

PSPASES: Building a High Performance Scalable Parallel Direct Solver for Sparse Linear Systems *

Mahesh Joshi George Karypis Vipin Kumar

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Anshul Gupta Fred Gustavson

Mathematical Sciences Department
IBM T.J.Watson Research Center
Yorktown Heights, NY 10598

Abstract

Many problems in engineering and scientific domains require solving large sparse systems of linear equations, as a computationally intensive step towards the final solution. It has long been a challenge to develop efficient parallel formulations of sparse direct solvers due to several different complex steps involved in the process. In this paper, we describe PSPASES, one of the first efficient, portable, and robust scalable parallel solvers for sparse symmetric positive definite linear systems that we have developed. We discuss the algorithmic and implementation issues involved in its development; and present performance and scalability results on Cray T3E and SGI Origin 2000. PSPASES could solve the largest sparse system (1 million equations) ever solved by a direct method, with the highest performance (51 GFLOPS for Cholesky factorization) ever reported.

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

*This work was supported by NSF CCR-9423082, by Army Research Office contract DA/DAAG55-98-1-0441, 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>.

$L^T x' = y$, 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. At the heart of this solver is a highly parallel algorithm based on multifrontal Cholesky factorization we recently developed [1]. This algorithm is able to achieve high computational rates (of over 50 GFLOPS on a 256 processor Cray T3E) and it successfully parallelizes the most computationally expensive phase of the sparse solver. Fill reducing ordering is obtained using a parallel formulation of the multilevel nested dissection algorithm [5] 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 scalable and achieve high computational rates [3].

We have integrated all these algorithms into a software library, PSPASES (Parallel SPArse Symmetric dirEct Solver). It 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. Various easy-to-use functional interfaces, and internal memory management are among other features of PSPASES.

In the following, first we briefly describe the parallel algorithms for each of the phases, and their theoretical scalability behavior. Then, we describe the functional interfaces and features of the PSPASES library. Finally, we demonstrate its performance and scalability for a wide range of matrices on Cray T3E and SGI Origin 2000 machines.

2 Algorithms

PSPASES implements scalable parallel algorithms in each of the phases. In essence, the algorithms are driven by the elimination tree structure of A [9]. The ordering phase computes an ordering with the aims of reducing the fill-ins and generating a balanced elimination tree. Symbolic factorization phase traverses the elimination tree in a bottom-up fashion to compute the non-zero structure of L . Numerical factorization phase is based on a highly parallel multifrontal algorithm. This phase determines the internal data distribution of A and L matrices. The triangular solve phase also uses multifrontal algorithms. Following subsections give more details of each of the phases.

2.1 Parallel Ordering

We use our multilevel nested dissection based algorithm for computing fill-reducing ordering.

The matrix A is converted to its equivalent graph, such that each row i is mapped to a vertex i of the graph and each nonzero $a_{ij}(i \neq j)$ is mapped to an edge between vertices i and j . A parallel graph partitioning algorithm [5] is used to compute a high quality p -way partition, where p is the number of processors. The graph is then redistributed according to this partition. This pre-partitioning phase helps to achieve high performance for computing the ordering, which is done by computing the $\log p$ levels of the elimination tree concurrently. At each level, multilevel paradigm is used to partition the vertices of the graph. The set of nodes lying at the partition boundary forms a *separator*. Separators at all levels are stored. When the graph is separated into p parts, it is redistributed among the processors such that each processor receives a single subgraph, which it orders using the multiple minimum degree (MMD) algorithm. Finally, a global numbering is imposed on all nodes such that nodes in each separator and each subgraph are numbered consecutively, respecting the MMD ordering.

Details of the multilevel paradigm can be found in [5]. Briefly, it involves gradual coarsening of a graph to a few hundred vertices, then partitioning this smaller graph, and finally, projecting the partitions back to the original graph by gradually refining them. Since the finer graph has more degrees of freedom, such refinements improve the quality of the partition.

When a matrix is reordered according to the ordering generated by above algorithm, it has a

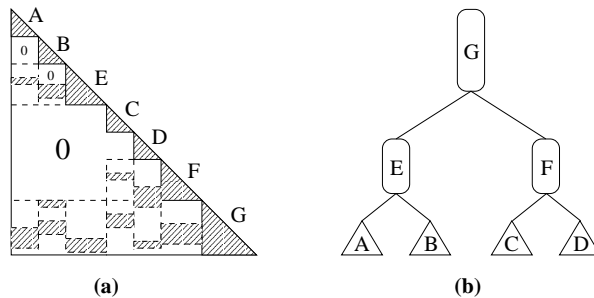


FIG. 1. Ordering a system for 4 processors. (a) Structure of a reordered matrix, (b) Separator tree corresponding to it.

format similar to that shown in Figure 1(a). The separator tree corresponding to the reordered graph has a structure of the form shown in Figure 1(b).

