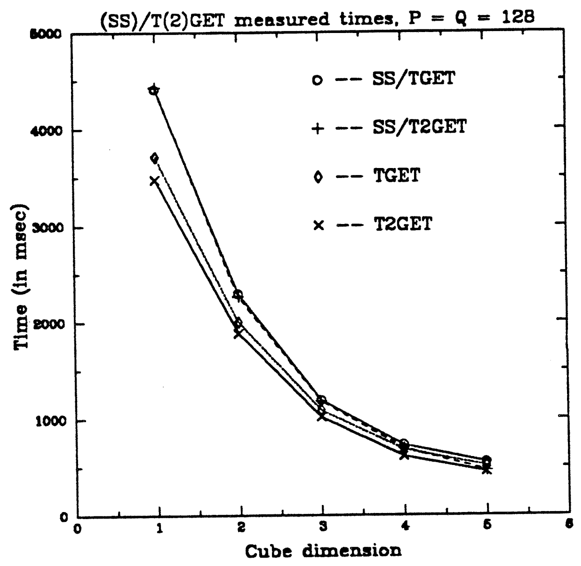
$$T_{SS/TGET}(P,Q;n,t_a,t_c,\tau)=T_{SS}(P,Q;n,t_a,t_c,\tau)+T_{TGET}(N,Q;n,t_a,t_c,\tau), \quad (20)$$

$$T_{SS/T2GET}(P,Q;n,t_a,t_c,\tau)=T_{SS}(P,Q;n,t_a,t_c,\tau)+T_{T2GET}(N,Q;n,t_a,t_c,\tau). \quad (21)$$

The reduction in the data transfer time offered by SS/T2GET compared to SS/TGET is approximately $\frac{1}{n}$. The number of start-ups is reduced by a factor of 0 to 2 depending on $\frac{Q}{N}$, B_{copy} and B_m. The factor of 2 applies when $\frac{4Q}{N} > \min(B_{copy}, B_m)$.

As in all previous cases an alternative to the optimized transpose algorithm used for the above estimate is to use the router. However, on the Intel iPSC 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 3.

For the arithmetic and communication parameters of the Intel iPSC, the complexity estimates of the TGET, SS/TGET, T2GET and SS/T2GET algorithms (expressions (18), (20), (19), and (21)) are compared in Figure 14 for up to 5-cubes. Measured times are given in the same Figure. We conclude that substructuring is not advantageous for the Intel iPSC. T2GET is always preferable over TGET (and SS/T2GET over SS/TGET).



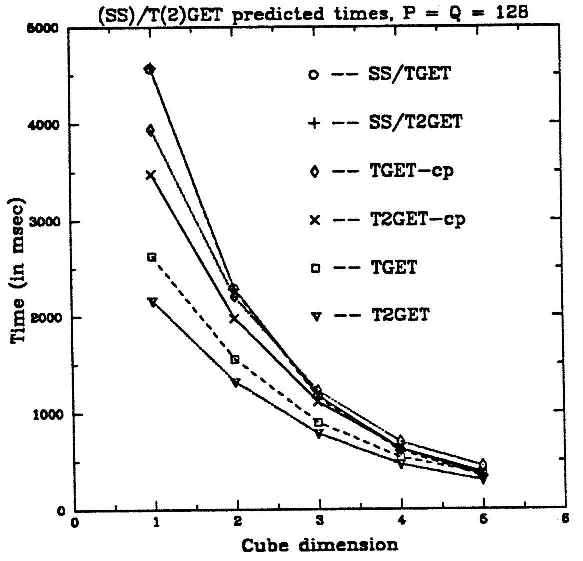


Figure 14: The measured and predicted times for the TGET, T2GET, SS/TGET and SS/T2GET algorithms. TGET-cp and T2GET-cp also include copy time.

Note the importance of including the copy time for a good agreement between measured and predicted times. A comparison for other parameters is made later.

In the substructured as well as the non-substructured estimates above we assume that all the data for an equation are 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.

We now make a simplified comparison between the SS/BCR and the SS/T2GET algorithms. The latter algorithm has about half the number of floating-point operations of the SS/BCR algorithm for the reduced systems. We consider two extreme cases for the evaluation of the communication complexity: 1) $B_m \geq \frac{16Q}{2}$, $B_{copy} \geq \frac{16Q}{4}$ and 2) $B_m = B_{copy} \leq \frac{4Q}{N}$. In the first case the number of start-ups for SS/BCR is $8n$ and that of SS/T2GET $4n+2$. The transpose algorithms have about half as many start-ups as the SS/BCR algorithm. In the second case, the number of start-ups compares as $120\frac{Q}{B_m}(1-\frac{1}{N})$ for SS/BCR and $20(n+1)\frac{Q}{B_m}$ for the SS/T2GET algorithm. In this case the SS/T2GET algorithm requires a factor of $\frac{1}{6}(n+1)$ more start-ups than the SS/BCR algorithm. The complexity of the data transmission term for the SS/T2GET algorithm is a factor of $\frac{1}{6}(n+1)$ larger than that of the SS/BCR algorithm. For a large maximum packet size the SS/T2GET algorithm may be more efficient than the SS/BCR algorithm, but for a small maximum packet size SS/BCR is of lower complexity for sufficiently large n. The total time for both cases are given in equations (22) and (23) respectively.

$$T^{case-1}_{SS/T2GET}(P,Q;n,t_a,t_c,\tau) \approx \tag{22}$$
$$17Q(\lceil\frac{P}{2^n}\rceil - 1)t_a + 2(16+4)Qt_c + 2(\lceil\frac{16Q}{B_m}\rceil + \lceil\frac{4Q}{B_m}\rceil)\tau$$

τ	0.1	2	8	2	2	2
$\frac{Q}{N}$	1	1	1	2	8	32
break-even dim.	12.0	8.2	8.8	7.7	7.8	9.6

Table 4: The break-even points of SS/BCR and SS/TGET as a function of the number of systems, Q, and the number of processors, N, or as a function of τ.

