

Reducing Communication Costs for Parallel Sparse Triangular Solution *

Keita Teranishi [†] Padma Raghavan [‡] Esmond Ng[§]

1 Introduction

The parallel solution of sparse linear systems is an important computational step in many large-scale simulations. Consider the case when the coefficient matrix is symmetric and positive definite. A sparse direct method [8, 10] uses Cholesky factorization $A = LL^T$ followed by triangular solution with L . In time-dependent applications, it is common to factor A once and use L in a sequence of sparse triangular solutions with different right-hand-side vectors. Additionally, the repeated triangular solution with sparse matrix factors is required for the class of preconditioned iterative methods. The latter can be viewed as hybrids of a direct solver and an iterative solver such as Conjugate Gradients (CG) [1, 11, 29]. Such hybrids can potentially realize the benefits of both classes of methods, i.e., the robustness of direct methods and the scalability of iterative methods. On multiprocessors and networks of workstations, repeated sparse triangular solution with substitution degrades due to the large latency of interprocessor communication. This paper concerns latency-tolerant schemes for repeated distributed sparse triangular solution.

In our earlier work, we developed the “Selective Inversion” scheme as a latency-tolerant triangular solution for parallel sparse direct solvers [27]. This scheme utilizes inverses of small submatrices within the factor to replace distributed substitution by latency-tolerant distributed matrix vector multiplication. In this paper, we show how the latency costs of SI can be further reduced by a factor of 2 by using a special two-way data mapping scheme. This method can be used for repeated triangular solution using sparse matrix factors in a direct method and for applying incomplete factor preconditioners.

The rest of paper is organized as follows. Section 2 discusses our data mapping scheme in detail. We also provide analytic results on the communication costs for sparse matrices of model two and three dimensional grids. In section 3, we provide empirical results on performance using parallel preconditioned CG with incomplete Cholesky factors as our application requiring repeated triangular solution. In this section, we also compare the our method with Parasails [3]; the latter represents a relatively new family of preconditioning methods called *sparse approximate inverses* [2, 4, 13, 18]. Section 4 contains some concluding remarks.

2 A New Two-Way Data Mapping For Sparse Triangular Solution

Background information on parallel sparse direct methods are available in the following papers [12, 9, 15, 22, 27]. Parallel incomplete factorization requires many of the same techniques for effective implementation. We discuss sparse triangular solution using a general tree formulation that applies to both complete and incomplete factors.

The factorization and triangular solution process is based on compute *tree* where each node of the tree, called a supernode, represents a subset of columns of L (or \hat{L} , the incomplete factor) [20, 21]. Without loss

*This work was funded in part by the National Science Foundation through grants NSF ACI-0102537 and NSF CCR-981834.

[†]Department of Computer Science and Engineering The Pennsylvania State University 220 Pond Lab University Park, PA 16802-6106 (teranish@cse.psu.edu).

[‡]Department of Computer Science and Engineering The Pennsylvania State University 220 Pond Lab University Park, PA 16802-6106 (raghavan@cse.psu.edu).

[§]Lawrence Berkeley National Laboratory. MS 50F-1602, 1 Cyclotron Road, Berkeley, CA 94720 (engng@lbl.gov).

of generality, assume that the tree is a balanced binary tree. The computation at a supernode depends on children supernodes; no sibling supernodes are involved. In a parallel implementation, the root supernode is assigned to all processors. Next, each child supernode is assigned to half of the processors at its parent and so on until each processor is assigned an independent subtree. In L , each supernode corresponds to a small dense submatrix that consists of columns with the same nonzero row structure. This is used to reduce the space for row index arrays and utilize efficient dense matrix algorithms [6, 7]. In incomplete Cholesky, only some elements of L are retained to obtain an approximation, \hat{L} and thus, each submatrix is no longer dense.

We now consider triangular solution using L (\hat{L}) with its submatrices assigned using the supernodal tree. Figure 1 sketches the process of forward triangular solution based on the earlier SI scheme for a supernode (A) assigned to 4 processors. Henceforth, we call it SI-1 and refer to it as SI with the one-way making. The computation at the supernode (A) in Figure 1 involves the matrix operation, $\hat{L}_A x_A = b_A$ (using the notation for incomplete factorization). The system can be viewed as:

$$\begin{bmatrix} \hat{L}_{11} & 0 \\ \hat{L}_{21} & \hat{L}_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

