SIAM J. SCI. COMPUT.
Vol. 29, No. 4, pp. 1683–1709
© 2007 Society for Industrial and Applied Mathematics

PARTITIONING SPARSE MATRICES FOR PARALLEL PRECONDITIONED ITERATIVE METHODS*

BORA UÇAR† AND CEVDET AYKANAT‡

Abstract. This paper addresses the parallelization of the preconditioned iterative methods that use explicit preconditioners such as approximate inverses. Parallelizing a full step of these methods requires the coefficient and preconditioner matrices to be well partitioned. We first show that different methods impose different partitioning requirements for the matrices. Then we develop hypergraph models to meet those requirements. In particular, we develop models that enable us to obtain partitionings on the coefficient and preconditioner matrices simultaneously. Experiments on a set of unsymmetric sparse matrices show that the proposed models yield effective partitioning results. A parallel implementation of the right preconditioned BiCGStab method on a PC cluster verifies that the theoretical gains obtained by the models hold in practice.

Key words. matrix partitioning, preconditioning, iterative method, parallel computing

AMS subject classifications. 05C50, 05C65, 65F10, 65F35, 65F50, 65Y05

DOI. 10.1137/040617431

1. Introduction.

We consider the parallelization of the preconditioned iterative methods that use explicit preconditioners such as approximate inverses or factored approximate inverses. Our objective is to develop methods for obtaining one-dimensional (1D) partitions on a coefficient matrix and a preconditioner matrix or factors of a preconditioner matrix simultaneously to efficiently parallelize a full step of the preconditioned iterative methods. We assume preconditioner matrices or their sparsity patterns are available beforehand. It has been shown that the rates of convergence of iterative methods depend on the partitioning method when the preconditioners are built from partitioned coefficient matrices [26]. With the above assumption in mind, we neither deteriorate nor improve the effects of the selected preconditioners on the rate of convergence. Our assumption is justified in applications where the preconditioner matrices can be reused; see, for example, [12] and a discussion of it in [10]. The assumption is also justified in the preconditioner constructing methods that require a priori sparsity patterns for the preconditioner matrices [44, 45], where techniques to develop effective sparsity patterns already exist in the literature [23, 24, 39].

Approximate inverse preconditioning techniques explicitly compute and store a sparse matrix $M \approx A^{-1}$ to be used as a preconditioner. Application of such preconditioners requires one or two matrix-vector multiply operations. Two types of approximate inverses exist in the literature. In the first type, an approximate inverse is stored as a single matrix, whereas in the second type it is stored as a product of two matrices. The second type of preconditioners are referred to as factored approximate inverses. Among the most notable approximate inverse preconditioners are AINV and its variants by Benzi et al. [5, 6, 7, 8]; SPAI by Grote and Huckle [33]; FSAI by Kolotilina and Yeremin [44, 45]; and MR by Chow and Saad [25]. See [4, 9, 31] for

*Received by the editors October 21, 2004; accepted for publication (in revised form) January 31, 2007; published electronically August 1, 2007. This work was partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under project 106E069.
http://www.siam.org/journals/sisc/29-4/61743.html
†CERFACS, 42 av. Gaspard Coriolis, 31057, Toulouse Cedex 1, France (ubora@cerfacs.fr).
‡Computer Engineering Department, Bilkent University, Ankara, 06800, Turkey (aykanat@cs.bilkent.edu.tr).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

a recent survey and the use of the approximate inverse preconditioning techniques. See [48, 52] for a general treatment of the preconditioning techniques.

Hendrickson and Kolda [36] give a thorough survey of the graph partitioning models used for partitioning sparse matrices. In these models, a K-way partition of the vertices of a given graph or hypergraph is computed. The partitioning constraint is to maintain a balance criterion on the number of vertices in each part; if the vertices are weighted, then the constraint is to maintain a balance criterion on the sum of the vertex weights in each part. The partitioning objective is to minimize the cutsize of the partition defined over the edges or hyperedges. The partitioning constraint and objective relate, respectively, to maintaining computational load balance and minimizing the total communication volume. Among those models, the hypergraph models by Aykanat, Pınar, and Çatalyürek [2], Çatalyürek and Aykanat [17, 18], and Pınar et al. [49] and the bipartite graph model by Hendrickson and Kolda [37, 43] are said to have more expressive power than the other models [18, 34, 36, 37]. These models have the flexibility of producing unsymmetric partitions on the input and output vectors of the sparse matrix-vector multiplies. A distinct advantage of the hypergraph models over both the standard graph and the bipartite graph models is that the partitioning objective in the hypergraph models is an exact measure of the total communication volume, whereas the objective in the graph models is an approximation [18, 34, 36, 37]. As noted in the survey [36] and in [34], all these graph and hypergraph models, except the bipartite graph model, are used to optimize a single sparse matrix-vector multiply operation. However, a single sparse matrix-vector multiply operation is only a piece of a larger computation in the preconditioned iterative methods. Therefore, new partitioning models that optimize a full step of these iterative methods are needed—this was also stated by Hendrickson [34].

Optimizing a full step of the preconditioned iterative methods requires partitioning the coefficient and preconditioner matrices. This problem has been formulated in terms of bipartite graph partitioning [37]. In the bipartite graph model, each vertex in one part represents a row of A and a column of M, and each vertex in the other part represents a column of A and a row of M. There is an edge between two vertices if the corresponding entries in A or M are nonzero. Partitioning this bipartite graph produces unsymmetric partitions on A and M, where the row partition of A and the column partition of M are the same, and the column partition of A and the row partition of M are the same. The formulation is based on the cut edges and hence does not capture the total communication volume exactly. The multiphase mesh partitioning method of Walshaw et al. [47, 64] and multiconstraint/multiobjective graph partitioning methods of Karypis et al. [41, 53] address the partitioning problem in scientific computations whose computational structures are similar to that of preconditioned iterative methods. These models can be used to partition a graph corresponding to the sparsity pattern of the matrix $A+M$ to find a single partition for both of the matrices. Partitioning such a graph will produce symmetric partition on the coefficient and preconditioner matrices.

In this work, we propose methods to build composite hypergraph models for preconditioned iterative methods from simple hypergraph models for a single matrix-vector multiply [18]. We show how to use the composite models to obtain 1D partitions on a matrix and its approximate inverse preconditioners for optimizing a full step of the preconditioned iterative methods. The composite hypergraph models encode the total communication volume exactly, can handle unsymmetric dependencies, and can produce unsymmetric partitions. Furthermore, they are more powerful than the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

bipartite graph model in the sense that they can produce a variety of partitions on the matrices. For example, given two matrices A and M, it is possible to obtain a symmetric partition on A and a unsymmetric partition on M while optimizing a full step of a preconditioned iterative method.

