A PARALLEL HYBRID LINEAR SOLVER FOR ACCELERATOR CAVITY DESIGN

I. Yamazaki*, X. S. Li†, and E. G. Ng‡
Lawrence Berkeley National Laboratory, Berkeley, CA 94720.

Abstract

Numerical simulations to design high-energy particle accelerators give rise to large-scale ill-conditioned highly-indefinite linear systems of equations that are becoming increasingly difficult to solve using either a direct solver or a preconditioned iterative solver alone. In this paper, we describe our current effort to develop a parallel hybrid linear solver that balances the robustness of a direct solver with the efficiency of a preconditioned iterative solver. We demonstrate that our hybrid solver is more robust and efficient than the existing state-of-the-art software to solve these linear systems on a large number of processors.

INTRODUCTION

Numerical simulations to design high-energy particle accelerators [10] give rise to large sparse linear systems of equations that are becoming increasingly difficult to solve using standard techniques [9]. Although significant progress has been made in the development of high-performance direct solvers [2, 4, 11], the dimension of the systems that can be directly factored is limited due to large memory requirements. Preconditioned iterative solvers [3, 14, 16] can reduce the memory requirements, but they often suffer from slow convergence.

To overcome these challenges, a number of parallel hybrid solvers have been developed based on a domain decomposition idea called the Schur complement method [5, 6]. In this method, the unknowns in interior domains are first eliminated using a direct method, and the remaining Schur complement system is solved using a preconditioned iterative method. These hybrid solvers often exhibit great parallel performance because the interior domains can be factored in parallel, and the direct solver is effective to factor the relatively-small interior domain. In addition, the preconditioned iterative solver is shown to be robust to solve the Schur complement systems, where most of the fill occurs, in a number of applications [5, 6]. In particular, for a symmetric positive definite system, the Schur complement has a smaller condition number than the original matrix [15, Section 4.2], and fewer iterations are often needed to solve the Schur complement system. Hence, these hybrid solvers have the potential to balance the robustness of the direct solver with the efficiency of the iterative solver.

Unfortunately, for a highly-indefinite linear system from the accelerator simulation, these existing hybrid solvers often suffer from slow convergence when solving the Schur complement system. This is true especially on a large number of processors because these solvers are designed to achieve good scalability of time to compute the preconditioners, but the quality of the preconditioner often degrades as more processors are used.

To overcome these drawbacks, we have been developing a new implementation of the Schur complement method which provides the robustness and flexibility to solve large highly-indefinite linear systems on a large number of processors [12, 17]. In this paper, we demonstrate the effectiveness of our hybrid solver to solve these linear systems on hundreds of processors using a linear system whose dimension is greater than those used in our previous papers. We also point out how our impelemtnation has been modified since the last publication in order to solve such large linear systems with millions of unknowns.

SCHUR COMPLEMENT METHOD

The Schur complement method is a non-overlapping domain decomposition method, which is also referred to as iterative substructuring. In this method, the original linear system is first reordered into a 2×2 block system of the following form:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \qquad (1)$$

where A_{11} and A_{22} respectively represent *interior domains* and *separators*, and A_{12} and A_{21} are the *interfaces* between A_{11} and A_{22}. By eliminating the unknows associated with the interior domains A_{11} in the bottom part of (1), we obtain the block-triangular system

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & S \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ \widehat{b}_2 \end{pmatrix}, \qquad (2)$$

where S is the Schur complement defined as

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12}, \qquad (3)$$

and $\widehat{b}_2 = b_2 - A_{21} A_{11}^{-1} b_1$. Hence, the solution of the linear system (1) can be computed by first solving the Schur complement system

$$S x_2 = \widehat{b}_2, \qquad (4)$$

then solving the interior system

$$A_{11} x_1 = b_1 - A_{12} x_2. \qquad (5)$$

Note that interior domains are independent of each other, and A_{11} is a block-diagonal matrix. Hence, the relatively

* ic.yamazaki@gmail.com
† xsli@lbl.gov
‡ egng@lbl.gov

small interior domains can be efficiently factored in parallel using a direct solver. On the other hand, a large amount of fill can be introduced in S. In order to reduce the memory requirement, the Schur complement system (4) is solved using a preconditioned iterative method. We note that within the iterative method, the matrix-vector product with the Schur complement S can be computed by applying the sequence of the sparse matrix operations (3) on the vector, and hence, S does not have to be stored explicitly for this phase of the solver.

