U. S. Department of Energy
Office of Science
Office of Advanced Scientific Computing Research

Exploratory Research for Extreme-Scale Science

Final Report:
Massive Asynchronous Parallelization of Sparse Matrix Factorizations

Period Covered: Sept. 1, 2014 – August 31, 2017

Date of Report: January 9, 2018

Award Number	DE-SC0012538
Applicant Institution and Address	Georgia Tech Research Corporation Office of Sponsored Programs 505 Tenth Street, NW Atlanta, GA 30332-0420
Lead Principal Investigator	Edmond Chow Associate Professor School of Computational Science and Engineering echow@cc.gatech.edu Tel.: (404) 894-3086
DOE Office of Science Program Office	Office of Advanced Scientific Computing Research
DOE Office of Science Technical Contact	Dr. Steven L. Lee

1 Introduction

Solving sparse problems is at the core of many DOE computational science applications. We focus on the challenge of developing sparse algorithms that can fully exploit the parallelism in extreme-scale computing systems, in particular systems with massive numbers of cores per node. Our approach is to express a sparse matrix factorization as a large number of bilinear constraint equations, and then solving these equations via an asynchronous iterative method. The unknowns in these equations are the matrix entries of the factorization that is desired.

For an incomplete LU (ILU) factorization, $A \approx LU$, define the sparsity pattern S to be the set of matrix locations where nonzeros are allowed, that is, $(i,j) \in S$ if l_{ij} in matrix L or u_{ij} in matrix U is permitted to be nonzero. The constraint equations are

$$(LU)_{ij} = a_{ij}, \quad (i,j) \in S \tag{1}$$

where $(LU)_{ij}$ denotes the (i,j) entry of the product of the computed factors L and U, and a_{ij} is the corresponding entry in matrix A.

To solve these constraint equations, we write

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right),\ i > j \qquad u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj},\ i \le j \tag{2}$$

which has the form $x = G(x)$, where x is a vector containing the unknowns l_{ij} and u_{ij} for $(i,j) \in S$. The equations are then solved by using the fixed-point iteration $x^{(p+1)} = G(x^{(p)})$, for $p = 0,1,\ldots$, starting with some initial $x^{(0)}$. Sweeps of the fixed-point iteration are highly parallel. In addition, the sweeps can be performed *asynchronously*, i.e., each thread uses the latest available values of the unknowns, rather than waiting to synchronize with other threads.

2 Convergence Results

A critical question in our research is the convergence of the nonlinear fixed-point iterations for the solution of the constraint equations. In particular, the question is whether the fixed-point mapping G is a contraction mapping for a given matrix, A. We have derived bounds for the 1-norm of the Jacobian, G', of the mapping as a function of the matrix A and the current approximation of x. We have also derived exact expressions when A is a finite difference matrix. These results show that indeed the mapping is a contraction in important cases, although it cannot be guaranteed for arbitrary matrices. More more details, please see [7].

3 Threshold-based ILU factorization

The ILU algorithm described above assumes that the sparsity pattern S is fixed before the factorization computation begins. In many cases, such as for anisotropic problems, a good choice of S is not known beforehand. Conventionally, this is addressed by using a threshold-based ILU factorization, where elements in S are chosen dynamically by dropping nonzeros during the factorization that are smaller than a threshold. However, because the sparsity pattern is not fixed beforehand, parallelization using conventional techniques such as level scheduling is impossible. There are no known techniques for fine-grained parallelization of threshold-based ILU factorizations (parallelization through coarse-grained domain decomposition is possible, where threshold-based ILU is performed sequentially on each subdomain).