$$+\frac{Q}{2^n}(8\cdot 2^n-4)t_a+(40(n-1)\frac{Q}{2}+48\frac{Q}{2^n})t_c$$
$$+20(n-1)Qt_{copy}+(4n-2)\tau.$$

$$T^{case-2}_{SS/T2GET}(P,Q;n,t_a,t_c,\tau)\approx \tag{23}$$
$$17Q(\lceil\frac{P}{2^n}\rceil-1)t_a+2(16+4)Qt_c+2(\lceil\frac{16Q}{B_m}\rceil+\lceil\frac{4Q}{B_m}\rceil)\tau$$
$$+\frac{Q}{2^n}(8\cdot 2^n-4)t_a+(40(n-1)\frac{Q}{2}+48\frac{Q}{2^n})t_c$$
$$+(10(n-1)+\frac{24}{2^n})\frac{Q}{B_m}2\tau.$$

With the same machine parameters as we used before ($\tau = 8$, 2 and 0.1 msec, $t_a = 33$ μsec, $t_c = 1$ μsec) the predicted break-even point between the SS/BCR and the SS/T2GET algorithms both based on exchanges are at approximately $n = 8.8$, 8.2 and 12.0, respectively for $Q = N$. The break-even points are computed from equations (17) and (21), Figure 15. As τ increases, the break-even point first decreases and then increases again. For a small τ, such as 0.1 msec, the arithmetic time dominates until the cube becomes sufficiently large for the data transfer time for T2GET to offset its arithmetic advantage over BCR. When τ is small, the copy time is relatively larger, and more direct sends and receives are made than for a large τ. Hence, with an optimized transpose algorithm the total start-up time may increase more slowly than the time for a single start-up for T2GET. The number of start-ups for T2GET is larger than that of BCR, and as τ increases the break-even point moves towards lower values of n. However, for a large τ such as 8 msec, the number of start-ups of T2GET is reduced for the optimum algorithm compared to the naive transpose by doing copying, so the effective number of start-ups for T2GET could be less than that of BCR. T2GET is less sensitive to the increase of τ than BCR.

The lower row of plots in Figure shows how the break-even point depends on $\frac{Q}{N}$. As it increases the break-even point first moves towards lower values of n, then again towards higher values of n as the required packet size of the SS/BCR exceeds the maximum packet size. Table 4 gives the values of the break-even points of Figure 15.

The break-even dimension between SS/BCR and SS/T2GET for $Q = N$ is mostly higher than that between SS/CR and SS/T2GET for $Q = 1$ (the single system), Figures 8 and 15.

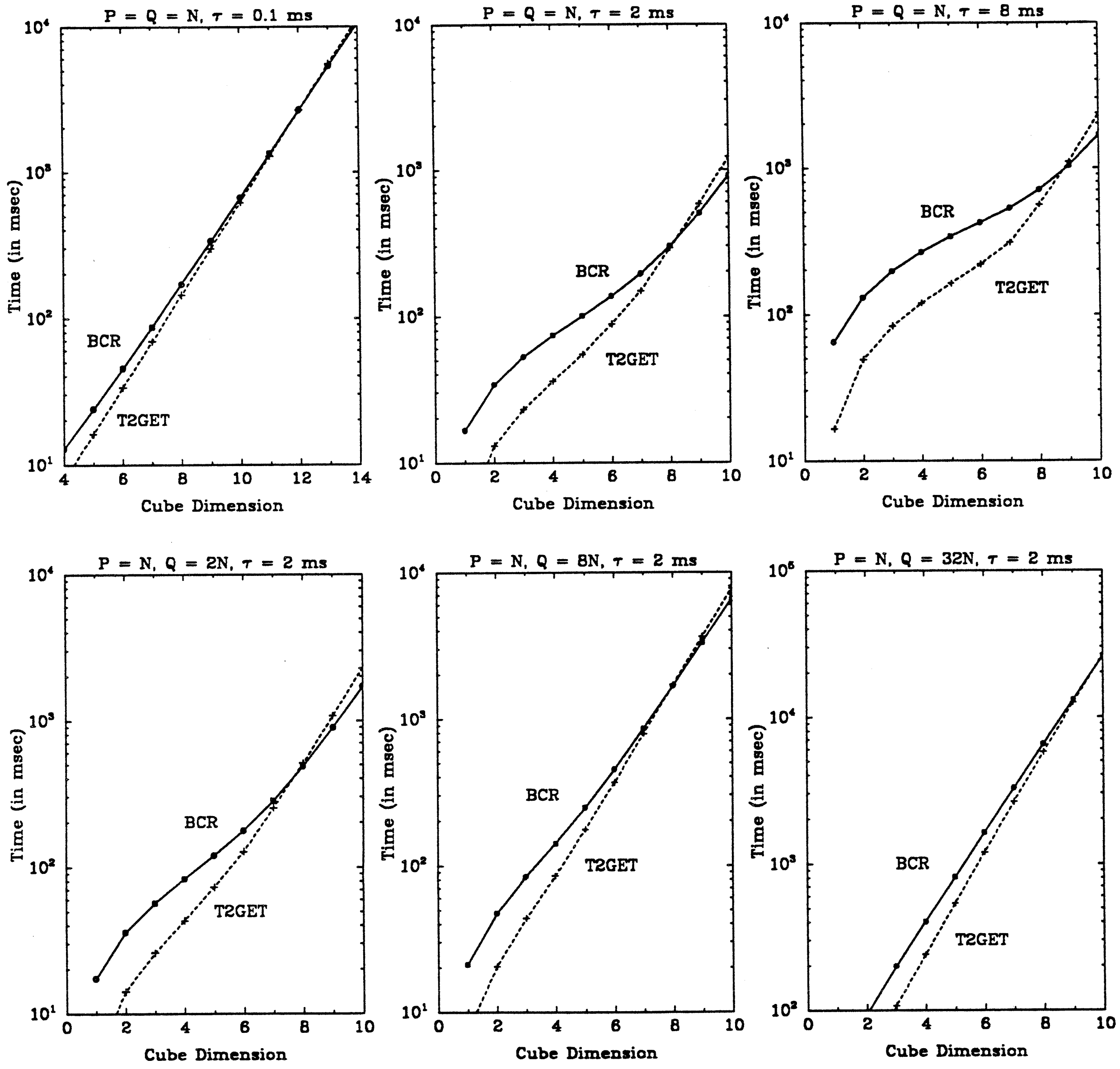


Figure 15: The predicted execution times for the SS/BCR (solid line) and SS/T2GET (dashed line) algorithms, $\tau = 0.1$, 2 and 8 msec for $P = Q = N$. The lower part shows the predicted times for $\tau = 2$ msec, $P = N$, and $\frac{Q}{N} = 2$, 8 and 32, respectively.

