On Partitioning and Reordering Problems in a Hierarchically Parallel Hybrid Linear Solver

Ichitaro Yamazaki, Xiaoye S. Li, François-Henry Rouet, Bora Uçar

On partitioning and reordering problems in a hierarchically parallel hybrid linear solver

Ichitaro Yamazaki
Electrical Engineering and Computer Science Department
University of Tennessee, Knoxville
Email: ic.yamazaki@gmail.com

Xiaoye S. Li
Computational Research Division
Lawrence Berkeley National Laboratory
Email: xsli@lbl.gov

François-Henry Rouet
ENSEEIHT-IRIT and
Lawrence Berkeley National Laboratory
Email: fhrouet@lbl.gov

Bora Uçar
CNRS and LIP, ENS Lyon, France
UMR CNRS-ENS-INRIA-UCBL
Email:bora.ucar@ens-lyon.fr

***Abstract*—PDSLin is a general-purpose algebraic parallel hybrid (direct/iterative) linear solver based on the Schur complement method. The most challenging step of the solver is the computation of a preconditioner based on the global Schur complement. Efficient parallel computation of the preconditioner gives rise to partitioning problems with sophisticated constraints and objectives. In this paper, we identify two such problems and propose hypergraph partitioning methods to address them. The first problem is to balance the work loads associated with different subdomains to compute the preconditioner. We first formulate an objective function and a set of constraints to model the preconditioner computation time. Then, to address these complex constraints, we propose a recursive hypergraph bisection method. The second problem is to improve the data locality during the parallel solution of a sparse triangular system with multiple sparse right-hand sides. We carefully analyze the objective function and show that it can be well approximated by a standard hypergraph partitioning method. Moreover, an ordering compatible with a postordering of the subdomain elimination tree is shown to be very effective in preserving locality. To evaluate the two proposed methods in practice, we present experimental results using linear systems arising from some applications of our interest. First, we show that in comparison to a commonly-used nested graph dissection method, the proposed recursive hypergraph partitioning method reduces the preconditioner construction time, especially when the number of subdomains is moderate. This is the desired result since PDSLin is based on a two-level parallelization to keep the number of subdomains small by assigning multiple processors to each subdomain. We also show that our second proposed hypergraph method improves the data locality during the sparse triangular solution and reduces the solution time. Moreover, we show that partitioning time can be greatly reduced while maintaining its quality by removing quasi-dense rows from the solution vectors.**

***Keywords*-Schur complement method; hypergraph partitioning.**

I. Introduction

In recent years, the Schur complement method [23] has gained popularity as a framework to develop scalable parallel hybrid (direct/iterative) linear solvers [13], [15], [25]. In this method, the original linear system $Ax = b$ is first reordered into a system of the following block structure:

$$\left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \cdots & F_k & C \end{array}\right)\left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_k \\ y \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ f_k \\ g \end{array}\right), \quad (1)$$

where D_ℓ is referred to as the ℓ-th *interior subdomain*, C consists of *separators*, and E_ℓ and F_ℓ are the *interfaces* between D_ℓ and C. Then, to compute the solution of the linear system (1), we first compute the solution vector y on the interface by solving the Schur complement system,

$$Sy = \widehat{g}\,, \quad (2)$$

where the Schur complement S is given by

$$S = C - \sum_{\ell=1}^{k} F_\ell D_\ell^{-1} E_\ell\,, \quad (3)$$

and $\widehat{g} = g - \sum_{\ell=1}^{k} F_\ell D_\ell^{-1} f_\ell$. Then, to compute the solution vector u_ℓ, we solve the ℓ-th subdomain system

$$D_\ell u_\ell = f_\ell - E_\ell y\,. \quad (4)$$

The Schur complement S often has a smaller condition number, but it is much denser than the original matrix A. Hence, a preconditioned iterative solver is typically used to solve (2) without explicitly forming S.

PDSLin (Parallel Domain decomposition Schur complement based Linear solver) [26], [25], [27] implements this Schur complement method based on a *two-level parallelism*: subdomains D_ℓ are factorized in parallel using a distributed-memory parallel solver such as SuperLU_DIST [12]. Hence, the numbers of subdomains can be far fewer than the number of processors. We emphasize that for a scalable parallel implementation, it is imperative to exploit this hierarchical parallelism. If a *single-level* parallelism is used (i.e., one processor per subdomain), the number of subdomains increases with the

number of processors, leading to a larger Schur complement S and increasing the cost of solving (3). This is especially true since PDSLin targets highly-indefinite linear systems, and the number of iterations increases dramatically when the size of S is larger [25]. For example, in our experiments, we use tens of subdomains (i.e., $k = 8$ or 32) to ensure that the iterative solver converges in a few iterations while the direct solver can efficiently factorize D_ℓ using tens to hundreds of processors.

Our focus in this paper is to reduce the time of computing a preconditioner for solving the Schur complement system (2), which is the most challenging step of PDSLin. In order to provide a robust preconditioner, PDSLin computes the preconditioner based on the global Schur complement S. Specifically, from the initial partition (1), PDSLin first extracts a *local* matrix A_ℓ associated with each subdomain D_ℓ:

$$A_\ell = \begin{pmatrix} D_\ell & \widehat{E}_\ell \\ \widehat{F}_\ell & O \end{pmatrix},$$

where $\widehat{E}_\ell$ and $\widehat{F}_\ell$ consist of the nonzero columns and rows of E_ℓ and F_ℓ, respectively. Then, the LU factors of D_ℓ are computed using SuperLU_DIST, i.e., $P_\ell D_\ell \bar{P}_\ell = L_\ell U_\ell$, where P_ℓ and $\bar{P}_\ell$ are the row and column permutation matrices, respectively. Next, a local *update* matrix T_ℓ is computed as

$$T_\ell = \widehat{F}_\ell D_\ell^{-1} \widehat{E}_\ell = W_\ell G_\ell, \tag{5}$$

where $W_\ell = \widehat{F}_\ell \bar{P}_\ell U_\ell^{-1}$ and $G_\ell = L_\ell^{-1} P_\ell \widehat{E}_\ell$. A large amount of fill may occur in W_ℓ and G_ℓ. To reduce the memory and computational costs, we compute the approximations $\widetilde{W}_\ell$ and $\widetilde{G}_\ell$ of W_ℓ and G_ℓ, respectively, by discarding nonzeros with magnitudes less than a prescribed threshold. Then, as an approximate update matrix $\widetilde{T}_\ell = \widetilde{W}_\ell \widetilde{G}_\ell$ is computed, it is gathered to form an approximate global Schur complement

$$\widehat{S} = C - \sum_{\ell=1}^{k} R_{F_\ell} \widetilde{T}_\ell R_{E_\ell}^T ,$$

where R_{E_ℓ} and R_{F_ℓ} are interpolation matrices to map the columns and rows of $\widehat{E}_\ell$ and $\widehat{F}_\ell$ to those of E_ℓ and F_ℓ, respectively. (They are not formed explicitly.) To further reduce the costs, small nonzeros are discarded from $\widehat{S}$ to form its approximation $\widetilde{S}$. Finally, the LU factors of $\widetilde{S}$ are computed using SuperLU_DIST and used as a preconditioner for solving (2).