To address the above issues, we have developed a new algorithm for computing an incomplete factorization that takes into account the values of the nonzeros in the coefficient matrix A, i.e., one that chooses the sparsity pattern S dynamically, to try to minimize

$$\|A - LU\|_F$$

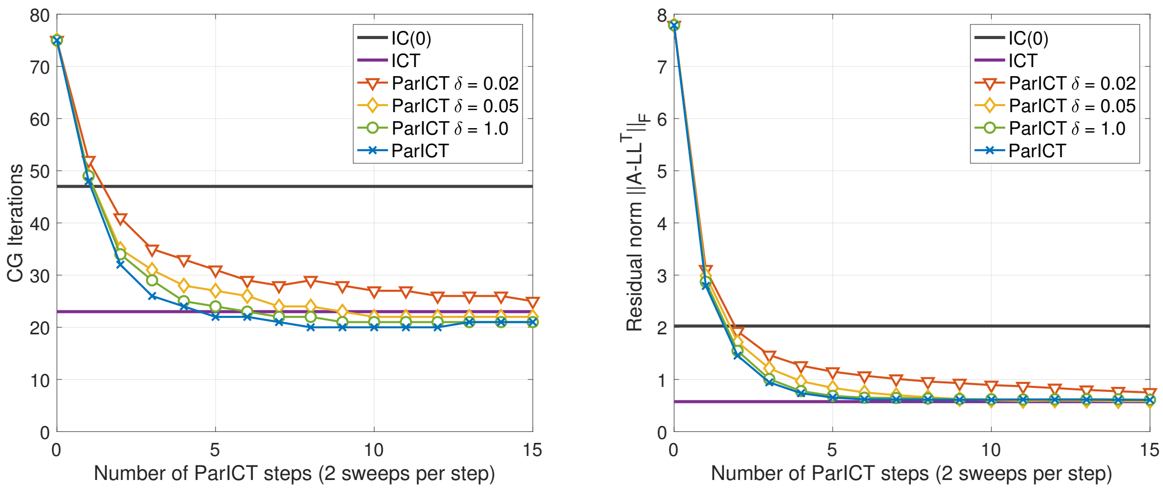

Figure 1: PCG iteration count and ILU residual norm for the ParICT preconditioner computed with different number of steps for the ani3 matrix. The results for IC(0) and classical ICT are also shown.

with the constraint that L and U are sparse. The idea is to interleave a parallel fixed-point iteration that approximates an incomplete factorization for a given nonzero pattern with a procedure that adaptively changes the pattern. Nonzeros are both added and removed from the nonzero pattern. Thus the thresholding strategy is different from existing threshold ILU techniques and generates different nonzero patterns.

While our initial goal was to match the preconditioner quality of threshold ILU factorizations, the new algorithm can generate factorizations that are better, for the same number of nonzeros in the factorizations. This is because by *removing* nonzeros, we can exclude certain nonzeros that must be retained in existing techniques. In particular, a given fill-in element may be the result of an earlier fill-in element. In existing threshold ILU techniques, the earlier fill-in element must be included in the pattern if the later fill-in element is included. This is not the case in our new strategy. Test results show the important consequence that it is possible with our new algorithm to compute threshold-based incomplete Cholesky factorizations with a given amount of nonzeros *that are impossible to compute via classical threshold-based algorithms* because these classical algorithms break down, i.e., encounter negative pivots.

Figure 1(left) shows the PCG iteration counts using the new preconditioner, called ICT in the incomplete Cholesky case. The number of solver iterations using IC(0) and classical ICT are also shown, as are variants using a parameter δ. Classical ICT preconditioning gives about half the number of iterations as IC(0) preconditioning. Compared to classical ICT, ParICT preconditioning gives about the same number of iterations after 4 or 5 steps. It is also observed that ParICT can be slightly better than classical ICT when many steps are taken. This is possible because the sparsity patterns for ICT and for ParICT are not guaranteed to be the same.

Figure 1(right) shows the incomplete factorization residual norm $||A - LL^T||_F$ where L is the computed incomplete Cholesky factor. The norm is larger for IC(0) than for classical ICT. The norm for ParICT after many steps is very close to that for classical ICT. The ParICT norm does not appear to smaller than the classical ICT norm when the ParICT preconditioner gives fewer PCG iterations than classical ICT. The incomplete factorization residual norm is not sensitive enough to identify differences between preconditioners that lead to small but noticeable differences in PCG iteration count.

For thorough results on a suite of test problems running on Intel Xeon Phi Knights Landing processors, please see [1].