System	$\frac{Q}{N}$	$\tau = 0.1$ msec	$\tau = 2$ msec	$\tau = 8$ msec
Real	1	$n-11$	$n-6$	$n-7$
	8	$n-16$	$n-6$	$n-5$
	32	$n-17$	$n-8$	$n-5$
	128	$n-17$	$n-9$	$n-5$
Complex	1	$0, n \leq 30$	$n-11$	$n-7$
	8	$0, n \leq 30$	$n-15$	$n-7$
	32	$0, n \leq 30$	$n-16$	$n-8$
	128	$0, n \leq 30$	$n-17$	$n-8$

Table 5: The optimum number of resuction steps, ℓ, as a function of τ and $\frac{Q}{N}$.

3.3 A Hybrid Scheme

The hybrid scheme (14) for the single system case has an obvious analogue for multiple systems. We perform ℓ steps of SS/BCR, then switch to the SS/T2GET algorithm. The cost of this hybrid scheme is given by combining (17) and (21) as follows:

$$\begin{aligned} T_{SS/H}(P,Q,\ell;n,t_a,t_c,\tau) &= T_{SS}(P,Q;n,t_a,t_c,\tau) + T_{BCR}(N,Q;\ell,t_a,t_c,\tau) \\ &\quad + T_{T2GET}(\frac{N}{2^\ell},\frac{Q}{2^\ell};n-\ell,t_a,t_c,\tau). \end{aligned} \tag{24}$$

In the context of the hybrid method a slight modification to the complexity expression for BCR, equation (16) is necessary. The number of start-ups is increased by $2(\lceil\frac{16Q}{2^\ell B_m}\rceil + \lceil\frac{4Q}{2^\ell B_m}\rceil)$ and the coefficient of t_c is increased by $2(16+4)\frac{Q}{2^\ell}$ if $\ell < n$. (There are fewer communications in the last step of BCR if run to completion). The optimal ℓ predicted by (24) is shown in Figure 16 and Table 5 for various $\frac{Q}{N}$ and τ. Figure 17 shows the optimum ℓ as a function of $\log\frac{Q}{N}$ and $\log N$ with different values of τ and the same values for t_a, and t_c used previously for the Intel iPSC. Figure 18 shows the optimum ℓ as a function of $\log\frac{\tau}{t_c}$ and $\log\frac{t_a}{t_c}$ with $P=Q=N=32$ (upper plots) and 1024 (lower plots) respectively. As expected, increasing $\frac{t_a}{t_c}$ favors the SS/T2GET method. Increasing $\frac{\tau}{t_c}$ first increases the competitive domain of the SS/BCR algorithm, later reduces it again. The reason for this behavior is that for small start-up times, the transpose algorithm [10] will choose *direct* communication for each non-contiguous block, and the number of start-ups is higher than for SS/BCR. For large start-up times, the SS/T2GET algorithm with the optimum transpose algorithm will copy to a buffer to reduce the number of start-ups, and hence the total number of start-ups is less than that of the SS/BCR.

Table 5 also lists the optimum values of ℓ for complex systems. The optimum number of reduction steps, ℓ, is decreased from real to complex systems for fixed cube sizes. The difference is more significant for a smaller value of τ.

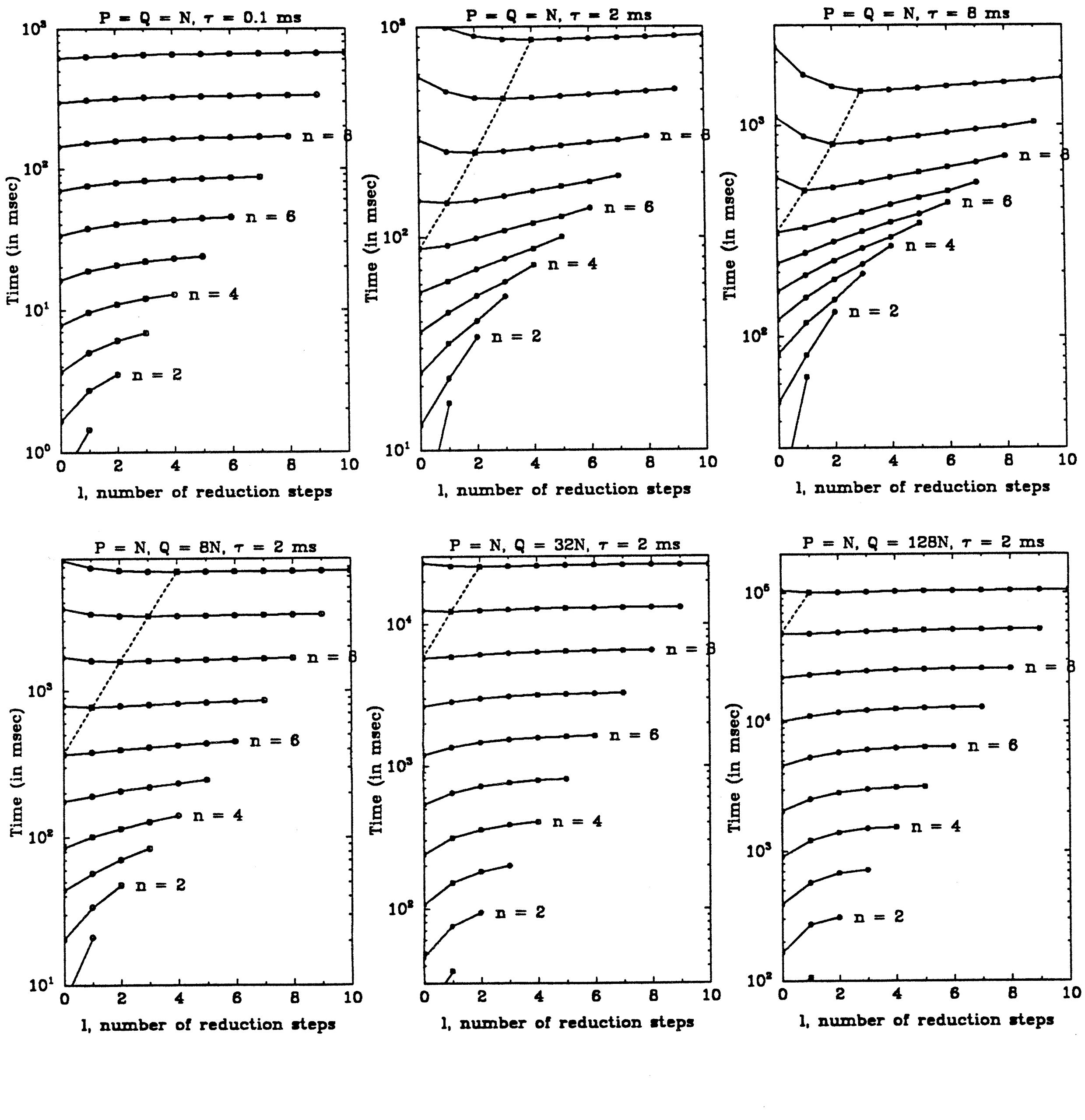


Figure 16: The predicted times for the Hybrid method as a function of the number of reduction steps, ℓ, with $\tau = 0.1$, 2, and 8 msec, $P = Q = N$ (upper plots), and $P = N$, $Q = 8N$, $32N$, and $128N$ for $\tau = 2$ msec (lower plots).