We review simple hypergraph models which are the building blocks of composite models in section 2. We discuss a procedure to analyze iterative methods in order to determine partitioning requirements for efficient parallelization and illustrate the procedure on a well-known iterative method in section 3. The partitioning requirements of a number of widely used iterative methods are also given in the same section. We propose methods to build composite hypergraph models for meeting the partitioning requirements in the preconditioned iterative methods in section 4. We discuss the applicability of the composite hypergraph models to a few additional scientific applications in section 5. The proposed methods are evaluated in section 6.

2. Preliminaries. The kernel operations in the iterative methods that use approximate inverse preconditioners are matrix-vector multiplies with both coefficient and preconditioner matrices. These multiply operations are performed on vectors that are dependent on each other through linear vector operations. For example, the computations of the form $y \leftarrow AMz$ are performed as $x \leftarrow Mz$ and then $y \leftarrow Ax$. If the partition on the x vector for the first multiply operation is different from that for the second multiply operation, then the x-vector entries should be reordered in between the two multiplies. Since the reordering requires communication, it should be avoided. In order to avoid this reordering operation, there should be only one partition on the intermediate vector x, and the partition should be helpful for both of the parallel multiply operations. We call the problem of finding partitions on two or more matrices in such a way that the common vectors are not reordered in a parallel implementation the simultaneous partitioning problem.

2.1. Parallel multiplies. Given a K-way rowwise partition on a matrix A, a conformable partition on the output vector, and a possibly different K-way partition on the input vector of a matrix-vector multiply operation, the matrix A can be permuted into a block structure:

$$PAQ = A_{BL} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix}. \tag{2.1}$$

Here, K is the number of processor and P and Q are permutation matrices. The partition on the rows of A is used to define the permutation P by permuting the rows belonging to processor P_k before those belonging to P_ℓ for $k < \ell$. The partition on the input vector is used to define the permutation Q by permuting the columns corresponding to the vector entries belonging to processor P_k before those corresponding to the vector entries belonging to P_ℓ for $k < \ell$. In either of the permutations, the order of the rows or columns associated with a common processor can be arbitrary.

Similarly, given a K-way columnwise partition on A, a conformable partition on the input vector, and a K-way partition on the output vector among K processors, the matrix A again can be permuted into a $K \times K$ block structure $PAQ = A_{BL}$ (2.1). Here, P is defined using the partition on the output vector, and Q is defined using the partition on the matrix columns (and input vector).

Let A be of size $m \times n$. Then the block $A_{k\ell}$ in (2.1) is of size $m_k \times n_\ell$, where

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$\sum_k m_k = m$ and $\sum_\ell n_\ell = n$. In a rowwise partitioning, the processor P_k holds the kth row stripe $[A_{k1} \cdots A_{kK}]$ of size $m_k \times n$. In a columnwise partitioning, P_k holds the kth column stripe $[A_{1k}^T \cdots A_{Kk}^T]^T$ of size $m \times n_k$. In the rowwise partitioning, the row stripes should have a nearly equal number of nonzeros for maintaining computational load balance among processors. The same requirement exists for the column stripes in the columnwise partitioning.

2.1.1. Row-parallel multiplies. Consider matrix-vector multiplies of the form $y \leftarrow Ax$, where A is partitioned rowwise. The rowwise partition of matrix A induces a partition on the output vector y, i.e., $y = [y_1^T \cdots y_K^T]^T$, where the processor P_k is set to be responsible for computing the subvector y_k of size m_k. According to the partition on the input vector $x = [x_1^T \cdots x_K^T]^T$, the processor P_k holds the subvector x_k of size n_k. Assume the matrix A has been permuted into the block structure (2.1) using these partitions. Then the *row-parallel* $y \leftarrow Ax$ executes the following steps at processor P_k [37, 57, 59, 61]:

1. For each nonzero off-diagonal block $A_{\ell k}$, send sparse vector $\hat{x}_k^\ell$ to processor P_ℓ, where $\hat{x}_k^\ell$ contains only those entries of x_k corresponding to the nonzero columns in $A_{\ell k}$.
2. Compute the diagonal block product $y_k^k = A_{kk} \times x_k$, and set $y_k = y_k^k$.
3. For each nonzero off-diagonal block $A_{k\ell}$, receive $\hat{x}_\ell^k$ from processor P_ℓ, then compute $y_k^\ell = A_{k\ell} \times \hat{x}_\ell^k$, and update $y_k = y_k + y_k^\ell$.

In step 1, P_k might be sending the same x_k-vector entry to different processors according to the sparsity pattern of column k of A. This multicast-like operation is referred to here as the *expand* operation.

2.1.2. Column-parallel multiplies. Consider matrix-vector multiplies of the form $w \leftarrow Az$, where A is partitioned columnwise. The columnwise partition of A induces a partition on the input vector z, i.e., $z = [z_1^T \cdots z_K^T]^T$, where the processor P_k holds the subvector z_k of size n_k. According to the partition on the output vector $w = [w_1^T \cdots w_K^T]^T$, the processor P_k is set to be responsible for computing the subvector w_k of size m_k. Assume the matrix A has been permuted into the block structure (2.1) using these partitions. Then the *column-parallel* $w \leftarrow Az$ executes the following steps at processor P_k:

1. For each nonzero off-diagonal block $A_{\ell k}$, form sparse vector $\hat{w}_\ell^k$, which contains only those results of $w_\ell^k = A_{\ell k} \times z_k$ corresponding to the nonzero rows in $A_{\ell k}$. Send $\hat{w}_\ell^k$ to processor P_ℓ.
2. Compute the diagonal block product $w_k^k = A_{kk} \times z_k$, and set $w_k = w_k^k$.
3. For each nonzero off-diagonal block $A_{k\ell}$ receive partial-result vector $\hat{w}_k^\ell$ from processor P_ℓ, and update $w_k = w_k + \hat{w}_k^\ell$.

In step 3, the multinode accumulation on the w_k-vector entries is referred to here as the *fold* operation.

