1

A Hybrid Parallel Preconditioner Using Incomplete Cholesky Factorization and Sparse Approximate Inversion

Keita Teranishi[1] and Padma Raghavan[1]

Department of Computer Science and Engineering, The Pennsylvania State University, 111 IST Bldg., University Park, PA 16802. E-mail `teranish,raghavan@cse.psu.edu`.

Summary. We have recently developed a preconditioning scheme that can be viewed as a hybrid of incomplete factorization and sparse approximate inversion methods. This hybrid attempts to deliver the strengths of both types of preconditioning schemes to accelerate the convergence of Conjugate Gradients (CG) for sparse linear system solution on multiprocessors. We provide an overview of our algorithm and report on initial results on some large sparse systems.

1.1 Introduction

Consider the solution of a sparse linear system $Ax = b$ on a distributed memory multiprocessor. When A is symmetric positive definite, preconditioned Conjugate Gradients (CG) [1, 2] can be used to solve the system. Although CG is scalable, the scalability and effectiveness of the preconditioner plays a pivotal role in the overall performance. Traditionally, incomplete Cholesky factorization with a drop threshold (ICT) [3] scheme can be used to construct a preconditioner $\hat{L}$ as an approximation to L, the sparse Cholesky factor of A ($A = LL^T$). Such an ICT preconditioner is often the method of choice on uniprocessors, but its scalable parallel implementation poses many challenges.

A parallel ICT scheme should ideally allow (i) efficient preconditioner construction, and (ii) latency-tolerant application at each CG iteration. Applying an ICT preconditioner requires distributed triangular solution which is typically inefficient due to the relatively large latencies of interprocessor communication on multiprocessors. We had earlier addressed this issue by developed a parallel ICT preconditioner with a feature called 'Selective Inversion' (SI) [4, 5, 6]. In our preconditioning scheme (ICT-SI), certain triangular submatrices were explicitly inverted to replace distributed substitution schemes

[⋆] This work was supported in part by the National Science Foundation through grants ACI-0102537, EIA-0221916, and DMR-0205232.

by latency-tolerant matrix-vector multiplication. Additionally, ordering, partitioning and blocking techniques from parallel sparse solvers were used to construct the ICT preconditioner efficiently. Our ICT-SI scheme enabled the scalable application of the preconditioner at each CG step while effectively accelerating the convergence of CG [5]. However, preconditioner construction ICT-SI was still relatively expensive. In this paper, we attempt to address this issue by using sparse approximate inversion techniques [7, 8, 9, 10] instead of the explicit inversion required in ICT-SI. We call our new scheme ICT-SSAI, i.e., ICT with 'selective sparse approximate inversion' [6].

We provide a brief overview of sparse approximate inverse preconditioning and our incomplete Cholesky preconditioner with SI (ICT-SI) in Section 1.2. We next describe in Section 1.3, our new ICT-SSAI scheme where sparse approximate inversion is used on selected submatrices in the incomplete factor $\hat{L}$ as an alternative to the SI scheme; we also provide some empirical results on the performance of our schemes and other preconditioners. We end with some concluding remarks in Section 1.4.

1.2 Background

Incomplete Cholesky factorization is a popular preconditioning scheme on uniprocessors. However, on multiprocessors with large latencies of interprocessor communication, the application of such preconditioners using parallel substitution does not scale well. This gave rise to a new class of preconditioning schemes that attempted to approximate an inverse of A which could then be applied using efficient parallel matrix-vector multiplication. However, these preconditioners may not be as effective as those from incomplete Cholesky [11] when systems from a wide range of applications are considered. Earlier, we had developed the ICT-SI scheme to enable latency tolerant application of ICT preconditioners on parallel multiprocessors. In this section, we provide a brief overview of sparse approximate inverse preconditioners and our ICT-SI. Our new ICT-SSAI preconditioner is in effect a hybrid of these two schemes.