The submatrix at the supernode (A) consists of \hat{L}_{11} and \hat{L}_{21} ; \hat{L}_{22} is located in the ancestor supernodes. The matrix is mapped by column block wrap-map to the processors 0, 1, 2, and 3. The SI scheme treats the triangular part \hat{L}_{11} and rectangular part \hat{L}_{21} in different ways. More specifically, \hat{L}_{11} is replaced by its inverse, \hat{L}_{11}^{-1} , in order to avoid parallel substitution for computing x_1 . In the lower part of the Figure 1, there are two supernodes computed by two processors. In the beginning of computation at (A), the right hand side vector b is stored in a scattered form on processors 0, 1 and 2, 3 assigned to children supernodes (B) and (C) respectively. Now a gather is required to collect b_1 to each processor. This gather is performed by a group reduction operation such as **MPI_AllReduce** for MPI [24]; we refer this operation to **Reduce_Sum**. Once b_1 is gathered, x_1 is computed by parallel matrix vector multiplication, **Parallel_Tri_MatVec**. As shown in the Figure 1, this operation produces x_1 scattered across the processors. Again, we need to gather x_1 to all working processors by using another **Reduce_Sum**. Next, the parallel matrix vector multiplication (**Parallel_MatVec**) updates the part of right hand side vector ($\tilde{b}_2 \leftarrow b_2 - \hat{L}_{22} \times x_1$). The vector \tilde{b}_2 is then available in a scattered form for computations at the parent supernode. At the parent supernode of (A), the same process is repeated but now it involves the 8 processors over both its children nodes ((A) and its sibling are each distributed on 4 processors). Thus, at each supernode, there are two **Reduce_Sum** operations. The algorithm is shown below:

Algorithm 1 (SI-1)

```

for each distributed supernode  $i$  (traversed in post-order on the tree)
  Let node  $i$  involves  $p_i$  processors
  Reduce_Sum( $b_1, p_i$ );
   $x_1 = \mathbf{Parallel\_Triangular\_MatVec}(\hat{L}_{11}, b_1, p_i)$ ;
  Reduce_Sum( $x_1, p_i$ );
   $\tilde{b}_2 = b_2 - \mathbf{Parallel\_MatVec}(\hat{L}_{21}, x_1, p_i)$ ;
end for

```

Backward solution is implemented in a similar way except the procedure is started from the root supernode down to the root nodes of local subtrees. The backward solution also requires two **Reduce_Sum** operations in each distributed supernode.

SI-1 needs two **Reduce_Sum** operations per supernode because the data mapping of distributed submatrices does not match with the distribution of vectors. We overcome this mismatch by using a new data mapping for the distributed submatrices at each supernode. Figure 2 shows our new two-way data mapping and the associated algorithm (we call it SI-2). The triangular part of the submatrix \hat{L}_{11} in a supernode is mapped in block row cyclic mapping while the rectangular part is mapped in column block cyclic fashion; block sizes are the same for both parts. Like SI-1, b_1 is gathered by **Reduce_Sum** to prepare for the solution of x_1 . In the new data structure, **Parallel_Triangular_MatVec** can be regarded as block vector dot products (see Figure 2). Now in the product, $x_1 \leftarrow \hat{L}_{11}^{-1} b_1$, every element of x_1 is computed independently by a single processor; x_1 is distributed across processors. As shown in Figure 2, the new