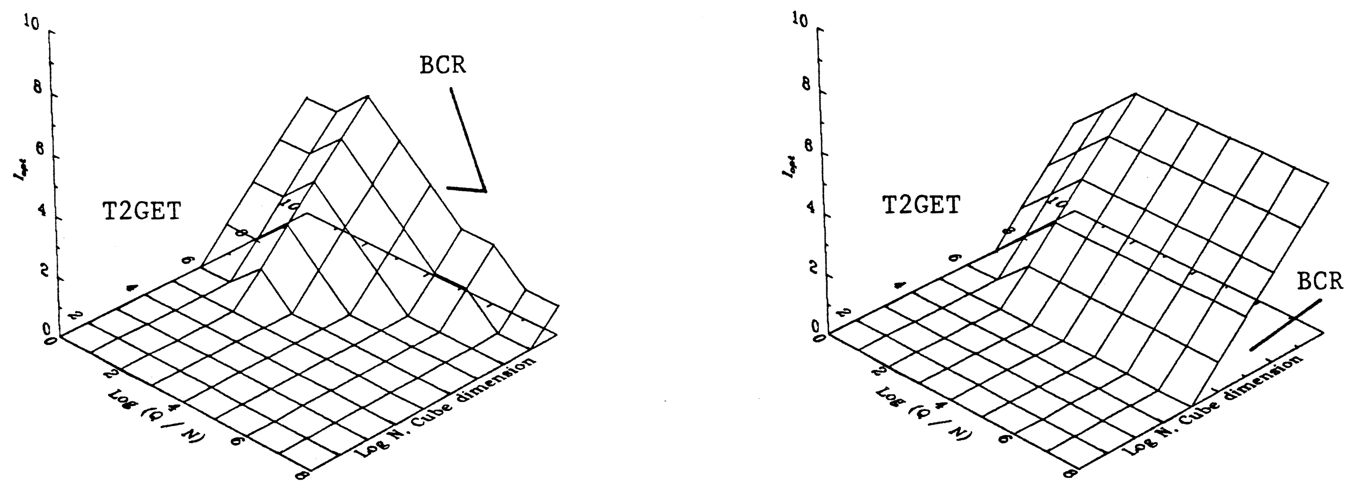


Figure 17: The optimum number of reduction steps, ℓ, as a function of $\log \frac{Q}{N}$ and $\log N$ with $\tau = 2$ (left) and 8 msec (right), and $t_a = 33$ μsec, $t_c = 1$ μsec.

Algorithm	Arithmetic	Element transfers	Min start-ups	Memory
TGET	$8\frac{PQ}{N} - 7\frac{Q}{N}$	$20\frac{nPQ}{N}$	$4n$	$\max(\lceil\frac{P}{N}\rceil Q, P\lceil\frac{Q}{N}\rceil)$
T2GET	$8\frac{PQ}{N} - 4\frac{Q}{N}$	$20\frac{(n-1)PQ}{N} + 48\frac{Q}{N}$	$4n-2$	$\max(\lceil\frac{P}{N}\rceil Q, \frac{P}{2}\lceil\frac{2Q}{N}\rceil)$
SS/TGET	$17\frac{PQ}{N} - 9Q - 7\frac{Q}{N}$	$20(n+2)Q$	$4n+4$	$\max(\lceil\frac{P}{N}\rceil Q, N\lceil\frac{Q}{N}\rceil)$
SS/T2GET	$17\frac{PQ}{N} - 9Q - 4\frac{Q}{N}$	$20(n+1)Q + 48\frac{Q}{N}$	$4n+2$	$\max(\lceil\frac{P}{N}\rceil Q, \frac{N}{2}\lceil\frac{2Q}{N}\rceil)$
SS/BCR	$17\frac{PQ}{N} - 17\frac{Q}{N}$	$120Q - 120\frac{Q}{N}$	$8n$	$\lceil\frac{P}{N}\rceil Q$

Table 6: Complexity comparison of algorithms for multiple tridiagonal systems.

3.4 Comparison of Methods for Multiple Tridiagonal Systems, $Q \bmod N = 0$

Table 6 summarizes the complexities of the algorithms. With respect to arithmetic complexity the TGET or T2GET algorithms shall always be used. The required number of operations is approximately a factor of two lower than that of the other methods. If the communication is a significant factor, then the choice of algorithm is more intricate.

Figure 19 (left) shows the measured iPSC times of the SS/BCR, SS/T2GET and T2GET algorithms for $P = Q = 128$, $Q \bmod N = 0$, for up to 5-cubes. The corresponding predicted times are shown on the right. The agreement between predicted and measured times is good. The difference in measured execution times for the SS/BCR and SS/T2GET algorithms is negligible, with a 25% advantage for the SS/T2GET algorithm on a 5-cube. These measurements agree with the predicted behavior.