2.2. Hypergraph partitioning. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$. Every net is a subset of vertices. The vertices of a net are also called its *pins*. The size of a net n_i is equal to the number of its pins, i.e., $|n_i|$. The set of nets that contain vertex v_j is denoted by $Nets(v_j)$. Weights can be associated with vertices.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is nonempty, i.e., $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$; parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$; and the union of parts gives $\mathcal{V}$, i.e., $\bigcup_k \mathcal{V}_k = \mathcal{V}$. In Π, a net is said to *connect* a part if it has at least one pin

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

in that part. The *connectivity set* Λ_i of a net n_i is the set of parts connected by n_i. The *connectivity* $\lambda_i = |\Lambda_i|$ of a net n_i is the number of parts connected by n_i. A net is said to be cut if it connects more than one part and uncut otherwise. The set of cut and uncut nets is also referred to as external and internal nets, respectively. In Π, the weight of a part is the sum of the weights of vertices in that part.

In the hypergraph partitioning problem, the objective is to minimize the *cutsize*:

$$cutsize(\Pi) = \sum_{n_i \in \mathcal{N}} (\lambda_i - 1). \tag{2.2}$$

This objective function is widely used in the VLSI community [46] and in the scientific computing community [2, 18, 59], and it is referred to as the *connectivity*-1 cutsize metric. The partitioning constraint is to satisfy a balancing constraint on part weights:

$$\frac{W_{max} - W_{avg}}{W_{avg}} \leq \epsilon. \tag{2.3}$$

Here W_{max} is the maximum part weight, W_{avg} is the average part weight, and ϵ is a predetermined imbalance ratio. This problem is NP-hard [46].

A recent variant of the above problem is the multiconstraint hypergraph partitioning [16, 20, 42] in which each vertex has a vector of weights associated with it. In this problem, the partitioning objective is the same as in (2.2), and the partitioning constraint is to satisfy a balancing constraint associated with each weight. Another variant is the multiobjective hypergraph partitioning [1, 53, 55], in which there are several objectives to be minimized. Specifically, a given net contributes different costs to different objectives.

2.3. Hypergraph models for row-parallel and column-parallel multiplies. It is inherent in the parallel matrix-vector multiply algorithms given in section 2.1 and existent in the literature [18, 36, 37, 59] that in partitioning a matrix the key is to find permutation matrices P and Q such that most of the nonzeros of the matrix $PAQ = A_{BL}$ (2.1) are in the diagonal blocks. Here, we propose a slight enhancement of the computational hypergraph models [18] to find permutation matrices P and Q. The proposed enhancement is to add a new set of vertices into the column-net and row-net hypergraph models.

In the column-net hypergraph model, an $m \times n$ matrix A is represented as a hypergraph $\mathcal{H} = (\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ for rowwise partitioning. Vertex and net sets, $\mathcal{V}_\mathcal{R}$ and $\mathcal{N}_\mathcal{C}$, correspond to the rows and columns of A, respectively. There exist one vertex r_i and one net c_j for each row i and column j, respectively. In this model, $r_i \in c_j$ if and only if $a_{ij} \neq 0$. The proposed enhancement is to add n new vertices each representing an input-vector entry of the $y \leftarrow Ax$ multiply. Each new vertex x_j is added to the net c_j, i.e., $c_j = c_j \cup \{x_j\}$ and $Nets(x_j) = \{c_j\}$. Each vertex r_i corresponds to the task of computing the inner product of the row i with the column vector x. Hence, the computational weight associated with the vertex r_i is equal to the number of nonzeros in row i. Each row vertex r_i also represents the vector entry y_i. Weights can be assigned to the vertices in regard to the vector entries. For example, a unit weight may be assigned to the vertex x_j for the corresponding vector entry, and a unit weight may be assigned to the vertex r_i for the vector entry y_i. These weights can be used in a multiconstraint formulation to balance the linear vector operations.

Figure 2.1 shows a sample matrix and its column-net hypergraph model enhanced with the x vertices. In the figure, the white and black circles represent, respectively,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

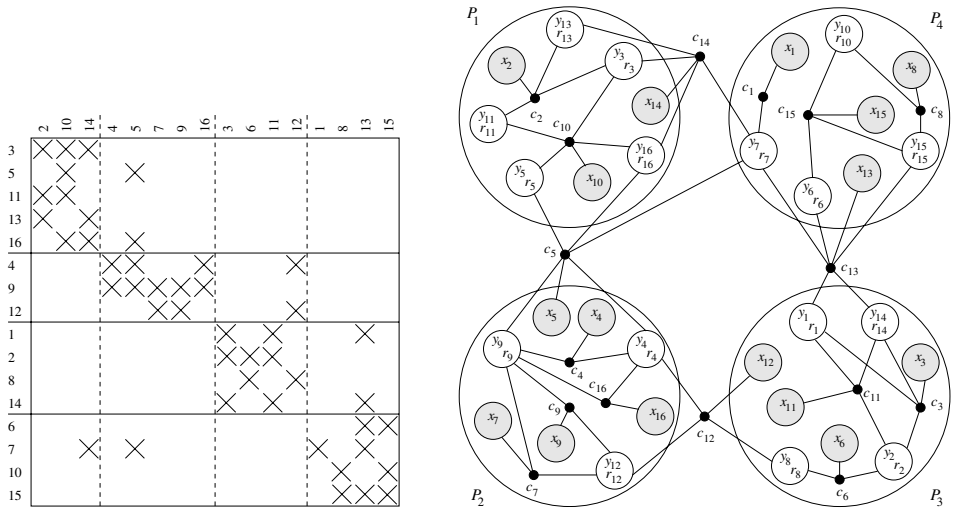


FIG. 2.1. *Matrix A, enhanced column-net hypergraph model for row-parallel $y \leftarrow Ax$, and a four-way rowwise partitioning.*

the vertices and nets of the original column-net hypergraph model [18], and straight lines show pins. The newly added vertices are shown with gray circles. A four-way partition is shown by four big circles encompassing the vertices of the hypergraph.

Given a partition Π on $\mathcal{H}$, the permutations P and Q can be found as follows. The permutation P is partially defined by the partition on the row vertices. The rows corresponding to the row vertices in $\mathcal{V}_k$ are mapped to processor P_k and therefore permuted before the rows corresponding to the row vertices in $\mathcal{V}_\ell$ for $1 \leq k < \ell \leq K$. In Figure 2.1, the permutation of the rows of A is shown by the permuted row indices, where the horizontal solid lines separate row stripes that belong to different processors. The permutation of the rows in the same row stripe can be arbitrary. The permutation Q is partially defined by the partition on the x vertices. The x-vector entries corresponding to the x vertices in $\mathcal{V}_k$ are mapped to processor P_k, and therefore the associated columns are permuted before the columns that are associated with the x vertices in $\mathcal{V}_\ell$ for $1 \leq k < \ell \leq K$. Figure 2.1 shows the permutation on the columns of A, where the vertical dashed lines separate virtual column stripes that are aligned with the x-vector entries belonging to different processors. Again, the permutation of the columns within the same column stripe can be arbitrary. In the figure, the processor P_2 is set to be responsible for computing the inner products of x with the rows in the second row stripe, i.e., the rows 4, 9, and 12. P_2 holds x_4, x_5, x_7, x_9, and x_{16} and thus expands x_5 to the processors P_1 and P_4. Observe that the net c_5 connects the parts P_1, P_2, and P_4. This association between the connectivity of nets and the communication requirements is not accidental, as shown by the following theorem.