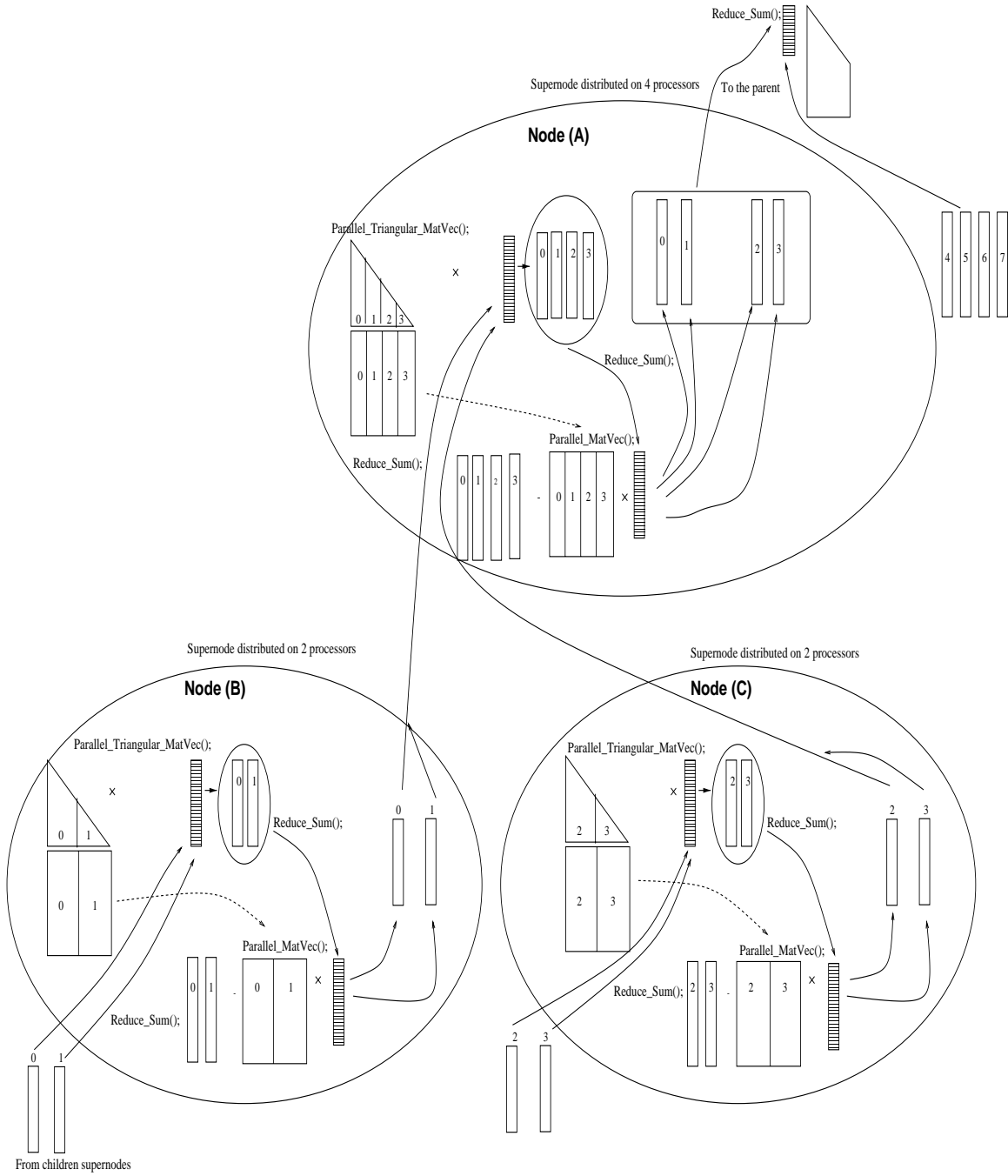


Figure 1: Forward solution at a supernode with selective inversion and the one-way data mapping (using 4 processors).

