# Parallel Algorithms for Forward Elimination and Backward Substitution in Direct Solution of Sparse Linear Systems*

Anshul Gupta and Vipin Kumar

Department of Computer Science
University of Minnesota
Minneapolis, MN - 55455

*agupta@cs.umn.edu* *kumar@cs.umn.edu*

Presenting Author: Anshul Gupta
Corresponding Author: Vipin Kumar; Phone: 612-624-8023; Fax: 612-625-0572.

**Abstract**

*A few parallel algorithms for solving triangular systems resulting from parallel factorization of sparse linear systems have been proposed and implemented recently. We present a detailed analysis of parallel complexity and scalability of the best of these algorithms and the results of its implementation on up to 256 processors of the Cray T3D parallel computer. It has been a common belief that parallel sparse triangular solvers are quite unscalable due to a high communication to computation ratio. Our analysis and experiments show that, although not as scalable as the best parallel sparse Cholesky factorization algorithms, parallel sparse triangular solvers can yield reasonable speedups in runtime on hundreds of processors. We also show that for a wide class of problems, the sparse triangular solvers described in this paper are optimal and are asymptotically as scalable as a dense triangular solver.*

*This work is sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army and Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008. The contents of of this paper do not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to computing facilities was provided by Cray Research Inc., Minnesota Supercomputer Institute and by the Pittsburgh Supercomputing Center. Related papers are available via at URL: http://www.cs.umn.edu/users/kumar/papers.html via WWW.

# 1 Introduction

The process of obtaining a direct solution to a sparse system of linear equations usually consists of four phases: reordering, symbolic factorization, numerical factorization, and solving the lower- and upper-triangular systems resulting from factorization. Since numerical factorization is computationally the most expensive phase, a significant research effort has been directed towards developing efficient and scalable parallel sparse factorization algorithms. We have recently proposed [4] a parallel sparse Cholesky factorization algorithm that is optimally scalable for a wide class of problems. Experiments have shown that this algorithm can easily speedup Cholesky factorization by a factor of at least a few hundred on up to 1024 processors. With such speedups in numerical factorization, it is imperative that the remaining phases of the solution process be parallelized effectively in order to scale the performance of the overall solver. Furthermore, without an overall parallel solver, the size of the sparse systems that can be solved may be severely restricted by the amount of memory available on a uniprocessor system.

In this paper, we address the problem of performing the final phase of forward and backward substitution in parallel on a distributed memory multiprocessor. We present a detailed analysis of the parallel complexity and scalability of parallel algorithm described briefly in [5] to obtain a solution to the system of sparse linear equations of the forms $LY = B$ and $UX = Y$, where $L$ is a lower triangular matrix and $U$ is an upper triangular matrix. Here $L$ and $U$ are obtained from the numerical factorization of a sparse coefficient matrix $A$ of the original system $AX = B$ to be solved. If $A$, $L$, and $U$ are $N \times N$ matrices, then $X$, $Y$, and $B$ are $N \times m$ matrices, where $m$ is the number of right-hand side vectors for which the solution to the sparse linear system with $A$ as the coefficient matrix is desired. Our analysis and experiments show that, although not as scalable as the best parallel sparse Cholesky factorization algorithms, parallel sparse triangular solvers can yield reasonable speedups in runtime on hundreds of processors. We also show that for a wide class of problems, the sparse triangular solvers described in this paper are optimal and are asymptotically as scalable as a dense triangular solver.

For a single right-hand side ($m = 1$), our experiments show a 256-processor performance of up to 435 MFLOPS on a Cray T3D, on which the single-processor performance for the same problem is $\approx 8.6$ MFLOPS. With $m = 30$, the maximum single-processor and 256-processor performance observed in our experiments is $\approx 30$ MFLOPS and $\approx 3050$ MFLOPS, respectively. To the best of our knowledge, this is the highest performance and speedup for this problem reported on any massively parallel computer.

In addition to the performance and scalability analysis of parallel sparse triangular solvers, we discuss the redistribution of the triangular factor matrix among the processors between numerical factorization and triangular solution, and its impact on performance. In [4], we describe an optimal data-distribution scheme for Cholesky factorization of sparse matrices. This distribution leaves groups of consecutive columns of $L$ with identical pattern of non-zeros (henceforth called *supernodes*) with a two-dimensional partitioning among groups of processors. However, this distribution is not suitable for the triangular solvers, which are scalable only with a one-dimensional partitioning of the supernodal blocks of $L$. We show that if the supernodes are distributed in a subtree-to-subcube manner [2] then the cost of converting the two-dimensional distribution to a one-dimensional distribution is only a constant times the cost of solving the triangular systems. From our experiments, we observed that this constant is fairly small on the Cray T3D—at most 0.9 for a single right-hand side vector among the test cases used in our experiments. Of course, if more than one systems need to be solved with the same coefficient matrix, then the one-time redistribution cost is amortized.

# 2 Algorithm Description

In this section, we describe parallel algorithms for sparse forward elimination and backward substitution, which have been discussed briefly in [5]. The description in this section assumes a single right-hand side vector; however, the algorithm can easily be generalized to multiple right-hand sides by replacing all vector operations by the corresponding matrix operations.

## 2.1 Forward elimination

The basic approach to forward elimination is very similar to that of multifrontal numerical factorization [12] guided by an elimination tree [13, 8] with the distribution of computation determined by a subtree-to-subcube mapping [2]. A symmetric sparse matrix, its lower triangular Cholesky factor, and the corresponding elimination tree with subtree-to-subcube mapping onto 8 processors is shown in Figure 1. The computation in forward elimination starts with the leaf supernodes of the elimination tree and progresses upwards to terminate at the root supernode. A supernode is a set of columns $i_1, i_2, \ldots, i_t$ of the sparse matrix such that all of them have non-zeros in identical locations and $i_{j+1}$ is the parent of $i_j$ in the elimination tree for $1 \leq j < t$. For example, in Figure 1, nodes 6, 7, and 8 form a supernode. The portion of the lower triangular matrix $L$ corresponding to a supernode is a dense trapezoidal block of width $t$ and maximum height $n$, where $t$ is the number of nodes in the supernode and $n$ is the number of non-zeros in the leftmost column of the supernode. Figure 2 outlines the forward elimination process across three levels of the left half of the elimination tree of Figure 1. One of the blocks of $L$ shown in Figure 2 is the dense trapezoidal supernode consisting of nodes 6, 7, and 8. For this supernode, $n = 4$ and $t = 3$.