2.2 Parallel Numerical Factorization

We will first describe the parallel numerical factorization phase, because algorithm used here decides the data distribution of L , and hence drives the symbolic factorization algorithm. We use our highly scalable algorithm [1] that is based on the multifrontal algorithm [8].

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.

A collection of consecutive nodes in the elimination tree, each with only one child, is called a *supernode*. Nodes in each supernode are collapsed together to form a *supernodal elimination tree*, which corresponds to the separator tree [Figure 1(b)] in the top $\log p$ levels.

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

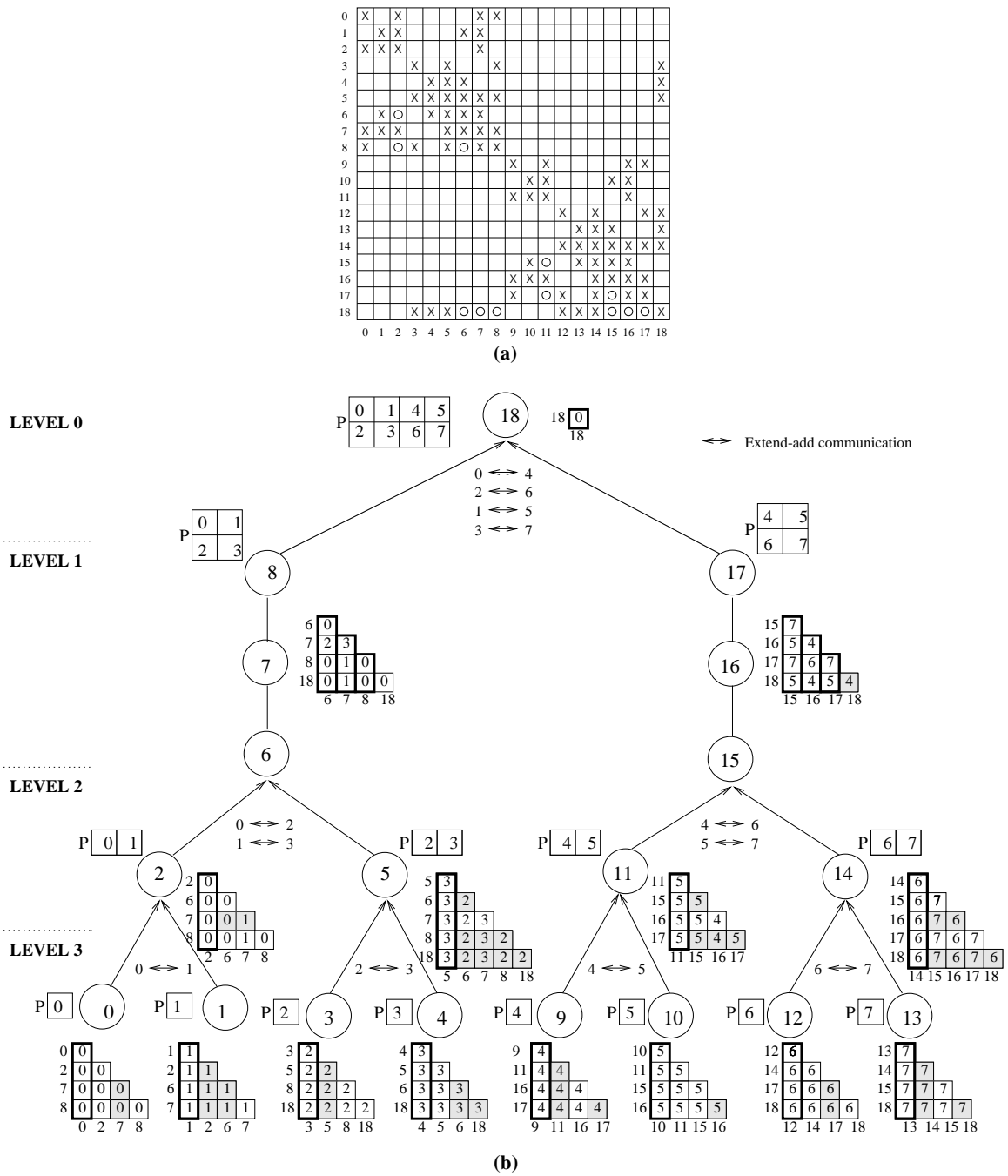


FIG. 2. (a). An example symmetric sparse matrix. The non-zeros of A are shown with symbol “x” in the upper triangular part and non-zeros of L are shown in the lower triangular part with fill-ins denoted by the symbol “o”. (b). The process of parallel multifrontal factorization using 8 processors. At each supernode, the factored frontal matrix, consisting of columns of L (thick columns) and update matrix (remaining columns), is shown.