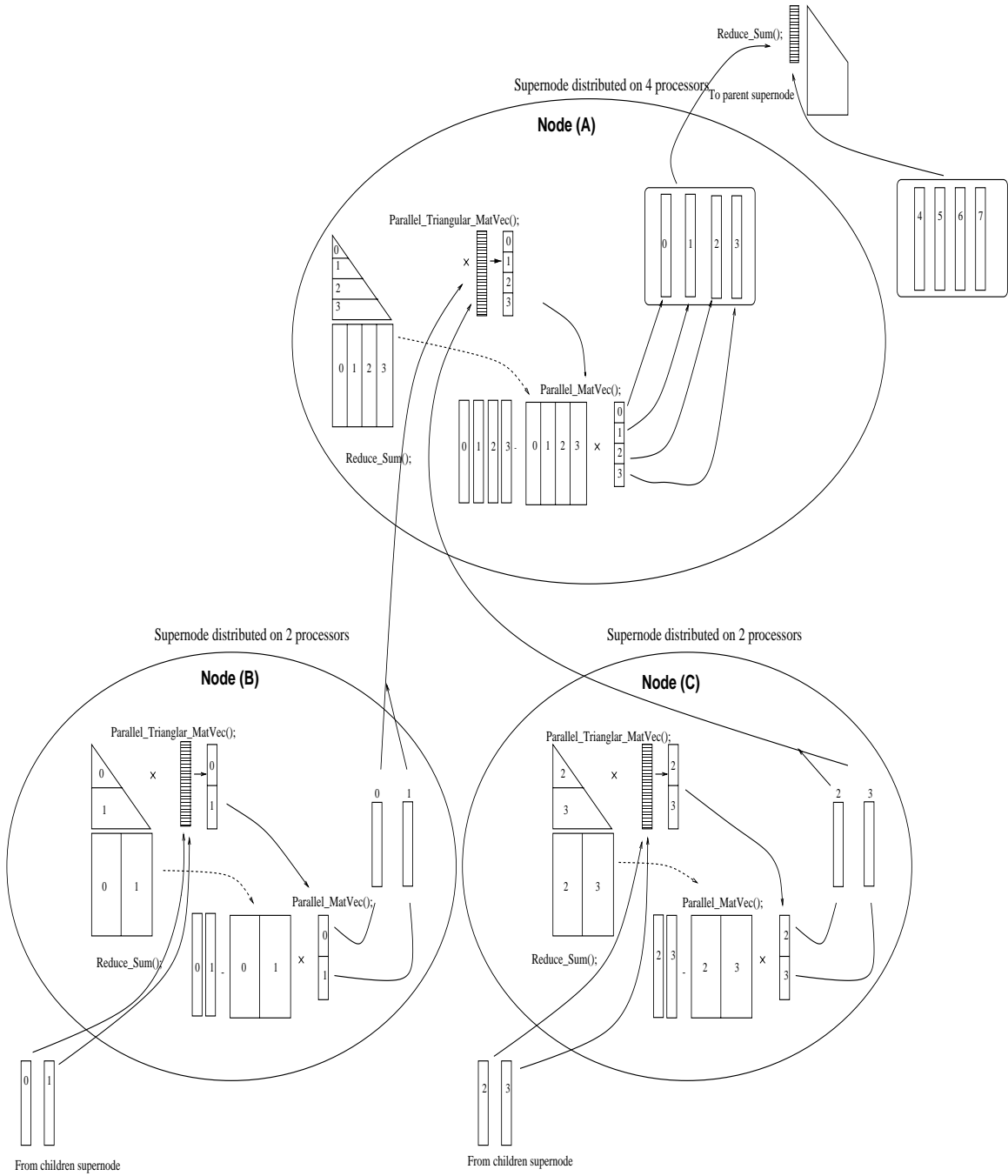


Figure 2: Forward solution at a supernode with selective inversion and the two-way data mapping (using 4 processors).

distribution of x_1 is suitable for parallel matrix vector multiplication to compute \tilde{b}_2 without any data exchange. Thus at each supernode, two **Reduce_Sum** operations are replaced by a single **Reduce_Sum** operation. SI-2 scheme is shown below.

Algorithm 2 (ICSI-2)

```

for each distributed supernode  $i$  (traversed in post-order on the tree)
  Let  $i$  involves  $p_i$  processors
  Reduce_Sum( $b_1, p_i$ );
   $x_1 = \mathbf{Parallel\_Triangular\_MatVec}(\hat{L}_{11}^{-1}, b_1, p_i)$ ;
   $\tilde{b}_2 = b_2\text{-Parallel\_MatVec}(\hat{L}_{21}, x_1, p_i)$ ;
end for

```

The main difference is that the **Reduce_Sum** between two matrix vector multiplications at each iteration is eliminated. Consequently, this modification halves the number of communication steps overall supernodes. This technique is also applicable for the backward solution.

We next summarize our analysis of the communication costs for SI-1 and SI-2 for model sparse matrices. We assume that **Reduce_Sum** with P processors can be implemented with $\log_2 P$ messages using a binary tree topology. The latency of the communication is proportional to the number of messages sent, and the data transfer cost is estimated by the data volume exchanged.

LEMMA 2.1. *Consider the $N \times N$ sparse linear system associated with the model five-point, finite-difference $K \times K$ ($N = K^2$) and $K \times K \times K$ ($N = K^3$) grids using P processors. The number of the messages for SI-2 is $\frac{(\log_2 P)^2 + \log_2 P}{2}$ ($(\log_2 P)^2 + \log_2 P$ for SI-1 [28]) and is independent of the matrix dimension. The amount of data sent for SI-2 is bounded by $3K(\log_2 P)$ ($6K(\log_2 P)$ for SI-1) on $K \times K$ grids and $3K^2(\log_2 P)$ ($6K^2(\log_2 P)$ for SI-1) on $K \times K \times K$ grids.*

The analysis is similar to an analysis of SI-1 and the traditional scheme (TS) in our earlier paper [28] where it is shown that the latency cost of TS grows as $O(\frac{N^{1/2}}{P})$ for $K \times K$ grid and $O(\frac{N^{2/3}}{P})$ for $K \times K \times K$ grids. Observe that for both SI-1 and SI-2 the communication latency costs are independent of the problem size and SI-2 has the half the costs of SI-1.

3 Empirical Results

This section contains preliminary performance results using our implementations of SI-1, SI-2 and the traditional method (TS). We use parallel preconditioned CG with drop-threshold incomplete Cholesky factors as an application with repeated triangular solution. Our implementation was based on DSCPACK [26], and we used parallel CG from the PETSc package [30]. For comparing the performance over all iterations of CG we used Parasails developed by Chow [3]. Parasails is a representative method from the class of parallel sparse approximate inverse preconditioners; we used level-of-fill parameters 0, 1, 2, and 4 to control the sparsity. The test matrices are diffusion problems associated with the $K \times K$ and the $K \times K \times K$ model grids. To keep the workload of triangular solution per processor fixed 9at the uniprocessor level), we increase K with the number of processors as shown in Table 1.