THEOREM 2.1. *Let Π be a partition on the enhanced column-net hypergraph model of a given matrix A, and let P and Q be the row and column permutations induced by the partition Π. Then the cutsize of the partition Π quantifies the total communication volume in the row-parallel $y \leftarrow Ax$ multiply.*

Proof. Consider an internal net c_i. Since the net is not cut, the row vertices that need x_i should be in the part that contains the vertex x_i. Hence, no communication occurs for the x-vector entries associated with the internal nets. Consider an external

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

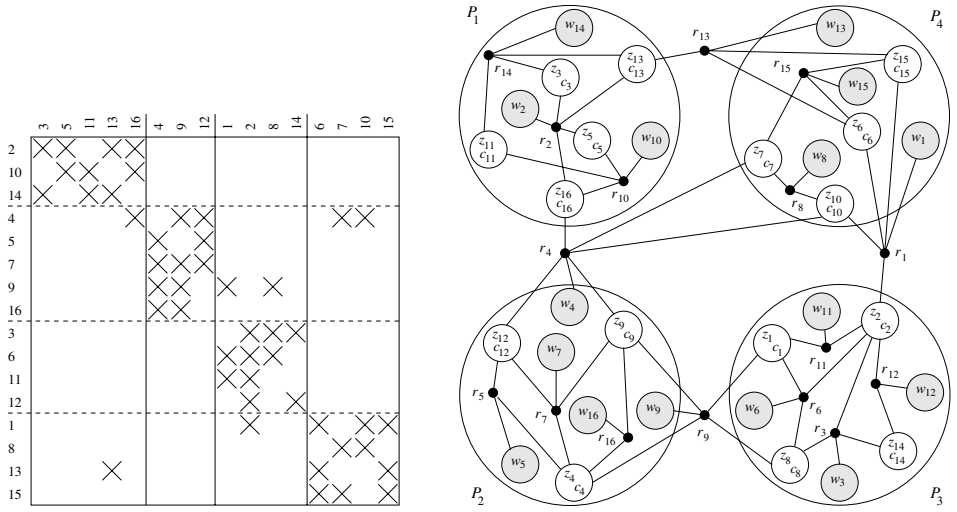


FIG. 2.2. *Matrix A, enhanced row-net hypergraph model for column-parallel $w \leftarrow Az$, and a four-way columnwise partitioning.*

net n_e with the connectivity set Λ_e. The processors corresponding to the parts in the set Λ_e need x_e. One of them owns x_e, since $x_e \in n_e$. The owner should send x_e to the others. That is, for each x_e there are a total of $|\Lambda_e| - 1 = \lambda_e - 1$ messages carrying x_e. The overall sum of these quantities matches the cutsize definition (2.2). □

With similar reasoning, it is concluded in [18] that the hypergraph partitioning objective and constraint correspond, respectively, to minimizing the total communication volume and maintaining the computational load balance. In Figure 2.1, the cutsize and hence the total communication volume is five units, and the part weights and hence the computational loads of the processors are 12, 12, 11, and 11.

The row-net hypergraph model [18] can be enhanced by adding m new vertices, each representing a w-vector entry to find the permutation matrices P and Q for the column-parallel $w \leftarrow Az$ multiplies. Recall that in the row-net model $\mathcal{H} = (\mathcal{V}_C, \mathcal{N}_R)$, the vertices and nets represent the columns and rows of A, respectively. In this model, $c_i \in r_j$ if and only if $a_{ji} \neq 0$. Each new w_i vertex is added to the net r_i, i.e., $r_i = r_i \cup \{w_i\}$ and $Nets(w_i) = \{r_i\}$. Upon partitioning the enhanced row-net hypergraph model, we use the partition on the w vertices and the partition on the column vertices to find the permutation matrices P and Q, respectively, with a procedure similar to that discussed for the column-net model. Figure 2.2 shows a sample matrix and its enhanced row-net hypergraph model. The column stripes determine the computational loads of processors. The virtual row stripes are defined by the partition on the w vertices and therefore designate the processors' responsibilities on folding the w-vector entries. For example, in Figure 2.2, the processor P_2 is set to be responsible for folding the w-vector entries that correspond to the rows in the second virtual row stripe. Therefore, the processors P_1 and P_4 have to send their contributions for w_4 to P_2. Again, there is the same association between the connectivity of the nets and the total communication volume. In the figure, the cutsize and hence the total communication volume is five units.

3. Determining partitioning requirements. In iterative methods, all vectors that participate in a linear vector operation should be partitioned conformally in order

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

to avoid the communication of the vector entries during the operation. To obtain such conformable partitions, we classify the vectors according to their relations to the inputs and outputs of the matrix-vector multiplies. In particular, we call a vector to be in the *input space* of a matrix A if it is multiplied by A or it undergoes linear vector operations with other vectors in the input space of A. Accordingly, we call a vector to be in the *output space* of a matrix A if it is obtained by multiplying A with another vector or it undergoes linear vector operations with other vectors in the output space of A. For example, in the $y \leftarrow Ax$ multiply, the y vector is in the output space of A, whereas x is in the input space of A.

In some iterative methods, e.g., conjugate gradients [30], the input space and output space of the A matrix coincide; i.e., the input-space vectors undergo linear vector operations with the output-space vectors. Such methods require a symmetric partition PAP^T in which all vectors are partitioned conformally with the permutation P. In some other methods, the input space and output space of A differ. Such methods allow unsymmetric partition PAQ in which all output-space vectors are partitioned conformally with P, whereas all input-space vectors are partitioned conformally with Q. If the method involves more than one multiply with different matrices, the output space of one matrix may coincide with the input space of another one. In this case, the output-space permutation for the first one becomes an input-space permutation for the other one.

