Design and Implementation of a Scalable Parallel Direct Solver for Sparse Symmetric Positive Definite Systems : Preliminary Results *

Anshul Gupta† Fred Gustavson† Mahesh Joshi‡ George Karypis‡
Vipin Kumar‡

Abstract

Solving large sparse systems of linear equations is at the core of many problems in engineering and scientific computing. It has long been a challenge to develop parallel formulations of sparse direct solvers due to several different complex steps involved in the process. In this paper, we describe one of the first efficient, practical, and robust parallel solvers for sparse symmetric positive definite linear systems that we have developed and discuss the algorithmic and implementation issues involved in its development.

1 Introduction

Solving large sparse systems of linear equations is at the heart of many engineering and scientific computing applications. There are two methods to solve these systems - direct and iterative. Direct methods are preferred for many applications because of various properties of the method and the nature of the application. A wide class of sparse linear systems arising in practice have a symmetric positive definite (SPD) coefficient matrix. The problem is to compute the solution to the system $Ax = b$, where A is a sparse and SPD matrix. Such a system is commonly solved using Cholesky factorization. A direct method of solution consists of four consecutive phases viz. ordering, symbolic factorization, numerical factorization and solution of triangular systems. During the ordering phase, a permutation matrix P is computed so that the matrix PAP^T will incur a minimal fill during the factorization phase. During the symbolic factorization phase, the non-zero structure of the triangular Cholesky factor L is determined. The symbolic factorization phase exists in order to increase the performance of the numerical factorization phase. The necessary operations to compute the values in L that satisfy $PAP^T = LL^T$, are performed during the phase of numerical factorization. Finally, the solution to $Ax = b$ is computed by solving two triangular systems viz. $Ly = b'$ followed by $L^T x' = y$, where $b' = Pb$ and $x' = Px$. Solving the former system is called forward elimination and the latter process of solution is called backward substitution. The final solution, x, is obtained using $x = P^T x'$.

In this paper, we describe one of the first scalable and high performance parallel direct solvers for sparse linear systems involving SPD matrices. Our parallel direct solver uses the MPI library for communication, making it portable to a wide range of parallel computers. Furthermore, high computational rates are achieved using serial BLAS routines to perform the computations at each processor. At the heart of this solver is a highly parallel algorithm based on multifrontal Cholesky

*This work was supported by NSF CCR-9423082, by Army Research Office contract DA/DAAH04-95-1-0538, by Army High Performance Computing Research Center cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, by the IBM Partnership Award, and by the IBM SUR equipment grant. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: http://www.cs.umn.edu/~kumar.

†IBM T.J.Watson Research Center, Yorktown Heights, NY 10598

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

factorization we recently developed [1]. This algorithm is able to achieve high computational rates (of over 20 GFlops on a 1024 processor Cray T3D) and it successfully parallelizes computationally the most expensive phase of the sparse solver. Fill reducing ordering is obtained using a parallel formulation of the multilevel nested dissection algorithm [2] that has been found to be effective in producing orderings that are suited for parallel factorization. For symbolic factorization and solution of triangular systems, we have developed parallel algorithms that utilize the same data distribution as used by the numerical factorization algorithm. Both algorithms are able to effectively parallelize the corresponding phases. In particular, our parallel forward elimination and parallel backward substitution algorithms are able to achieve a high degree of concurrency and high computational rates. In this paper we briefly describe the parallel algorithms for numerical factorization, symbolic factorization and solution of triangular systems, and present some preliminary experimental results on an IBM SP2. Details about the parallel algorithm for the ordering phase can be found in [2].

2 Parallel Numerical Factorization

For the numerical factorization phase we use a highly scalable algorithm that we developed recently [1] that is based on the multifrontal algorithm [5].

Given a sparse matrix and the associated elimination tree, the multifrontal algorithm can be recursively formulated as follows. Consider an $N \times N$ matrix A. The algorithm performs a postorder traversal of the elimination tree associated with A. There is a frontal matrix F^k and an update matrix U^k associated with any node k. The row and column indices of F^k correspond to the indices of row and column k of L, the lower triangular Cholesky factor, in increasing order. In the beginning, F^k is initialized to an $(s+1) \times (s+1)$ matrix, where $s+1$ is the number of non-zeros in the lower triangular part of column k of A. The first row and column of this initial F^k is simply the upper triangular part of row k and the lower triangular part of column k of A. The remainder of F^k is initialized to all zeros.