PARALLEL IMPLEMENTATION

To obtain high-performance, our implementation takes full advantage of the state-of-the-art software. Specifically, the 2×2 block system (1) is computed using PT-SCOTCH [8], which implements a parallel nested bisection algorithm.[1] Then, to compute the LU factors of the interior domains in parallel, our hybrid solver uses either a "1-level" or "2-level" configuration, where each interior domain is factored using either a single processor or multiple processors, respectively. Currently, we use a serial direct solver SuperLU [4] or a parallel direct solver SuperLU_DIST [11] for the 1-level or 2-level configuration, respectively. To preserve the sparsity of the LU factors, each interior domain is permuted using METIS [7] or PT-SCOTCH. Then, the Schur complement system (4) is solved using a Krylov subspace method from PETSc [13] combined with an ILU preconditioner. Finally, the solution of the interior system (5) is computed using the previously computed LU factors of the interior domains.

Since computing the ILU preconditioner of the Schur complement is often the computational and memory bottleneck, we give a brief description of how the preconditioner is computed. In our current implementation, the preconditioners are the exact LU factors of a sparsified Schur complement. Specifically, let us denote the ℓ-th interior domain and corresponding interfaces by $A_{11}^{(\ell)}$, and $A_{12}^{(\ell)}$ and $A_{21}^{(\ell)}$, respectively, such that the coefficient matrix of the 2×2 block system (1) with k interior domains can be written as

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{cccc|c} A_{11}^{(1)} & & & & A_{12}^{(1)} \\ & A_{11}^{(2)} & & & A_{12}^{(2)} \\ & & \ddots & & \vdots \\ & & & A_{11}^{(k)} & A_{12}^{(k)} \\ \hline A_{21}^{(1)} & A_{21}^{(2)} & \dots & A_{21}^{(k)} & A_{22} \end{array}\right). \quad (6)$$

In the 1-level configuration, the ℓ-th processor stores the nonzeros of $A_{11}^{(\ell)}$ and $A_{21}^{(\ell)}$ in row-wise order, and the nonzeros of $A_{12}^{(\ell)}$ in column-wise order. If the 2-level configuration is used, then the rows of $A_{11}^{(\ell)}$ and $A_{12}^{(\ell)}$, and the columns of $A_{21}^{(\ell)}$ are evenly distributed among the processors assigned to the ℓ-th interior domain. Furthermore, the rows of A_{22} are evenly distributed among the processors that solve the Schur complement system (4).

With the block structure (6) and the LU factorization of the interior domain $A_{11}^{(\ell)}$, which is denoted by $A_{11}^{(\ell)} = L_{11}^{(\ell)} U_{11}^{(\ell)}$,[2] the Schur complement is computed as

$$\begin{aligned} S &= A_{22} - \sum_{\ell=1}^{k} A_{21}^{(\ell)} (A_{11}^{(\ell)})^{-1} A_{12}^{(\ell)} \\ &= A_{22} - \sum_{\ell=1}^{k} ((U_{11}^{(\ell)})^{-T} (A_{21}^{(\ell)})^T)^T ((L_{11}^{(\ell)})^{-1} A_{12}^{(\ell)}) \\ &= A_{22} - \sum_{p=1}^{p_A} E^{(\ell)}(:, j_1^{(p)} : j_2^{(p)}) F^{(\ell)}(j_1^{(p)} : j_2^{(p)}, :), \end{aligned}$$