In this paper, we study the following two combinatorial problems to reduce the preconditioner computation time: computing the partition (1) with multiple constraints to improve parallel load balance at various stages of preconditioner computation (Section III); and reordering columns of $\widehat{E}_\ell$ and $\widehat{F}_\ell^T$ to improve data locality of the triangular solver to form G_ℓ and W_ℓ respectively (Section IV). To demonstrate the effectiveness of the proposed methods, in Section V, we present experimental results with large-scale linear systems of equations arising from some applications of our interest. We then conclude in Section VI.

To highlight our contribution, Figure 1 shows the PDSLin runtime with two-level parallelization for one of our test matrices (see Table I). Here, for each core count, we compare the solution times using our new Recursive Hypergraph Bisection (RHB) algorithm with that using the state-of-the-art nested graph dissection algorithm of PT-Scotch [10], [21]. As designed, the new RHB algorithm reduces the time to compute $\widetilde{S}$ without a significant increase in the time to compute LU factors of D_ℓ in all cases. The details of the algorithms and experimental setups will be described in the following sections.

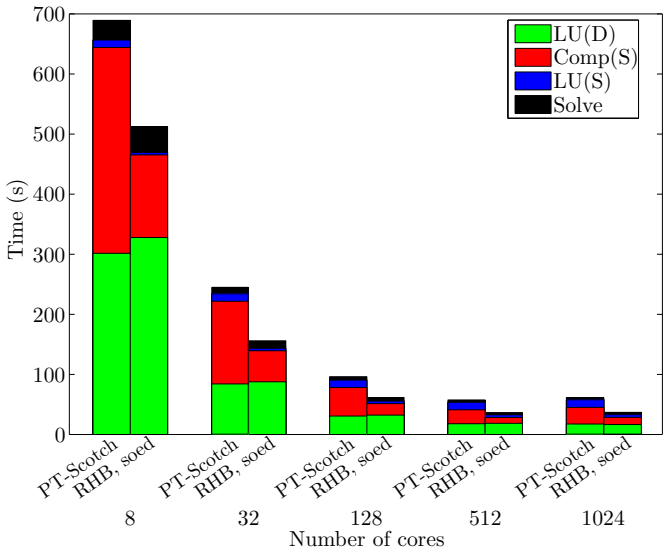


Fig. 1. Run time of PDSLin for matrix **tdr455k** as a function of the number of processes, and using two different partitioning strategies. The number of subdomains is set to 8.

II. Preliminaries

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a generalization of a graph, where every net (or hyperedge) in $\mathcal{N}$ is a subset of vertices $\mathcal{V}$. Usually, a nonnegative weight $w(i)$ and a nonnegative cost $c(j)$ are associated with the i-th vertex v_i and the j-th net n_j, respectively. In this paper, each net has a unit cost unless otherwise stated. In a k-way partition $\Pi_k(\mathcal{V}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ of the vertex set $\mathcal{V}$, a net is said to *connect* a part if it has at least one vertex in that part. The *connectivity set* $\Lambda(j)$ of the j-th net n_j is the set of parts connected by n_j, while the *connectivity* $\lambda(j)$ of n_j is the number of parts connected by n_j, i.e., $\lambda(j) = |\Lambda(j)|$, where $|\cdot|$ denotes the cardinality of a set. The net n_j is said to be a *cut net* if $\lambda(j) > 1$.

The solution of a hypergraph partitioning problem must satisfy at least one balancing constraint and one objective function. Let $W(\mathcal{V}_\ell) = \sum_{v_i \in \mathcal{V}_\ell} w(i)$ be the weight of the ℓ-th part, then the balancing constraint is formally defined as

$$\frac{W_{max} - W_{avg}}{W_{avg}} \leq \varepsilon \, , \tag{6}$$

where W_{max} and W_{avg} are the largest and average part weights, respectively, and ε is a given allowable imbalance ratio. The objective of a partitioning problem is to minimize

a cutsize metric for which there are three standard definitions [9], [16] of $cutsize(\Pi_k)$:

$$\sum_{n_j \in \mathcal{N}} (\lambda(j) - 1) \quad \equiv \text{connectivity-1 (con1)}, \tag{7}$$

$$\sum_{\substack{n_j \in \mathcal{N}, \\ \lambda(j) > 1}} 1 \quad \equiv \text{cut-net (cnet)}, \tag{8}$$

$$\sum_{\substack{n_j \in \mathcal{N}, \\ \lambda(j) > 1}} \lambda(j) \quad \equiv \text{sum-of-external-degree (soed)}. \tag{9}$$

Each of the above metrics has been used in various application domains (for some see [4, Section 2.2]). In Section III-C, we examine all of these three cutsize metrics for our partitioning problem.

In this paper, we use the column-net and the row-net hypergraph models [5], [8] of a sparse matrix. The column-net hypergraph model $\mathcal{H}_\mathcal{C} = (\mathcal{R}, \mathcal{C})$ of an $m \times n$ sparse matrix M has m vertices and n nets. Each vertex in $\mathcal{R}$ and each net in $\mathcal{C}$ correspond to a row and a column of M, respectively. Furthermore, for a vertex r_i and net c_j, $r_i \in c_j$ if and only if $m_{ij} \neq 0$. A k-way partitioning of the column-net model can be used to permute the matrix M into a singly-bordered form

$$P_r M P_c^T = \begin{pmatrix} M_1 & & & C_1 \\ & M_2 & & C_2 \\ & & \ddots & \vdots \\ & & M_k & C_k \end{pmatrix}, \tag{10}$$

where the permutation matrices P_r and P_c are defined as follows. The matrix P_r permutes the rows of M such that the rows corresponding to the vertices in $\mathcal{V}_i$ come before those in $\mathcal{V}_j$ for $1 \leq i < j \leq k$. The matrix P_c permutes the columns corresponding to the nets that connect only $\mathcal{V}_i$ before those that connect only $\mathcal{V}_j$ for $1 \leq i < j \leq k$, and permutes the columns corresponding to the cut nets to the end. A similar partition where the interface is along the rows can be obtained using a row-net model of M, which is the column-net hypergraph model of M^T.

The hypergraph partitioning problem is NP-hard [19]. There are a number of publicly-available software packages implementing efficient heuristics (e.g., Zoltan [6], PaToH [9], and Mondriaan [24]).