After the algorithm has traversed all the subtrees rooted at a node k, it ends up with a $(t+1) \times (t+1)$ frontal matrix F^k, where t is the number of non-zeros in the strictly lower triangular part of column k in L. The row and column indices of the final assembled F^k correspond to $t+1$ (possibly) noncontiguous indices of row and column k of L in increasing order. If k is a leaf in the elimination tree of A, then the final F^k is the same as the initial F^k. Otherwise, the final F^k for eliminating node k is obtained by merging the initial F^k with the update matrices obtained from all the subtrees rooted at k via an extend-add operation. The extend-add is an associative and commutative operator on two update matrices such the index set of the result is the union of the index sets of the original update matrices. Each entry in the original update matrices is mapped onto some location in the accumulated matrix. If entries from both matrices overlap on a location, they are added. Empty entries are assigned a value of zero. After F^k has been assembled, a single step of the standard dense Cholesky factorization is performed with node k as the pivot. At the end of the elimination step, the column with index k is removed from F^k and forms the column k of L. The remaining $t \times t$ matrix is called the update matrix U^k and is passed on to the parent of k in the elimination tree.

In this paper, a collection of consecutive nodes in the elimination tree, each with only one child, is called a *supernode*. The nodes in a supernode are collapsed together to form the *supernodal elimination tree*. The serial multifrontal algorithm can be extended to operate on this supernodal tree by extending the single node operations performed while forming and factoring the frontal matrix. The frontal matrix corresponding to a supernode with l nodes is obtained by merging the frontal matrices of the individual nodes, and the first l columns of this frontal matrix are factored during the factorization of this supernode.

In our parallel formulation of the multifrontal algorithm, we assume that the supernodal tree is binary in the top $\log p$ levels [1]. The portions of this binary supernodal tree are assigned to processors using a subtree-to-subcube strategy illustrated in Figure 1(b), where eight processors are used to factor the example matrix of Figure 1(a). The subcubes of processors working on various subtrees

[1] The separator tree obtained from the recursive nested dissection parallel ordering algorithm used in our solver yields such a binary tree.