As in the case of multifrontal numerical factorization [12], the computation in forward and backward triangular solvers can also be organized in terms of dense matrix operations. In forward elimination (see Figure 2), before the computation starts at a supernode, the

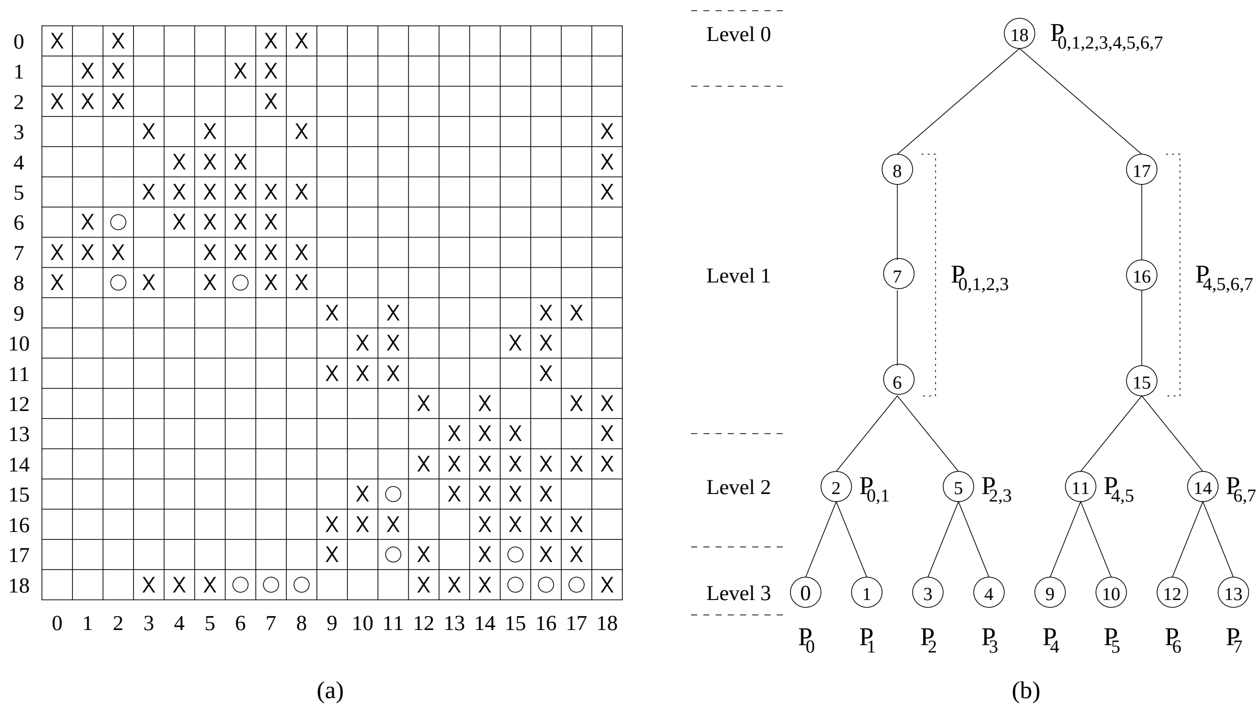



Figure 1: A symmetric sparse matrix and the associated elimination tree with subtree-to-subcube mapping onto 8 processors. The nonzeros in the original matrix are denoted by the symbol "$\times$" and fill-ins are denoted by the symbol "$\circ$".

elements of the right-hand side vector with the same indices as the nodes of the supernode are collected in the first $t$ contiguous locations in a vector of length $n$. The remaining $n-t$ entries of this vector are filled with zeros. The computation corresponding to a trapezoidal supernode, which starts at the leaves, consists of two parts. The first computation step is to solve the dense triangular system at the top of the trapezoid (above the dotted line in Figure 2). The second step is to subtract the product of the vector of length $t$ (above the dotted line) with the $(n-t) \times t$ submatrix of $L$ (below the dotted line) from the vector of length $n-t$ (below the dotted line). After these two computation steps, the entries in the lower part of the vector of length $n-t$ are subtracted from the corresponding (i.e., with the same index in the original matrix) entries of the vector accompanying the parent supernode. The computation at any supernode in the tree can commence after the contributions from all its children have been collected. The algorithm terminates after the computation at the triangular supernode at the root of the elimination tree.

In a parallel implementation on $p$ processors, a supernode at level $l$ (see Figure 1) from the top is distributed among $p/2^l$ processors. The computation at a level greater than or equal to $\log p$ is performed sequentially on a single processor assigned to that subtree. However, the computation steps mentioned above must be performed in parallel on $p/2^l$ processors for a supernode with $0 \leq l < \log p$.