III. Multi-constraint partitioning problems and algorithms

The initial partition (1) has a significant impact on the performance of PDSLin. To obtain (1), PDSLin currently uses a parallel nested graph dissection (NGD) algorithm which is implemented in software packages like PT-Scotch [21] or ParMETIS [17]. In NGD, an input graph is recursively bisected by a set of vertices called separator until a desired number k of parts is obtained. This can be visualized as a binary tree, where the leaf nodes represent the k parts, and the internal nodes represent the separators at each level of bisection. By aggregating the separator vertices into $\mathcal{V}_S$, one can permute the matrix into a doubly-bordered block diagonal form (1). Typically, the subdomain constraint of balancing the subdomain sizes is locally enforced at each bisection. We use the performance of this standard algorithm as the baseline algorithm and compare its performance with those of the proposed partitioning algorithms.

An NGD algorithm often effectively addresses our objective of reducing the separator size. However, at each branch of the tree, the bisection is performed independently from the other bisections, and the imbalance of the subdomains in the global partition may grow as more subdomains are extracted. Furthermore, it does not address most of the load balancing constraints which will be discussed below. Our goal of this section is to develop a hypergraph partitioning algorithm which improves the parallel load balance of PDSLin and reduces its solution time. In Sections III-A and III-B, we first describe a partitioning problem whose objective function and balancing constraints model the runtime of PDSLin. Then, in Section III-C, we outline the hypergraph partitioning algorithm that is designed to globally satisfy our objective function and balancing constraints. We emphasize that this is a very difficult problem. The main challenge is to formulate an objective function and a set of constraints that accurately capture the solution time of PDSLin. The partitioning algorithms in this section work on the symmetrized matrix $|A| + |A|^T$. To simplify the notations, we let A denote this symmetrized matrix in this section.

A. Objective

Our primary objective is to minimize the size of the Schur complement S. This is important since the number of iterations required to solve (2) often increases as the size of S increases. Furthermore, minimizing the size of S is likely to reduce both the number of nonzeros and columns in the interfaces $\widehat{E}_\ell$ (and $\widehat{F}_\ell^T$). Hence, the computation of a smaller $\widetilde{S}$, which approximates S, typically requires a smaller computational and memory costs.

B. Constraints

PDSLin relies on SuperLU_DIST to obtain a good *intra-processor* load balance among the processors assigned to the same subdomain [25]. Therefore, our focus here is the *inter-processor* load balance to compute $\widetilde{S}$ among the processors assigned to different subdomains. Moreover, the runtime of PDSLin is often dominated by the computation of the approximate Schur complement $\widetilde{S}$. Hence, our partitioning algorithm should balance the computational costs associated with different subdomains to compute $\widetilde{S}$. Below, we list some constraints for balancing the costs of computing $\widetilde{S}$:

- *subdomain constraints*: To balance the costs of the LU factorizations of the diagonal blocks D_ℓ, our partitioning algorithms try to balance the dimension of D_ℓ, i.e., $|\mathcal{V}_\ell|$, and/or the number of nonzeros in D_ℓ, i.e., $|\mathcal{E}_\ell|$. Even though the exact cost of an LU factorization is well understood (see [14], [20], for example), it is difficult to assign weights to a vertex, which reflect how much

cost the corresponding row/column will introduce to the LU factorization. Furthermore, these costs depend on the permutation which is used to preserve the sparsity of LU factors, and on the pivoting which is used to enhance the numerical stability of factorization. Neither permutation nor pivoting can be determined until the partition is computed.

- *interface constraints*: Balancing the cost of solving the sparse triangular systems (5) requires not only the balanced subdomains but also the balanced interfaces. Specifically, balance in the numbers of columns of $\widehat{E}_\ell$, i.e., $|\mathcal{V}_S^{(\ell)}|$, or balance in the numbers of nonzeros in $\widehat{E}_\ell$, i.e., $|\mathcal{E}_S^{(\ell)}|$, is required. Balancing these interface constraints also helps to balance the cost of sparse matrix-matrix multiplication (5), which can become expensive when many subdomains are generated.

C. Proposed partitioning algorithms

Hypergraph-based algorithms have been proposed to reorder a matrix A into the doubly-bordered form (1). These approaches first use a structural factorization of A. For example, the approach discussed in [7] uses the structural factorization

$$\text{str}(A) \equiv \text{str}(M^T M)\ , \tag{11}$$

where $\text{str}(A)$ represents the nonzero structure of a matrix A. Once this factorization is obtained, the column-net hypergraph $\mathcal{H}_\mathcal{C}(M)$ of M is used to obtain $P_r M P_c^T$ in the singly-bordered form (10). Subsequently, we have

$$\begin{array}{l} \text{str}(P_c A P_c^T) \\ \equiv \text{str}\begin{pmatrix} M_1^T M_1 & & & M_1^T C_1 \\ & \ddots & & \vdots \\ & & M_k^T M_k & M_k^T C_k \\ C_1^T M_1 & \dots & C_k^T M_k & \sum_{\ell=1}^k C_\ell^T C_\ell \end{pmatrix}. \end{array} \tag{12}$$

When we use the cut-net metric (8) with unit vertex weights, a k-way column-net hypergraph partitioning of $\mathcal{H}_\mathcal{C}(M)$ minimizes the separator size in $P_c A P_c^T$ and balances the number of rows in the blocks of $P_r M P_c$. Unfortunately, this constraint does not satisfy any of our balancing constraints, and it has been experimentally shown that the imbalance in the diagonal block sizes can be much greater than that of NGD [18]. In [7], a partitioning method that balances the number of columns in the diagonal blocks of $P_r M P_c$ was proposed. This would balance the subdomain sizes in $P_c A P_c^T$. However, its implementation is not publicly available, and it does not balance the number of nonzeros in the diagonal blocks nor addresses our interface constraints.

To satisfy our specific objective function and balancing constraints, we propose a recursive hypergraph bisection (RHB) method which is based on the column-net hypergraph $\mathcal{H}_\mathcal{C}(M)$ to permute the matrix $M^T M$ into the doubly-bordered form (12). As described above, we have multiple balancing constraints (subdomains constraints and interface constraints); furthermore, these constraints cannot be assessed from a set of priorly given vertex weights (they are said to be *complex* [18]). To overcome these challenges, we first use, at each bisection, the information from the previous bisection steps to dynamically assign vertex weights which approximately capture the balancing constraints in Section III-B. We then use a multi-constraint hypergraph bisection algorithm to approximately satisfy the exact balancing constraints at each bisection step, even though the weights to capture the exact balances can only be determined after the bisection. Since we do not have any information at the first-level bisection, a unit weight is assigned to each vertex. We illustrate this framework in Algorithm 2. We have studied many weighting schemes and found that the following two weights to be most effective:

- $w_1(i) = \text{nnz}(M_\ell(i,:))$: An upper-bound on the number of nonzeros in D_ℓ for the current partition is given by $\sum_{v(i)\in M_\ell} w_1(i)^2$. Hence, this weight tries to balance the numbers of nonzeros in the subdomains after the next-level bisection by predicting them based on the current partition.
- $w_2(i) = \text{nnz}(M(i,:))$: This is simply the nonzeros in the corresponding row in the matrix M. An upper-bound on the total number of nonzeros introduced in the ℓ-th interface and separator by $v(i) \in M_\ell$ of the current partition is $\sum_{v(i)\in M_\ell}(w_2(i)^2 - w_1(i)^2)$. Hence, this weight is designed to balance the numbers of nonzeros in the interfaces when it is used as a complementary constraint to $w_1(i)$.

