Enhancing Scalability of Sparse Direct Methods

X.S. Li[1], J. Demmel[2], L. Grigori[3], M. Gu[4], J. Xia[5], S. Jardin[6], C. Sovinec[7], L.-Q. Lee[8]

[1]One Cyclotron Road, Lawrence Berkeley National Laboratory, MS 50F-1650, Berkeley, CA 94720.
[2]Computer Science Division, University of California at Berkeley, Berkeley CA 94720-1776.
[3]INRIA Rennes, Campus Universitaire Beaulieu, Rennes, 35042, France.
[4]Mathematics Department, University of California at Berkeley, Berkeley CA 94720-3840.
[5]Department of Mathematics, University of California, Los Angeles, Los Angeles, CA 90095.
[6]Princeton Plasma Physics Laboratory, P.O. Box 451, Princeton, NJ 08543.
[7]Engineering Physics Department, University of Wisconsin, Madison, WI 53706.
[8]Stanford Linear Accelerator Center, 2575 Sand Hill Road, Mail Stop 27, Menlo Park, CA 94025.

E-mail: `xsli@lbl.gov`

Abstract. TOPS is providing high-performance, scalable sparse direct solvers, which have had significant impacts on the SciDAC applications, including fusion simulation (CEMM), accelerator modeling (COMPASS), as well as many other mission-critical applications in DOE and elsewhere. Our recent developments have been focusing on new techniques to overcome scalability bottleneck of direct methods, in both time and memory. These include parallelizing symbolic analysis phase and developing linear-complexity sparse factorization methods. The new techniques will make sparse direct methods more widely usable in large 3D simulations on highly-parallel petascale computers.

1. The SciDAC applications

1.1. Fusion energy research

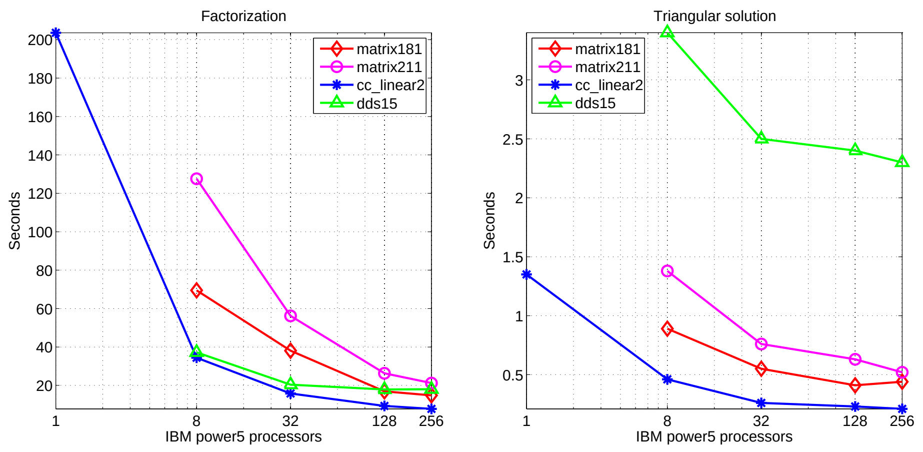
The Center for Extended Magnetohydrodynamic Modeling (CEMM) [1] is developing simulation codes for studying the nonlinear macroscopic dynamics of MHD-like phenomena in fusion plasmas, and address critical issues facing burning plasma experiments such as ITER. Their code suite includes M3D-C1, NIMROD. The PDEs include many more physical effects than the standard MHD equations; they involve large, multiple time-scales, and are very stiff temporally, therefore requiring implicit methods. The linear systems are extremely ill-conditioned, and 50-90% of the execution time is spent in linear solvers. TOPS sparse direct solver SuperLU has played significant roles in both codes. For the large 3D matrix-free formulation in NIMROD, SuperLU is also used as an effective preconditioner for the global GMRES solver.