where p_A is the number of processors used to solve the whole system, $E^{(\ell)} = ((U_{11}^{(\ell)})^{-T} (A_{21}^{(\ell)})^T)^T$, $F^{(\ell)} = (L_{11}^{(\ell)})^{-1} A_{12}^{(\ell)}$, the p-th processor owns the $j_1^{(p)}$-th through $j_2^{(p)}$-th columns of $A_{21}^{(\ell)}$ and the corresponding rows of $A_{12}^{(\ell)}$, and $E(:, j_1 : j_2)$ and $F(j_1 : j_2, :)$ are the matrices consisting of the j_1-th through j_2-th columns of a matrix E and the corresponding rows of F, respectively. In other words, after $E^{(\ell)}$ and $F^{(\ell)}$ are computed by the processors assigned to the ℓ-th interior domain, the $j_1^{(p)}$-th through $j_2^{(p)}$-th columns of $E^{(\ell)}$ and the corresponding rows of $F^{(\ell)}$ are sent to the p-th processors. Then, the p-th processor computes the corresponding outer-product updates of the Schur complement S, and sends the rows of the updates to the processor that owns the corresponding rows of A_{22}. Subsequently, the rows of S are evenly distributed among the processors that solve the Schur complement system. We note that the matrices $E^{(\ell)}$ and $F^{(\ell)}$ are computed with regards to the sparsity of interface $A_{21}^{(\ell)}$ and $A_{12}^{(\ell)}$.

In the above expression, the columns of $E^{(\ell)}$ and rows of $F^{(\ell)}$ are distributed in the same way as those of $A_{21}^{(\ell)}$ and $A_{12}^{(\ell)}$, respectively. This is not necessary. In our new implementation, the matrices $E^{(\ell)}$ and $F^{(\ell)}$ are distributed to balance the computational cost to compute the outer-product updates $E^{(\ell)}(:, j_1^{(p)} : j_2^{(p)}) F^{(\ell)}(j_1^{(p)} : j_2^{(p)}, :)$ among the processors assigned to the ℓ-th interior domain. Furthermore, to reduce the costs, the sparsity of the matrices $E^{(\ell)}$ and $F^{(\ell)}$ are enforced by discarding nonzeors with magnitudes less than a prescribed drop tolerance σ_1. Hence, an approximation $\widetilde{S}$ to the Schur complement S is computed.

After the approximate Schur complement $\widetilde{S}$ is computed, it is preprocessed to enhance numerical stability, and then its sparsity is enforced using a drop tolerance σ_2. Finally, SuperLU_DIST is used to compute the LU factor of $\widetilde{S}$, which is used as the preconditioner. To preserve the sparsity of the LU factor, $\widetilde{S}$ is permuted based on a parallel nested bisection of the supernobal graph of $\widetilde{S}$. See [17], for the preprocessing techniques used on $\widetilde{S}$.

[1] In the previous implementation [12], HID of HIPS [5] was used to compute the block system using a single processor.

[2] The matrix $A_{11}^{(\ell)}$ is scaled and permuted to enhance numerical stability and preseve the sparsity of $L_{11}^{(\ell)}$ and $U_{11}^{(\ell)}$. For clarity, the scaling and permutation are not shown in the expression.

NUMERICAL EXPERIMENTS

We present preliminary results of our hybrid solver to solve a highly-indefinite linear system on hundreds of processors. Our test matrix **tdr455k** is from a simulation of an international linear collider [1, 10], and its dimension is $2,738,556$. To solve the Schur complement system, we used the unrestarted GMRES from PETSc [13]. The GMRES iteration was started with the zero vector, and the computed solution was considered to have converged when the ℓ_2-norm of the initial residual was reduced by at least twelve order of magnitude. This is the solution accuracy enforced in the actual simulations. All the experiments were conducted on the Cray XT4 machine at the National Energy Research Scientific Computing Center.

To demonstrate the effectiveness of our hybrid solver, we compare its performance with that of a direct solver SuperLU_DIST [11], and that of a state-of-the-art hybrid solver HIPS [5]. The primary difference between our hybrid solver and HIPS is the way the preconditioner is computed for solving the Schur complement system. HIPS computes the preconditioner based on an ILU factorization of $\widetilde{S}$, where the sparsity of the preconditioner is enforced based on both numerical values and locations of nonzeros. Specifically, the fill is allowed only between separators adjacent to the same domain. Furthermore, HIPS factors each interior domain using a single processor, and the number of interior domains needs to be at least as large as the number of processors used to solve the whole linear system. This allows HIPS to achieve good parallel scalability of time to compute the prconditioner. However, for a highly-indefinite system, we found that HIPS often suffers from slow convergence. This was especially true on a large number of processors since a large number of interior domains must be generated, which increases the size of the Schur complement.