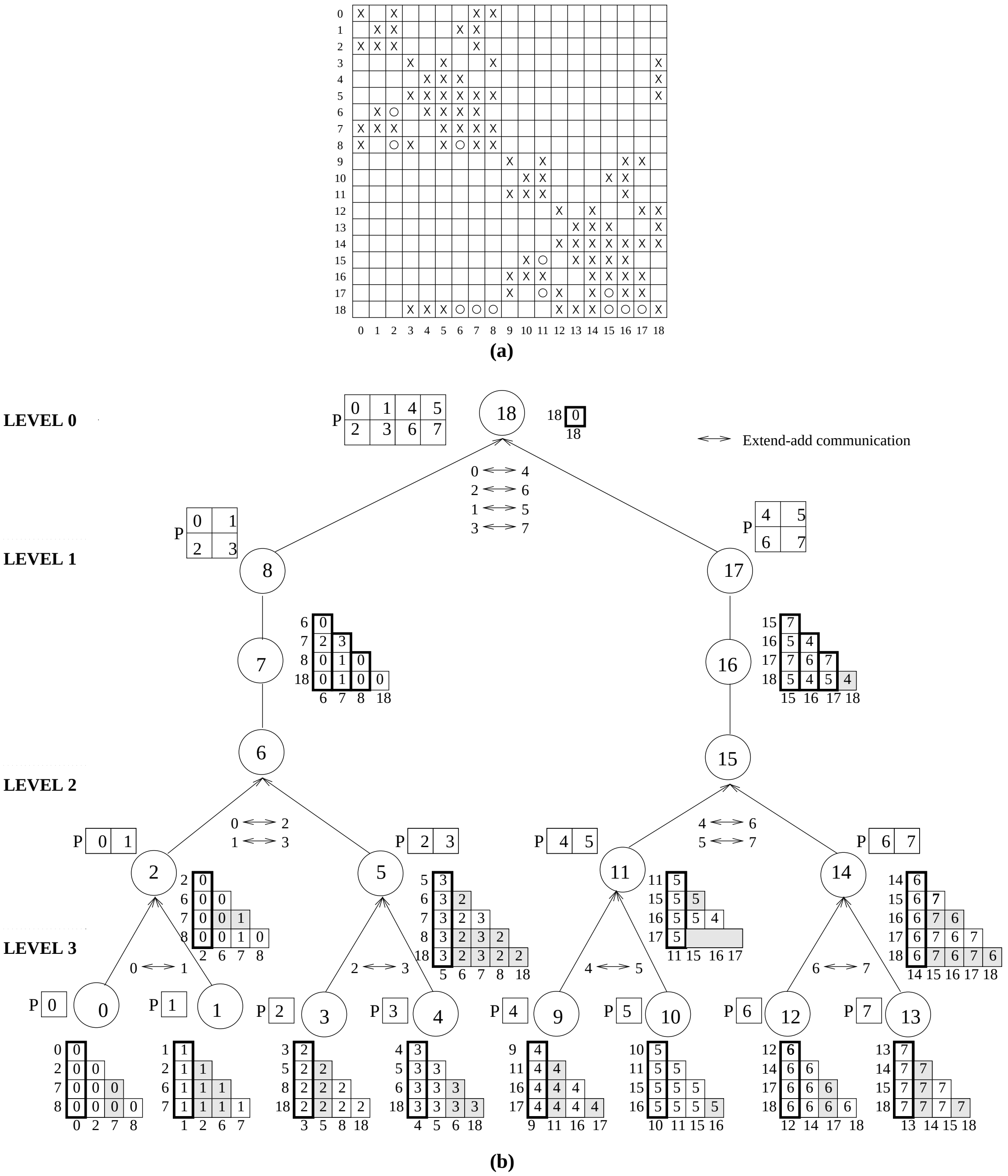

FIG. 1. **(a).** *An example symmetric sparse matrix. The non-zeros of A are shown with symbol "$\times$" in the upper triangular part and non-zeros of L are shown in the lower triangular part with fill-ins denoted by the symbol "$\circ$".* **(b).** *The process of parallel multifrontal factorization using 8 processors. At each supernode, the factored frontal matrix, consisting of columns of L (thick columns) and update matrix (remaining columns), is shown.*