These weights can be used as either single or multiple constraints at each bisection. Notice that the weights $w_1(\cdot)$ changes at each bisection step, and RHB is different from a standard partitioning method with static vertex weights. We did not try to use $w_2(\cdot)$ as a single constraint alone because this is equivalent to the standard hypergraph partitioning methods.

In this RHB algorithm, we can use any of the three standard cut-metrics discussed in Section II, which have the following meanings in our particular partitioning problem:

- `con1` of (7): This corresponds to the total number of

Input: A: a sparse matrix. R: row indices. C: column indices. K: number of parts. low, up: id of the lowest and highest numbered parts.
Output: $partition$: partition information for the rows

1: Form the model of the matrix $A(R, C)$
2: **if** this is not the first bisection step **then**
3: Use previous bisection information to set up the constraints
4: **end if**
5: Partition into two $\langle R_1, R_2\rangle \leftarrow$BISECTROWS$(A(R, C))$ // with standard tools
6: Set $partition(R_1) \leftarrow low$ and set $partition(R_2) \leftarrow up$
7: Create the two column sets, either use net splitting or net discarding, giving C_1 and C_2
8: RHB$(A, R_1, C_1, K/2, low, (low + up - 1)/2)$
9: RHB$(A, R_2, C_2, K/2, (low + up - 1)/2 + 1, up)$

Fig. 2. Algorithm RHB(A, R, C, K, low, up).

nonzero columns in the interfaces $C_1, C_2, \ldots, C_k$ of M, and gives an upper-bound on the total number of nonzero columns in the interfaces $E_1, E_2, \ldots, E_k$ of A since $E_i = M_i^T C_i$.
- `cnet` of (8): This corresponds to the number of columns in C_ℓ, which is the separator size of $P_c A P_c^T$.
- `soed` of (9): This sums the above two functions, and tries to minimize both the separator size and the total number of nonzero columns in the interfaces at the same time.

The implementation of the `con1` and `cnet` metrics have been previously described in details [9], [18]. On the other hand, the implementation of the `soed` metric was not discussed neither in [9] nor in [18]. Therefore, we summarize our implementation here. Initially we set the cost of each net to be two. Then, when a net is cut during bisections, we divide its cost by two and round up the cost to the next smallest integer; this implies that the cost of a net is either two (the net is not cut) or one (the net is cut). We then proceed the recursive bisection using the following net-splitting technique described in [9]: for a cut net n whose vertices are in the two parts $\mathcal{V}_A$ and $\mathcal{V}_B$, a net $n_A = n \cap \mathcal{V}_A$ is put in part A and another one $n_B = n \cap \mathcal{V}_B$ is put in part B to continue with recursive bisections. When a net is cut, two new nets with unit costs are created; therefore, the total cost of the nets that represent the same net in the initial hypergraph is the connectivity λ of that net. Therefore, summing the cost of all cut nets provides the `soed` metric.

In Section V, we present numerical results to demonstrate that this RHB algorithm satisfies the balancing constraints of Section III-B better than the NGD algorithm, while increasing the separator size only slightly. As a result, the runtime of PDSLin was reduced using the RHB algorithm. An approach similar to RHB was described in [18] for some other partitioning problems.

IV. Reordering sparse right-hand sides for triangular solution

When solving the triangular systems to form G_ℓ as in (5), we exploit the sparsity of the right-hand side (RHS) columns $\widehat{E}_\ell$. A similar argument follows for forming W_ℓ with $\widehat{F}^T$. Furthermore, since there could be thousands of columns in $\widehat{E}_\ell$, these columns are partitioned into m parts, and the triangular system is simultaneously solved for the multiple columns within each part. There are several advantages of the simultaneous solution with multiple columns: 1) the symbolic algorithm needs to be invoked only once for a part, 2) the total number of messages is reduced, and 3) the data locality of accessing the L-factor may be improved. However, the disadvantage is that we need to pad zeros so that these multiple columns have the same nonzero pattern. This introduces unnecessary operations with zeros. To minimize the number of padded zeros, in this section, we develop two techniques to reorder the columns of $\widehat{E}_\ell$ so that the structural similarity among the adjacent columns is maximized. The first technique uses insights from the sparse linear algebra to develop an efficient heuristic. In order to minimize the total number of padded zeros in the m parts, the second technique transforms the problem into a standard hypergraph partitioning problem, for which there exists effective heuristics and tool support.

In the rest of this section, we drop the subscript ℓ in D_ℓ, G_ℓ, $\widehat{E}_\ell$, and use ℓ to denote the ℓ-th part of the m-way partition of $\widehat{E}$. Detailed discussion of our triangular solver implementation can be found in [25].

A. Reordering based on elimination tree structure

The first technique is based on a postordering of the elimination tree (e-tree for short) of the matrix D; for an unsymmetric D, we use the e-tree of the symmetrized matrix $|D| + |D^T|$. Each node of e-tree corresponds to a column of the matrix. The e-tree structure gives the column dependency during the factorization of D. Moreover, it can be used to determine where the nonzero fill-ins would be generated during the triangular solution $D^{-1}b$ when b is a sparse vector. Specifically, if the i-th element $b(i)$ of b is nonzero, then the fill-ins will be generated at the positions corresponding to the nodes on the fill-in path from the i-th node to the root of the e-tree [14].

Our reordering technique works as follows. Given the e-tree of D, we permute the rows and columns of D so that the corresponding nodes of the e-tree are in a postorder, that is, all the nodes in a subtree are numbered consecutively before the root of the subtree. We then permute the rows of the RHS columns $\widehat{E}$ conforming to the row permutation of D. Finally, the columns of $\widehat{E}$ are permuted such that their row indices of the first nonzeros are in ascending order. The reason this ordering may reduce the number of padded zeros is the following. Let i and j be the first nonzero indices in two consecutive columns. Since the RHS columns are sorted by the first nonzero row indices, the two nodes i and j are likely to be close together in the postordered e-tree, and the two fill-in paths from the i-th node and the j-th node are likely to have a large degree of overlapping in the e-tree. As a result, this reordering technique is likely to increase the structural similarity among consecutive columns. This simple heuristic is easy to implement and is effective in practice. However, it only considers the first nonzeros in the columns, and ignores the fill-ins generated by the other nonzeros. Furthermore, it is based on the properties of the elimination tree and the input and the output of sparse triangular matrix solutions procedures. Hence, it is not applicable to maximize the structural similarity among the adjacent columns of a general sparse matrix. Similar topological orderings have been previously used for triangular solution with multiple sparse RHSs, nullspace computations, and computing elements of the inverse of a sparse matrix [3], [22].