The arithmetic complexities of the SS/TGET and SS/T2GET algorithms are lower than

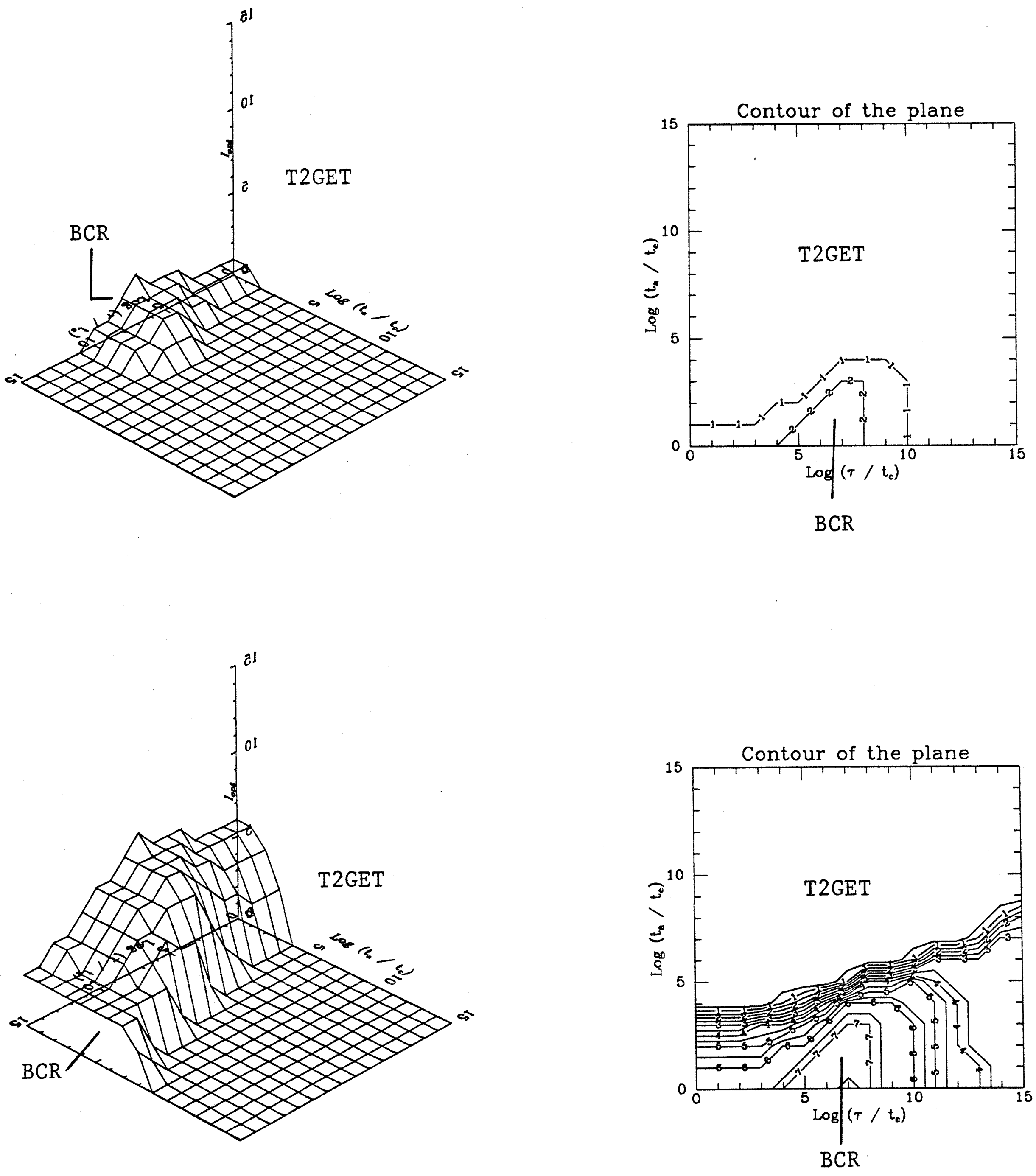
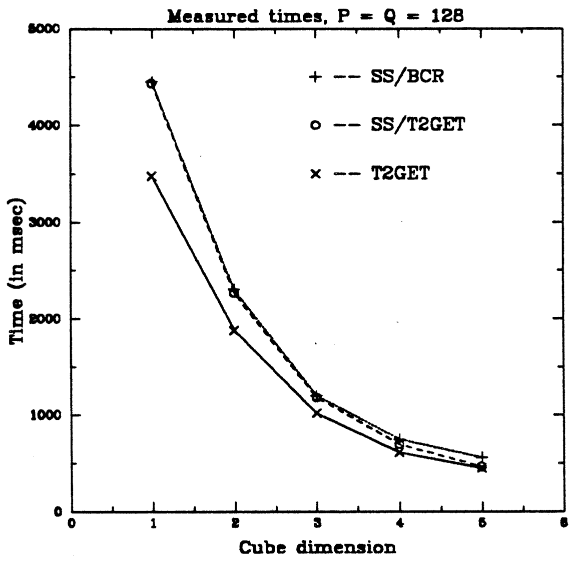


Figure 18: The optimum number of reduction steps, ℓ, as a function of $\log \frac{\tau}{t_c}$ and $\log \frac{t_a}{t_c}$ with $P = Q = N = 32$ (upper plots) and 1024 (lower plots) respectively.



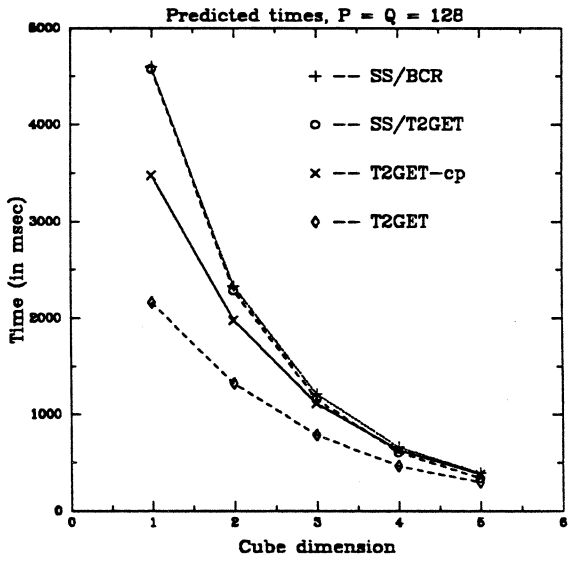


Figure 19: The measured (left) and the predicted (right) times of the SS/BCR, SS/T2GET and T2GET algorithms for $P = Q = 128$.

that of the SS/BCR by approximately $9Q$. The data transfer volume of the TGET and T2GET algorithms are larger than that of the SS/TGET and SS/T2GET algorithms by a factor of $\frac{P}{N}$, and a factor of $\approx \frac{nP}{6N}$ greater that that of the SS/BCR algorithm. The number of start-ups depends on the maximum packet size, B_m. For a sufficiently large maximum packet size, the number of start-ups of the TGET and T2GET algorithms is less than that of the SS/TGET and SS/T2GET algorithms by a small constant factor, since there is no communication for substructuring. The number of start-ups for SS/BCR is approximately twice that of the algorithms based on transposition. However, for a relatively small packet size compared to $\frac{PQ}{N}$, the number of start-ups may become proportional to the number of element transfers, in which case the TGET and T2GET algorithms require a substantially higher number of start-ups than the SS/TGET and SS/T2GET algorithms, which in turn might require a higher number of start-ups than the SS/BCR algorithm. Figure 20 shows the algorithm of lowest predicted execution time as a function of $\log \frac{\tau}{t_c}$ and $\log \frac{t_a}{t_c}$ with different values of P, Q and N. It is assumed that B_m = 1k bytes for the plots on the left and $B_m = \infty$ for the plots on the right. With slow arithmetic T2GET is the method of choice with respect to execution time. With the time for an arithmetic operation of the same order as the time for communication of an element substructuring shall always be used, and the choice is between SS/H and T2GET. The region for BCR increases with N, and decreases with $\frac{Q}{N}$.