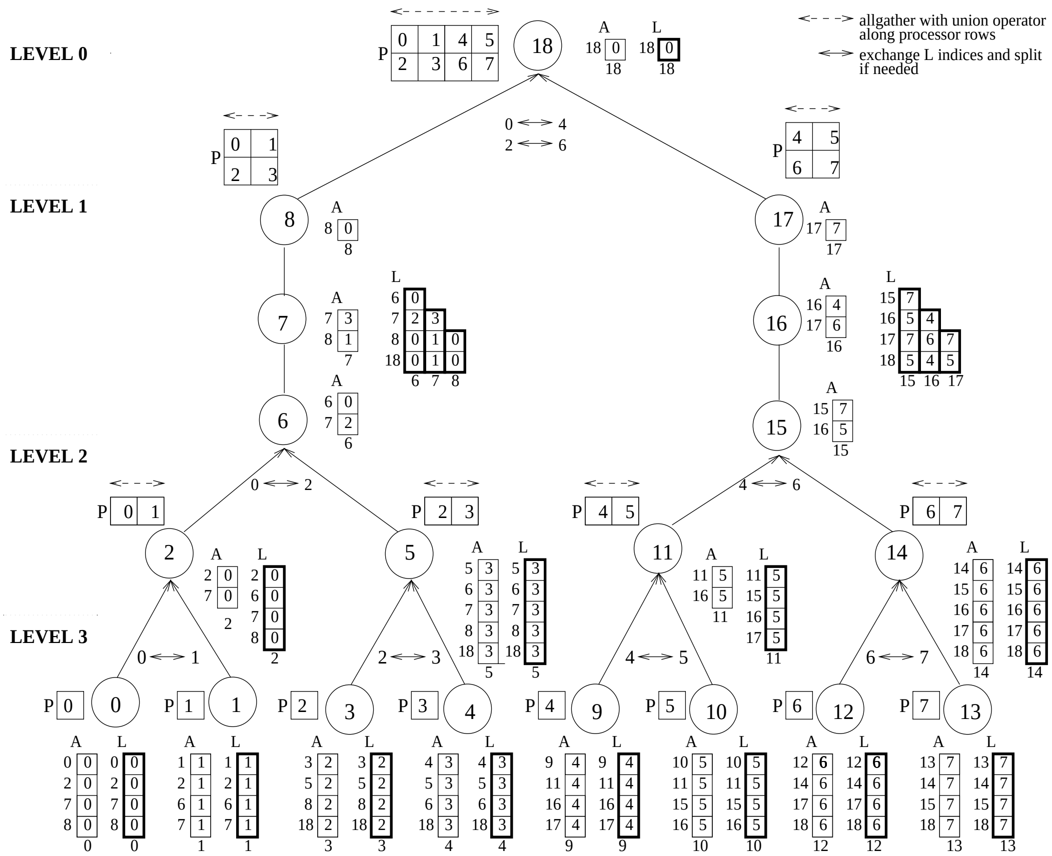

FIG. 2. *Parallel Symbolic Factorization*

are shown in the form of a logical mesh labeled with P. The frontal matrix of each supernode is distributed among this logical mesh using a bitmask based block-cyclic scheme [1]. Figure 1(b) shows such a distribution for unit blocksize. This distribution ensures that the extend-add operations required by the multifrontal algorithm can be performed in parallel with each processor exchanging roughly half of its data *only* with its partner from the other subcube. Figure 1(b) shows the parallel extend-add process by showing the pairs of processors that communicate with each other. Each processor sends out the shaded portions of the update matrix to its partner. The parallel factor operation at each supernode is a pipelined implementation of the dense column Cholesky factorization algorithm.

3 Parallel Symbolic Factorization

During the symbolic factorization phase the non-zero structure of the factor matrix L is determined. The serial algorithm to generate the structure of L performs a postorder traversal of the elimination tree. At each node k, the L indices of all its children node (excluding the children nodes themselves) and the A indices of k are merged together to form the L indices of node k.

This algorithm can be effectively parallelized using the same subtree-to-subcube mapping used by the numerical factorization algorithm. The basic structure of the algorithm is illustrated in Figure 2. Initially, matrix A is distributed such that the columns of A of each supernode are distributed using the same bitmask based block-cyclic distribution required by the numerical factorization phase [2]. Now, the non-zero structure of L is determined in a bottom-up fashion. First the non-zero structure of the leaf nodes is determined and it is sent upwards in the tree,