4 Solving Sequences of Linear Systems

When solving a sequence of related linear systems by iterative methods, it is common to reuse the preconditioner for several systems, and then to recompute the preconditioner when the matrix has changed significantly. Rather than recomputing the preconditioner from scratch, it is potentially more efficient to update the previous preconditioner. Unfortunately, it is not always known how to update a preconditioner, for example, when the preconditioner is an incomplete factorization. Our iterative algorithm for computing incomplete factorizations, however, is able to exploit an initial guess, unlike existing algorithms for incomplete factorizations. By treating a previous factorization as an initial guess to this algorithm, an incomplete factorization may thus be updated. We use a sequence of problems from model order reduction. Experimental results using an optimized GPU implementation show that updating a previous factorization can be inexpensive and effective, making solving sequences of linear systems a potential niche problem for the iterative incomplete factorization algorithm.

Figure 2 shows a typical result for a sequence of 34 linear systems, each having a different "shift index." Exact and iterative IC factorizations were computed every ℓ systems, and were used to precondition ℓ consecutive systems. The figure shows the case with $\ell = 5$. The iterative IC factorizations were computed using a single sweep applied to either the standard or previous factorization initial guesses (SIG or PFIG). In the case of PFIG, larger ℓ means that the factorization used as initial guess for the current factorization is more "stale" or numerically farther. It could be expected that for larger ℓ, the PFIG updating strategy may be worse than the SIG strategy without updating.

A typical sawtooth pattern can be observed, as the PCG iteration count degrades as a factorization is reused, and then improves again when it is recomputed. From the graphs on the left side of Figure 2, fewer total number of PCG iterations are needed when factorizations are updated using PFIG, compared to when factorizations are computed from scratch using SIG. When the recomputation interval is larger (not shown) the overall cost decreases (due to requiring fewer total number of sweeps) but the benefit of PFIG also decreases compared to SIG. However, even for large values of ℓ, the PFIG strategy is better than the SIG strategy. Additional results can be found in the journal article [4].

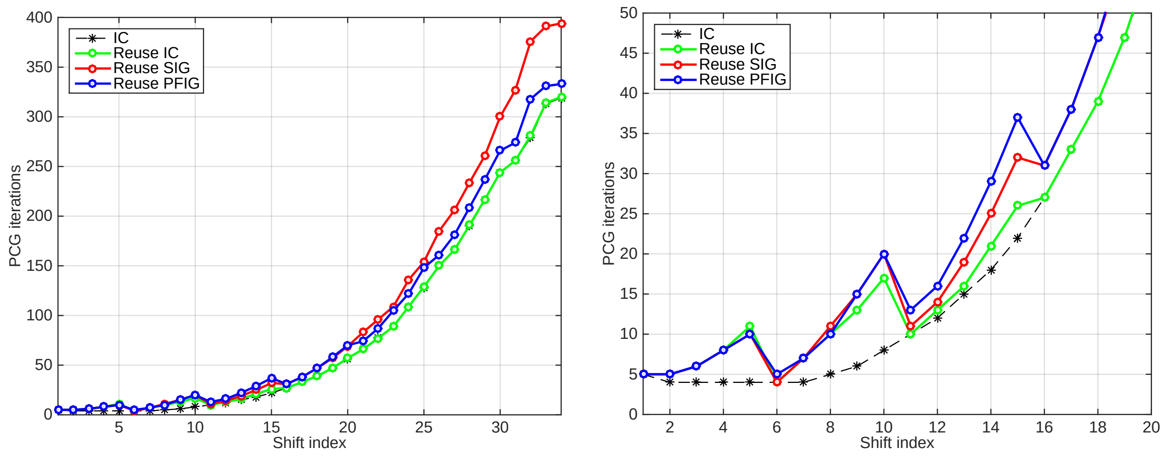

Figure 2: PCG iteration counts when each factorization is computed every 5 systems and reused for 5 consecutive systems. Right side graph zoom in on the left side graph. A single sweep was used in the SIG and PFIG cases. "IC" denotes the case where the exact IC factorization is used for every system.