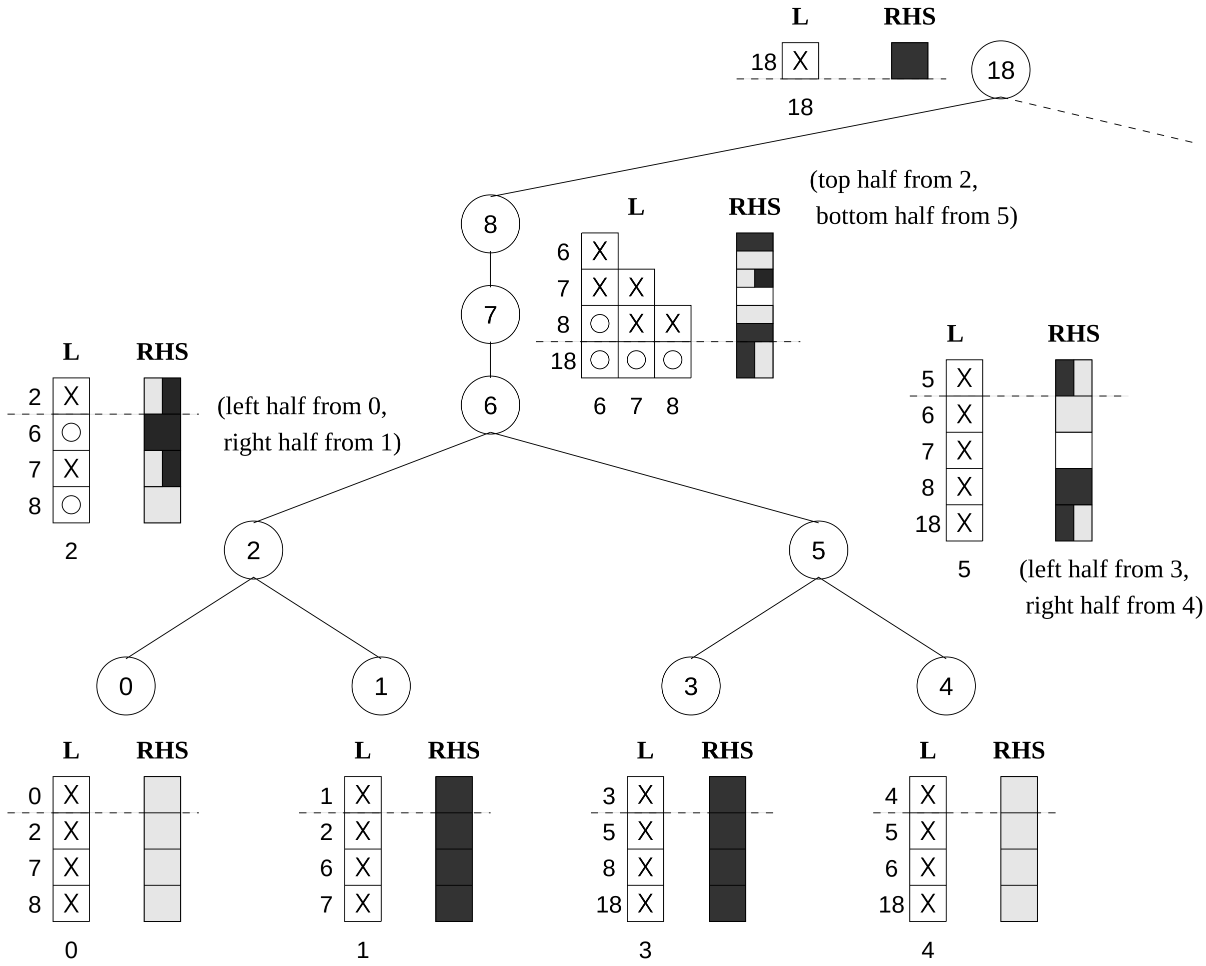
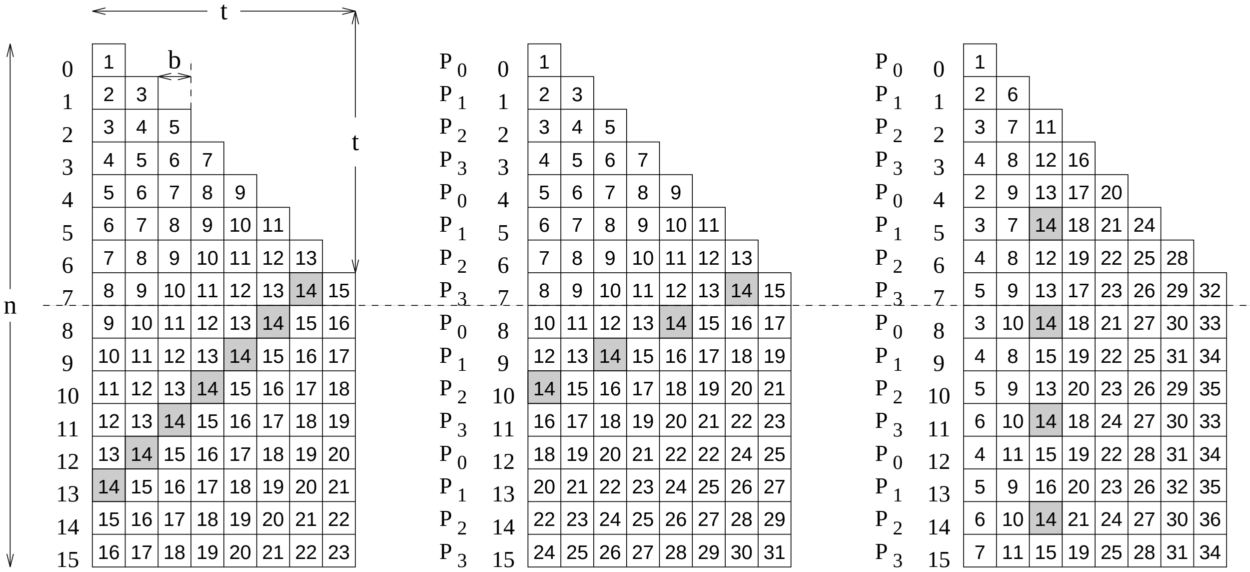



Figure 2: Pictorial representation of forward elimination along three levels of an elimination tree. The color of an RHS box is determined by the color(s) of the box(es) at the next lower level that contribute to its value.

In [6], Heath and Romine describe efficient pipelined or wavefront algorithms for solving dense triangular systems with block-cyclic row-wise and column-wise partitioning of the triangular matrices. We use variations of the same algorithms on the dense trapezoidal supernodes at each of the parallel levels of the elimination tree. the number of processors among which a supernode is partitioned varies with its level in the tree, but the same basic parallel algorithm is used for each supernode. Figure 3(a) shows hypothetical forward elimination on a supernode with an unlimited number of processors on an EREW-PRAM. From this figure, it is clear that, due to data dependencies, at a time only $max(t, n/2)$ processors can remain busy. Since the computation proceeds along a diagonal wave from the upper-left to the lower-right corner of the supernode, at any given time, only one block per row and one element per column is active. From this observation, it can be shown that an efficient parallel algorithm (an algorithm capable of delivering a speedup of $\Theta(p)$ using $p$




(a) Pipelined computation on an EREW-PRAM with unlimited number of processors.

(b) Row-priority pipelined computation with cyclic mapping of rows onto four processors.

(c) Column-priority pipelined computation with cyclic mapping of rows onto four processors.

Figure 3: Progression of computation consistent with data dependencies in parallel pipelined forward elimination in a hypothetical supernode of the lower-triangular factor matrix $L$. The number in each box of $L$ represents the time step in which the coresponding element of $L$ is used in the computation. Communication delays are ignored in this figure and the computation time for each box is assumed to be identical. In parts (b) and (c), the supernode is partitioned among the processors using a cyclic mapping. A block-cyclic mapping can be visualized by regarding each box as a $b \times b$ block (the diagonal boxes will represent triangular blocks).

processors) for forward elimination must employ a one-dimensional row-wise or column-wise partitioning of the supernode so that all processor can be busy at all times (or most of the time). From a practical perspective, we chose a row-wise block-cyclic partitioning because $n \geq t$ and a more uniform partitioning with reasonable block sizes can be obtained if the rows are partitioned. Figures 3(b) and (c) illustrate two variations of the pipelined forward elimination with block-cyclic row-wise partitioning of the supernode. Each box in the figure can be regarded as a $b \times b$ square block of the supernode (note that the diagonal boxes represent lower triangular blocks). In the column-priority algorithm, the computation along a column of the supernode is finished before a new column is started. In the row-priority algorithm, the computation along a row is finished before a new row is started.

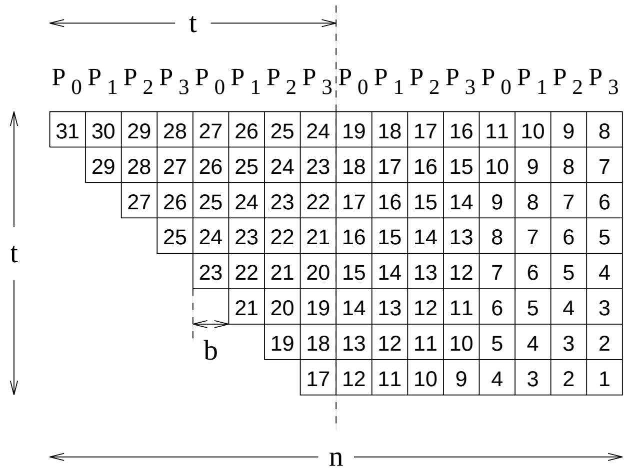


Figure 4: Column-priority pipelined backward substitution on a hypothetical supernode distributed among 4 processors using column-wise cyclic mapping.