[2]This distribution is performed by the ordering algorithm

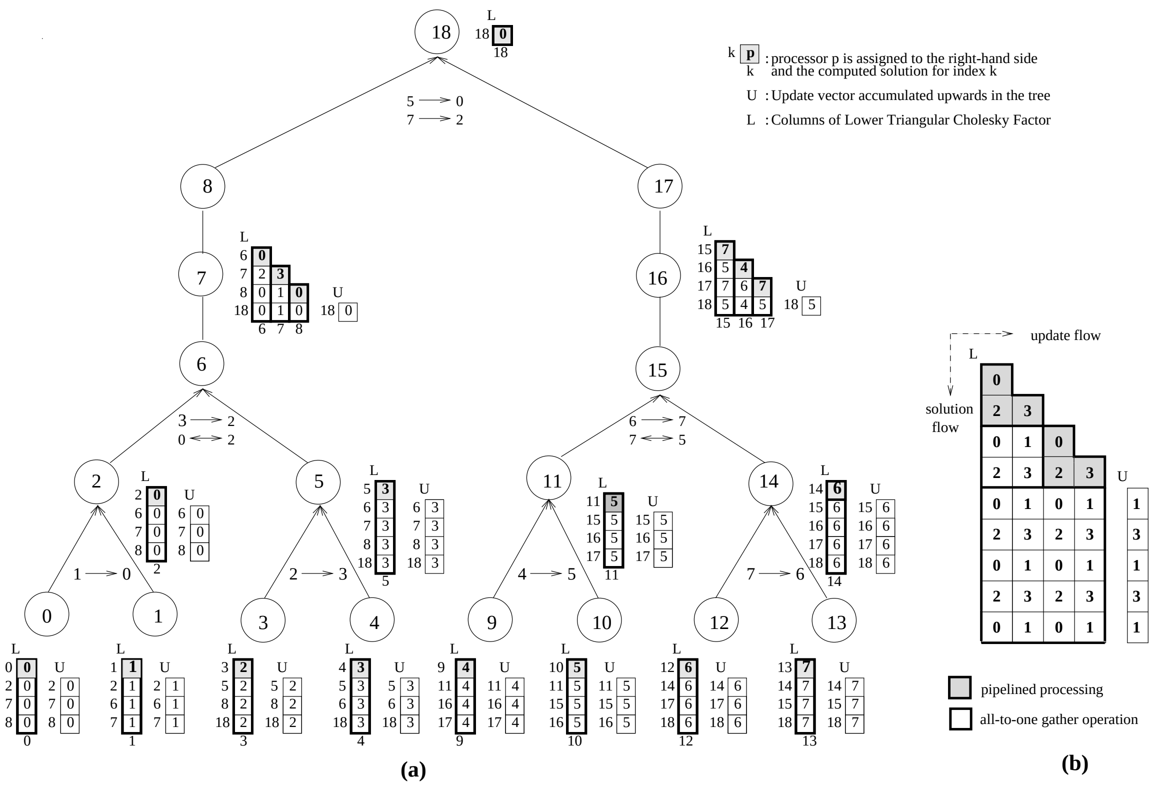

FIG. 3. *Parallel Triangular Solve.* **(a).** *Entire process of parallel forward elimination for the example matrix.* **(b).** *Processing within a hypothetical supernodal matrix for forward elimination.*

to the processors that store the next level supernode. These processors determine the non-zero structure of their supernode and merge it with the non-zero structure received from their children nodes. For example, consider the computation involved in determining the structure of L for the supernode consisting of the nodes {6,7,8} which are distributed among a 2×2 processor grid. First, the processors determine the non-zero structure of L by performing a union along the rows of the grid to collect the distinct row indices of the columns of A corresponding to the nodes {6,7,8}. The result is {6,8} and {7} at the first and second row of processors, respectively. Now the non-zero structure of the children nodes are received (excluding the nodes themselves). Since these non-zero structures are stored in the two 1×2 sub-grids of the 2×2 grid, this information is sent using a similar communication pattern described in Section 2, however only the processors in the first column of the subgrids need to communicate. In particular, processor 0 splits the list {6,7,8} into two parts {6,8} and {7}, retains {6,8} and sends {7} to processor 2. Similarly processor 2 splits the list {6,7,8,18} into two parts {6,8,18} and {7}, retains {7} and sends {6,8,18} to processor 0. Now processors 0 and 2 merge these lists. In particular, processor 0 merges {6,8}, {6,8} and {6,8,18} and processor 2 merges {7}, {7} and {7}.

4 Parallel Triangular Solve

During the phase of solving triangular systems, a forward elimination $Ly = b'$, where $b' = Pb$, is performed followed by a backward substitution $L^T x' = y$ to determine the solution $x = P^T x'$. Our parallel algorithms for this phase are guided by the supernodal elimination tree. They use the same subtree-to-subcube mapping and the same two-dimensional distribution of the factor matrix L as used in the numerical factorization.

Figure 3(a) illustrates the parallel formulation of the forward elimination process. The right hand side vector b', is distributed to the processors that own the corresponding diagonal blocks

phase \ analysis	2-D constant node-degree graphs: Serial Complexity	2-D constant node-degree graphs: Parallel Complexity	3-D constant node-degree graphs: Serial Complexity	3-D constant node-degree graphs: Parallel Complexity
Order	$O(N \log N)$	$O(\frac{N \log N}{p^{1/2}})$	$O(N \log N)$	$O(\frac{N \log N}{p^{1/2}})$
SymFact	$O(N \log N)$	$O(\frac{N \log N}{p}) + O(\frac{N^{1/2}}{p^{1/2}})$	$O(N^{4/3})$	$O(\frac{N^{4/3}}{p}) + O(\frac{N^{2/3}}{p^{1/2}})$
NumFact	$O(N^{3/2})$	$O(\frac{N^{3/2}}{p}) + O(N p^{1/2})$	$O(N^2)$	$O(\frac{N^2}{p}) + O(N^{4/3} p^{1/2})$
TriSolve	$O(N \log N)$	$O(\frac{N \log N}{p}) + O(N^{1/2})$	$O(N^{4/3})$	$O(\frac{N^{4/3}}{p}) + O(N^{2/3})$