5 Parallel Sparse Triangular Solves

Sparse triangular solvers are typically parallelized using level-scheduling techniques, but parallel efficiency is poor on high-throughput architectures like GPUs. We have studied an iterative approach, which is highly parallel, for solving sparse triangular systems when an approximation is suitable. This approach will not work for all problems, but can be successful for sparse triangular matrices arising from incomplete factorizations, where an approximate solution is acceptable.

A triangular system $Rx = b$ may be solved approximately by a fixed small number of steps of a Jacobi iteration

$$x_{k+1} = (I - D^{-1}R)x_k + D^{-1}b$$

where D is the matrix consisting of the diagonal of R. Thus the iteration matrix $G = I - D^{-1}R$ is strictly triangular (the diagonal is all zeros) and has spectral radius 0. This is similar to the ILU case, providing trivial asymptotic convergence. Convergence within a small number of iterations is desired, however, and this depends on the norm or non-normality of G.

To improve robustness and convergence, we need to decrease the non-normality of the iteration matrix. This can be done simply by using a block Jacobi iteration, rather than a scalar Jacobi iteration. Our experiments have shown that this can dramatically reduce the number of sweeps required for the approximate triangular solves to achieve a given level of accuracy. Blocks should be chosen such that the diagonal blocks have large norm, which can be enhanced by permuting rows and columns.

A simple blocking technique that we are using is the following based on the idea of aggregation. Using the graph theoretic description of sparse matrices, each node of the sparse matrix is initially in its own set. The technique considers all edges by decreasing edge weight. Each edge, considered in order, aggregates two sets, unless the size of the new set exceeds a user-defined maximum. After all edges have been considered the merged sets define the rows and columns that are grouped to define a block.

Reordering to create a heavily weighted block diagonal part of the matrix, however, changes the ordering of the matrix which can adversely affect the accuracy of an incomplete factorization. We have further experimented with applying a block RCM ordering to this block matrix, as well as using a simple blocking of the matrix in the original ordering, for cases where the large entries are already near the diagonal.

Experimental results are shown in [2]. Performance results using GPUs (but not not using blocking) are in [8]. This latter paper also empirically studies the effect of using asynchronous iterations for the triangular solves.

6 Software

We have designed a GPU implementation of the asynchronous iterative algorithm for computing incomplete factorizations. Our GPU implementation considers several non-traditional techniques that can be important for asynchronous algorithms to optimize convergence and data locality. These techniques include controlling the order in which variables are updated by controlling the order of execution of thread blocks, taking advantage of cache reuse between thread blocks, and managing the amount of parallelism to control the convergence of the algorithm. These techniques can be considered radical because the order of execution of thread blocks cannot be controlled. However, by reverse engineering this order on different GPUs, this ordering can be exploited, in particular to reuse cache in a temporal fashion. Traditionally, cache cannot be reused in a temporal fashion, again because the thread block execution order is not controlled. A description of the GPU implementation and these optimizations are described in [9].

Software for fine-grained asynchronous ILU computation has been developed in collaboration with Sandia National Laboratories. The software is part of the ShyLU Trilinos package. The software uses Kokkos to provide performance portability across different types of manycore architectures including GPUs, Intel Xeon Phi, and general purpose CPUs. The software was presented in [17].

Software for fine-grained asynchronous ILU computation and sparse triangular solves is also part of the MAGMA Sparse library developed with our collaborators at the Innovative Computing Lab at the University of Tennessee.

7 Publications

[1] H. Anzt, E. Chow, and J. Dongarra, PARILUT: A New Parallel Threshold ILU Factorization, SIAM Journal on Scientific Computing, 2017, submitted.

[2] E. Chow, H. Anzt, J. Scott, and J. Dongarra, Experimental Study of Iterative Methods and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning, Journal of Parallel and Distributed Computing, 2017, submitted.

[3] H. Anzt, E. Chow, T. Huckle, and J. Dongarra, Batched Generation of Incomplete Sparse Approximate Inverses on GPUs, Proceedings of the 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA'16), Salt Lake City, UT, pp. 49-56 (2016).

[4] H. Anzt, E. Chow, J. Saak, and J. Dongarra, Updating Incomplete Factorization Preconditioners for Model Order Reduction, Numerical Algorithms, 73, 611-630 (2016).