B. Reordering based on a hypergraph model

In order to minimize the total number of padded zeros in the m parts, we propose a reordering technique based on a hypergraph model which can be applied to general sparse matrices. To partition the RHS columns $\widehat{E}$ into m parts, we use the row-net hypergraph model of the solution vectors G, whose nonzero structure is obtained by a symbolic triangular

solution. Our goal is to partition the columns of $\widehat{E}$ into m parts, where the similarity of the row structure among the corresponding columns of G in the same part is maximized.

Let B be the number of columns in each part, and consider a partition $\Pi_m = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m\}$ of the columns of G into m parts. To simplify our discussion, we assume that the number of columns is divisible by B. Let r_i denote the set of columns of G, whose i-th row entry is nonzero, i.e., $r_i \equiv \{j : G(i,j) \neq 0\}$. Then, for a given part $\mathcal{V}_\ell$, the zeros to be padded in the i-th row is given by the formula

$$cost(r_i, \mathcal{V}_\ell) = \begin{cases} |\mathcal{V}_\ell| - |r_i \cap \mathcal{V}_\ell| & \text{if } r_i \cap \mathcal{V}_\ell \neq \emptyset \\ 0 & \text{otherwise} \end{cases} . \quad (13)$$

If the i-th row does not have any nonzero in any columns of $\mathcal{V}_\ell$, then clearly no zeros are padded in the i-th row of $\mathcal{V}_\ell$. On the other hand, if $\mathcal{V}_\ell$ has a nonzero in the i-th row, then for each column in $\mathcal{V}_\ell$ for which $G(i,j) = 0$, there will be a padded zero. Hence, this cost function counts the number of padded zeros in the i-th row of $\mathcal{V}_\ell$. The total cost of Π_m is the total number of padded zeros and given by

$$cost(\Pi_m) = \sum_{i=1}^{n_G} \sum_{\mathcal{V}_\ell \in \Lambda_i} (|\mathcal{V}_\ell| - |r_i \cap \mathcal{V}_\ell|) , \quad (14)$$

where n_G is the number of rows in G.

Since each part has B columns, the cost function (14) reduces to

$$cost(\Pi_m) = \sum_{i=1}^{n_G} (\lambda_i B - |r_i|) . \quad (15)$$

We can further manipulate the formula (15) and obtain

$$\sum_{i=1}^{n_G} (\lambda_i B - |r_i|) = \sum_{i=1}^{n_G} \lambda_i B - nnz(G) = \sum_{i=1}^{n_G} (\lambda_i - 1)B + \sum_{i=1}^{n_G} B - nnz(G) = \sum_{i=1}^{n_G} (\lambda_i - 1)B + n_G B - nnz(G) .$$

Hence, for a given G, the cost function (15) and the connectivity-1 metric (7) with each net having the constant cost of B differ only by the constant value $n_G B - nnz(G)$. Therefore, one can minimize (15) by minimizing (7).

In our numerical experiments, we used PaToH to partition the first $m \times B$ columns of G enforcing each part to have B columns by setting the imbalance parameter ε of (6) to be zero. The remaining columns of G are gathered into one part at the end.

V. Numerical results

We now present numerical results of partitioning and reordering techniques described in this paper. The numerical experiments were conducted on a Cray XE6 machine at the National Energy Research Scientific Computing Center (NERSC). Each node of the machine has two 12-core AMD 2.1GHz Magny-Cours processors and 32GB of memory. For our experiments, we used eight cores on each node and up to 32 cores (for experimental results using up to 1024 cores, see [28]). The codes were written in C, and the **pgcc** compiler with **-fastsse** optimization flag were used to compile the codes. The test matrices were taken from the numerical simulations of modeling particle accelerator cavities [2], the Tokamak design for fusion energy [1] and the circuit simulations [11]. Some properties of the test matrices are shown in Table I.

TABLE I
Test matrices.

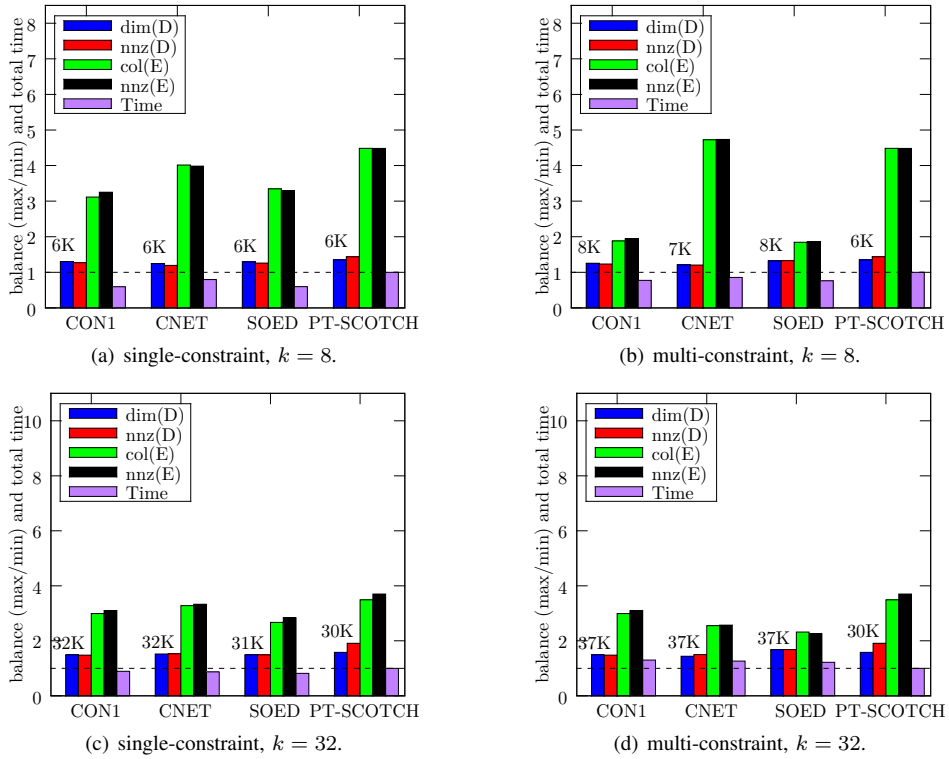
name	source	n	nnz/n	symmetry pattern	symmetry value	pos. def.
tdr190k	cavity	$1,110,242$	39	yes	yes	no
tdr455k	cavity	$2,738,556$	41	yes	yes	no
dds.quad	cavity	$380,698$	42	yes	yes	no
dds.linear	cavity	$834,575$	16	yes	yes	no
matrix211	fusion	$801,378$	70	no	no	no
ASIC_680ks	circuit	$682,712$	2	yes	no	no
G3_circuit	circuit	$1,585,478$	5	yes	yes	yes

As discussed in Section III-B, PDSLin relies on SuperLU_DIST to obtain a good intra-processor load balance, and our focus in this paper is the inter-processor load balance. Hence, in this section, we present the numerical results of PDSLin running in a one-level parallel configuration, that is, the number of processors is the same as the number of subdomains. In this way, the intra-processor load balance does not interfere with the inter-processor load balance that is encapsulated as the constraints of the partitioning problems discussed. We note that in two-level parallel configuration, the improvements in the running time can be higher (see Fig. 1 and our technical report [28] where the overall running time is reduced significantly).