## 2.2 Backward substitution

The algorithm for parallel backward substitution is very similar. Since an upper triangular system is being solved, the supernodes are organized as dense trapezoidal matrices of height $t$ and width $n$ ($n \geq t$) and a column-wise block-cyclic partitioning is used at the top $\log p$ levels of the elimination tree. In backward substitution, the computation starts at the root of the elimination tree and progresses down to the leaves. First, the entries from the right-hand side vector with the same indices as the nodes of a supernode are collected in the first $t$ contiguous locations of a vector of length $n$. The remaining $n-t$ entries of this vector are copied from the entries with same indices in the vector accompanying the parent supernode. This step is not performed for the root supernode, which does not have a parent and for which $n = t$. The computation at a supernode consists of two steps and can proceed only after the computation at its parent supernode is finished. The first computation step is to subtract the product of the $t \times (n-t)$ rectangular portion of the supernode with the lower part of the vector of size $n-t$ from the upper part of the vector of size $t$. The second step is to solve the triangular system formed by the $t \times t$ triangle of the trapezoidal supernode and the upper part of the vector of size $t$. Just like forward elimination, these steps are carried out serially for supernodes at levels greater than or equal to $\log p$ in the elimination tree. For the supernodes at levels 0 through $\log p - 1$, the computation is performed using a pipelined parallel algorithm. Figure 4 illustrates the pipelined algorithm on four processors with column-wise cyclic mapping. The algorithm with a block-cyclic mapping can be visualized by regarding each box in Figure 4 as a square block (the blocks along the diagonal of the trapezoid are triangular) of size $b \times b$.

In both forward and backward triangular solvers described in this section, if the system needs to be solved with respect to more than one, say $m$, right-hand sides, then the vectors of length $n$ are replaced by rectangular $n \times m$ matrices. The overall algorithms remain identical except that all vector operations are replaced by the corresponding matrix operations, the size of the matrix being the length of the vector times the number of vectors.

# 3 Analysis

In this section we derive expressions for the communication overheads and analyze the scalability of the sparse supernodal multifrontal triangular solvers described in Section 2. We will present the analysis for the forward elimination phase only; however, the reader can verify that the expressions for the communication overhead are identical for backward substitution.

## 3.1 Communication overheads

It is difficult to derive analytical expressions for general sparse matrices because the location and amount of fill-in, and hence, the distribution and the number if non-zeros in $L$, is a function of the the number and position of nonzeros in the original matrix. Therefore, we will focus on the problems in which the original matrix is the adjacency matrix of a two- or three-dimensional neighborhood graph [14]. These classes of matrices include the coefficient matrices generated in all two- and three-dimensional finite element and finite difference problems. We also assume as that a nested-dissection based fill-reducing ordering is used, which results in an almost balanced elimination tree. The subtree-to-subcube assignment of the elimination tree to the processors relies heavily on a balanced tree. Although there are bound to be overheads due to unequal distribution of work, it is not possible to model such overheads analytically because the extent of such overheads is data-dependent. From our experience with actual implementations of parallel triangular solvers as well as parallel factorization codes [4], we have observed that such overheads are usually not excessive. Moreover, the overhead due to load imbalance in most practical cases tends to saturate at 32 to 64 processors for most problems and does not continue to increase as the number of processors are increased. In the remainder of this section, we will concentrate on overheads due to inter-processor communication only.

Consider the column-priority pipelined algorithm for forward elimination shown in Figure 3(c). Let $b$ be the block size in the block-cyclic mapping. A piece of the vector of size $b$ is transferred from a processor to its neighbor in each step of the algorithm until the computation moves below the upper triangular part of the trapezoidal supernode. If a supernode is distributed among $q$ processors, then during the entire computation at a supernode, $q + t/b - 1$ such communication steps are performed; $q - 1$ steps are required for the computation to reach the last processor in the pipeline and $t/b$ steps to pass the entire data (of length $t$) through the pipeline. Thus, the total communication time is proportional to $b(q-1) + t$, which is $O(q) + O(t)$, assuming that $b$ is a constant.

Besides the communication involved in the pipelined processing over a supernode, there is some more communication involved in collecting the contributions of the vectors associated with the children of a supernode into the vector associated with the parent supernode. If the two child supernodes are each distributed among $q$ processors, then this communication is equivalent to an all-to-all personalized communication [8] among $2q$ processors with a data size of roughly $t/q$ on each processor. This communication can be accomplished in time proportional to $t/q$, which is asymptotically smaller than the $O(q) + O(t)$ time spent during the pipelined computation phase at the child supernodes. Therefore, in the remainder of this section, we will ignore the communication required to transfer the contributions of the vector across the supernodes at different levels of the elimination tree.

So far we have established that a time proportional to $b(q-1)+t$ (or roughly, $bq+t$) is spent while processing an $n \times t$ trapezoidal supernode on $q$ processors with a block-cyclic mapping that uses blocks of size $b$. We can now derive an expression for the overall communication time for the entire parallel forward elimination process by substituting for $q$ and $t$ in the expression $bq + t$ for a supernode at level $l$ and summing up the resulting expression over all levels.

Let us first consider a sparse linear system of $N$ equations resulting from a two-dimensional finite element problem being solved on $p$ processors. As a result of using the subtree-to-subcube mapping, $q$ at a level $l$ is $p/2^l$. If a nested-dissection based ordering scheme is used to number the nodes of the graph corresponding to the coefficient matrix, then the number of nodes $t$ in a supernode at level $l$ is $\alpha\sqrt{N/2^l}$, where $\alpha$ is a small constant [11, 10, 1, 4]. Thus the overall communication time is proportional to $\Sigma_{l=0}^{\log p-1}(bp/2^l) + \Sigma_{l=0}^{\log p-1}(\alpha\sqrt{N/2^l})$, which is $O(p) + O(\sqrt{N})$.