Substructuring shall always be used whenever the BCR algorithm is used. If the T2GET algorithm is used, then measurements as well as predictions for arithmetic and communication parameters such as those of the Intel iPSC show that the transpose algorithm without substructuring shall be used, whenever SS/BCR is not the algorithm of choice.

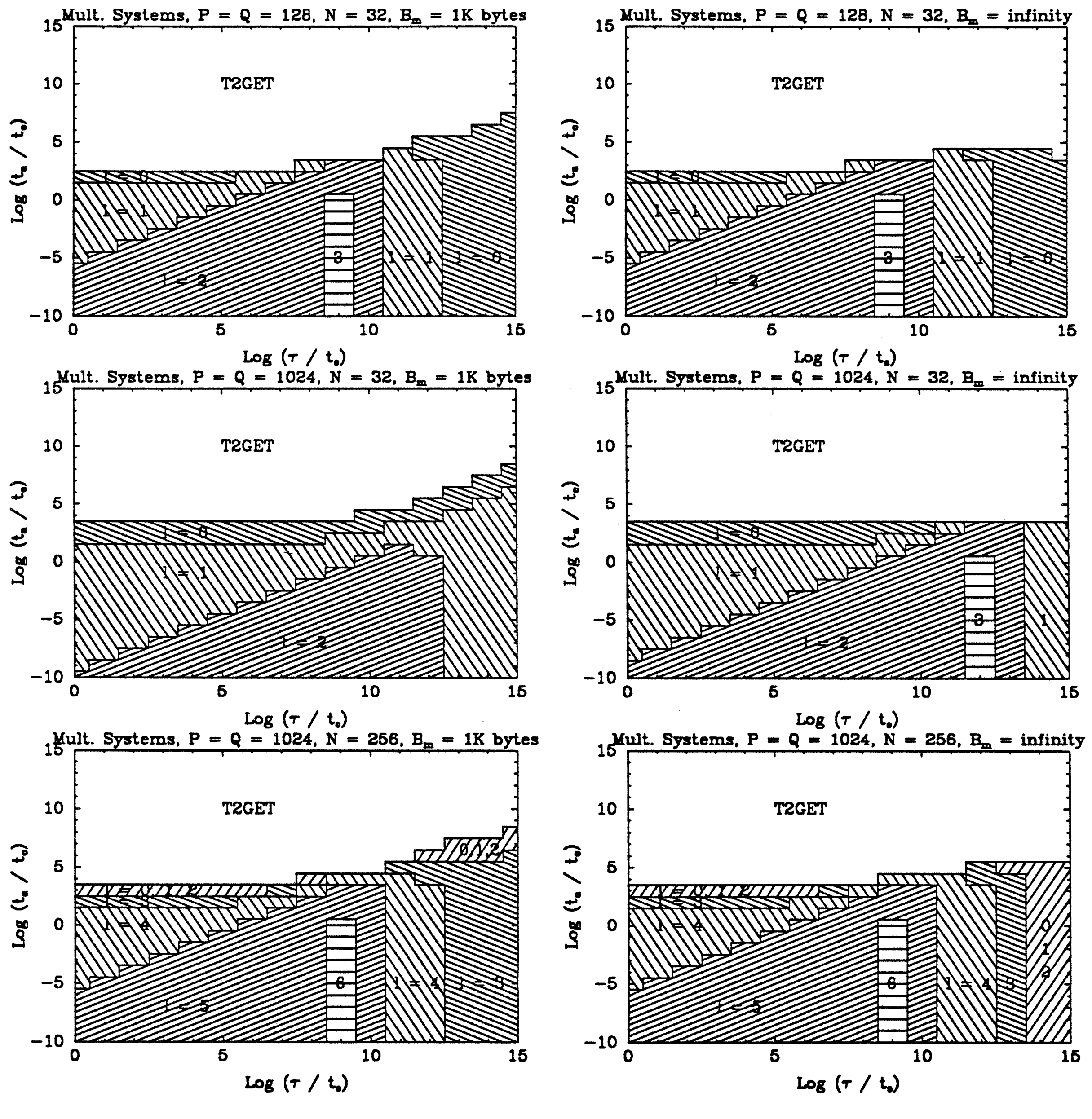


Figure 20: Algorithms of lowest estimated execution time as a function of $\log \frac{\tau}{t_c}$ and $\log \frac{t_a}{t_c}$.

3.4.1 $Q \bmod N \neq 0$

In the above derivations we assumed that N divides Q. To address the case where this condition is not true we consider the case of $Q < N$ (or $Q \bmod N \neq 0$ if $Q > N$). We first consider the SS/BCR algorithm. The number of systems treated by a processor after ℓ reduction steps is $\frac{Q}{2^\ell}$ and the size of each such system is $\frac{N}{2^\ell}$. Hence, after $\log Q$ steps of BCR we have one system of order $\frac{N}{Q}$ on a subcube with $\frac{N}{Q}$ processors. There are Q independent subcubes. Each of the systems can be solved by one of the methods described above for a single system, e.g. (8).

$$\begin{aligned}
&T_{SS/BCR}(P,Q;n,t_a,t_c,\tau) \approx \qquad (25)\\
&\quad 17Q(\lceil\frac{P}{N}\rceil - 1)t_a + 2(16Qt_c + \lceil\frac{16Q}{B_m}\rceil\tau) + 2(4Qt_c + \lceil\frac{4Q}{B_m}\rceil\tau)\\
&\quad + 17(Q-1)t_a + 4(16+4)(Q-1)t_c + 4\sum_{i=1}^{\log Q}(\lceil\frac{16Q}{2^i B_m}\rceil + \lceil\frac{4Q}{2^i B_m}\rceil)\tau\\
&\quad + (8\frac{N}{Q} - 7)t_a + 20(\frac{N}{Q} - 1)t_c + \sum_{i=0}^{\log\frac{N}{Q}-1}(\lceil\frac{16\times 2^i}{B_m}\rceil + \lceil\frac{4\times 2^i}{B_m}\rceil)\tau.
\end{aligned}$$