All vectors in a full step of an iterative algorithm should be analyzed in terms of their relations to the input and output spaces of all matrices to determine the partitioning requirements. We analyze the right preconditioned BiCGStab[1] method [62] given in Figure 3.1 and determine its partitioning requirements as an example. There are ten vectors in the method: r, b, $\tilde{r}$, x, p, v, $\hat{p}$, s, $\hat{s}$, and t. Because of the linear vector operations in lines 1, 2, 4, 7, 10, 14, 15, 19, 20, and 21, the vectors r, b, $\tilde{r}$, p, v, s, t, and x should be partitioned conformally. All these vectors are in the output space of A because of the matrix-vector multiplies in lines 13 and 18. We are left with the vectors $\hat{p}$ and $\hat{s}$. Because of the matrix-vector multiplies in lines 13 and 18, these two are in the input space of A, and thus can have a different partition Q. Since we have completed the classification of vectors, we can determine the partitioning requirements for A and M. The input and output spaces of A differ. Therefore, the partitioning requirement for the A matrix is PAQ. The vectors $\hat{p}$ and $\hat{s}$ are in the output space of M because of the matrix-vector multiplies in lines 12 and 17. Therefore, the output space of M coincides with the input space of A. Similarly, the input space of M coincides with the output space of A because of the vectors p and s. Therefore, the partitioning requirement for the M matrix is $Q^T MP^T$. The overall requirement is thus PAQ and $Q^T MP^T$. We express this requirement as $PAMP^T$ to simplify the notation.

We examined a number of widely used preconditioned iterative methods whose codes are given in the literature. We noticed that different methods have different partitioning requirements, as shown in Table 3.1. Several caveats are necessary for the table to be useful.

1. We analyze the methods in their original form as given in the references; i.e., we do not consider any type of code optimizations for performance gains.

[1] Note that the given code works with preconditioned x vector; i.e., the solution vector x obtained at the termination is a solution to $AMx = b$. That is, in order to get a solution to $Ax = b$ we have to multiply x with the approximate inverse preconditioner M at the termination. However, using $\hat{p}$ and $\hat{s}$ instead of p and s in lines 20 and 16 would yield the solution to $Ax = b$ as given in [3]. In this case, the analysis would yield different classification of vectors.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

BiCGStab(A, M, x, b)#Solve $Ax = b$ using the right preconditioner M
begin
(1) $r^{(0)} = b - AMx^{(0)}$ for some initial $x^{(0)} = x$
(2) $\tilde{r} = r^{(0)}$
(3) **for** $i = 1, 2, \ldots$ **do**
(4) $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
(5) **if** $\rho_{i-1} = 0$ method fails
(6) **if** $i = 1$
(7) $p^{(i)} = r^{(i-1)}$
(8) **else**
(9) $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
(10) $p^{(i)} = r^{(i-1)} + \beta_{i-1}\left(p^{(i-1)} - \omega_{i-1}v^{(i-1)}\right)$
(11) **endif**
(12) $\hat{p} = Mp^{(i)}$
(13) $v^{(i)} = A\hat{p}$
(14) $\alpha_i = \rho_{i-1}/\tilde{r}^T v^{(i)}$
(15) $s = r^{(i-1)} - \alpha_i v^{(i)}$
(16) check norm of s; if small enough; set $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ and stop
(17) $\hat{s} = Ms$
(18) $t = A\hat{s}$
(19) $\omega_i = t^T s/t^T t$
(20) $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} + \omega_i s$
(21) $r^{(i)} = s - \omega_i t$
(22) check convergence; continue if necessary
(23) for continuation it is necessary that $\omega_i \neq 0$
(24) **endfor**
end

FIG. 3.1. *Preconditioned BiCGStab using the approximate inverse M as a right preconditioner.*

TABLE 3.1
Iterative methods and partitioning requirements.

Method	Partitioning requirement	Number of distinct vector partitions
BiCGStab right precond. [3, 33]	$PAMP^T$	2
BiCGStab right factor. precond. [3]	$PAM_1M_2P^T$	3
symmetric GMRES right precond. [21]	PAP^T–PMP^T	1
GMRES right precond. [52]	$PAMP^T$	2
GMRES left precond. [52]	$PMAP^T$	2
TFQMR symmetric precond [29]	PAP^T–$PM_2M_1P^T$	2
TFQMR original form [28]	$PM_1AM_2P^T$	3
CGNE [52]	PAQ–PMP^T	2
CGNR [52]	QAP^T–PMP^T	2
CGS right precond. [3]	$PAMP^T$	2
PCG [3, 30, 45]	PAP^T–PMM^TP^T	2
	PAP^T–PMP^T	2

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

2. If two matrices are written consecutively in a partitioning requirement, then the two matrix-vector multiplies involving these matrices follow each other without any interleaving synchronization. For example, in the $PAMP^T$ case, the input space of A and the output space of M coincide. In other words, there is an arbitrary permutation matrix and its transpose in between the two matrices which designates a distinct vector partition. Therefore, $PAMP^T$ means unsymmetric partitions PAQ for A and Q^TMP^T for M. We write the number of distinct vector partitions for each method in the rightmost column of Table 3.1.

3. Listing two partitioning requirements separated by "–" means that there is at least one synchronization point between the two matrix-vector multiplies. Therefore, we distinguish the partitioning requirement PAP^T and PMP^T from PAP^T–PMP^T. The first one states only that the output spaces of the matrices coincide with the input spaces of the matrices. The second partitioning requirement further states that the two matrix-vector multiplies are interleaved with synchronizations.

4. Factored approximate inverse M_1M_2 can be used (the table contains such an example for BiCGStab) instead of M by just writing the factors consecutively in place of M to determine their partitioning requirement. For example, the use of a factored approximate inverse in the right preconditioned CGNE necessitates PAQ–$PM_1M_2P^T$, which in turn gives the requirements PAQ, PM_1R, and $R^TM_2P^T$.

5. The given requirements are independent of the dimensions along which the matrices are partitioned. That is, matrices can be partitioned rowwise or columnwise, whichever is preferable.

In choosing a partitioning dimension, three issues should be considered. The first issue is the individual matrix characteristics, i.e., the number of nonzeros per row and column. If, for example, a matrix has dense rows but no dense columns, then it is advisable to partition it along the columns [37]. This choice will more likely lead to reduced communication volume and better computational load balance compared to partitioning along the rows [37].

