STRUMPACK: Scalable Preconditioning using Low-Rank Approximations and Random Sampling

Pieter Ghysels and Xiaoye S. Li
Computational Research Division
Lawrence Berkeley Nat'l Laboratory
Berkeley, California 94720-8150
Email: {pghysels,xsli}@lbl.gov

Christopher Gorman
Mathematics Department
UC Santa Barbara
Santa Barbara, California 93106
Email: gorman@math.ucsb.edu

François-Henry Rouet
Livermore Software Technology Corporation
Livermore, California 94551
Email: fhrouet@gmail.com,
fhrouet@lbl.gov

***Abstract*—We present a parallel and fully algebraic preconditioner based on an approximate sparse factorization using rank-structured matrix compression. The sparse factorization is a multifrontal algorithm with a nested-dissection fill-in-reducing ordering. The fill-in occurs in so-called frontal matrices, which are approximated as Hierarchically Semi-Separable (HSS) rank-structured matrices. The HSS matrices are constructed using an efficient randomized sampling technique. We show that this preconditioner has optimal or close to optimal complexity – in terms of floating point operations and memory usage – for matrices from several types of discretized Partial Differential Equations (PDEs). Through a number of numerical experiments, we illustrate the robustness and performance of this new preconditioner for a wide range of problems, including non-PDE based problems. This shows that the rank-structured preconditioner is a viable alternative to for instance algebraic multigrid and incomplete factorization preconditioners such as ILU. The code is released with a BSD license and a PETSc interface is available to allow for easy integration in existing applications.**

I. Algorithm Description

The sparse solver and preconditioner implemented in STRUMPACK is based on the method originally described in [1]. However, the method presented here is a generalization to non-symmetric and indefinite linear systems. The shared memory implementation of STRUMPACK-sparse is described in [2]. Here, the distributed memory hybrid MPI+OpenMP code is presented.

The STRUMPACK-sparse solver is based on multifrontal sparse LU factorization [3]. The main bottleneck in sparse LU factorization is excessive memory usage due to fill-in and the corresponding floating point operations on this fill-in. In multifrontal LU, the fill-in occurs in dense frontal matrices. However, it has been observed by a number of authors [4], [1] that for certain applications, like discretizations of partial differential equations, these dense matrices have low-rank blocks, typically away from the diagonal. Therefore, in STRUMPACK-sparse the large dense frontal matrices are approximated by Hierarchically Semi-Separable (HSS) matrices [5]. HSS is an example of a data-sparse or hierarchical ($\mathcal{H}$ [6]) matrix format. An HSS matrix uses a hierarchical partitioning where the off-diagonal blocks are approximated as low-rank. For the efficient construction of HSS matrices we use a randomized sampling scheme as presented in [7]. This randomized scheme has the benefit that it avoids constructing large dense frontal matrices explicitly, hence saving memory for the factorization as well as maximum working memory. However, the main benefit of random sampling is that the dimension of the problem is reduced. This makes the application of a rank-revealing factorization much cheaper. STRUMPACK uses rank-revealing QR factorization, implemented as QR factorization with column pivoting. Moreover, the random sampling (also referred to as random projection) approach requires multiplication with a dense random matrix, which has several advantages for modern computer architectures. On multi-core machines it can efficiently use BLAS3 routines and on distributed memory clusters communication can be avoided since random matrices do not need to be exchanged between nodes, instead they can just be generated with the properly initialized random generator. STRUMPACK provides a distributed memory implementation of the randomized HSS construction [8], as well as fast ULV-like factorization. Unlike in traditional ULV factorization, the U and V in STRUMPACK's ULV factorization are not orthogonal.

II. Design of the Distributed Memory Algorithm

The parallel implementation of the STRUMPACK sparse solver and preconditioner relies on MPI, ScaLAPACK with BLACS, LAPACK and BLAS and ParMetis or PT-Scotch for nested-dissection matrix reordering. For on-node parallelism STRUMPACK uses OpenMP.

At the distributed memory level, subtrees of the sparse elimination or separator tree are assigned statically to MPI sub-communicators based on estimates of the amount of work in those subtrees, an assignment referred to as proportional mapping. Operations on nodes of the separator tree that are mapped to multiple MPI processes use multi-threaded ScaLAPACK through multi-threaded system BLAS. A subtree assigned to a single MPI process is traversed using OpenMP task parallelism. This allows for dynamic runtime task scheduling, which deals well with load imbalances due to irregular trees. However, multi-threaded BLAS cannot be used from within OpenMP tasks. Therefore, we implemented parallel version of a number of BLAS routines using recursive OpenMP tasking. This allows us to exploit multiple levels of parallelism (within a subtree mapped on a single MPI process), all within the same OpenMP runtime task scheduler.

An interface from PETSc to STRUMPACK-sparse is available in the current master branch of PETSc and will be part of the next major release (3.8). This makes it trivial for applications that already rely on the very powerful PETSc infrastructure to use the STRUMPACK solver or preconditioner.

III. Comparison With Other Solvers & Preconditioners

Like algebraic multigrid (AMG), the sparse approximate solver presented here is fully algebraic and has linear or close to linear scaling with respect to the problem size for a range of PDE based problems. However, unlike AMG our solver is based on a direct (factorization) method, which makes it more robust and applicable to broader classes of problems. The STRUMPACK preconditioner remains effective in cases where AMG preconditioning does not converge well (for instance indefinite problems, problems with irregular meshes and large jumps in coefficients). Numerical experiments also show our method to be more robust than incomplete LU type preconditioners. Moreover, methods like ILU are hard to scale up in parallel, whereas our algorithm scales well to thousands of processes.

IV. Conclusion

Our poster presents the STRUMPACK sparse solver and preconditioner. This is a parallel implementation of a fast sparse solver using HSS matrix compression. The work presented here and on the poster is an extension of our earlier work in [2] and [8] but the current work has not appeared in publication before.

Acknowledgment

Partial support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] J. Xia, "Randomized sparse direct solvers," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 1, pp. 197–227, 2013.

[2] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, and A. Napov, "An efficient multi-core implementation of a novel hss-structured multifrontal solver using randomized sampling," 2014, to appear in SIAM Journal on Scientific Computing.

[3] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *ACM Transactions on Mathematical Software (TOMS)*, vol. 9, no. 3, pp. 302–325, 1983.

[4] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam, "On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2261–2290, 2010.

[5] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, "A bibliography on semiseparable matrices," *Calcolo*, vol. 42, no. 3-4, pp. 249–270, 2005.

[6] S. Börm, L. Grasedyck, and W. Hackbusch, "Introduction to hierarchical matrices with applications," *Engineering Analysis with Boundary Elements*, vol. 27, no. 5, pp. 405–422, 2003.

[7] P.-G. Martinsson, "A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1251–1274, 2011.

[8] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, "A distributed-memory package for dense Hierarchically Semi-Separable matrix computations using randomization," 2014, to appear in ACM Transactions on Mathematical Software.