1.2.1 Sparse Approximate Inverse Preconditioners

Sparse approximate inverse techniques are based on the Frobenius norm minimization [8, 9, 10] of $\|I - AM\|_F$, where M is the preconditioner. This problem can be formulated as multiple least square problems of the form: $\|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2$. In this expression, e_j is a canonical vector and m_j is a jth column of M. The least-square solution is performed using dense matrix algebra kernels [12] on nonzero elements (blocks) in M. The sparsity pattern of M is selected a priori. If A is symmetric and positive definite (SPD), the method tries to minimize $\|I - LG\|_F$, where L is a Cholesky factor of A and G is a lower triangular preconditioner matrix. Since each least-square solution can be computed independently, the method is highly suitable for parallel implementation as shown by Grote and Huckle [9], and Chow [8]. However, preconditioning quality may lag that of ICT preconditioners.

1.2.2 Incomplete Cholesky with Selective Inversion

Our parallel incomplete Cholesky with SI uses many of the ideas from parallel sparse direct multifrontal solution. We start with a good fill-reducing strategy such as minimum-degree and nested dissection [13]; the latter also helps provide a natural data partitioning for the parallel implementations. We then compute an approximate $\hat{L}$ corresponding to the true factor L for the given ordering.

The parallel factorization and triangular solution of $\hat{L}$ is guided by the traversal of the elimination tree [14] in conjunction with supernodes [15, 16] The elimination tree represents the data dependency between columns during factorization, and a supernode comprised a set of consecutive columns with nested sparsity structure to enable the use of cache-efficient techniques. The relationship between the separators and their supernodes in the tree is illustrated in Figure 1.1; the separators recursively partition the domain to form supernodes in a tree structure.

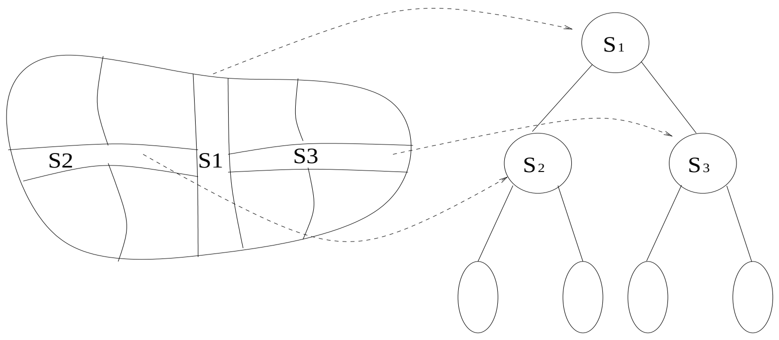

Fig. 1.1. Two levels of separators applied to a domain and corresponding levels in the supernodal tree.

During our ICT factorization some nonzero elements are dropped based on the drop threshold condition. Consequently, the column dependencies and the structure for supernodes derived from the coefficient matrix are not exact. However, these structures are used to manage the implementation and allow flexible dropping schemes to compute $\hat{L}$ for a range of fill to meet a variety of preconditioning needs. In addition, we utilize efficient dense matrix kernels [12] to perform the factorization at a supernode before applying drop threshold condition. During our ICT factorization some nonzero elements are dropped based on the drop threshold condition. Consequently, the column dependencies and the structure for supernodes derived from the coefficient matrix are not exact. However, these structures are used to manage the implementation and allow flexible dropping schemes to compute $\hat{L}$ for a range of fill to meet a variety of preconditioning needs. In addition, we utilize efficient dense matrix kernels [12] to perform the factorization at a supernode before applying drop threshold condition.

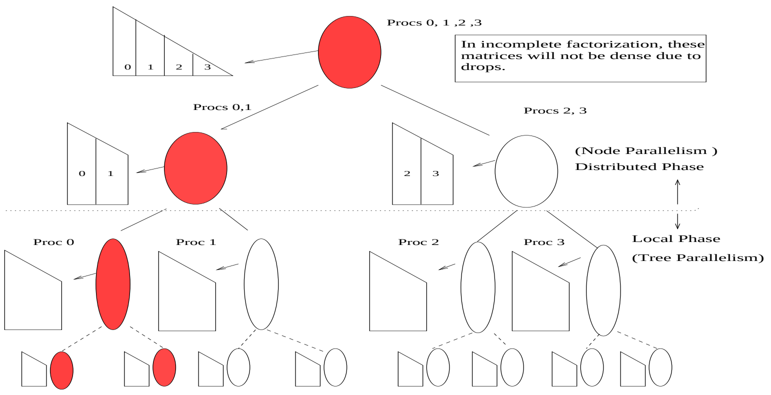

Fig. 1.2. The structure of the supernodal tree and submatrices associated with each node; a 4-processor computation is shown.