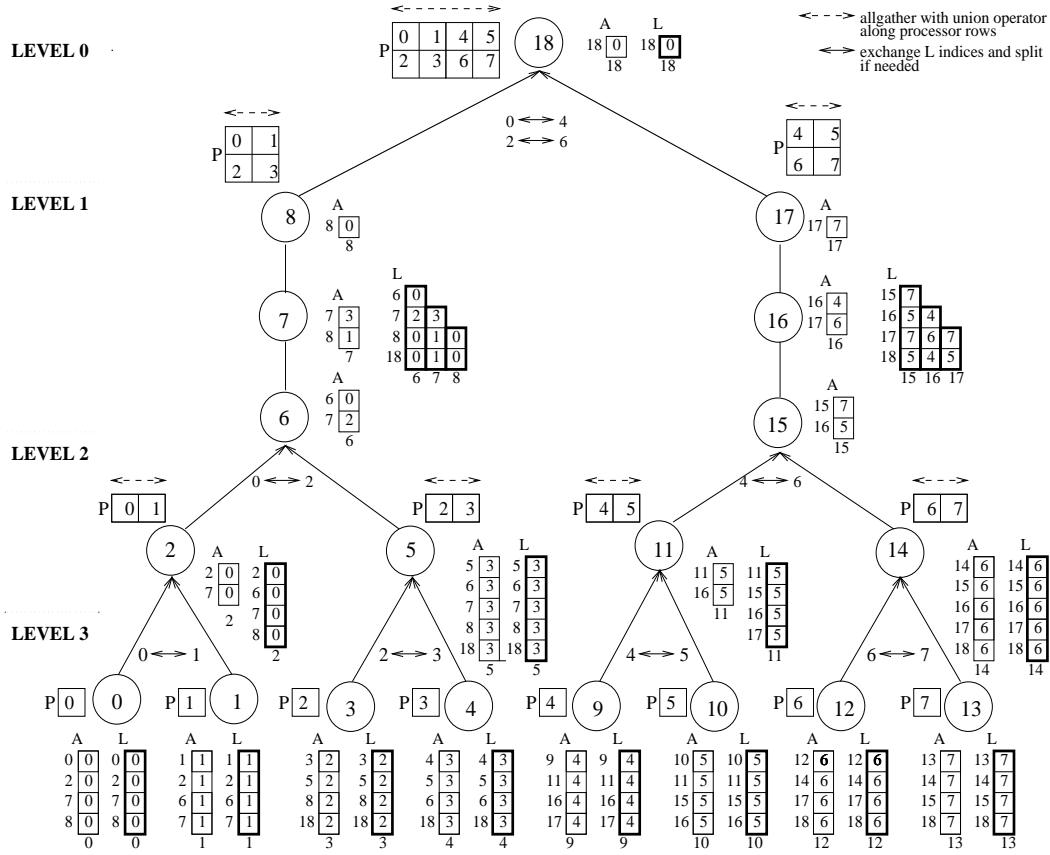


FIG. 3. Parallel Symbolic Factorization

is binary in the top $\log p$ levels¹. The portions of this binary supernodal tree are assigned to processors using a subtree-to-subcube strategy illustrated in Figure 2(b), where eight processors are used to factor the example matrix of Figure 2(a). The subcubes of processors working on various subtrees 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 2(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 2(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.

2.3 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

¹The separator tree obtained from the recursive nested dissection parallel ordering algorithm used in our solver yields such a binary tree (Fig. 1(b)).

Figure 3. 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, 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 2.2, 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 $\{7\}$.

2.4 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 4(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 of the L matrix as shown in the shaded blocks in Figure 4(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 4(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 4(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.

²This distribution is performed after computing ordering.