Figure 3 shows the time spent to apply a preconditioner in a single CG iteration. We use ICSI-1 for IC with SI-1, ICSI-2 for the IC with SI-2, ICTS for the IC with TS and Parasails for the sparse approximate inverse method. For all methods, we report results with the number of nonzeros in the preconditioner relative to $|L|$ set as close as possible to .1 and .5 for 2-dimensional problems (at .1 and .4 for 3-dimensional problems). There is some variation in the sparsity because of the nature of the underlying algorithms. Ideally, the scaled performance should not result in any increase in time with the number of processors and the problem size. As expected ,the performance of ICTS is the worst; ICSI-2 performs better than ICSI-1 with larger numbers of processors and is close to the ideal performance. The performance of both SI methods is comparable to that of Parasails even though the latter uses only a single distributed matrix vector product; recall that SI methods use a sequence of smaller distributed matrix-vector products on the tree.

Table 1: Description of sparse test matrices; N is the matrix dimension, $|A|$ contains the number of nonzeros in the matrix, and $|L|$ contains number of nonzeros in L , $A = LL^T$.

Processors (P)	2-dimensional grids				3-dimensional grids			
	K	N	$ A $	$ L $	K	N	$ A $	$ L $
			(10^6)	(10^6)			(10^6)	(10^6)
1	100	10000	0.030	0.221	30	27000	0.105	4.394
2	134	17956	0.054	0.440	35	42875	0.167	8.345
4	181	32761	0.098	0.863	42	74088	0.291	17.959
8	244	59536	0.178	1.731	50	125000	0.493	37.197
16	330	108900	0.326	3.399	58	195112	0.770	69.194
32	450	202500	0.606	6.712	68	314432	1.244	133.223

Figure 4 shows the number of iterations to converge for IC and Parasails. In terms of the number of iterations, IC is better than the Parasails. For the sake of completeness, we also provide total time over all preconditioned iterations. This quantity is the product of the time per iteration and the number of iterations to convergence. Our ICSI preconditioners are comparable to Parasails and they perform better especially for 2-dimensional problems. Our results clearly indicate that SI-2 makes parallel preconditioning with incomplete factors a very viable and effective method.

4 Conclusions

Our selective inversion method with the two-way data mapping scales better than selective inversion with the one-way data mapping. The performance difference is related to reducing message latency costs by halving the total number of messages that are sent for a single triangular solution using sparse matrix factors. We have shown that applying our method to the problem of scalable parallel preconditioning with incomplete factors, provides a solution with performance comparable to Parasails (a representative method from the class of sparse approximate inverse preconditioners), when the preconditioner is very sparse. Additionally, this method allows us to benefit from potentially better quality preconditioners that can be developed using incomplete factorization with larger fill-in. We plan to provide comprehensive empirical results on scalability in our final paper.

References

- [1] O. AXELSSON, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.
- [2] MICHELE BENZI AND CARL D. MEYER AND MIROSLAV TŮMA, *A Sparse Approximate Inverse Preconditioner for the Conjugate Gradient Method*, SIAM J. SCI. COMPUT., 17 (1996), pp. 1135-1149.
- [3] E. CHOW, *ParaSails User's Guide*, TECH. REPORT UCRL-MA-137863, LAWRENCE LIVERMORE NATIONAL LABORATORY, LIVERMORE, CA, 2000.
- [4] E. CHOW, *Parallel implementation and performance characteristics of least squares sparse approximate inverse preconditioners*, TECH. REPORT UCRL-JC-138883, LAWRENCE LIVERMORE NATIONAL LABORATORY, LIVERMORE, CA, 2000.
- [5] P. CONCUS, G. GOLUB, AND D. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, J. R. BUNCH AND D. J. ROSE, EDS., SPARSE MATRIX COMPUTATIONS, pp. 309-332, ACADEMIC PRESS, 1976.
- [6] J. J. DONGARRA, J. D. CROZ, S. HAMMARLING, AND I. S. DUFF, *An extended set of basic linear algebra subprograms*, ACM TRANS. MATH. SOFTWARE, 14 (1988), pp. 1-17.
- [7] J. J. DONGARRA, J. DU CROZ, I. DUFF, AND S. HAMMARLING, *An Set of Level 3 FORTRAN Basic Linear Algebra Subprograms*, ACM TRANS. MATH. SOFTWARE, 16 (1990), pp. 1-17.