A. Partitioning with multiple constraints

Figure 3 compares the performance of our proposed RHB algorithm using the three cut-metrics `con1`, `cnet`, and `soed` with that of the NGD algorithm of PT-Scotch for the matrix **tdr190k**. We performed two sets of experiments: one with 8 subdomains (the top two plots (a) and (b)), and the other with 32 subdomains (the bottom two plots (c) and (d)). The load balance metric is computed as W_{max}/W_{min} among all the subdomains. Since PT-Scotch was used as our baseline algorithm, we include its data in every plot in the rightmost group of bars, even though they are the same in (a) and (b), and in (c) and (d), respectively. The last bar of each group of bars shows the solution time of PDSLin, which is normalized to the baseline time of the NGD algorithm. The number above each group of the bars is the separator size.

In the figure, we see that both the single-constraint and the multi-constraint RHB algorithms improved both subdomain and interface balances with only a modest increase in the separator size, although the increase was slightly greater using the multi-constraint algorithms. As a result, using the RHB algorithm, the solution time of PDSLin was reduced from that using the NGD algorithm. For example, with $k = 8$ and $k = 32$, the solution time was reduced by a factor of 1.68 and 1.22, respectively, using the single constraint algorithm with the `soed` metric.



(a) single-constraint, $k = 8$. (b) multi-constraint, $k = 8$.

(c) single-constraint, $k = 32$. (d) multi-constraint, $k = 32$.

Fig. 3. Load balancing and solution time with matrix **tdr190k**. In the legend, "dim(D)" refers to the dimension of D_ℓ, "nnz(D") refers to the number of nonzeros in D_ℓ, "col(E)" refers to the number of nonzero columns in E_ℓ, "nnz(E)" refers to the number of nonzeros in E_ℓ, and "time" refers to the total runtime of PDSLin. The number on top of each group of bars is the separator size.

Let us summarize the above results. When the number of subdomains was moderate (e.g., $k = 32$), the RHB algorithm improved the load balance which well offset the modest increase in the separator size. As a result, the runtime of PDSLin was reduced. This is encouraging since in a two-level parallel configuration, the modest number of subdomains gives the best trade-off between the parallelism and the small size of Schur complement (hence fast convergence), and allows us to efficiently solve large-scale linear systems using a large number of processors. We also note that the single-constraint RHB algorithm usually gave a better result than the multi-constraint algorithm. Table II shows the partitioning results of the other test matrices using the single-constraint algorithm and the `soed` metric. Under "Time," we separately report the time needed for the preconditioner computation and for the iterative solution. We see that in comparison with the NGD algorithm, the balance in the subdomain dimensions, which were not directly addressed by our partitioning algorithm, got worse for the matrices from the fusion and circuit simulations. On the other hand, the balance in the numbers of nonzeros in both subdomains and interfaces were improved. Furthermore, the RHB algorithm reduced the size of the Schur complement for the two matrices from circuit simulations, in particular for the matrix **ASIC_680ks**, yielding a better runtime. For the **dds** matrices, the RHB algorithm had less significant improvements. At the end, the RHB algorithm obtained the speedups between 1.08 and 8.58.

B. *Reordering sparse right-hand-side vectors*

We now study the performance of the two reordering techniques, namely the technique based on the postordering of the elimination tree and the technique based on the hypergraph model. For our numerical experiments, we used the parallel nested dissection algorithm of PT-Scotch to extract eight subdomains and used a minimum degree ordering on each subdomain to preserve sparsity of its LU factors (a very common setting in direct and hybrid linear solvers). Due to the limited space, in this section, we only present the results of the matrices from accelerator and fusion simulations. Some statistics of the partitions are shown in Table III. As discussed in Section IV, the objective of the reordering is to reduce the number of the padded zeros in the solution vectors and to reduce the triangular solution time.

a) Fraction of the padded zeros: We first examine the effects of the reordering techniques on the number of the padded zeros in the supernodal blocks. In Figure 4, we show the fraction of padded zeros as a function of the block size B, where for each block size with each algorithm, the markers show the minimum, average, and maximum fraction of padded zeros (over the eight subdomains that we have generated). In general, it is easier to find a good reordering to reduce the number of padded zeros for a smaller block size B. This is because as more columns are included in each part, it becomes harder to find the columns with similar sparsity structures. In the extreme case of block size of one, there is no padded zero.

TABLE II
PARTITIONING STATISTICS OF THE EIGHT INTERIOR SUBDOMAINS USING THE SINGLE-CONSTRAINT ALGORITHM AND THE SOED METRIC. "n_{D_ℓ}" AND "NNZ$_{D_\ell}$" ARE THE DIMENSION AND NUMBER OF NONZEROS OF D_ℓ RESPECTIVELY;

Matrix	Alg.	Time (s)	#Iter.	n_S $\times 10^2$		n_{D_ℓ} $\times 10^3$	nnz$_{D_\ell}$ $\times 10^3$	nnzcol$_{E_\ell}$ $\times 10^0$	nnz$_{E_\ell}$ $\times 10^0$
dds.quad	NGD	98.3+5.5	18	95	min	35	1408	980	18792
					max	58	2372	3292	61880
	RHB	90.4+5.3	19	99	min	37	1504	956	17548
					max	58	2162	3614	66416
dds.linear	NGD	108.7+7.5	11	44	min	87	1355	305	1695
					max	114	1792	2593	14622
	RHB	100.7+6.7	10	38	min	87	1346	305	1685
					max	112	1762	2267	12566
matrix211	NGD	89.8+8.9	17	121	min	80	3328	1290	15480
					max	106	8782	5580	133056
	RHB	73.3+9.9	18	130	min	78	6290	1428	17136
					max	173	7223	4380	104256
ASIC_680ks	NGD	34.3+0.5	1	92	min	84	133	864	5223
					max	85	201	4493	13024
	RHB	4.0+3.5	19	11	min	76	179	98	1812
					max	104	253	431	4501
G3_circuit	NGD	26.3+6.9	11	66	min	192	925	975	1718
					max	205	985	2493	3944
	RHB	22.9+5.3	8	51	min	193	933	899	1749
					max	201	969	1750	3300

As a result, we clearly see that the fraction of the padded zeros increases as B increases.