The parallel implementation is based on the supernodal tree. Individual subtrees rooted at vertices at $\approx \log P$ levels are computed independently on each processor; these correspond to local sub-domains. At level above the local subtrees, each supernode corresponds to a distributed separator and it is processed by multiple processors using data-parallel dense/blocked operations. Figure 1.2 illustrates the computational scheme using four processors. The incomplete factorization proceeds up the tree; local computations at processor 0 are highlighted followed by distributed processing with processors 0,1 and then at the root with all processors. The forward solution proceeds bottom-up on the tree followed by a top-down backward solution.

Parallel triangular solution using substitution becomes a performance bottleneck at each distributed supernode because the algorithm is latency-bound. Triangular solution at a supernode a involves:

$$\begin{bmatrix} {L_{11}}^a \\ {L_{21}}^a \end{bmatrix} \begin{bmatrix} {x_1}^a \end{bmatrix} = \begin{bmatrix} {b_1}^a \\ {b_2}^a \end{bmatrix}.$$

The submatrices in the expression above are incomplete forms of the factor submatrix at supernode a. Parallel substitution is performed to obtain of ${x_1}^a$ using ${L_{11}}^a$ and ${b_1}^a$; next ${b_2}^a$ is updated as $b_2 - {L_{21}}^a {x_1}^a$ and used in computations at the ancestor supernode of a.

The SI scheme [4, 5] includes a parallel matrix inversion of ${L_{11}}^a$ for each distributed supernode a. Subsequently, parallel substitution is replaced by sparse matrix vector multiplication ${x_1}^a \leftarrow {L_{11}^a}^{-1}{b_1}^a$. The scheme incurs the extra computational cost of inversion, but the improvements in applying the preconditioner are substantial [5].

1.3 ICT with Selective Sparse Approximate Inversion

Although the ICT-SI scheme described earlier achieves scalable application of the preconditioner, the construction of the preconditioner is relatively expensive. One of the reasons is that explicit inversion of the diagonal portion of sparse supernodal matrix causes fill-in which must then be reduced by applying a drop-condition. Consequently, even if a very sparse preconditioner is required, the cost of the construction is close to that for a true sparse factorization in a sparse direct solver. To alleviate this problem, we use a sparse approximate inverse schemes to compute an approximation of ${L_{11}}^{a-1}$ at a distributed supernode a. For model sparse matrices from five-point finite-difference schemes using regular grids in two and three dimensions, we can show analytically that the the arithmetic and communication costs for constructing the preconditioner using ICT-SSAI is lower in the order of magnitude sense than for ICT and ICT-SI [6].

We now provide some preliminary results on the performance of parallel ICT-SI and ICT-SSAI and comparisons with Block SSOR, Block Jacobi [17], level-0 incomplete Cholesky (IC(0)) [18], and the sparse approximate inverse preconditioner (Parasails) [8]. We also report on the performance of DSC-PACK [19], a parallel sparse direct solver as another point of comparison.

Our experiments were performed on a cluster with Intel Xeon processors and a Myrinet interconnect using the CG implementation in the PETSc package [17]. We terminate the CG iteration when relative residual is smaller than 10^{-8}. For ICT-SI and ICT-SSAI, we applied drop tolerance thresholds ranging from 0.002 to 0.02 and diagonal shifts in the range 0.01 to 0.05 if required. For the sparse approximate inversion in the ICT-SSAI scheme, we used the default parameters in Parasails excluding the parameter for dropping elements after approximate inversion; the dropping parameter is the same as the drop tolerance condition of the ICT-SI and ICT-SSAI. We report the best performing instance of ICT-SI and ICT-SSAI with respect to the time for the entire solution including preconditioner construction and PCG iterations.

We use three large sparse matrices from finite-element and finite-difference applications described in Table 1.1. We used 1–16 processors for the first two smaller matrices and 4–64 processors to solve the largest matrix, **augustus7**.