The overall computation is proportional to the number of non-zeros in $L$, which is $O(N \log N)$ for an $N \times N$ sparse coefficient matrix resulting from a two-dimensional finite element problem [1] with a nested-dissection based ordering. Assuming that the computation is divided uniformly among the processors, each processor spends $O((N \log N)/p)$ time in computation. Hence, the parallel runtime for forward elimination algorithm described in Section 2 is as follows[1]:

$$T_P = O(\frac{N \log N}{p}) + O(\sqrt{N}) + O(p). \tag{1}$$

If the underlying graph corresponding to the coefficient matrix is a three-dimensional neighborhood graph [14] (as is the case in three-dimensional finite element and finite difference problems), then the value of $t$ at level $l$ is roughly $\alpha(N/2^l)^{2/3}$, where $\alpha$ is a small constant [1, 4]. The value of $q$ at level $l$ is $p/2^l$. Thus, the total communication time is

[1]Depending on the way the pipelining is implemented for processing a supernode, there may be $O(t)$ and/or $O(q)$ steps of pipeline delay (one step performs $2b^2$ operations) at each supernode. However, the aggregate of such terms does not exceed the asymptotic communication complexity of $O(p) + O(\sqrt{N})$

proportional to $\Sigma_{l=0}^{\log p-1}(bp/2^l) + \Sigma_{l=0}^{\log p-1}(\alpha(N/2^l)^{2/3}$, which is $O(p) + O(N^{2/3})$. Assuming that the overall computation of $O(N^{4/3})$ [1] is uniformly distributed among the processors, the parallel runtime is given by the following equation:

$$T_P = O(\frac{N^{4/3}}{p}) + O(N^{2/3}) + O(p). \tag{2}$$

If more than one (say $m$) right-hand side vectors are present in the system, then each term in Equations 1 and 2 is multiplied with $m$.

## 3.2 Scalability analysis

The scalability of a parallel algorithm on a parallel architecture refers to the capacity of the algorithm-architecture combination to effectively utilize an increasing number of processors. In this section we use the isoefficiency metric [8, 9, 3] to characterize the scalability of the algorithm described in Section 2. The isoefficiency function relates the problem size to the number of processors necessary to maintain a fixed efficiency or to deliver speedups increasing proportionally with increasing number of processors.

Let $W$ be the size of a problem in terms of the total number of basic operations required to solve a problem on a serial computer. For example, $W = O(N^2)$ for multiplying a dense $N \times N$ matrix with an $N$-vector. The serial run time of a problem of size $W$ is given by $T_S = t_c W$, where $t_c$ is the time to perform a single basic computation step. If $T_P$ is the parallel run time of the same problem on $p$ processors, then we define an overhead function $T_o$ as $pT_P - T_S$. Both $T_P$ and $T_o$ are functions of $W$ and $p$, and we often write them as $T_P(W,p)$ and $T_o(W,p)$, respectively. The efficiency of a parallel system with $p$ processors is given by $E = T_S/(T_S + T_o(W,p))$. If a parallel system is used to solve a problem instance of a fixed size $W$, then the efficiency decreases as $p$ increases. This is because the total overhead $T_o(W,p)$ increases with $p$. For many parallel systems, for a fixed $p$, if the problem size $W$ is increased, then the efficiency increases because for a given $p$, $T_o(W,p)$ grows slower than $O(W)$. For these parallel systems, the efficiency can be maintained at a desired value (between 0 and 1) with increasing $p$, provided $W$ is also increased. We call such systems *scalable* parallel systems. Note that for different parallel algorithms, $W$ may have to increase at different rates with respect to $p$ in order to maintain a fixed efficiency. As the number of processors are increased, the smaller the growth rate of problem size required to maintain a fixed efficiency, the more scalable the parallel system is.

Given that $E = 1/(1+T_o(W,p)/(t_c W))$, in order to maintain a fixed efficiency, $W$ should be proportional to $T_o(W,p)$. In other words, the following relation must be satisfied in order to maintain a fixed efficiency: $W = \frac{e}{t_c}T_o(W,p)$, where $e = E/(1-E)$ is a constant depending on the efficiency to be maintained. Therefore, $W \propto T_o$ is the central relation that is used to determine the isoefficiency function of a parallel algorithm-architecture combination. This is accomplished by abstracting $W$ as a function of $p$ through algebraic manipulations on the relation $W \propto T_o$. If the problem size needs to grow as fast as $f_E(p)$ to maintain an efficiency $E$, then $f_E(p)$ is defined as the isoefficiency function of the parallel algorithm-architecture combination for efficiency $E$.

Recall that for a triangular system resulting from the factorization of an $N \times N$ sparse matrix corresponding to a two-dimensional neighborhood graph,

$$W = O(N \log N). \tag{3}$$

If we assume that only the last two terms in Equation 1 contribute to the overhead, then from the relation $T_o = pT_P - T_S$, it is easy to see that

$$T_o = O(p^2) + O(p\sqrt{N}). \tag{4}$$

Balancing $W$ against the first term in the expression for $T_o$ yields the following (see Appendix A for details):

$$W \propto p^2, \tag{5}$$

and balancing it against the second term in the expression for $T_o$ yields

$$W \propto \frac{p^2}{\log p}. \tag{6}$$

Since $p^2$ is the dominant term in the two isoefficiency expressions, the overall rate at which the problem size must increase with the number of processors to maintain a fixed efficiency is $O(p^2)$, as given by Equation 5.

For a triangular system resulting from the factorization of an $N \times N$ sparse matrix corresponding to a three-dimensional neighborhood graph,

$$W = O(N^{4/3}) \tag{7}$$

and

$$T_o = O(p^2) + O(pN^{2/3}). \tag{8}$$

Balancing $W$ against each term in the expression for $T_o$ yields the following isoefficiency function (see Appendix A for details):

$$W \propto p^2. \tag{9}$$

In this section, we have shown that the isoefficiency function for solving sparse triangular systems resulting from the factorization of a wide class of sparse matrices is $O(p^2)$. In [4], we described parallel algorithms for sparse Cholesky factorization of the same class of matrices with an isoefficiency function of $O(p^{1.5})$, which is better than the $O(p^2)$ isoefficiency function of the corresponding triangular solver. However, the amount of computation involved in numerical factorization is much higher than that in a triangular solver. Therefore, as we experimentally demonstrate in Section 5, despite being less efficient than parallel numerical factorization, triangular solvers can still be speeded up enough n parallel so as to claim only a fraction of the factorization time on the same number of processors.

### 3.3 Comparison with the scalability of dense triangular solvers

The communication time for solving a dense $N \times N$ triangular system using a row-wise block-cyclic mapping onto $p$ processors with block size $b$ is proportional to $b(p-1)+N$, which is $O(p)+O(N)$. The problem size $W$ is $O(N^2)$ and the total communication overhead $T_o$ is $O(p^2)+O(Np)$ (note that the total communication overhead or the overhead function is the product of $p$ and the communication time). It is easy to see $W$ must grow in proportion to $p^2$ in order to satisfy the relation $W \propto T_o$ for maintaining a constant efficiency. Thus, the isoefficiency function of a parallel dense triangular solver is $O(p^2)$, indicating that the parallel algorithms described in Section 2 for sparse forward and backward triangular solvers are asymptotically as scalable as their dense counterparts. From this observation, it can be argued that the sparse algorithms, at least in the case of matrices associated with three-dimensional neighborhood graphs are optimal. The topmost supernode in such a matrix is an $N^{2/3} \times N^{2/3}$ dense triangle. Solving a triangular system corresponding to this supernode involves asymptotically a computation of the same complexity as solving the entire sparse triangular system. Thus, the overall scalability cannot be better than that of solving the topmost $N^{2/3} \times N^{2/3}$ dense triangular system in parallel, which is $O(p^2)$.