1.2. Accelerator design

The Community Petascale Project for Accelerator Science and Simulation (COMPASS) [2] is developing parallel simulation tools with integrated capabilities in beam dynamics, electromagnetics, and advanced accelerator concept modeling for accelerator design, analysis, and discovery. Their code suite includes Omega3P eigenanalysis. The eigen computations for cavity mode frequencies and field vectors are on the critical path of the shape optimization cycles. These need to be done repeatedly, accurately, and quickly. Another challenge is that a large number of small nonzero eigenvalues, which are tightly clustered, is desired. The matrix

Name	Codes	Type	Order (N)	nnz(A)/N	Fill-ratio
matrix181	M3D-C1	Real	589,698	161	9.3
matrix211	M3D-C1	Real	801,378	161	9.3
cc_linear2	NIMROD	Complex	259,203	109	7.5
dds15	Omega3P	Real	834,575	16	40.2

Table 1. Characteristics of the sample matrices. The sparsity is measured as average number of nonzeros per row (i.e., nnz(A)/N), and the Fill-ratio shows the ratio of number of nonzeros in L+U over that in A. Here, MeTiS is used to reorder the equations to reduce fill.



(a) Factorization time. (b) Triangular solution time.

Figure 1. SuperLU runtime (seconds) for the linear systems from the SciDAC applications. This was done on the IBM Power 5 machine at NERSC. The factorization reached **161 Gflops/s** flop rate for matrix211.

dimensions can be tens to hundreds of millions. The main method used is shift-invert Lanczos, for which the shifted linear systems are solved with a combination of direct and iterative methods.

1.3. SuperLU efficiency with these applications

SuperLU [6] is a leading scalable solver for sparse linear systems using direct methods, of which the development is mainly funded through the TOPS SciDAC project (led by David Keyes) [7].

Table 1 shows the characteristics of a few typical matrices taken from these simulation codes. Figure 1 shows the parallel runtime of the two important phases of SuperLU: factorization and triangular solution. The experiments were performed on an IBM Power 5 parallel machine at NERSC. In strong scaling sense, the factorization routine scales very well, although performance varies with applications. The triangular solution takes very small fraction of the total time. On the other hand, it does not scale as well as factorization, mainly due to large communication to computation ratio and higher degree of sequential dependencies. One of our future tasks is to improve scalability of this phase, since in these application codes, the triangular solution often needs to be done several times with respect to one factorization.

In the last year or so, we have been focusing on developing new algorithms to enhance scalability of our direct solvers. The new results are summarized in the next two sections.

2. Improving memory scalability of SuperLU – parallelizing symbolic factorization

Symbolic factorization is a phase to determine the nonzero locations of the L. U factors. In most parallel sparse direct solvers, this phase is performed in serial, with matrix A being available on

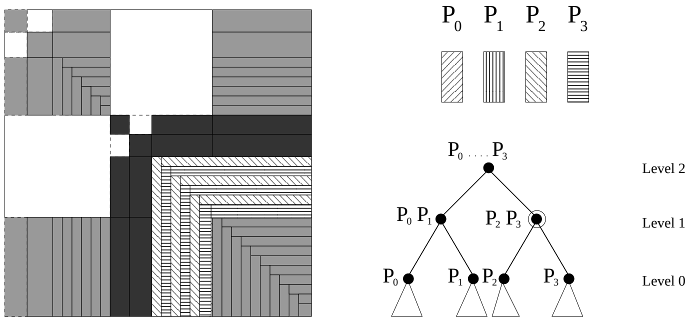


Figure 2. Subtree-to-subgroup processor to matrix mapping. The figure on the right illustrates the separator tree.

matrix181		P=8	P=256
Fill (millions)	Sequential	888.1	888.1
	Parallel	1094.2	1445.3
Symbolic	Sequential	365.5	365.5
	Parallel	18.0	8.1
	Ratio Seq/Par	20	45
Entire solver	Old	1445.1	377.2
	New	1262.8	84.3

dds15		P = 8	P = 256
Fill (millions)	Sequential	526.6	526.6
	Parallel	528.9	583.7
Symbolic	Sequential	295.8	295.8
	Parallel	27.2	10.8
	Ratio Seq/Par	11	27
Entire solver	Old	1061.9	341.3
	New	817.0	113.1

Table 2. Memory usage (Megabytes) (maximum among all processors). "Old" or "New" solver means using the serial or parallel symbolic factorization algorithm, respectively.

one node . The reasons for not doing it in parallel are: 1) the algorithm involves only integer operations and is very fast in practice—the complexity is larger than the nonzero count of $L+U$, but much smaller than the flops for LU decomposition, and 2) it is very difficult to design an efficient parallel algorithm, which is partly due to sequentiality of the tasks—computation of the i-th column/row depends on results of the previous columns/rows, and partly due to lower computation-to-communication ratio.