The second issue in choosing a partitioning dimension is the relation between the partitioning requirement and the set of concurrent tasks to be partitioned. For example, in the $PAMP^T$ case, we have four partitioning choices for the pair of A (of size $m \times n$) and M (of size $n \times m$) matrices: *rowwise-rowwise* (RR), *rowwise-columnwise* (RC), *columnwise-rowwise* (CR), and *columnwise-columnwise* (CC). In the RR scheme, the partitioning determines the output-space permutation for the two matrices. Since the output spaces of the two matrices differ, there are a total of $m+n$ tasks, each representing computations involving either A or M. In the CC scheme, the partitioning determines the input-space permutation. Since the input spaces differ, there are a total of $n+m$ tasks, each representing computations involving either A or M. In the RC and CR schemes, the partition determines the permutation for the coinciding input and output spaces. Therefore, in the RC and CR schemes, the number of tasks reduces to m and n, respectively, each representing computations involving both A and M. In any reasonable task partitioning for the RC and CR schemes, each processor is guaranteed to take part in both of the multiplies. Therefore, these two schemes may lead to more efficient parallel algorithms.

The third issue is the arrangement of computations and communications. Consider the partitioning requirement of $PAMP^T$ for the multiplies of the form $y \leftarrow AMz$, which are performed as $x \leftarrow Mz$ and then $y \leftarrow Ax$. For each multiply, there exists an expand or a fold communication operation. The partitioning dimension determines whether these communications take place before or after the local com-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

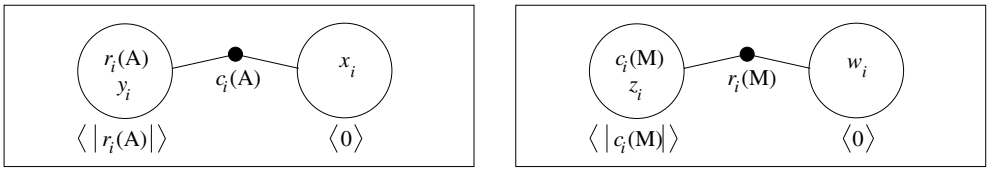
putations. In the RC partitioning scheme, the fold and expand operations take place successively in between the two multiplies. There are dependencies between these two communication operations; before expanding a particular x-vector entry it should be folded. Because of these dependencies, the successive fold and expand operations are likely to incur a local synchronization point which separates the two multiplies. Consequently, processor loads should be balanced for the individual matrix-vector multiplies in the RC scheme. The RR and CC schemes have either an expand or a fold in between the two multiply operations. Although such communications do not incur synchronization points under the assumption that each processor has enough local computation which overlaps with the incoming messages, it is advisable to balance processors' loads for the individual matrix-vector multiplies when the matrices have a comparable number of nonzeros. The CR scheme is unique in that the two multiply operations are performed successively without any interleaving communication. Therefore, processor loads may be balanced for the overall computation $y \leftarrow AMz$.

4. Building composite hypergraph models. We combine enhanced hypergraph models of the matrix-vector multiplies with the coefficient matrix A and the preconditioner matrix M or its factors M_1 and M_2 into a composite hypergraph whose partitioning meets the requirements given in section 3. We define two operations to combine the enhanced hypergraph models. These are called vertex amalgamation and vertex weighting. The first operation is used to enforce identical partitions on the vertices of the individual hypergraphs. The second operation is used to enable load balancing. The key point in combining a number of hypergraphs is to preserve the identities of the nets of the individual hypergraphs.

In the following discussion, we assume that A is to be partitioned rowwise and M is assumed to be partitioned columnwise. In the factored case, where $M = M_1M_2$, M_1 is to be partitioned columnwise, and M_2 is to be partitioned rowwise. That is, we have the enhanced column-net hypergraphs for A and M_2 and the enhanced row-net hypergraph for M_1. As discussed in section 2.3, weights can be associated with the vertices in reference to the vector entries. However, we do not use weights for the vector entries and show only the weights corresponding to the matrix rows or columns to simplify the discussion. Figure 4.1 shows portions of the enhanced column-net hypergraph model for $y \leftarrow Ax$ and the enhanced row-net hypergraph model for $w \leftarrow Mz$. These portions will be used while building composite hypergraph models.

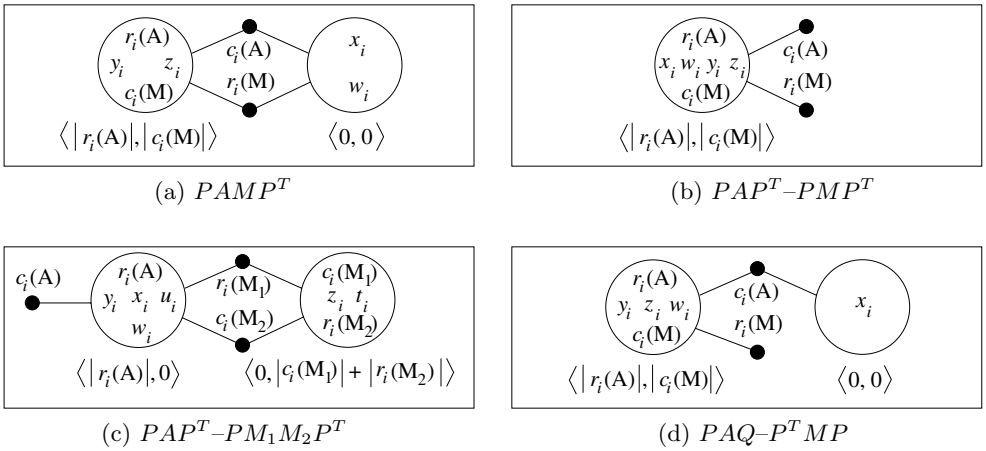
Vertex amalgamation. In this operation, the vertices of the individual hypergraphs are combined into a single vertex. The net set of the resulting composite vertex is set to the union of the nets of the constituent vertices. For example, in the $PAMP^T$ case for the operations $w \leftarrow Mz$ and $y \leftarrow Ax$, we amalgamate the row-vertex $r_i(A)$ with the column-vertex $c_i(M)$ into v_i so that $Nets(v_i) = Nets\,(r_i(A)) \cup Nets(c_i(M))$. Furthermore, we amalgamate the vertex x_i with the vertex w_i to avoid a vector reordering operation. Figure 4.2(a) shows a portion of the resulting composite hypergraph. In a partition of the composite hypergraph, a row or a column vertex v_i being in a part $\mathcal{V}_k$ shows that the processor P_k is responsible for performing multiplications with the ith row or ith column of the matrices. Similarly, a vector-entry vertex x_i being in a part $\mathcal{V}_k$ shows that the processor P_k is responsible for folding or expanding x_i. Figures 4.2(b) to 4.2(d) show portions of the composite hypergraph models built using a vertex amalgamation operation for some other partitioning requirements from Table 3.1.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) (b)

FIG. 4.1. *Portions of enhanced hypergraph models for* (a) *row-parallel* $y \leftarrow Ax$, (b) *column-parallel* $w \leftarrow Mz$. *We use* $c_i(\cdot)$ *and* $r_i(\cdot)$ *to represent, respectively, the* i*th column and* i*th row of the matrices;* $|c_i(\cdot)|$ *and* $|r_i(\cdot)|$ *to represent the number of nonzero elements in the* i*th column and row, respectively; and* $\langle\cdot,\cdot\rangle$ *to represent the weight(s) of a vertex. The nets are labeled with a single* $r_i(\cdot)$ *or a single* $c_i(\cdot)$.