# 4 Data Distribution for Efficient Triangular Solution

In Section 2 and in [8], we discuss that in order to implement the steps of dense triangular solution efficiently, the matrix must be partitioned among the processors along the rows or along the columns. However, as we have shown in [4], the dense supernodes must be partitioned along both dimensions for the numerical factorization phase to be efficient. The table in Figure 5 shows the communication overheads and the isoefficiency functions for parallel dense and sparse factorization and triangular solution using one- and two-dimensional partitioning schemes. The most efficient scheme in each category is denoted by a shaded box in the table. From Figure 5, it is clear that, for both the factorization and triangular solution to be efficient together, each supernode must be redistributed among the processors that share it. This redistribution must convert the original two-dimensional block-cyclic partitioning into a one-dimensional block-cyclic partitioning. In this section we show that the time spent in this redistribution is not asymptotically higher than the parallel run time of the triangular solvers.

Consider an $n \times t$ dense supernode mapped onto a $\sqrt{q} \times \sqrt{q}$ logical grid of processors using a two dimensional partitioning. As shown in Figure 6, the redistribution is equivalent to a transposition of each $(n/\sqrt{q}) \times t$ rectangular block of the supernode among the $\sqrt{q}$ processor on which it is horizontally partitioned. This is an all-to-all personalized communication operation [8] among $\sqrt{q}$ processors with each processor holding $nt/q$ words of data. Although Figure 6 illustrates redistribution with a plain block partitioning, both the procedure and the

| Matrix Type | Partitioning | Factorization, no Pivoting | | Forward/Backward Solution | |
|---|---|---|---|---|---|
| | | Communication Overhead | Isoefficiency Function | Communication Overhead | Isoefficiency Function |
| Dense | One-dimensional | $O(N^2 p)$ | $O(p^3)$ | $O(p^2) + O(Np)$ | $O(p^2)$ |
| | Two-dimensional | $O(N^2 p^{0.5})$ | $O(p^{1.5})$ | $O(N^2 p)$ | Unscalable |
| Sparse (resulting from 2-D graphs) | One-dimensional with subtree-subcube | $O(Np)$ | $O(p^3)$ | $O(p^2) + O(N^{0.5} p)$ | $O(p^2)$ |
| | Two-dimensional with subtree-subcube | $O(Np^{0.5})$ | $O(p^{1.5})$ | $O(Np \log(N))$ | Unscalable |
| Sparse (resulting from 3-D graphs) | One-dimensional with subtree-subcube | $O(N^{1.3} p)$ | $O(p^3)$ | $O(p^2) + O(N^{0.7} p)$ | $O(p^2)$ |
| | Two-dimensional with | $O(N^{1.3} p^{0.5})$ | $O(p^{1.5})$ | $O(N^{1.3} p)$ | Unscalable |

Figure 5: A table of communication overheads and isoefficiency functions for sparse factorization and triangular solution with different partitioning schemes.

cost of redistribution are the same with block-cyclic partitioning as well. The communication time for this all-to-all personalized operation is $O(nt/q)$ [8]. Note that for solving a triangular system with a single right-hand side, each processor performs $O(nt/q)$ computation while processing an $n \times t$ supernode on $q$ processors. Thus, the total overhead for redistribution is of the same order as the time spent by the parallel triangular solver while working on the top $\log p$ levels of the elimination tree, which is less than the total parallel runtime. The actual ratio of the redistribution time and the parallel triangular solution time will depend on the relative communication and computation speeds of the parallel computer being used. In Section 5, we show that on a Cray T3D, the redistribution time is at most 0.9 times (0.6 times on an average) the parallel triangular solution time with a single right-hand side vector. If more than one right-hand side vectors are used, then the cost of redistribution can be amortized further because the redistribution needs to be done only once.

# 5 Experimental Results

We implemented the algorithms described in Section 2 and integrated them with our sparse Cholesky factorization algorithms described in [4]. Figures 7 and 8 show the performance of the parallel triangular solvers on a Cray T3D.

In the table in Figure 7, we show the time in seconds and the performance in MFLOPS on a selected number of processors for five test matrices with the number of right-hand side

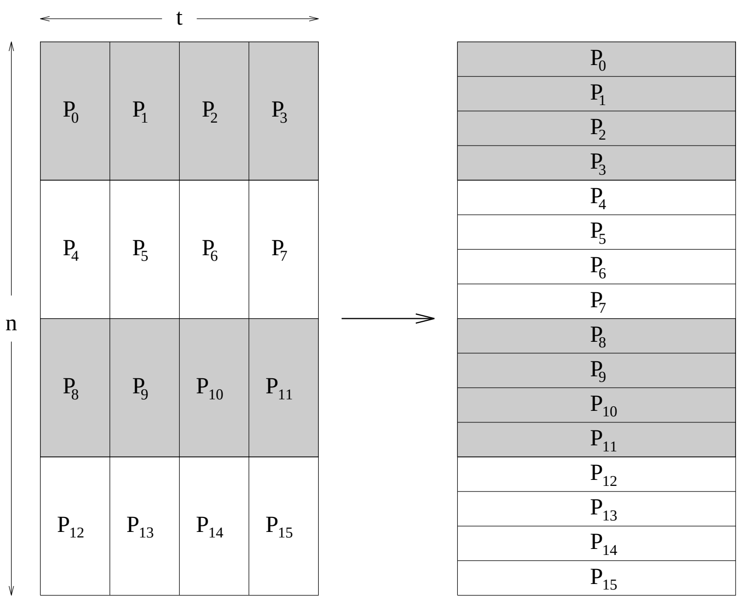



Figure 6: Converting the two-dimensional partitioning of a supernode into one-dimensional partitioning.

vectors varying from 1 to 30. To facilitate a comparison of the times for various phases of the solution processes, the table also contains the factorization run time and MFLOPS, as well as the time to redistribute the factor matrix to convert the supernodes from a two-dimensional to a one-dimensional partitioning among the processors. As shown in Figure 7, for a single right-hand side vector, the highest performance achieved on a 256-processor Cray T3D is approximately 435 MFLOPS, which increases to over 3 GFLOPS if a solution with 30 right-hand side vectors is obtained. Comparing with the single-processor performance for BCSSTK15, this represents roughly 50- and 100-fold enhancement in performance on 256 processors for 1 and 30 right-hand side vectors, respectively. There are two other important observations to be made from the table in Figure 7. First, despite a highly scalable implementation of sparse Cholesky factorization, parallelization of the relatively less scalable triangular solvers can speed them enough so that their runtime is still a small fraction of the factorization time. Second, although efficient implementations of factorization and triangular solvers use different data partitioning schemes, the redistribution of the data, on an average, takes much less time than the triangular solvers for a single right-hand side vector on the T3D.