[5] J. Wolfson-Pou and E. Chow, Reducing Communication in Distributed Asynchronous Iterative Methods, ICCS Workshop on Mathematical Methods and Algorithms for Extreme Scale, Procedia Computer Science, 80, 1906-1916 (2016).

[6] H. Anzt, E. Chow, D. Szyld, and J. Dongarra, Domain Overlap for Iterative Sparse Triangular Solves on GPUs, Lecture Notes in Computational Science and Engineering (LNCSE), 113, 527-545 (2016).

[7] E. Chow and A. Patel, Fine-grained Parallel Incomplete LU Factorization, SIAM Journal on Scientific Computing, 37, C169-C193 (2015).

[8] H. Anzt, E. Chow, and J. Dongarra, Iterative Sparse Triangular Solves for Preconditioning, 21st International European Conference on Parallel and Distributed Computing (Euro-Par 2015), Vienna, Austria, Aug. 24-28, 2015, Lecture Notes in Computer Science, 9233, 650-661 (2015).

[9] E. Chow, H. Anzt, and J. Dongarra, Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs, ISC High Performance, Frankfurt, Germany, July 12-16, 2015, Lecture Notes in Computer Science, 9137, 1-16 (2015).

Presentations

[10] E. Chow, Householder Symposium XX, Blacksburg, VA, June 18-23, 2017.

[11] E. Chow, Linear Algebra and Optimization Seminar, Institute for Computational and Mathematical Engineering, Stanford University, Palo Alto, CA, May 11, 2017.

[12] E. Chow, 26th Advanced Supercomputing Environment (ASE) Seminar, Information Technology Center (ITC), The University of Tokyo, Japan, Jan. 6, 2017.

[13] E. Chow, New Parallel Iterative Methods with Reduced Communication for Solving Large Sparse Linear Systems, Workshop on Recent Development of Matrix Computations, National Center for Theoretical Sciences, Taipei, Taiwan, May 13, 2016.

[14] E. Chow and J. Wolfson-Pou, Reducing Communication Costs in Distributed Relaxation Methods, Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, March 20-25, 2016.

[15] E. Chow, Very Fine-grained Parallelization of Preconditioning Operations, International Workshop on Software for Peta-Scale Numerical Simulation, Tokyo, Japan, Dec. 3-4, 2015.

[16] E. Chow, Very Fine-grained Parallelization of Preconditioning Operations, Numerical Analysis Group, Applied Mathematics Department, Delft University of Technology, Delft, The Netherlands, Nov. 9, 2015.

[17] A. Patel, S. Rajamanickam, E. G. Boman, and E. Chow, Parallel Iterative Incomplete Factorizations and Triangular Solves SIAM Conference on Applied Linear Algebra, Atlanta, GA, Oct. 26-30, 2015.

[18] E. Chow, Very fine-grained parallelization of sparse linear algebra computations, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, Oct. 13, 2015.

[19] E. Chow, Very fine-grained parallelization of sparse linear algebra computations, Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, Oct. 12, 2015.

[20] E. Chow, Asynchronous Algorithms for Approximate Sparse Matrix Factorizations, ICIAM 2015, Beijing, China, Aug. 10-15, 2015.

[21] E. Chow, Fine-Grained Parallel Incomplete Factorization Preconditioning, 2015 International Conference on Preconditioning Techniques for Scientific and Industrial Applications, Eindhoven University of Technology, Eindhoven, The Netherlands, June 17-19, 2015.

[22] E. Chow, Asynchronous Preconditioning on Accelerators, SIAM Conference on Computational Science and Engineering, Salt Lake City, UT, March 14-18, 2015.

[23] E. Chow, Fine-grained Parallel Computation of Approximate Sparse Matrix Factorizations, Applied Math and Scientific Computing Seminar, Department of Mathematics, Temple University, Philadelphia, PA, Feb. 25, 2015.

[24] E. Chow, A Parallel Iterative Approach for Computing and Updating an Approximate Sparse Matrix Factorization, Department of Mathematics Colloquium, Virginia Tech, Blacksburg, VA, Oct. 10, 2014.