Phase	2-D constant node-degree graph			3-D constant node-degree graph		
	T_s	T_o	IsoEff	T_s	T_o	IsoEff
Order	$O(N \log N)$	$O(p^2 \log p)$	$O(p^2 \log p)$	$O(N \log N)$	$O(p^2 \log p)$	$O(p^2 \log p)$
SymFact	$O(N \log N)$	$O(\sqrt{Np} \log p)$	$O(p \log p)$	$O(N^{4/3})$	$O(N^{2/3} \sqrt{p})$	$O(p)$
NumFact	$O(N^{3/2})$	$O(N \sqrt{p})$	$O(p \sqrt{p})$	$O(N^2)$	$O(N^{4/3} \sqrt{p})$	$O(p \sqrt{p})$
TriSolve	$O(N \log N)$	$O(\sqrt{Np})$	$O(p^2 / \log p)$	$O(N^{4/3})$	$O(N^{2/3} p)$	$O(p^2)$

TABLE 1

*Complexity Analysis of various phases of our parallel solver for N -vertex constant node-degree graphs. Notations: T_s = Serial Runtime Complexity, T_o = Parallel Overhead = $pT_p - T_s$, where T_p is the Parallel Runtime. *IsoEff* is the isoefficiency function indicating the rate at which the problem size should increase with the number of processors to achieve same efficiency.*

3 Library Implementation and Experimental Results

We have implemented all the parallel algorithms presented in previous section, and integrated them into a library called PSPASES [4]. This library uses MPI call interface for communication, and BLAS interface for computation within a processor. This makes PSPASES portable to a wide range of parallel computers. Furthermore, using BLAS, it is able to achieve high computational performance especially on the platforms with vendor-tuned BLAS libraries.

We have incorporated some more features into PSPASES to make it easy-to-use. The library provides simple functional interfaces, and allows user to call its functions from both C and Fortran 90 programs. It uses the concept of a communicator which relieves the user from managing the memory required by various internal data structures. The communicator also allows to solve multiple systems with same sparsity structure, or same system for multiple sets of right hand side matrix, B . PSPASES also provides the user with useful statistical information such as the load imbalance of the supernodal elimination tree, number of fill-ins (both of which can be used to judge the quality of ordering), operation count for factor operation, and estimation of memory for internal data structures.

We have tested PSPASES on a wide range of parallel machines, for a wide range of sparse matrices. Tables 3 and 4 give experimental results obtained on a 32-processor SGI Origin 2000 and a 256-processor Cray T3E-1200, respectively. We used native versions of both the MPI and BLAS libraries. The notations used in presenting the results are described in Table 2. We give numbers for the numerical factorization and triangular solve phases only. The parallel ordering algorithm we use is incorporated into a separate library called ParMetis [6], and PSPASES library functions just call the ParMetis library functions. ParMetis performance numbers are reported elsewhere [6]. The symbolic factorization took relatively very small time, and hence, we do not report those numbers here.

It should be noted that the blocksize of distributing supernodal matrices and the load balance of the supernodal tree played important roles in determining the performance of the solver. Also, since most of today's parallel computers have better communication performance for larger data being communicated, blocksize optimal for numerical factorization need not be optimal for triangular solve phase, which has relatively small data communication per computational block. For the results presented, blocks of 1024 or 4096 double precision numbers worked best for numerical factorization phase.

It can be seen from Tables 3 and 4, that numerical factorization phase demonstrates high performance and scalability for all the matrices. Triangular solve phase also yields a very high performance, but for larger number of processors it tends to show an unscalable behavior. We profiled various parts of the code to see where the unscalability was appearing and found that it was in the part where b and x' are being permuted in parallel ($b' = Pb$ and $x = P^T x'$). Analyzing it further showed that bad performance of all-to-all operations of underlying MPI library for large chunks of data being exchanged among large number of processors caused such a behavior. The basic algorithms for both forward elimination and backward substitution performed scalably.

N	: Dimension of A.
NNZ_LowerA	: Number of non-zeros in Lower triangular part of A (including diagonal).
np	: Number of Processors.
NNZ_L	: Number of non-zeros in L.
Imbal	: Computational Load Imbalance due to supernodal tree structure.
FAC_OPC	: Operation count for Numerical Factor phase.
Ntime	: Numerical Factor Time (in seconds).
Nperf	: Numerical Factor Performance (in MFLOPS).
Ttime	: Redistribution of b and x + Triangular Solve Time for nrhs = 1.

TABLE 2

Notations used in the presented results

The highest performance of 51 GFLOPS was obtained for numerical factorization for the 1-million equation system CUBE100, on 256 processors of Cray T3E-1200. To our knowledge, this is the largest sparse system ever solved by a direct solution method, with the highest performance ever reported.