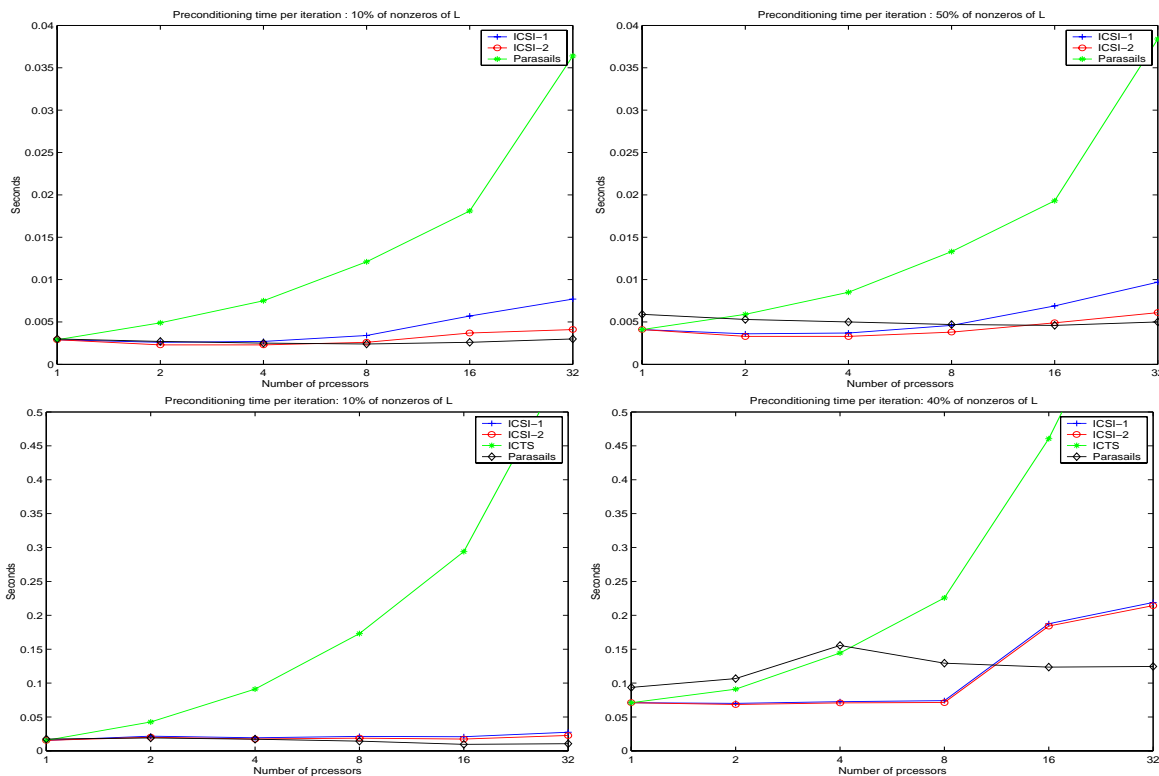


Figure 3: Time to apply the preconditioner in one CG iteration using: $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.5 for sparse matrices of 2-dimensional grids (top left and top right), and using $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.4 for sparse matrices of 3-dimensional grids (bottom left and bottom right). The problem size is scaled with the number of processors to keep the work per processor fixed.

[8] I. S. DUFF, A. M. ERISMAN, AND J. K. RAID, *Direct Methods for sparse Matrices*, CLAREDON PRESS, OXFORD, 1986.

[9] A. GEORGE AND M. T. HEATH AND J. W-H. LIU AND E. G-Y. NG *Symbolic Cholesky factorization on a local-memory multiprocessor*, PARALLEL COMPUTING 5 (1987), PP. 85–95.

[10] J. A. GEORGE, AND J. W-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, PRENTICE-HALL INC. , ENGLEWOOD CLIFFS, NJ, 1981.

[11] G. GOLUB AND C. F. VAN LOAN, *Matrix Computations, third edition*, THE JOHNS HOPKINS UNIVERSITY PRESS, BALTIMORE, MD, 1996.

[12] ANSHUL GUPTA AND VIPIN KUMAR, *A scalable parallel algorithm for sparse matrix factorization*, TECHNICAL REPORT 94-19, DEPARTMENT OF COMPUTER SCIENCE , UNIVERSITY OF MINNESOTA, MINNEAPOLIS, 1994.

[13] MARCUS J. GROTE AND THOMAS HUCKLE, *Parallel Preconditioning with Sparse Approximate Inverses*, SIAM J. SCI. COMPUT. , 18 (1997), PP. 838–853.