To compare the performance of our hybrid solver with that of SuperLU_DIST, Figure 1 shows the total solution times as functions of the number of processors used to solve the linear system. In the figure, "1-level" and "2-level" are the two configurations of our hybrid solver, which use a single processor and multiple processors to factor each interior domain, respectively. In our numerical experiments with the 1-level configuration, we set the number of interior domains to be equal to the number of processors used to solve the linear system. On the other hand, for the 2-level configuration, the number of interior domains was fixed to be 16, and the processors were evenly distributed among the interior domains.

By looking at the left plot of Figure 1, we see that the solution time with our hybrid solver scaled better than that with SuperLU_DIST, (i.e., the hybrid solver could reduce the solution time using up to 512 processors, while SuperLU_DIST did not scale beyond 128 processors). Furthermore, we see that the solution times with the 1-level configuration scaled similarly to that with the 2-level configuration. This is because with the small drop tolerances $(\sigma_1, \sigma_2) = (10^{-6}, 10^{-5})$, the number of GMRES iterations was nearly independent of the number of interior domains, and GMRES converged within 20 iterations even when the number of interior domains needed to be increased for the 1-level configuration to run on more processors (see Table 1). In comparison, HIPS required 151 iterations on 16 processors, and it failed to converge within $1,000$ iterations on 32 processors even though the drop tolerances were set to be zero. Furthermore, even when HIPS converged, our hybrid solver solved the linear system faster since it takes full advantage of the state-of-the art software.

	Number of domains				
(σ_1, σ_2)	16	32	64	128	256
$(10^{-6}, 10^{-5})$	11	15	15	17	16
$(10^{-5}, 10^{-4})$	32	60	116	205	290

Table 1: Number of iterations with our hybrid solver.

We note that larger drop tolerances reduce the memory requirement of our hybrid solver. For example, in Figure 1, less memory was needed in the right plot since the drop tolerances were increased by an order of magnitude from those in the left plot. Specifically, in the left plot, about 15% of the nonozeros were discarded from the matrices E and F, and about 50% of the nonzeros were discarded from the approximate Schur complement $\widetilde{S}$. On the other hand, in the right plot, about 30% of the nonzeros were discarded from E and F, and about 75% of the nonzeros were discarded from $\widetilde{S}$. Unfortunately, with large drop tolerances, the number of GMRES iterations may increase as more interior domains are generated. For example with the drop tolerance $(\sigma_1, \sigma_2) = (10^{-5}, 10^{-4})$, the number of iterations increased from 32 to 290 when the number of interior domains increased from 16 to 256 (see Table 1). As a result, with the 1-level configureation, the solution time did not scale beyond 64 processors (see Figure 1). On the other hand, with the 2-level configuration, we can increase the number of processors while keeping the size of the Schur complement the same. Subsequently, with the 2-level configuration, the solution time was reduced using up to 256 processors in Figure 1.

Unfortunately, with the drop tolerance $(\sigma_1, \sigma_2) = (10^{-5}, 10^{-4})$, the 2-level configuration could not reduce the solution time using 512 processors from that using 256 processors (see Figure 1). This is because with the 2-level configuration, the size of the Schur complement is fixed, and we used only 16 processors to solve the Schur complement system even when more processors were available. As a result, the time to solve the Schur complement system became the dominant part of the total solution time as the number of processors increased. However, the total solution time was still reduced from that of SuperLU_DIST by a factor of 2.3 on 512 processors.