For the transpose based methods not all processors will have a system to solve for $Q < N$, but those that do carry out the same amount of arithmetic as in the case $Q = N$. In the transposition the same number of steps is required, but in each exchange operation the data volume is different for a send and a receive. However, the maximum is the same as in the case $Q = N$. The data transfer time is lower than that of the $Q = N$ case by at most a factor of 2, if the communication system allows one send *or* one receive, but remains the same if sends and receives can be performed concurrently. Similarly, the number of start-ups may be smaller for $Q < N$ than $Q = N$, if the data volume is large compared to the maximum packet size. We do not give explicit expressions for the expected execution time.

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 estimate for the hybrid method and two-dimensional domain decomposition with $N_1 > 1$ is

$$\begin{aligned}
&T_{SS/H}(P, \frac{Q}{N_2}, \ell; n_1, t_a, t_c, \tau) \approx \\
&17\frac{Q}{N_2}(\lceil\frac{P}{N_1}\rceil - 1)t_a + 2(16\frac{Q}{N_2} + 4\frac{Q}{N_2})t_c + 2(\lceil\frac{16Q}{N_2 B_m}\rceil + \lceil\frac{4Q}{N_2 B_m}\rceil)\tau \\
&+ 17\frac{Q}{N_2}(1 - \frac{1}{2^\ell})t_a + 80\frac{Q}{N_2}(1 - \frac{1}{2^\ell})t_c + 4\sum_{i=1}^{\ell}(\lceil\frac{16Q}{2^i N_2 B_m}\rceil + \lceil\frac{4Q}{2^i N_2 B_m}\rceil)\tau \\
&+ \frac{Q}{N_1 \cdot N_2}(8\frac{N_1}{2^\ell} - 4)t_a + \{40(n_1 - \ell - 1)\frac{Q}{N_2 2^{\ell+1}} + 48\frac{Q}{N_2 \cdot N_1}\}t_c \\
&+ \{16\frac{Q}{N_2 2^\ell}\max(0, n_1 - \ell - 1 - \log\lceil\frac{16Q}{N_2 2^\ell B_{copy}}\rceil) \\
&+ 4\frac{Q}{N_2 2^\ell}\max(0, n_1 - \ell - 1 - \log\lceil\frac{4Q}{N_2 2^\ell B_{copy}}\rceil)\}t_{copy} \\
&+ \{\min(n_1 - \ell - 1, \log\lceil\frac{16Q}{N_2 2^\ell B_m}\rceil)\lceil\frac{16Q}{N_2 2^{\ell+1} B_m}\rceil + \min(\frac{N_1}{2^{\ell+1}}, \frac{16Q}{N_2 2^\ell B_{copy}}) \\
&- \min(\frac{N_1}{2^{\ell+1}}, \frac{16Q}{N_2 2^\ell B_m}) + \lceil\frac{16Q}{N_2 2^{\ell+1} B_m}\rceil\max(0, n_1 - \ell - 1 - \log\lceil\frac{16Q}{N_2 2^\ell B_{copy}}\rceil) \\
&+ \min(n_1 - \ell - 1, \log\lceil\frac{4Q}{N_2 2^\ell B_m}\rceil)\lceil\frac{4Q}{N_2 2^{\ell+1} B_m}\rceil + \min(\frac{N_1}{2^{\ell+1}}, \frac{4Q}{N_2 2^\ell B_{copy}}) \\
&- \min(\frac{N_1}{2^{\ell+1}}, \frac{4Q}{N_2 2^\ell B_m}) + \lceil\frac{4Q}{N_2 2^{\ell+1} B_m}\rceil\max(0, n_1 - \ell - 1 - \log\lceil\frac{4Q}{N_2 2^\ell B_{copy}}\rceil)\}2\tau.
\end{aligned} \tag{26}$$

The expression (26) is simply the time estimate of the hybrid approach (24) applied with $N \leftarrow N_1$, $Q \leftarrow Q/N_2$ and $N = N_1 \times N_2$. The substructuring part has the same number of floating-point operations as in the one-dimensional domain decomposition cases, (17), (21) and (24), but the complexity of the other components is lower. Clearly, the time is minimized if N_2 is maximized, and indeed the formula shows that 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 for this equation consists of two half steps,

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

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

In each half step computations are performed independently on grid rows or columns, i.e., on independent subcubes [20] . 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

$$T_{MPY}(P, \frac{Q}{N_2}; n_1, t_a, t_c, \tau) = 5\frac{P}{2^{n_1}}\frac{Q}{N_2}t_a + 16\frac{Q}{N_2}t_c + \lceil\frac{4Q}{N_2 B_m}\rceil 4\tau, \qquad (27)$$

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. For a small cube, like the Intel iPSC, the T2GET algorithm is likely to be the most efficient and the estimated time for a half step is

$$T_{ADI_{\frac{1}{2}}}(P, Q; n_1, n_2, t_a, t_c, \tau) = T_{MPY}(P, \frac{Q}{N_2}; n_1, t_a, t_c, \tau) + T_{SS/H}(Q, \frac{P}{N_1}, \ell_2; t_a, t_c, \tau), \quad (28)$$

and for a complete step

$$\begin{aligned} & T_{ADI_1}(P, Q; n_1, n_2, t_a, t_c, \tau) \qquad (29) \\ = \; & T_{MPY}(P, \frac{Q}{N_2}; n_1, t_a, t_c, \tau) + T_{MPY}(Q, \frac{P}{N_1}; n_2, t_a, t_c, \tau) \\ & + T_{SS/H}(Q, \frac{P}{N_1}, \ell_2; t_a, t_c, \tau) + T_{SS/H}(P, \frac{Q}{N_2}, \ell_1; t_a, t_c, \tau). \end{aligned}$$