The performance of all methods for the first two matrices (**cfd2** and **engine**) is summarized in Table 1.2. The best value of each measure (Time for solution, and number of iterations) is shown in bold. The parallel performance for **augustus7** is shown in detail in Figure 1.3 when the number of processors is increased from 1–16.

Matrix	N	$\|A\|$	Description
cfd2	123,440	3,087,898	CFD: pressure matrix
engine	143,571	2,424,822	Engine head, linear tetrahedral elements
augustus7	1,060,864	9,313,876	Diffusion equation from 3D mesh

Table 1.1. Description of sparse matrices. N is the matrix dimension, $|A|$ is the number of nonzeroes in the matrix.

ICT-SI leads to the least number of iterations and ICT-SSAI requires a slightly larger number of iterations. The increase in the number of iterations is a consequence of selectively using sparse approximate inversion. Observe that Parasails leads to a higher number of iterations than ICT-SI and ICT-SSAI but fewer iterations than IC(0) and simpler preconditioners like SSOR and Jacobi. Preconditioner construction is less expensive in ICT-SSAI than in ICT-SI and this difference results in reduced total time on the larger number of processors; ICT-SSAI becomes the fastest method on 16 processors. We expect the benefits of ICT-SSAI to be more significant in applications where the preconditioner construction costs can be amortized over solutions for a sequence of right-hand-side vectors.

Method	Number of Processors										Mem
	1		2		4		8		16		
	Time	Its	Time	Its	Time	Its	Time	Its	Time	Its	
cfd2	Matrix size: 123,440 Nonzeroes: 3,087,898										
SSOR		NC		NC		NC		NC		NC	1.0
Jacobi		NC		NC		NC		NC		NC	1.72
IC(0)		NC		NC		NC		NC		NC	2.0
Parasails	192.8	782	115.7	747	61.35	776	31.70	777	17.50	776	3.89
ICT-SI	**88.73**	**451**	**45.24**	**461**	39.08	**463**	24.21	**464**	21.80	**471**	4.17
ICT-SSAI	**88.73**	**451**	50.83	498	**29.88**	554	**18.79**	569	**12.22**	583	3.96
engine	Matrix size: 143,571 Nonzeroes: 2,424,822										
SSOR	170.4	1153		NC		NC		NC		NC	1.0
Jacobi	97.50	994	76.53	1436		NC		NC		NC	1.75
IC(0)		NC		NC		NC		NC		NC	2.0
Parasails	130.4	760	103.3	760	77.35	761	62.29	762	46.09	761	3.89
ICT-SI	**48.12**	**282**	**35.24**	**252**	36.08	**287**	47.21	**306**	37.80	**308**	2.56
ICT-SSAI	**48.12**	**282**	36.69	336	**24.87**	356	**19.96**	356	16.64	387	2.36

Table 1.2. Performance of parallel preconditioners on three sparse matrix problems using 1 – 16 processors with the best instances shown in bold. The column labeled 'Time' is the total time (in seconds). The column labeled 'Its' is number of CG iterations; NC indicates that convergence was not achieved after 1,500 iterations. The column labeled 'MEM' contains the memory usage as a multiple of the space for the coefficient matrix.

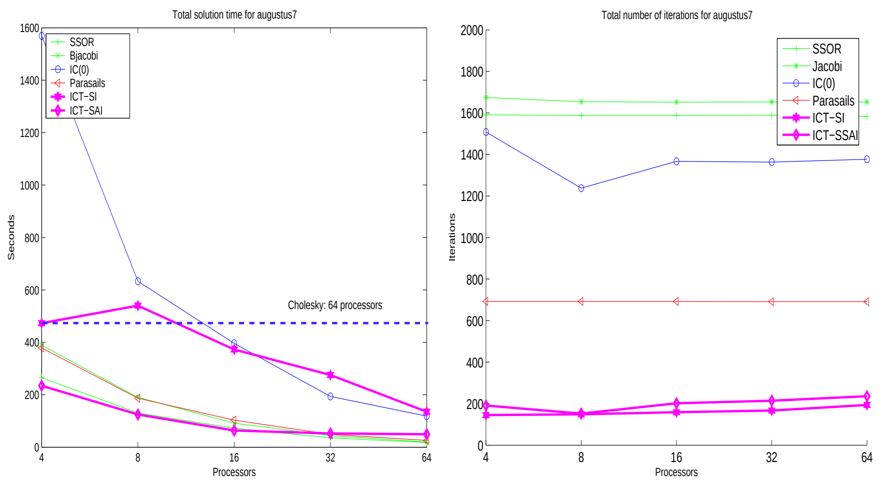

Fig. 1.3. Time to solve **augustus7** (left) and and the number of iterations (right).