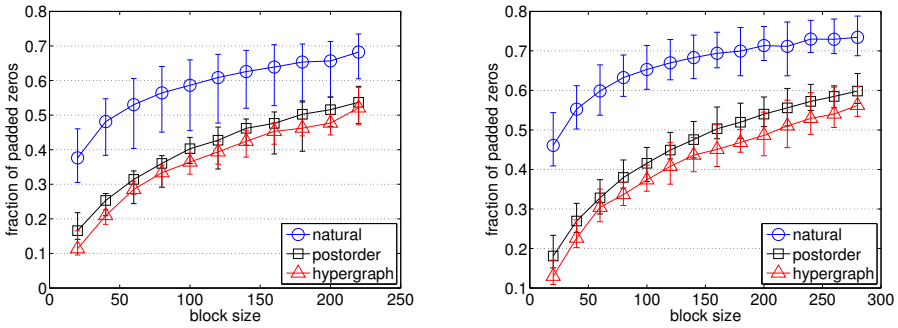
In Figure 4, the "natural ordering" is in fact the nested dissection ordering of the global matrix, and it obtained a reasonable performance by reducing the fill-ins in the interface. However, we see that postordering the RHS vectors can significantly reduce the fraction of the padded zeros, and the hypergraph ordering can further reduce the padded zeros in some cases. For the matrices **tdr190k** and **dds.quad**, the hypergraph ordering induced less padded zeros than the postordering for all the block sizes, while for the matrix **dds.linear**, the hypergraph ordering was better than the postordering for almost all the block sizes. However, the situation was quite different for **matrix211**, and the postordering globally performed better than the hypergraph ordering. We believe this is mainly because the interfaces of **matrix211** are much sparser as shown by both the effective density and the fill-ratio in Table III. The larger effective density provides more chance for the reordered columns to have similar row structures. The hypergraph model seems to exploit this property better than the postordering, and it obtained greater improvements for **tdr190k**. On the other hand, the postordering takes only the first nonzero positions into account ignoring the other fill-in positions. This works reasonably well if the fill-ratio is small, which was precisely the case for **matrix211**.

b) Triangular solution time: Figure 5 shows the total solution time spent to compute $L_\ell^{-1}E_\ell$ after the three reordering techniques are applied. As in Figure 4, for each block size and each reordering technique, we show the minimum, average, and maximum solution time among the eight subdomains. The best time was obtained with B around 60, which is the default in PDSLin. For matrices larger than the ones shown in this section, the triangular solution time increases significantly, and the triangular solves can become the computational bottleneck. Hence, we are more interested in the speedups gained over the natural ordering than the actual solution time. The figure shows that the hypergraph ordering often obtains a greater speedup as the problem becomes more difficult with a larger B. In Figure 5, the triangular solution time was reduced by a factor of up to 1.3 using the hypergraph ordering from that using the natural ordering.

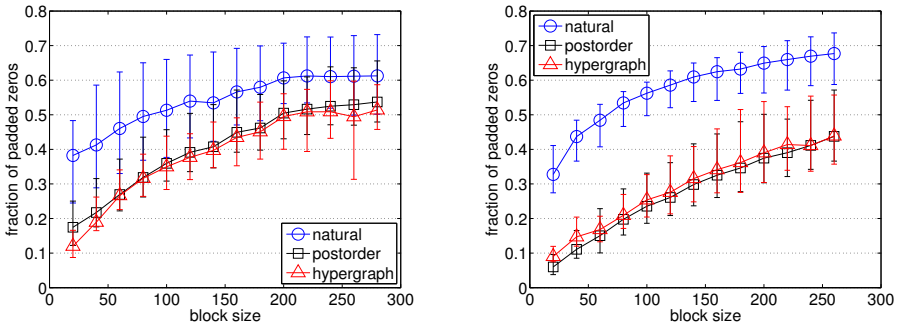
TABLE III
STATISTICS OF THE EIGHT INTERIOR SUBDOMAINS AND INTERFACES OF THE TEST MATRICES. "NNZCOL$_{G_\ell}$" ("NNZROW$_{G_\ell}$") IS THE NUMBER OF COLUMNS (ROWS) WITH AT LEAST ONE NONZERO IN G_ℓ. "EFF. DENS." IS THE EFFECTIVE DENSITY GIVEN BY NNZ$_{G_\ell}$/(NNZCOL$_{G_\ell}$ $\times$ NNZROW$_{G_\ell}$), AND "FILL-RATIO" IS NNZ$_{G_\ell}$/NNZ$_{E_\ell}$.

		nnz$_{G_\ell}$ $\times 10^6$	nnzcol$_{G_\ell}$ $\times 10^3$	nnzrow$_{G_\ell}$ $\times 10^3$	eff. dens. $\times 10^{-2}$	fill-ratio
tdr190k	min	3.55	0.60	23.0	2.20	186
	max	7.64	2.12	30.4	4.66	338
dds.quad	min	4.99	1.02	25.2	14.5	139
	max	13.7	3.25	13.9	39.1	290
dds.linear	min	1.46	0.31	6.62	33.6	830
	max	13.0	1.95	20.7	73.1	1330
matrix211	min	0.38	1.58	12.5	1.10	20
	max	4.44	4.74	35.0	3.71	42

c) Effect of removing quasi-dense rows: We say that a row is *quasi-dense* if the fraction of the number of nonzeros is greater than or equal to a density threshold τ. We found that the majority of the rows in the solution vectors are sparse. For example, when eight subdomains are extracted from **tdr190k** test matrix, with $\tau = 0.4$, only about 15% of the rows were quasi-dense. However, the time for computing the hypergraph partitioning can be significantly reduced by removing the empty and quasi-dense rows (we observed factors up to 4), while the fraction of padded zeros (i.e., the quality of the partitioning) is largely independent of the threshold until the threshold becomes too small (i.e., $\tau < 0.1$). Therefore, the total solution time (setup time for hypergraph partitioning and sparse triangular solution) can be reduced by removing the quasi dense rows. For more detailed studies, we refer the

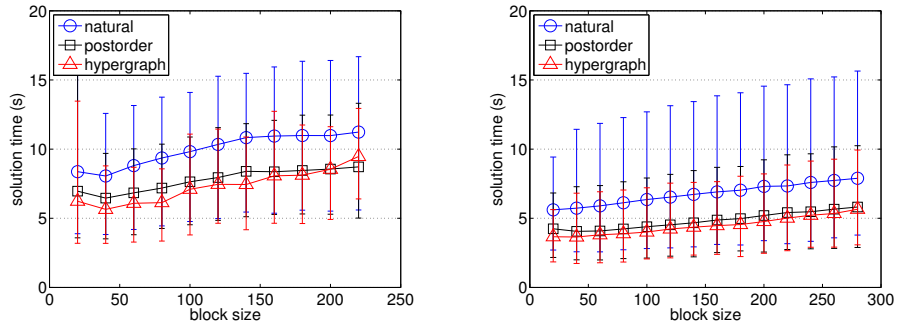


(a) **tdr190k**. (b) **dds.quad**.