Obviously, with this sequential part, the solver's scalability will suffer by the Amdahl's law. Another reason we have to do this in parallel is that when the simulations increase in size, A itself will not fit in one node. Motivated mainly by memory scalability, we have designed and implemented a an efficient parallel algorithm for this phase. Our approach is based on a graph partitioning for reordering/partitioning the input matrix. Specifically, we apply ParMeTiS [5] to the structure of $A + A^T$. We exploit parallelism given by this partition (coarse level with the separator tree) and by a block cyclic distribution (fine level at each separator). We also identify dense separators, dense columns of L and rows of U to decrease the algorithmic complexity, see details in [4]. The processors-to-matrix assignment is based on a top-down subtree-to-subgroup mapping scheme, as shown in Figure 2.

Table 2 compares, for two matrices, the memory water marks before and after parallelizing the symbolic factorization phase. Considering only the symbolic phase itself, the memory reduction is dramatic, up to a factor of 45 for matrix181 and 27 for dds15. When all the other phases are included (entire solver), the new solver reduces the memory usage by nearly 5-fold for matrix 181. Now, the numerical phase consumes more memory.

Table 3 shows the runtime of the parallel symbolic algorithm, and that reasonable speedups are obtained.

One remark worth noting is that the fill quality of ParMeTiS is worsening with increasing processor count. This remains an open problem for future research.

matrix181		P = 8	P = 256
Symbolic	Sequential	6.8	6.8
	Parallel	2.6	2.7
Entire solver	Old	84.7	26.6
	New	159.2	26.5

dds15		P = 8	P = 256
Symbolic	Sequential	4.6	4.6
	Parallel	1.6	0.5
Entire solver	Old	64.1	43.2
	New	66.3	31.4

Table 3. Parallel runtime (seconds). "Old" or "New" solver means using the serial or parallel symbolic factorization algorithm, respectively.

3. Linear-complexity sparse factorization

As is well known, sparse Gaussian elimination has super-linear time complexity. For example, for a Laplacian-type of PDE model problem with 2D $n \times n$ mesh or 3D $n \times n \times n$ mesh, using nested dissection numbering, the operation counts are $O(n^3)$ and $O(n^6)$, respectively. This was shown to be optimal when performing exact elimination. However, we have observed that by exploiting *numerical low rankness* in Schur complements, we can design a nearly-linear time, and sufficiently accurate factorization algorithm.

The idea comes from structured matrix computations. Specifically, we use *semi-separable* matrices in our study. For example, a semi-separable matrix with 4×4 blocks has the following structure:

$$A \simeq \begin{pmatrix} D_1 & U_1V_2^T & U_1W_2V_3^T & U_1W_2W_3V_4^T \\ V_2U_1^T & D_2 & U_2V_3^T & U_2W_3V_4^T \\ V_3W_2^TU_1^T & V_3U_2^T & D_3 & U_3V_4^T \\ V_4W_3^TW_2^TU_1^T & V_4W_3U_2^T & V_4U_3^T & D_4 \end{pmatrix}$$

where,

- The first and second off-diagonal blocks of A are

$$U_1 \begin{pmatrix} V_2^T & W_2V_3^T & W_2W_3V_4^T \end{pmatrix} \quad \text{and} \quad \begin{matrix} U_1W_2 \\ U_2 \end{matrix} \begin{pmatrix} V_3^T & W_3V_4^T \end{pmatrix}.$$

- For general $i < j$, the (i, j) block-entry is $U_iW_{i+1}W_{i+2}\cdots W_{j-1}V_j^T$.
- Every upper off-diagonal block has the form

$$\begin{pmatrix} U_1W_2W_3\cdots W_i \\ U_2W_3\cdots W_i \\ \vdots \\ U_i \end{pmatrix} \begin{pmatrix} V_{i+1}^T & W_{i+1}V_{i+2}^T & \cdots & W_{i+1}W_{i+2}\cdots W_{N-1}V_N^T \end{pmatrix}.$$

D, U, W and V matrices have dimensions $k \times k$. A is $n \times n$ with $n = Nk$ and uses $O(nk)$ memory, which is very economical for $k \ll n$. The examples of semi-separable matrix include banded matrices and their inverses. This representation can be numerically constructed.