TABLE 1

Complexity Analysis of various phases of our parallel solver for N-vertex constant node-degree graphs.

of the L matrix as shown in the shaded blocks in Figure 3(a). The computation proceeds in a bottom-up fashion. Initially, for each leaf node k, the solution y_k is computed and is used to form the update vector $\{l_{ik}y_k\}$ (denoted by "U" in Figure 3(a)). The elements of this update vector need to be subtracted from the corresponding elements of b', in particular $l_{ik}y_k$ will need to be subtracted from b'_i. However, our algorithm uses the structure of the supernodal tree to accumulate these updates upwards in the tree and subtract them only when the appropriate node is being processed. For example consider the computation involved while processing the supernode {6,7,8}. First the algorithm merges the update vectors from the children supernodes to obtain the combined update vector for indices {6,7,8,18}. Note that the updates to the same b' entries are added up. Then it performs forward elimination to compute y_6, y_7 and y_8. This computation is done using a two dimensional pipelined dense forward elimination algorithm. At the end of the computation, the update vector on processor 0 contains the updates for for b'_{18} due to y_6, y_7 and y_8 as well as the updates received from supernode {5}. In general, at the end of the computation at each supernode, the accumulated update vector resides on the column of processors that store the last column of the L matrix of that supernode. This update vector needs to be sent to the processors that store the first column of the L matrix of the parent supernode. Because of the bitmask based block-cyclic distribution, this can be done by using at most two communication steps [3].

The details of the two-dimensional pipelined dense forward elimination algorithm are illustrated in Figure 3(b) for a hypothetical supernode. The solutions are computed by the processors owning diagonal elements of L matrix and flow down along a column. The accumulated updates flow along the row starting from the first column and ending at the last column of the supernode. The processing is pipelined in the shaded regions and in other regions the updates are accumulated using a reduction operation along the direction of the flow of update vector.

Our algorithm for parallel backward substitution is similar except for two differences. First, the computation proceeds from the top supernode of the tree down to the leaf. Second, the computed solution that gets communicated across the levels of the supernodal tree instead of accumulated updates and this is achieved with at most one communication per processor. Refer to [3] for details.

5 Analysis and Preliminary Experimental Results

Table 1 shows the serial and parallel time complexities of the various phases of our parallel direct solver for matrices corresponding to 2-D and 3-D finite element meshes [2, 1, 3]. From this table we see that the overall time complexity is dominated by the numerical factorization phase. The isoefficiency function of the parallel solver is determined by the numerical factorization phase and it is $O(p^{1.5})$ for both 2-D and 3-D finite element problems. As discussed in [4], the isoefficiency function of the dense Cholesky factorization algorithm is also $O(p^{1.5})$. Thus our sparse direct solver is as scalable as the dense factorization algorithm.