Figure 8 shows the plots of MFLOPS versus number of processors of the Cray T3D for triangular solutions with different number of right-hand side vectors. The curves for these four test matrices show that both the overall performance and the speedups are much higher if a block of right-hand side vectors is available for solution. The use of multiple right-hand side vectors enhances the single processor performance due to effective use of BLAS-3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **BCSSTK15:** N = 3948; Factorization Opcount = 85.5 Million; Nonzeros in factor = 0.49 Million | | | | | | | | |
| | Factorization time = 2.46 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 1** | Factorization MFLOPS = 34.8 | FBsolve time (sec.) | .228 | .284 | .452 | .740 | 1.33 | 1.92 |
| | Time to redistribute L = 0.0 sec. | FBsolve MFOLPS | 8.6 | 13.7 | 21.5 | 26.5 | 29.4 | 30.0 |
| | Factorization time = .107 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 64** | Factorization MFLOPS = 800 | FBsolve time (sec.) | .024 | .027 | .034 | .048 | .074 | .100 |
| | Time to redistribute L = .009 sec. | FBsolve MFOLPS | 81.5 | 145 | 285 | 405 | 527 | 583 |
| **BCSSTK31:** N = 35588; Factorization Opcount = 2791 Million; Nonzeros in factor = 6.64 Million | | | | | | | | |
| | Factorization time = 5.59 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 16** | Factorization MFLOPS = 499 | FBsolve time (sec.) | .227 | .274 | .398 | .614 | 1.05 | 1.51 |
| | Time to redistribute L = .071 sec. | FBsolve MFOLPS | 115 | 194 | 330 | 427 | 498 | 523 |
| | Factorization time = .721 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 256** | Factorization MFLOPS = 3871 | FBsolve time (sec.) | .073 | .082 | .107 | .152 | .242 | .334 |
| | Time to redistribute L = .035 sec. | FBsolve MFOLPS | 363 | 646 | 1240 | 1738 | 2199 | 2385 |
| **HSCT21954:** N = 21954; Factorization Opcount = 2822 Million; Nonzeros in factor = 5.84 Million | | | | | | | | |
| | Factorization time = 2.48 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 32** | Factorization MFLOPS = 1138 | FBsolve time (sec.) | .113 | .133 | .189 | .284 | .472 | .672 |
| | Time to redistribute L = .10 sec. | FBsolve MFOLPS | 203 | 347 | 609 | 809 | 973 | 1025 |
| | Factorization time = .619 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 256** | Factorization MFLOPS = 4560 | FBsolve time (sec.) | .091 | .099 | .122 | .161 | .234 | .312 |
| | Time to redistribute L = .067 sec. | FBsolve MFOLPS | 255 | 471 | 953 | 1452 | 1991 | 2244 |
| **CUBE35:** N = 42875; Factorization Opcount = 7912 Million; Nonzeros in factor = 9.95 Million | | | | | | | | |
| | Factorization time = 7.528 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 32** | Factorization MFLOPS = 1051 | FBsolve time (sec.) | .245 | .296 | .436 | .681 | 1.21 | 1.72 |
| | Time to redistribute L = .13 sec. | FBsolve MFOLPS | 162 | 269 | 456 | 583 | 660 | 693 |
| | Factorization time = 1.43 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 256** | Factorization MFLOPS = 5527 | FBsolve time (sec.) | .108 | .120 | .154 | .216 | .340 | .468 |
| | Time to redistribute L = .08 sec. | FBsolve MFOLPS | 369 | 665 | 1289 | 1838 | 2345 | 2548 |
| **COPTER2:** N = 55476; Factorization Opcount = 8905 Million; Nonzeros in factor = 12.77 Million | | | | | | | | |
| | Factorization time = 5.764 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 64** | Factorization MFLOPS = 1545 | FBsolve time (sec.) | .186 | .220 | .320 | .492 | .832 | 1.10 |
| | Time to redistribute L = .11 sec. | FBsolve MFOLPS | 274 | 463 | 795 | 1036 | 1226 | 1277 |
| | Factorization time = 1.846 sec. | NRHS | 1 | 2 | 5 | 10 | 20 | 30 |
| **p = 256** | Factorization MFLOPS = 4825 | FBsolve time (sec.) | .117 | .130 | .167 | .232 | .364 | .500 |
| | Time to redistribute L = .07 sec. | FBsolve MFOLPS | 434 | 785 | 1526 | 2195 | 2805 | 3053 |

Figure 7: A partial table of experimental results for sparse forward and backward substitution on a Cray T3D. In the above table, "NRHS" denotes the number of right-hand side vectors, "FBsolve time" denotes the total time spent in both the forward and the backward solvers, and "FBsolve MFLOPS" denotes the average performance of the solvers in million floating point operations per second.