We can devise many fast algorithms based on this compressed representation. For example, Figure 3 compares the fast semi-separable Cholesky factorization (SS-Cholesky) to the standard Cholesky in LAPACK (DPOTRF), with the rank k chosen to be 16 and 64. For large matrices, we see orders of magnitude speedups.

This SS-Cholesky kernel can be used in a sparse factorization. Consider using a nested dissection ordering for the discretization mesh, then the submatrix corresponding to the separator node is essentially dense during Schur complement updates. Furthermore, the maximum off-diagonal rank of those matrices are small and nearly constant. Based on this observation, we devised a new multifrontal factorization scheme, which compresses the frontal matrix in semi-separable form, and performs SS-Cholesky on it. The superfast multifrontal method works as follows [3]:

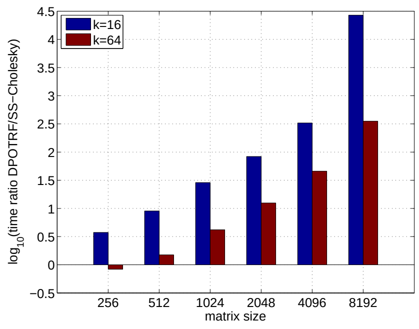


Figure 3. Speedup of fast semi-separable Cholesky factorization over DPOTRF.

Mesh	255×255	511×511	1023×1023	2047×2047	4095×4095
MF	0.22	1.4	8.7	55.8	383.6
fast MF	0.24	1.3	6.1	27.0	113.4

Table 4. Superfast solver runtime (seconds) on a SGI Altix.

- Perform traditional Cholesky factorization at the bottom levels of the separator tree.
- At the higher levels, convert the fill-in submatrices into semi-separable structures, and perform structured semi-separable Cholesky factorizations at these levels.

The overall operation count is reduced from $O(n^3)$ operations to $O(n^2k)$ for 2D problems, where k is tolerance-dependent constant. Table 3 compares the performance of traditional multifrontal (MF) solver to that of the superfast multifrontal (fast MF) solver. As can be seen, with increasing problem size, the superfast solver can be more than three times faster than the traditional one.

Our fast solver framework can be extended to construct effective and robust preconditioners for more general, non-elliptic PDEs.

4. Acknowledgments

This work was partly supported by the US Department of Energy under contract No. DE-AC03-6SF00098.

5. References

[1] Center for Extended MHD Modeling (CEMM). URL: http://w3.pppl.gov/cemm/.
[2] The Community Petascale Project for Accelerator Science and Simulation (COMPASS). URL: http://www.scidac.gov/.
[3] S. Chandrasekaran, M. Gu, X.S. Li, and J. Xia. Superfast multifrontal method for structured linear systems of equations. Technical Report LBNL-62898, Lawrence Berkeley National Laboratory, June 2007.
[4] Laura Grigori, James W. Demmel, and Xiaoye S. Li. Parallel symbolic factorization for sparse LU with static pivoting. *SIAM J. Scientific Computing*, 2007. To appear.
[5] G. Karypis, K. Schloegel, and V. Kumar. ParMETIS*: Parallel Graph Partitioning and Sparse Matrix Ordering Library – Version 3.1.* University of Minnesota, August 2003. `http://www-users.cs.umn.edu/~karypis/metis/parmetis/`.
[6] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Mathematical Software*, 31(3):302–325, September 2005. `http://crd.lbl.gov/~xiaoye/LBNL-53848.pdf`.
[7] Towards Optimal Petascale Simulations, URL: http://www-unix.mcs.anl.gov/scidac-tops/.