(a) $PAMP^T$ (b) PAP^T–PMP^T

(c) PAP^T–$PM_1M_2P^T$ (d) PAQ–P^TMP

FIG. 4.2. *Composite hypergraph models for different partitioning requirements. The computations to be carried out are* $y \leftarrow Ax$, $w \leftarrow Mz$, $w \leftarrow M_1z$, *and* $t \leftarrow M_2u$.

Vertex weighting. This operation is used to enable load balancing. Remember that in some of the iterative methods there are synchronization points between different matrix-vector multiplies. That is, computations occur in phases. Therefore, we define multiple weights for vertices—one for each computation phase—and use a multiconstraint formulation to obtain load balance for each computation phase. For a certain phase, the weight of a vertex is set to the weight of the constituent vertex in the hypergraph of the matrix associated with that phase. In case the vertices of the simple hypergraphs bear weights for the vector entries, another weight component should be defined to account for those weights. Consider right preconditioned symmetric-GMRES [21] and its partitioning requirement PAP^T–PMP^T which designates $w \leftarrow Mz$ and $y \leftarrow Ax$, where all vectors are to be partitioned conformally. As seen in Figure 4.2(b), we apply vertex amalgamation to the vertices x_i and w_i. Since symmetric partitions are required for both of the multiplies, we also amalgamate the row-vertex $r_i(A)$ and the column-vertex $c_i(M)$ with the vertex composed of x_i and w_i. The vertex in Figure 4.2(b) has two weights, since the multiplies with A and M occur in different phases. The first weight represents the computational load associated with the ith row of A, i.e., $|r_i(A)|$. The second weight represents the computational load associated with the ith column of M, i.e., $|c_i(M)|$. In some cases, different matrix-vector multiplies occur successively

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

without any interleaving synchronization. In these cases, the weight of a composite vertex is set to the sum of the weights of its constituent vertices. Consider the TFQMR method using symmetric preconditioning and its partitioning requirement PAP^T–$PM_1M_2P^T$. Since M_1 and M_2 are partitioned columnwise and rowwise, respectively, there is no synchronization between the respective matrix-vector multiplies. Therefore, the weights of $c_i(M_1)$ and $r_i(M_2)$ are added up as seen in Figure 4.2(c).

Note that partitioning a composite hypergraph results in partitioning the coefficient and preconditioner matrices simultaneously, since the vertices of the composite hypergraph cover the rows or columns of each matrix. Given a partition on the composite hypergraph, we obtain a rowwise or columnwise partition and row and column permutations for each matrix by using the partitions on the vertices pertaining to the respective matrix-vector multiply operation. For example, when a matrix A is to be partitioned rowwise, we use the partition on the composite vertices that contain the rows of A to obtain both a rowwise partition on A and a row permutation, and we use the partition on the composite vertices that contain the input vector entries to obtain a column permutation. Having defined the row and column permutations for each matrix, we obtain the following theorem.

THEOREM 4.1. *The cutsize of a partition in a composite hypergraph formed by applying the vertex amalgamation operation on the enhanced hypergraph representations of a number of matrices quantifies the total volume of communication in the respective sparse matrix-vector multiplies.*

Proof. The proof follows easily by using the same arguments stated in the proof of Theorem 2.1 under the following observations. First, the identities of the nets are preserved while building the composite models. Second, each vertex of a net contains at least one vertex of the original enhanced hypergraph model. Third, the connectivity of a net can be calculated by using only the vertices of the respective enhanced hypergraph model. □

Illustration. Consider the right preconditioned BiCGStab method and its partitioning requirement $PAMP^T$ obtained in section 3. Let A and M be the matrices shown in Figures 2.2 and 2.1, respectively. Suppose that A is to be partitioned columnwise and M is to be partitioned rowwise. We generate the enhanced row- and column-net hypergraph models of A and M, respectively, as shown in Figures 2.2 and 2.1. The partitioning requirement imposes identical partitions on the columns of A and rows of M. Hence, we amalgamate the vertices $c_i(A)$ and $r_i(M)$. The method has no synchronization point between the two multiplies, and the separated expand and fold operations do not cause synchronization. Therefore, we add the weights of the vertices $c_i(A)$ and $r_i(M)$. Since the partitioning requirement imposes identical partitions on the output vector y of the first multiply ($y \leftarrow Ax$) and the input vector z of the second multiply ($w \leftarrow Mz$), we apply vertex amalgamation to the vertices y_i and z_i. A 4-way partition of the resulting hypergraph is shown in Figure 4.3(a). In order to distinguish the nets, the source matrix names are written next to them. The pins of the internal nets are not shown for the sake of clarity. The permutations on the matrices induced by the composite hypergraph partitioning are shown in Figure 4.3(b). Note that the resulting partitions and permutations are identical to those shown in Figures 2.2 and 2.1. As seen from Figure 4.3(a), the cutsize is 10, and hence the total volume of communication is 10 units, where each multiply contributes 5.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