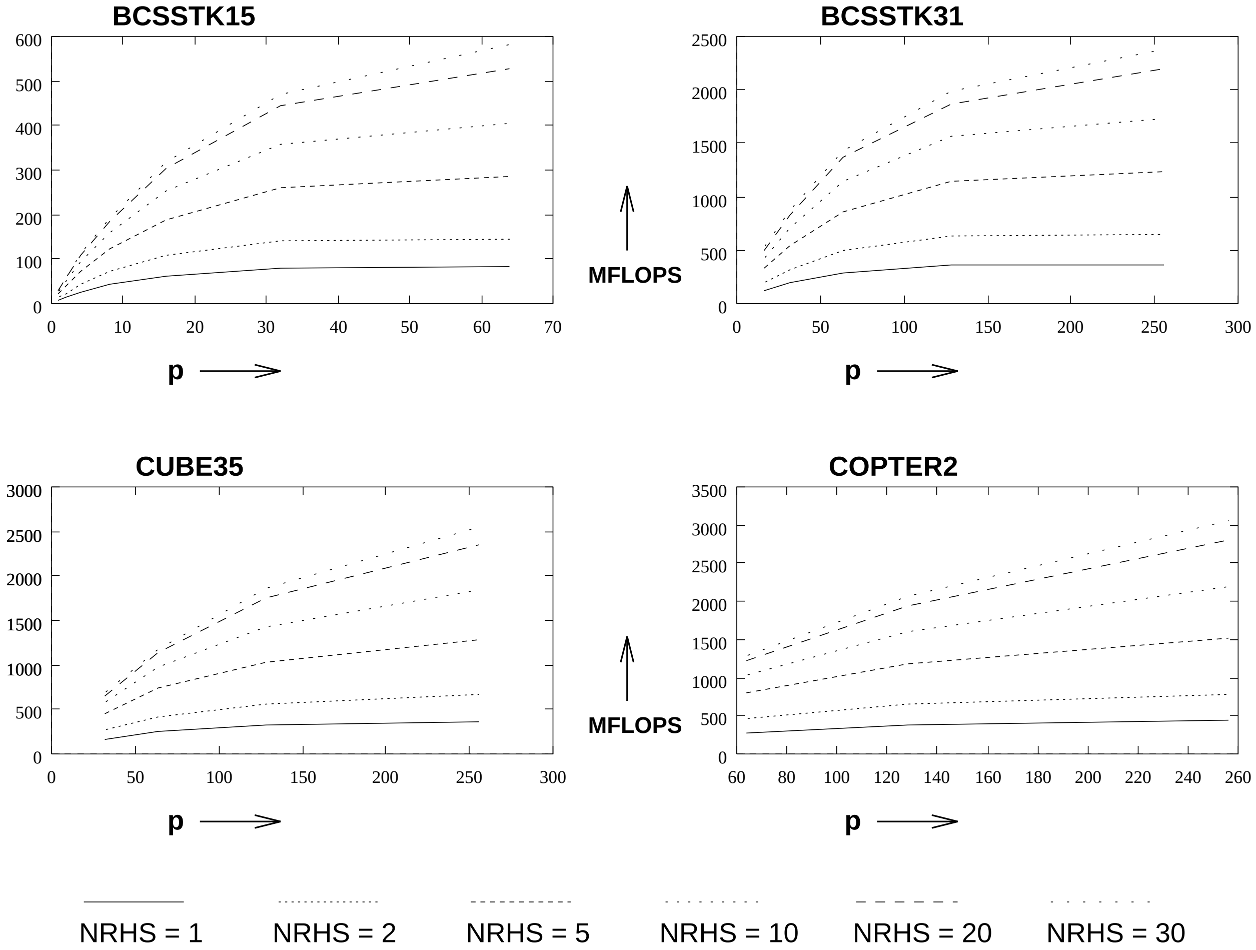



Figure 8: Performance versus number of processors for parallel sparse triangular solutions with different number of right-hand side vectors.

routines. It also improves speedups because the cost of certain index computations required in the parallel implementation can be amortized over all the right-hand side vectors.

# 6 Concluding Remarks

Despite more inherent parallelism that dense linear systems, it has been a challenge to develop scalable parallel direct solvers for sparse linear systems. The process of obtaining a direct solution to a sparse system of linear equations usually consists of four phases: reordering, symbolic factorization, numerical factorization, and forward elimination and backward substitution. A scalable parallel solver for sparse linear systems must implement all these phases effectively in parallel. In [4], we introduced a highly scalable parallel algorithm for sparse Cholesky factorization, which is the most time consuming phase of solving a sparse linear system with s symmetric positive definite (SPD) matrix of coefficients. In [7], Karypis and Kumar present an efficient parallel algorithm for a nested-dissection based fill-reducing ordering for such sparse matrices. The results of this paper bring us another step closer to a complete scalable direct solver for sparse SPD systems. In this paper, we have shown that although less scalable than numerical factorization, the forward and backward substitution steps can obtain sufficient speedup on hundreds of processors so that numerical factorization still dominates the overall time taken to solve the system in parallel. In addition, we show that, although efficient implementations of factorization and triangular solvers use different data partitioning schemes, the time spent in redistributing the data to change the partitioning schemes is not a bottleneck when compared to the time spent in factorization and triangular solutions.

# References

[1] A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[2] A. George, J. W.-H. Liu, and E. G.-Y. Ng. Communication reduction in parallel sparse Cholesky factorization on a hypercube. In M. T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 576–586. SIAM, Philadelphia, PA, 1987.

[3] Ananth Grama, Anshul Gupta, and Vipin Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel and Distributed Technology*, 1(3):12–21, August, 1993. Also available as Technical Report TR 93-24, Department of Computer Science, University of Minnesota, Minneapolis, MN.

[4] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. Technical Report 94-63, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994. Submitted for publication in *IEEE Transactions on Parallel and Distributed Computing*. Postscript file available in *users/kumar* at anonymous FTP site *ftp.cs.umn.edu*.

[5] M. T. Heath and Padma Raghavan. Distributed solution of sparse linear systems. Technical Report 93-1793, Department of Computer Science, University of Illinois, Urbana, IL, 1993.

[6] M. T. Heath and C. H. Romine. Parallel solution of triangular systems on distributed-memory multiprocessors. *SIAM Journal on Scientific and Statistical Computing*, 9(3):558–588, 1988.

[7] George Karypis and Vipin Kumar. Parallel graph partitioning and matrix reordering. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995. Submitted to *Supercomputing '95*.

[8] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, CA, 1994.

[9] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, 22(3):379–391, 1994. Also available as Technical Report TR 91-18, Department of Computer Science Department, University of Minnesota, Minneapolis, MN.

[10] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16:346–358, 1979.

[11] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[12] J. W.-H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. Technical Report CS-90-04, York University, Ontario, Canada, 1990. Also appears in *SIAM Review*, 34:82–109, 1992.

[13] J. W.-H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

[14] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of 31st Annual Symposium on Foundations of Computer Science*, pages 538–547, 1991.

# Appendix A

## Derivation of the isoefficiency function for parallel triangular solvers

Consider solving the triangular systems resulting from the factorization of a sparse matrix associated with a two-dimensional neighborhood graph. From Equation 3, $W = O(N \log N$ and from Equation 4, $T_o = O(p^2) + O(p\sqrt{N})$. In order to maintain a fixed efficiency, $W \propto T_o$s; i.e., the following two conditions must be satisfied:

$$W \propto p^2 \tag{10}$$

and

$$W \propto p\sqrt{N}. \tag{11}$$

Equation 10 clearly suggests an isoefficiency function of $O(p^2)$. From Equation 11,

$$N \log N \propto p\sqrt{N},$$

$$\sqrt{N} \log N \propto p, \tag{12}$$

$$\log N + \log\log N \propto \log p. \tag{13}$$

Discarding the lower order term $\log\log N$ from Equation 13, we get

$$\log N \propto \log p. \tag{14}$$

From Equations 12 and 14,

$$\sqrt{N} \propto \frac{p}{\log p},$$

$$N \propto (\frac{p}{\log p})^2,$$

$$N \log N \propto \frac{p^2}{\log p},$$

$$W \propto \frac{p^2}{\log p} \tag{15}$$

Thus, we have derived Equation 6. Similarly, we can derive the isoefficiency function for the triangular systems resulting from the factorization os sparse matrices associated with three-dimensional neighborhood graphs. Recall from Section 3.2 that for such systems, $W = O(N^{4/3})$ and $T_o = O(p^2) + O(pN^{2/3})$. If $W \propto T_o$, then $W \propto p^2$ (the first term of $T_o$) and $W \propto pN^{2/3}$. The second condition yields

$$N^{4/3} \propto pN^{2/3},$$

$$N^{2/3} \propto p,$$

$$N^{4/3} \propto p^2,$$

$$W \propto p^2, \tag{16}$$

which is same as Equation 9.