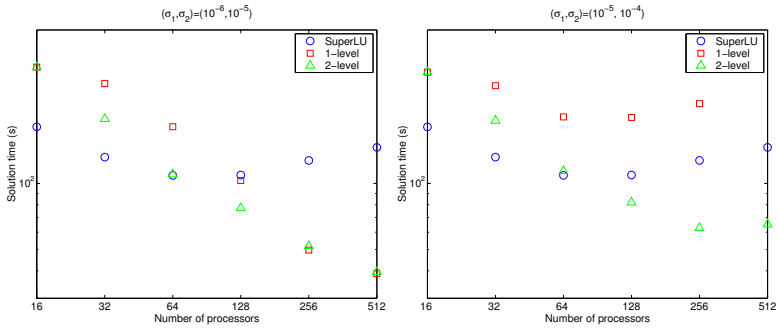

Figure 1: Solution times required by SuperLU_DIST and our hybrid solver.

CONCLUSIONS

For a large-scale particle accelerator simulation, solving large highly-indefinite linear systems becomes the memory bottleneck. We have described our current effort to address this challenge with a new parallel hybrid solver. The preliminary results have demonstrated that our hybrid solver improves the parallel scalability of a state-of-the-art direct solver. Furthermore, in comparison to a state-of-the-art hybrid solver, our hybrid solver is more robust and efficient to solve these linear systems on a large number of processors. As a result, our hybrid solver has the potential to enable a large-scale particle accelerator simulation by using a large number of processors and reducing the memory required by a processor. We are working to improve the parallel performance of our solver and conducting further experiments to solve larger systems using thousands of processors.

ACKNOWLEDGMENTS

The authors thank Lie-Quan Lee at the SLAC National Accelerator Laboratory for providing the test matrix. This research was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We used the resources at the National Energy Research Scientific Computing Center.

REFERENCES

[1] Community Petascale Project for Accelerator Science and Simulation (ComPASS). https://compass.fnal.gov.

[2] P. Amestoy, I. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[3] O. Axelsson. *Iterative solution methods*. Cambridge University Press, New York, 1994.

[4] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20:720–755, 1999.

[5] J. Gaidamour and P. Henon. HIPS: a parallel hybrid direct/iterative solver based on a schur complement. In *Proc. PMAA*, 2008.

[6] L. Giraud, A. Haidar, and L. T. Watson. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Computing*, 34:363–379, 2008.

[7] Karypis Lab, Digital Technology Center, Department of Computer Science and Engineering, University of Minesota. METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis.

[8] Laboratoire Bordelais de Recherche en Informatique (LaBRI). SCOTCH - Software package and libraries for graph, mesh and hypergraph partitioning, static mapping, and parallel and sequential sparse matrix block ordering. http://www.labri.fr/perso/pelegrin/scotch/.

[9] L.-Q. Lee, L. Ge, M. Kowalski, Z. Li, C.-K. Ng, G. Schussman, M. Wolf, and K. Ko. Solving large sparse linear systems in end-to-end accelerator structure simulations. *Parallel and Distributed Processing Symposium, International*, 1:8a, 2004.

[10] L.-Q. Lee, Z. Li, C.-K. Ng, and K. Ko. Omega3P: A parallel finite-element eigenmode analysis code for accelerator cavities. Technical Report SLAC-PUB-13529, Stanford Linear Accelerator Center, 2009.

[11] X. Li and J. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, 2003.

[12] X. Li, M. Shao, I. Yamazaki, and E. Ng. Factorization-based sparse solvers and preconditioners. In *Journal of Physics: Conference Series, SciDAC 2009*.

[13] Mathematics and Computer Science Division, Argonne National Laboratory. The portable, extensible, toolkit for scientific computation (PETSc). www.mcs.anl.gov/petsc.

[14] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, 2004.

[15] B. Smith, P. Bjorstad, and W. Gropp. *Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.

[16] H. van der Vorst. *Iterative Krylov methods for large linear systems*. Cambridge University Press, New York, 2003.

[17] I. Yamazaki, X. Li, and E. Ng. Preconditioning schur complement systems of highly-indefinite linear systems for a parallel hybrid solver. In *Submitted to the proceedings of the international conference on preconditioning techniques for scientific and industrial applications*, 2009.