1.4 Conclusions

We have developed a parallel hybrid ICT-SSAI scheme which can potentially meet the preconditioning needs of sparse systems from complex applications. Initial empirical results are indeed encouraging and we are currently collaborating with Barry Smith to further test and refine our methods [20]. Our results indicate that ICT-SSAI successfully trades a slight decrease in the quality of the preconditioner for faster and more scalable preconditioner construction. We expect our methods to serve as a scalable limited memory scheme for applications that have traditionally relied on a direct solver for robust sparse linear system solution.

Acknowledgments

We gratefully acknowledge several useful discussions with Edmond Chow and Barry Smith. We thank Michele Benzi for providing sparse matrices from his test collection. We also thank the Mathematics and Computer Sciences Division at the Argonne National Laboratory for allowing us to use their network of workstations.

References

1. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. National Bureau Standard J. Res. **49** (1952) 409–436

2. Saad, Y.: Iterative method for sparse linear systems. second edn. SIAM, Philadelphia, PA (2003)
3. Zlatev, Z.: Use of iterative refinement in the solution of sparse linear systems. SIAM J. Numer. Anal. **19** (1982) 381–399
4. Raghavan, P.: Efficient parallel triangular solution using selective inversion. Parallel Processing Letters **9** (1998) 29–40
5. Raghavan, P., Teranishi, K., Ng, E.G.: A latency tolerant hybrid sparse solver using incomplete cholesky factorization. Numer. Linear Algebra Appl. **10** (2003) 541–560
6. Teranishi, K.: Scalable Hybrid Sprase Lienar Solvers. PhD thesis, Department of Computer Science and Engineering, The Pennsylvania State University (2004)
7. Benzi, M., Meyer, C.D., Tůma, M.: A sparse approximate inverse preconditioner for the conjugate gradient method. SIAM Journal on Scientific Computing **17** (1996) 1135–1149
8. Chow, E.: Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns. Int. J. High Perf. Comput. Apps. **15** (2001) 56–74
9. Grote, M.J., Huckle, T.: Parallel preconditioning with sparse approximate inverses. SIAM Journal on Scientific Computing **18** (1997) 838–853
10. Kolotilina, L.Y., Yeremin, A.Y.: Factorized sparse approximate inverse preconditionings. I. Theory. SIAM Journal on Matrix Analysis and Applications **14** (1993) 45–58
11. Benzi, M., Tůma, M.: A comparative study of sparse approximate inverse preconditioners. Applied Numerical Mathematics: Transactions of IMACS **30** (1999) 305–340
12. Dongarra, J., Croz, J.D., Duff, I.S., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Software **16** (1990) 1–17
13. George, A., Liu, J.W.H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs, NJ, USA (1981)
14. Liu, J.W.H.: The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications **11** (1990) 134–172
15. Liu, J.W.H., Ng, E., Peyton, B.W.: On finding supernods for sparse matrix ccomputation. SIAM J. Matrix Anl. Appl. **14** (1993) 242–252
16. Ng, E.G., Peyton, B.W.: Block sparse cholesky algorithm on advanced uniprocessor computers. SIAM J. Sci. Comput. **14** (1993) 1034–1056
17. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.1, Argonne National Laboratories (2002)
18. Jones, M., Plassmann, P.: The efficient parallel iterative solution of large sparse linear systems. In George, A., Gilbert, J.R., Liu, J.W.H., eds.: Graph Theory and Sparse Matrix Computations. Volume 56 of IMA. Springer-Verlag (1994) 229–245.
19. Raghavan, P.: DSCPACK: Domain-separator codes for solving sparse linear systems. Technical Report CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University (2002)
20. Teranishi, K., Raghavan, P., Smith, B.F.: Tree-based parallel drop threshold incomplete cholesky preconditioning using matrix inversion heuristics. In: 2005 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications. (2005) in preparation.