[14] DAVID HYSOM AND ALEX POTHEN, *A Scalable Parallel Algorithm for Incomplete Factor Preconditioning*, SIAM J. SCI. COMPUT. , 22 (2001),PP. 2194–2225.

[15] M. T. HEATH, E. NG, AND B. W. PAYTON, *Parallel algorithms for sparse linear systems*, SIAM REVIEW, 33 (1991),PP. 420–460.

[16] M. T. JONES AND P. E. PLASSMANN, —*The efficient parallel iterative solution of large sparse linear systems*, in A. George, J. R. Gilbert and J. W. H. Liu, eds, Graph Theory and Sparse Matrix Computations, IMA Vol 56, Springer-Verlag, pp. 229–245.

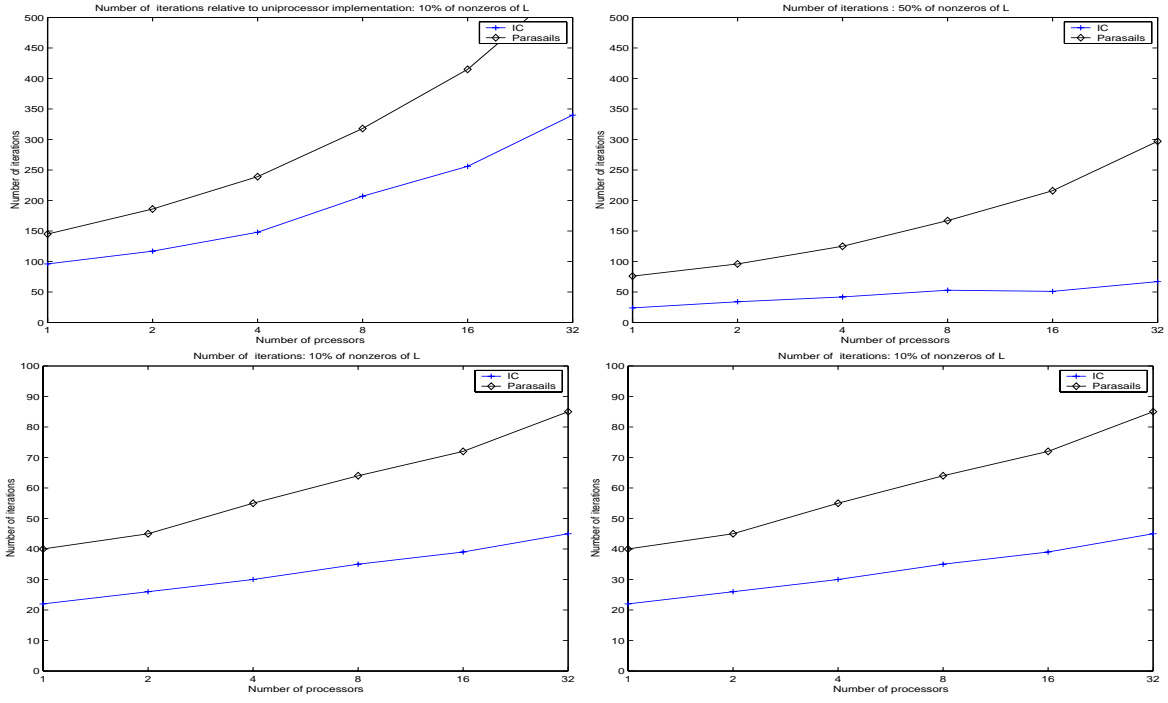


Figure 4: Number of iterations to convergence using: $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.5 for sparse matrices of 2-dimensional grids (top left and top right), and using $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.4 for sparse matrices of 3-dimensional grids (bottom left and bottom right).