For a given cube a decision has to be made as to how to choose N_1 and N_2. In the matrix multiplication and substructuring phases these parameters only enters in expressions of the form $\frac{P}{N_1} + \frac{Q}{N_2}$. For the balanced cyclic reduction part there are terms of this form and also terms of the form $\frac{P}{N_1}\frac{1}{2^{\ell}_2} + \frac{Q}{N_2}\frac{1}{2^{\ell}_1}$. If ℓ_1 and ℓ_2 are 0 then clearly these terms are like the previous ones. Our previous analysis of the hybrid method showed that it is approximately true that $n - \ell = const$ for the optimum choice of the number of reduction steps. Hence, for such a choice of ℓ it is also true that N_1 and N_2 appear in expressions of the form $\frac{P}{N_1} + \frac{Q}{N_2}$ for Balanced Cyclic Reduction. It can also be verified that for the optimum choice of the number of reduction steps the same property is true for the T2GET phase. Hence, with an optimum choice of the number of reduction steps, the total execution time is minimized to first order, if $\frac{P}{N_1} + \frac{Q}{N_2}$ is minimized. Hence,

$$\frac{P}{N_1} = \frac{Q}{N_2} \quad \text{or} \quad N_1 = \sqrt{\frac{PN}{Q}}, \quad \text{which implies} \quad (\frac{P}{N_1} + \frac{Q}{N_2}) = 2\sqrt{\frac{PQ}{N}}.$$

The optimal aspect ratio of the processor grid is equal to the aspect ratio of the computational grid.

5 Summary and Conclusions

The execution time for multiprocessors with a packet switched communication systems with 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. The model was verified on the Intel iPSC, for which it is also necessary to account for the copy time.

The analysis as well as the experiments on the Intel iPSC show that odd-even cyclic reduction can be competitive with parallel cyclic reduction with respect to arithmetic, and in particular with respect to communication. Odd-even cyclic reduction performed considerably better than parallel cyclic reduction on the Intel iPSC. A similar result has also been observed on the AMETEK system S14 [21].

For sufficiently small cubes and high start-up times for communication data transposition followed by local Gaussian elimination may be preferable to a parallel algorithm such as odd-even cyclic reduction, or in the case of multiple systems, balanced cyclic reduction. For any given set of arithmetic and communication parameters, a parallel algorithm eventually becomes preferable as the machine size grows. For the particular parameters that apply for the Intel iPSC a sequential algorithm and data transposition is preferable.

It follows that for a machine with many processors a parallel algorithm should be used for the first several reduction steps, then a switch be made to an algorithm using transposition, local elimination, and transposition. The number of reduction steps replaced by data transposition and local solution decreases with increased start-up time, and increases with the number of systems per processor. The number of reduction steps replaced by a transposition – local solution algorithm is higher for complex systems than for real systems. For complex systems, $\tau = 0.1$ msec, $t_a = 33$ μsec, and $t_c = 1$ μsec the transpose based algorithms is always preferable for $n \leq 30$. For real systems and the same set of values of τ, t_a and t_c, the optimum switching point to transposed based algorithms ranges from 11 to 17 for the number of systems per subcube in the range 1 – 128. The optimum is rather flat.

For transpose based algorithms we note that the transposition can be terminated one step before completion, and two-way elimination used. The total arithmetic complexity of two-way elimination is the same as that of standard Gaussian elimination, but the communication need is less.

For a one-directional solution of multiple systems of tridiagonal equations the optimum assignment of equations to processors corresponds to the embarrassingly parallel case. For solution in alternating directions of equations associated with a lattice the optimum aspect ratio of the processor grid is to first order equal to that of the lattice forming the base for the equations.

We conclude that combining in the routing system is important for the performance of several of the algorithms. The required combining is of the merge/split type (rather than copy).

Acknowledgement

Many discussions with Faisal Saied, who independently implemented a version of the BCR algorithm in the context of fast Poisson solvers, contributed to the development of our implementation of the same algorithm. This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564.

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

[3] J.O. Eklundh. A fast computer method for matrix transposing. *IEEE Trans. Computers*, C-21(7):801–803, 1972.

[4] D. Evans and M. Hatzopoulus. The solution of certain banded systems of linear equations using the folding algorithm. *Computer Journal*, 19:184–187, 1976.

[5] Dennis Gannon and John Van Rosendale. On the impact of communication complexity in the design of parallel numerical algorithms. *IEEE Trans. Computers*, C-33(12):1180–1194, December 1984.

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

[7] W. Daniel Hillis. *The Connection Machine.* MIT Press, 1985.

[8] Ching-Tien Ho and S. Lennart Johnsson. Algorithms for matrix transposition on Boolean n-cube configured ensemble architectures. In *Int. Conf.on Parallel Processing*, IEEE Computer Society, 1987.

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

[10] Ching-Tien Ho and S. Lennart Johnsson. *Matrix Transposition on Boolean n-cube Configured Ensemble Architectures.* Technical Report YALEU/CSD/RR-494, Yale University, Dept. of Computer Science, September 1986.

[11] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *Int. Conf.on Parallel Processing*, IEEE Computer Society, 1987.

[12] Ching-Tien Ho and S. Lennart Johnsson. *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes.* Technical Report YALEU/CSD/RR-500, Yale University, Dept. of Computer Science, November 1986.

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

[14] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, 4(2):133–172, April 1987. Report YALEU/CSD/RR-361, January 1985, Dept. of Computer Science, Yale University.

[15] S. Lennart Johnsson. *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures.* Technical Report YALEU/CSD/RR-367, Yale University, Dept. of Computer Science, February 1985.

[16] S. Lennart Johnsson. Fast PDE solvers on fine and medium grain architectures. In *Int. Assoc. for Mathematics and Computers in Simulation*, IMACS, 1987.

[17] S. Lennart Johnsson. *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations.* Technical Report YALE/CSD/RR-339, Department of Computer Science, Yale University, October 1984.

[18] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Stat. Comp.*, 8(3):354–392, May 1987. Report YALEU/CSD/RR-436, November 1985.

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

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

[21] David S. Lim and Rex V. Thanakij. A survey of alternating direction implicit (ADI) method implementations on hypercubes. In *The 1986 Hypercube Conference*, SIAM, 1987.

[22] Abhiram Ranade and S. Lennart Johnsson. The communication efficiency of meshes, Boolean cubes, and cube connected cycles for wafer scale integration. In *Int. Conf. on Parallel Processing*, IEEE Computer Society, 1987.

[23] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice Hall, 1977.

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

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

[26] Harold S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Computers*, C-20:153–161, 1971.

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

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

[29] J. Wilkinson. Unpublished note.