We have made PSPASES library and more experimental results available via WWW at URL: <http://www.cs.umn.edu/~mjoshi/pspases>.

References

- [1] Anshul Gupta, George Karypis and Vipin Kumar, *Highly Scalable Parallel Algorithms for Sparse Matrix Factorization*, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 5, pp. 502-520, 1997.
- [2] Anshul Gupta, *Analysis and Design of Scalable Parallel Algorithms for Scientific Computing*, Ph.D. Thesis, Department of Computer Science, University of Minnesota, Minneapolis, 1995.
- [3] Mahesh Joshi, Anshul Gupta, George Karypis and Vipin Kumar, *A High Performance Two Dimensional Scalable Parallel Algorithm for Solving Sparse Triangular Systems*, Proc. of 4th Int. Conf. on High Performance Computing (HiPC'97), Bangalore, India, 1997.
- [4] Mahesh Joshi, George Karypis, Vipin Kumar, Anshul Gupta, and Fred Gustavson, *PSPASES: Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems (Version 1.0), User's Manual*, Available via <http://www.cs.umn.edu/~mjoshi/pspases>.
- [5] George Karypis and Vipin Kumar, *Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering*, Journal of Parallel and Distributed Computing, vol. 48, pp. 71-95, 1998.
- [6] George Karypis, Kirk Schlogel, and Vipin Kumar *ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library, (Version 1.0), User's Manual*, Available via <http://www.cs.umn.edu/~metis>.
- [7] Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms* Benjamin/Cummings, Redwood City, CA, 1994.
- [8] Joseph W. H. Liu, *The Multifrontal Method for Sparse Matrix Solution: Theory and Practice*, SIAM Review, vol. 34, no.1, pp. 82-109, March 1992.
- [9] Joseph W. H. Liu, *The Role of Elimination Trees in Sparse Factorization*, SIAM Journal of Matrix Analysis and Applications, vol. 11, pp. 134-172, 1990.

np	NNZ_L	Imbal	FAC_OPC	Ntime	Nperf	Ttime
matrix : MDUAL2, N: 988605, NNZ_LowerA: 2.9357E+06						
4	1.77e+8	1.29	3.20e+11	685.420	466.56	8.899
8	1.90e+8	1.66	3.98e+11	490.000	811.57	5.151
16	1.96e+8	1.32	4.09e+11	256.588	1592.17	3.217
32	1.92e+8	1.43	3.63e+11	140.110	2590.74	2.399
matrix : AORTA, N: 577771, NNZ_LowerA: 1.7067E+06						
4	7.64e+7	1.48	7.95e+10	163.648	485.87	4.475
8	7.79e+7	1.74	8.47e+10	92.727	913.89	2.566
16	8.20e+7	2.59	9.46e+10	79.850	1185.14	1.633
32	8.49e+7	2.46	1.02e+11	52.877	1936.23	1.314
matrix : CUBE65, N: 274625, NNZ_LowerA: 1.0858E+06						
4	1.33e+8	1.11	4.04e+11	737.626	547.68	4.878
8	1.32e+8	1.15	4.04e+11	485.296	832.73	2.550
16	1.36e+8	1.27	4.19e+11	306.904	1363.67	1.423
32	1.40e+8	1.27	4.27e+11	137.730	3100.99	1.379
matrix : S3DKQ4M2, N: 90449, NNZ_LowerA: 2.4600E+06						
2	1.90e+7	1.05	8.87e+09	22.418	395.49	0.946
4	1.92e+7	1.10	8.72e+09	12.351	705.93	0.570
8	2.17e+7	1.44	1.29e+10	12.249	1051.67	0.583
16	2.18e+7	1.43	1.25e+10	7.357	1702.44	0.613
32	2.29e+7	1.58	1.36e+10	6.404	2124.37	0.596
matrix : COPTER2, N: 55476, NNZ_LowerA: 4.0771E+05						
2	9.88e+6	1.01	5.84e+09	14.433	404.61	0.624
4	1.02e+7	1.08	6.47e+09	9.102	710.73	0.438
8	1.06e+7	1.50	6.85e+09	6.960	984.09	0.397
16	1.11e+7	1.39	7.38e+09	4.616	1599.81	0.380
32	1.19e+7	1.39	8.29e+09	4.121	2011.69	0.422
matrix : BCSSTK18, N: 11948, NNZ_LowerA: 8.0500E+04						
2	6.18e+5	1.35	1.01e+08	0.672	149.58	0.084
4	6.54e+5	1.88	1.17e+08	0.496	234.91	0.051
8	6.71e+5	1.47	1.21e+08	0.272	444.55	0.034
16	8.10e+5	2.30	1.82e+08	0.302	602.09	0.034
32	7.56e+5	1.95	1.45e+08	0.270	537.83	0.051