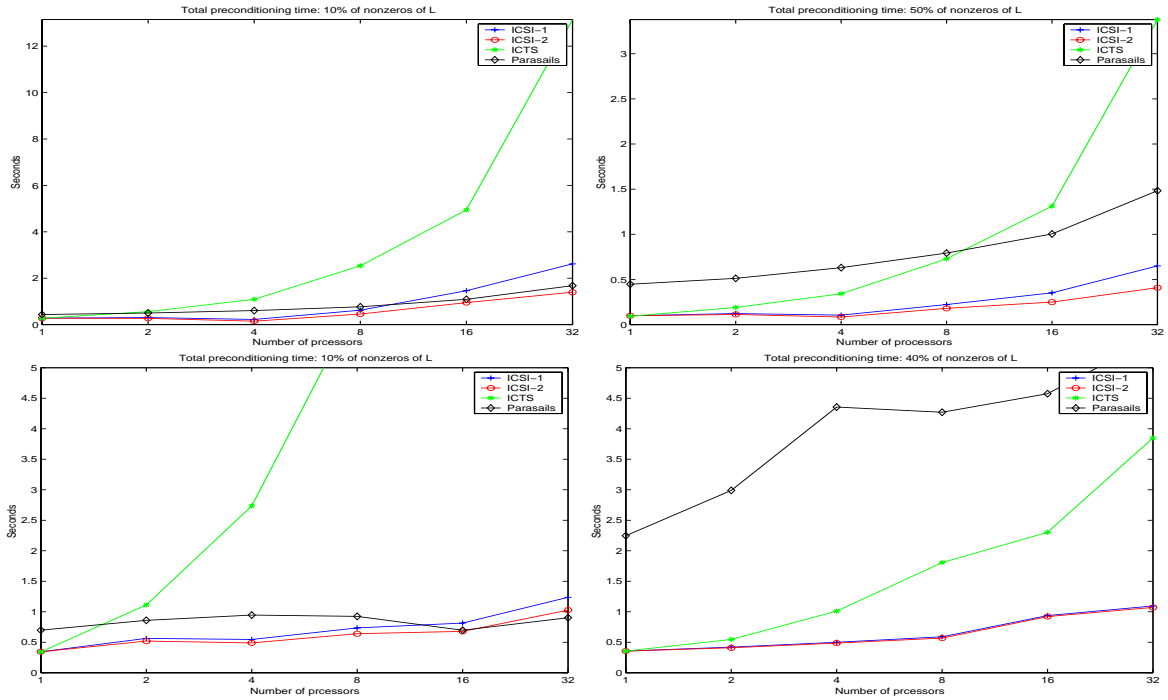


Figure 5: Total time for applying preconditioner over all iterations using: $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.5 for sparse matrices of 2-dimensional grids (top left and top right), and using $\frac{|\hat{L}|}{|L|} = 0.1$, and 0.4 for sparse matrices of 3-dimensional grids (bottom left and bottom right).

- [17] G. KARYPIS AND V. KUMAR, *Parallel threshold-based ILU factorization*, in Proceedings of the ACM Conference on Supercomputing, San Jose, CA, 1997, CD-ROM, ACM, New York, 1997.
- [18] L. YU. KOLOTILINA AND A. YU. YEREMIN, *Factorized sparse approximate inverse preconditionings I. Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
- [19] Z. LI, Y. SAAD, AND M. SOSONKINA, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Technical Report, umsi-2001-100, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
- [20] JOSEPH W. H. LIU, *The role of elimination trees on sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.
- [21] J. W. H. LIU, E. NG, AND B. W. PAYTON, *On finding supernodes for sparse matrix computations*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 242–252.
- [22] R. F. LUCAS, *Solving planar systems of equations on distributed-memory multiprocessors*, Ph. D. Thesis, Department of Electrical Engineering, Stanford University (1987).
- [23] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.
- [24] MESSAGE PASSING INTERFACE FORUM, *MPI: A Message-Passing Interface Standard*, Int. J. of Supercomputer Applications, 8, (1994) pp. 165–414.
- [25] E. G. NG, B. W. PEYTON AND P. RAGHAVAN, *A Blocked incomplete Cholesky preconditioner for hierarchical-memory computers*, Proceedings of the Fourth IMACS International Symposium on Iterative Methods in Scientific Computation, University of Texas, October 18–20, 1998. Published in IMACS Series in Computational and Applied Mathematics: Iterative Methods in Scientific Computation IV, D. R. KINCAID AND A. C. ELSTER, eds., 1999, pp. 211–222.
- [26] PADMA RAGHAVAN, *DSCPAC: A Domain-Separator Codes For The Parallel Solution Of Sparse Linear Systems*, Technical Report CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University.
- [27] P. RAGHAVAN, *Efficient parallel sparse triangular solution using selective inversion*, Parallel processing Letters, Vol. 8 No. 1 (1998), pp. 29–40.
- [28] PADMA RAGHAVAN, KEITA TERANISHI AND ESMOND NG, *Towards Scalable Preconditioning Using Incomplete Factorization*, Proceedings of 2001 International Conference On Preconditioning Techniques For Large Sparse Matrix Problems In Industrial Applications, pp. 63–65, Tahoe City CA, April 29–May 1, 2001. Longer version submitted to Numerical Linear Algebra.
- [29] YOUSEF SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
- [30] B. SMITH, L. C. MCINNES, AND W. GROPP, *PETsc 2.0 user's manual*, Technical Report ANL 95/11 - Revision 2.0.22 (1997), Mathematics and Computer Science Division Argonne National Laboratory, Argonne IL 60439.