Recent Developments to Enhance Scalability of Sparse Direct Solvers

Xiaoye S. Li[1] and Laura Grigori[2]

[1] Lawrence Berkeley National Laboratory, MS 50F-1650
One Cyclotron Road, Berkeley, CA 94720, USA.
xsli@lbl.gov
[2] INRIA Rennes
Campus Universitaire de Beaulieu, 35042 Rennes, France.
Laura.Grigori@irisa.fr

Abstract. We describe our recent efforts to improve scalability of the sparse factorization algorithms. One such development is to exploit more dense operations by switching to a full matrix representation towards a later stage of factorization. Another effort is the parallelization of the symbolic analysis algorithm which is used to determine the nonzero positions of the factored matrices. The first technique reduces amount of communication and indirect addressing, while the second one improves memory scalability of the solver.

1 Introduction

Sparse direct methods play a critical role in large scale computer simulations of various disciplines because they are robust and reliable, especially for ill-conditioned problems. However, it is a daunting task to implement sparse factorizations well on a distributed memory machine because of the need for managing irregular data access and communication patterns, and a high communication-to-computation ratio. Although a number of high quality parallel solvers have been developed, such as SuperLU_DIST [4] and MUMPS[1], there has been no demonstration that these solvers could scale to thousands of processors [2]. This scaling bottleneck will be worsened as the architectural trends indicate that the gap between the computation and the interconnect communication speeds will become wider. This motivates us to develop algorithms that has lower communication demand and thus reduces the bandwidth and latency costs. Here, we present two such ideas implemented in the SuperLU_DIST solver, which is based on sparse LU factorization. Our techniques are directly applicable to the other sparse factorization algorithms such as Cholesky and QR.

2 Performance optimization via switch-to-dense

Sparse matrices are often represented in a compact format in which only the nonzero values are stored. Auxiliary arrays are needed to store the position indices (integer) of those nonzero entries. In a fully parallel, distributed factorization algorithm, all these data structures are distributed on different processors.

During factorization, the nonzero values as well as their position indices must be transferred among the processors. Compared to a dense factorization, the extra costs include indirect addressing and many small-sized messages.

For many well structured matrices, such as those from FEM applications, the unfactored trailing submatrix can become very dense towards a later stage of factorization. It is then beneficial to represent the trailing block as a full matrix, and continue factorization using dense operations. The benefits include the elimination of indirect addressing and transferring of the indices, and use of larger block size in BLAS calls. On uniprocessor platforms, Vuduc et al. showed that this technique yielded up to 80% performance gain compared to a non-blocked compressed row storage [5].

We have implmented a parallel switch-to-dense algorithm in SuperLU_DIST. Since SuperLU_DIST already exploited the variable blocking scheme (i.e., supernodes), the payoff here mainly comes from reduction of the messages and larger block size. Depending on matrix-architecture combinations, we have observed 30-40% performance gain even using a relatively small number of processors (e.g., 32). An interesting research issue is the design of a good heuristic to select the siwtch point that best trades off the extra computation with the reduction of data movement.

3 Parallel symbolic factorization

The purpose of symbolic factorization is to compute the nonzero structure of the factors L and U, which contains the original nonzero elements as well as the filled elements. There has not been much motivation to parallelize this phase for two reasons: (1) Computationally, this phase is much faster than numerical factorization, and (2) The analysis involves combinatorial algorithms that are difficult to parallelize. Most direct solvers choose to do this sequentially on one processor, then broadcast the results to all processors. However, as larger and larger problems are solved on the emerging petascale computers and as numerical factorization has been made more and more scalable, symbolic factorization will ultimately become a bottleneck both in memory and in time.

We are the first group tackling this difficult problem in SuperLU_DIST [3]. Our parallel algorithm uses a graph partitioning approach, applied to the graph of $A + A^T$, to partition the matrix in such a way that is good for sparsity preservation as well as for parallel factorization. The partitioning yields a so-called separator tree which represents the dependencies among the computations. We use the separator tree to distribute the input matrix over the processors using a block cyclic approach and a subtree to sub-processor mapping. The parallel algorithm performs a bottom up traversal of the separator tree. With a combination of right-looking and left-looking partial factorizations, the algorithm obtains one column structure of L and one row structure of U at each step. Our prototype code led to five-fold reduction in maximum per-processor memory requirement. The already moderate serial runtime of the sequential algorithm is often further reduced by the parallel algorithm.

References

1. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
2. Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Xiaoye S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, 27(4):388–421, December 2001.
3. Laura Grigori, James W. Demmel, and Xiaoye S. Li. Parallel symbolic factorization algorithm. Technical Report LBNL-59031, Lawrence Berkeley National Laboratory, November 2005.
4. Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.
5. R. Vuduc, S. Kamil, J Hsu, R. Nishtala, J. W. Demmel, and K. A. Yelick. Automatic performance tuning and analysis of sparse triangular solve. In *Proceedings of the ICS 2002: Workshop on Performation Optimizations via High-Level Languages and Libraies*, New York, June 2002.