TABLE 3

Performance for factorization and solve phases on SGI Origin 2000

np	NNZ_L	Imbal	FAC_OPC	Ntime	Nperf	Ttime
matrix : CUBE100 N: 1000000, NNZ_LowerA: 3.9700E+06						
256	9.06e+08	1.28	6.03e+12	117.084	51471.67	4.361
matrix : MDUAL2 N: 988605, NNZ_LowerA: 2.9357E+06						
32	1.64e+08	1.89	2.51e+11	47.787	5247.15	1.062
64	1.64e+08	1.63	2.52e+11	25.044	10073.15	0.896
128	1.64e+08	1.59	2.54e+11	16.180	15699.65	1.258
256	1.63e+08	1.72	2.47e+11	8.501	29035.15	2.267
matrix : AORTA N: 577771, NNZ_LowerA: 1.7067E+06						
8	7.86e+07	1.65	8.43e+10	60.566	1391.39	1.448
16	7.72e+07	1.92	8.20e+10	33.599	2440.07	0.855
32	7.84e+07	2.33	8.58e+10	26.651	3220.07	0.614
64	7.76e+07	2.08	8.26e+10	11.056	7472.44	0.530
128	7.96e+07	2.11	8.63e+10	7.088	12175.47	0.639
256	7.92e+07	2.03	8.53e+10	3.675	23198.01	1.190
matrix : CUBE65 N: 274625, NNZ_LowerA: 1.0858E+06						
32	1.20e+08	1.18	3.46e+11	49.944	6927.02	0.446
64	1.21e+08	1.16	3.50e+11	27.313	12822.10	0.471
128	1.20e+08	1.13	3.47e+11	14.931	23259.87	0.567
256	1.21e+08	1.20	3.49e+11	8.665	40239.15	1.112
matrix : S3DKQ4M2 N: 90449, NNZ_LowerA: 2.4600E+06						
2	1.81e+07	1.00	7.21e+09	14.910	483.39	0.512
4	1.81e+07	1.01	7.24e+09	7.797	928.73	0.278
8	1.81e+07	1.15	7.22e+09	4.444	1625.43	0.163
16	1.82e+07	1.27	7.27e+09	2.632	2761.03	0.116
32	1.82e+07	1.26	7.31e+09	1.432	5106.93	0.093
64	1.82e+07	1.31	7.24e+09	0.859	8420.89	0.109
128	1.79e+07	1.33	7.05e+09	0.517	13625.52	0.090
256	1.74e+07	1.40	6.81e+09	0.361	18877.46	0.114
matrix : COPTER2 N: 55476, NNZ_LowerA: 4.0771E+05						
2	9.54e+06	1.03	5.43e+09	11.271	482.08	0.431
4	9.46e+06	1.14	5.26e+09	5.912	890.14	0.240
8	9.34e+06	1.32	5.16e+09	3.355	1537.90	0.146
16	9.68e+06	1.44	5.57e+09	2.390	2328.89	0.103
32	9.62e+06	1.38	5.46e+09	1.211	4510.29	0.074
64	9.55e+06	1.46	5.37e+09	0.737	7284.67	0.066
128	9.64e+06	1.59	5.45e+09	0.526	10350.46	0.075
256	9.65e+06	1.68	5.55e+09	0.413	13432.96	0.140
matrix : BCSSTK18 N: 11948, NNZ_LowerA: 8.0500E+04						
2	6.18e+05	1.35	1.01e+08	0.540	186.20	0.076
4	6.54e+05	1.88	1.17e+08	0.411	283.40	0.045
8	6.72e+05	2.04	1.22e+08	0.258	473.10	0.029
16	8.10e+05	2.30	1.82e+08	0.220	824.45	0.023
32	7.84e+05	1.75	1.61e+08	0.125	1288.02	0.022
64	7.08e+05	2.14	1.20e+08	0.101	1191.23	0.020
128	6.80e+05	2.44	1.12e+08	0.087	1282.71	0.024

TABLE 4

Performance for factorization and solve phases on Cray T3E-1200