np	Factor MFLOPS	Forward Solve MFLOPS				Backward Solve MFLOPS			
		nr = 1	2	4	8	nr = 1	2	4	8
bcsstk13 (N=2003,blocksize=32,factor mflop=53.4,solve mflop=1.08*nr)									
1	69.3	39.3	36.7	52.6	86.8	49.1	52.6	68.0	103.0
2	101.0	57.3	59.3	85.1	127.6	77.2	87.3	117.6	170.5
4	149.3	64.7	75.5	106.3	160.4	78.9	98.5	137.5	217.1
8	235.0	68.9	86.8	126.5	187.2	74.9	100.5	148.1	240.7
16	400.9	68.5	94.4	139.0	217.8	76.8	109.6	163.2	271.0
bcsstk15.snd6 (N=3948,blocksize=32,factor mflop=96.9,solve mflop=2.04*nr)									
1	72.7	39.7	37.1	54.4	89.5	47.3	50.8	68.7	105.2
2	106.8	56.1	56.6	81.5	126.5	64.8	80.2	106.9	163.3
4	130.6	57.7	61.1	88.7	134.1	67.8	83.1	109.1	172.7
8	148.0	61.1	66.5	97.4	149.4	72.0	86.6	121.3	179.6
16	237.8	65.6	79.7	111.6	174.3	71.9	95.0	135.1	210.5
32	314.5	63.9	73.9	121.4	204.1	74.8	102.3	150.7	245.4
hsct1 (N=16146,blocksize=32,factor mflop=791.2,solve mflop=10.8*nr)									
1	108.1	58.0	55.2	75.8	118.0	53.0	76.3	96.1	140.3
2	160.8	47.8	83.9	116.5	179.2	61.3	119.6	152.2	219.4
4	296.0	123.2	128.3	181.0	266.0	156.0	200.7	260.1	372.2
8	463.3	157.2	172.2	241.2	353.7	183.7	239.3	318.7	464.3
16	761.9	180.0	227.9	314.0	455.3	223.6	282.7	377.7	571.6
32	1068.2	200.8	259.5	358.3	550.0	218.1	302.4	415.1	650.0
hsct16152 (N=16152,blocksize=32,factor mflop=660.3,solve mflop=10.5*nr)									
1	117.7	57.5	53.6	74.5	116.1	62.8	77.2	96.2	138.4
2	155.4	41.2	76.2	106.0	165.5	51.9	109.5	138.3	199.6
4	187.5	91.8	89.8	125.2	191.8	103.7	130.3	166.2	237.8
8	225.1	120.1	121.6	163.2	240.3	132.4	160.0	202.4	289.5
16	385.1	147.2	162.1	217.0	308.1	172.6	209.1	232.1	387.1
32	453.5	148.3	142.2	214.4	345.3	166.1	211.8	281.6	420.8

TABLE 2

Performance for factorization and solve phases (np: number of processors, mflop: million floating point operations, nr: number of right-hand sides)

We have implemented our parallel direct solver using the MPI library for communication and the BLAS library for computation within a processor. Use of MPI makes our solver portable to a wide range of parallel computers, and by using BLAS, it is able to achieve high computational performance especially on the platforms with vendor-tuned BLAS libraries. We tested the solver on the IBM SP2 using IBM's BLAS and MPI libraries. Some preliminary performance results on up to 32 processors are shown in Table 2 for four test matrices. The table shows the performance of the numerical factorization phase and the performance of forward and backward solution phases for different number of right-hand sides. These results are obtained using a blocksize of 32 that has been found to produce good results on SP2 for the numerical factorization phase. However, this blocksize may not be the best for the forward and backward solution phases because of the difference in the amount of computation and communication performed.

From the table, we see that the overall MFLOPS achieved for the numerical factorization as well as the forward and backward solution increases with the number of processors. In particular for hsct1, the numerical factorization algorithm achieves a speedup of 10 on 32 processors, despite the fact that the problem is quite small (it takes 0.74 seconds to factor on 32 processors). High performance is also achieved by both forward and backward solution algorithms. In particular for hsct1, the entire triangular solution takes 0.05 seconds on 32 processors for a single right-hand side. Note that the MFLOPS performance of the triangular solutions improves as the number of right-hand sides increases. This is because the algorithms are able to achieve level 3 BLAS performance.

We have made our parallel solver library and the detailed experimental results available via WWW at URL: **http://www.cs.umn.edu/~kumar**.

References

[1] 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.

[2] George Karypis and Vipin Kumar, *Parallel multilevel graph partitioning*, Technical Report TR 95-036, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[3] Mahesh Joshi and Vipin Kumar, *Two-Dimensional Scalable Parallel Algorithms for Solution of Triangular Systems*, Technical Report, Department of Computer Science, University of Minnesota, MN, 1997.

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

[5] J. W. H. Liu, *The Multifrontal Method for Sparse Matrix Solution: Theory and Practice*, SIAM Review, vol.34, no.1, pp. 82-109, March 1992.