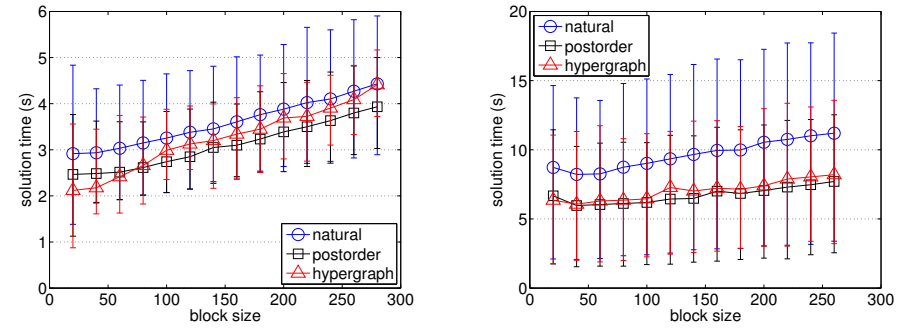


(c) **dds.linear**. (d) **matrix211**.

Fig. 4. Fraction of the padded zeros using different block size B with three different reordering techniques.



(a) **tdr190k**. (b) **dds.quad**.



(c) **dds.linear**. (d) **matrix211**.

Fig. 5. Sparse triangular solution time using different block size B with three different reordering techniques.

reader to the accompanying technical report [28].

VI. Conclusion

We studied two combinatorial problems to enhance the performance of a hybrid linear solver PDSLin that is based on the Schur complement method. First, we have designed the Recursive Hypergraph Bisection partitioning algorithm with various metrics to improve parallel load balance. We presented the numerical results of the new algorithms using the matrices from several applications (more extensive results can be found in our technical report [28]). Among these algorithms, the most promising one was based on a RHB using single constraints with dynamic vertex weights assigned at each bisection step, and either with `soed` or `cnet` cut-metrics. When the number of subdomains is moderate (e.g., $k = 32$), in comparison to a state-of-the-art nested graph bisection algorithm, our RHB algorithm improved the load balance which well offset the modest increase in separator size. As a result, the runtime of PDSLin was reduced by up to 1.68x.

For our experiments, we used the serial partitioner PaToH and a serial algorithm to compute the structural decomposition (11) of A. To avoid these serial bottlenecks, we plan to investigate the use of a parallel partitioner and develop an efficient algorithm to compute the structural decomposition (11).

We also studied two sparse RHS reordering strategies to improve the performance of a supernodal triangular solver, which is used to eliminate the unknowns associated with each interface; one based on a postordering of the elimination tree and the other based on a hypergraph partitioning. The numerical results have shown that these strategies reduce the number of padded zeros in the RHSs, and reduces the runtime of the triangular solver by up to 1.3x.

Acknowledgments

This research was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. DOE under Contract No. DE-AC02-05CH11231, and 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. Most of the work of Ichitaro Yamazaki was done while he was at Lawrence Berkeley National Laboratory.

References

[1] "Center for Extended MHD Modeling," http://w3.pppl.gov/cemm/.

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

[3] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, Y. Robert, F.-H. Rouet, and B. Uçar, "On computing inverse entries of a sparse matrix in an out-of-core environment," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. A1975–A1999, 2012.

[4] C. Aykanat, B. Cambazoglu, and B. Uçar, "Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices," *Journal of Parallel and Distributed Computing*, vol. 68, pp. 609–625, 2008.

[5] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, "Permuting sparse rectangular matrices into block-diagonal form," *SIAM Journal on Scientific Computing*, vol. 25, no. 6, pp. 1860–1879, 2004.

[6] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco, *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide*, Sandia National Laboratories, Albuquerque, NM, 2007.

[7] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, "Hypergraph partitioning-based fill-reducing ordering for symmetric matrices," *SIAM J. Scientific Computing*, vol. 33, no. 4, pp. 1996–2023, 2011.

[8] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, Jul 1999.

[9] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/~umit/software.htm, 1999.

[10] C. Chevalier and F. Pellegrini, "PT-Scotch," *Parallel Computing*, vol. 34, no. 6–8, pp. 318–331, 2008.

[11] T. A. Davis, "University of Florida Sparse Matrix Collection," http://www.cise.ufl.edu/research/sparse/matrices.

[12] J. Demmel, J. Gilbert, and X. Li, "SuperLU Users' Guide," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-44289, September 1999, http://crd.lbl.gov/~xiaoye/SuperLU/. Last update: September 2007.

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

[14] J. Gilbert, "Predicting structure in sparse matrix computations," *SIAM J. Matrix Analysis and Applications*, vol. 15, pp. 62–79, 1994.

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

[16] G. Karypis and V. Kumar, *hMeTiS: Hypergraph and Circuit Partitioning*, University of Minnesota. http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview.

[17] G. Karypis, K. Schloegel, and V. Kumar, "ParMeTiS: Parallel graph partitioning and sparse matrix ordering," University of Minnesota. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

[18] K. Kaya, F.-H. Rouet, and B. Uçar, "On partitioning problems with complex objectives," in *Euro-Par 2011: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, vol. 7155. Springer, 2012, pp. 334–344.

[19] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, U.K.: Wiley–Teubner, 1990.

[20] J. W. H. Liu, "The role of elimination trees in sparse factorization," *SIAM J. Matrix Anal. Appl.*, vol. 1, pp. 134–172, 1990.

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

[22] Tz. Slavova, "Parallel triangular solution in an out-of-core multifrontal approach for solving large sparse linear systems," Ph.D. dissertation, Institut National Polytechnique de Toulouse, Toulouse, France, 2009.

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

[24] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005.

[25] I. Yamazaki and X. S. Li, "On techniques to improve robustness and scalability of a parallel hybrid linear solver," in *Proceedings of the ninth international meeting on high performance computing for computational science, Springer's Lecture Notes in Computer Science*, 2010, pp. 421–434.

[26] I. Yamazaki, X. S. Li, and E. G. Ng, "Preconditioning schur complement systems of highly-indefinite linear systems for a parallel hybrid solver," in *Numerical Mathematics: Theory, Methods and Applications*, vol. 3, no. 3, 2010, pp. 352–366.

[27] ——, "PDSLin Users Guide," lawrence Berkeley National Laboratory, Tech. Rep. LBNL-4825E, June 2011.

[28] I. Yamazaki, X. S. Li, F.-H. Rouet, and B. Uçar, "Partitioning, Ordering and Load Balancing in a Hierarchically Parallel Hybrid Linear Solver," Institut National Polytechnique de Toulouse, Toulouse, France, Technical Report RT-APO-12-2, octobre 2011.