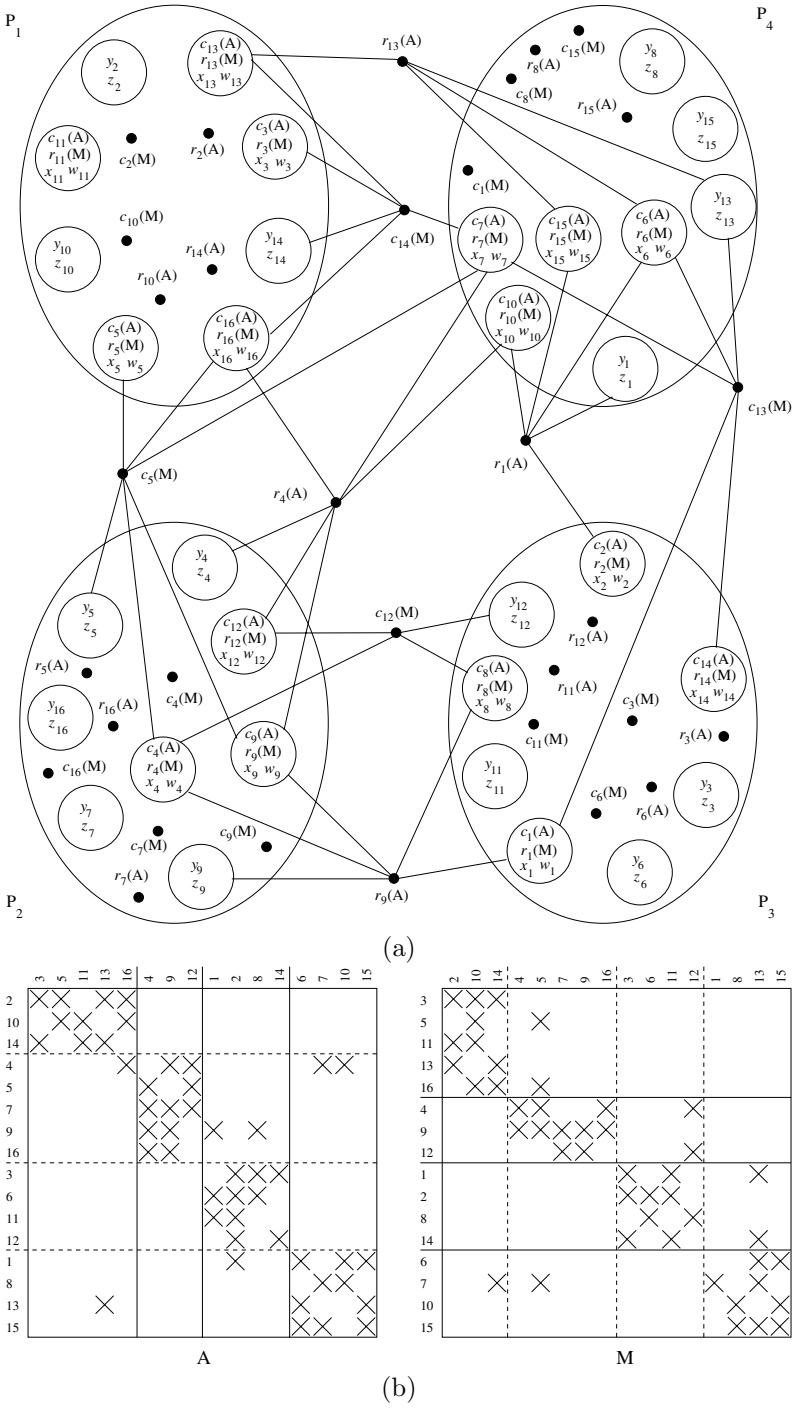


FIG. 4.3. (a) *A composite hypergraph formed by combining the enhanced row-net hypergraph of A and the enhanced column-net hypergraph of M for the computations $y \leftarrow Ax$ and $w \leftarrow Mz$ and a partition which meets the requirement $PAMP^T$. The pins of the internal nets and the weights of the vertices are not shown.* (b) *A columnwise partition of A and a rowwise partition of M induced by the partition on the vertices of the composite hypergraph.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

5. Further notes.

5.1. Investigations on composite models. In the enhanced hypergraph model, the partition on the vertices corresponding to the vector entries are used to obtain permutation matrices. In the previous computational hypergraph models [18], each net is mapped to a part in its connectivity set to obtain permutation matrices. The freedom in mapping a net to one of the parts in its connectivity set has been exploited to minimize communication cost metrics such as the total number of messages and the maximum volume per processor defined in terms of sends [58, 59], and maximum volume per processor [13, 63] defined in terms of both sends and receives. These works achieve their goals by assigning the vector entries to processors upon partitioning the matrix with the computational hypergraph models. In the enhanced hypergraph model and the composite hypergraph model, the vertices that do not contain row or column vertices can be reassigned with the same freedom. For example, the vertex formed by x_i and w_i in Figure 4.2(a) and the vertex x_i in Figure 4.2(d) can be reassigned to optimize the aforementioned communication cost metrics.

5.2. Generalizations. The computational structure of the preconditioned iterative methods is similar to that of a more general class of scientific computations, including multiphase, multiphysics, and multimesh simulations.

In multiphase simulations, there are a number of computational phases separated by global synchronization points. The existence of the global synchronizations necessitates each phase to be load balanced individually. In our model, the multiple weights associated with vertices can be used to achieve this goal as described in [41, 64].

In multiphysics simulations, a variety of materials and processes are analyzed using different physics procedures. In these types of simulations, computational as well as memory requirements are not uniform across the mesh [54]. For scalability issues, processor loads should be balanced in terms of these two components. The multiconstraint partitioning framework also addresses these problems [54].

In multimesh simulations, a number of grids with different discretization schemes and with arbitrary overlaps are used. The existence of overlapping grid points necessitates the simultaneous partitioning of the grids [54]. This simultaneous partitioning should balance the computational loads of the processors and minimize the communication cost due to interactions within a grid as well as interactions among different grids. The vertex amalgamation operation used in our models can be applied to overlapped grid points to build a composite hypergraph. With the use of vertex weighting operations, our models can be used to address the partitioning problem in the multimesh computations. Although simultaneous partitioning seems to be more adequate for these types of problems, independent partitioning is also possible (see, for example, [51] for independent partitionings on a two-grid system).

In some contact/impact problems, there is a priori knowledge about the to-be-contacting surfaces. The implementation in [38] uses this information to decompose the underlying mesh among processors. The implementation uses the graph model and adds edges between the to-be-contacting surface elements. Partitioning such a graph using two constraints balances the loads of processors, both for the finite element analysis and for the contact detection phases. The partitioning algorithm tries to minimize the communication cost by minimizing the edge cut. By modeling the interactions among the to-be-contacting surface elements with hypergraphs, we can build a composite hypergraph to address these problems. However, the implementation in [38] is reported to be suffering from load imbalances and to be limited to a small number of processors; see the comments on it in [50].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

In obtaining partitions for two or more computation phases interleaved with synchronization points, our models lead to the minimization of the overall sum of the total volume of communication in all phases. For the preconditioned iterative methods, this approach will likely minimize the communication cost in one full step. However, in more sophisticated simulations, the magnitude of the interactions in one phase may be different from that of the interactions in another one. In such settings, minimizing the total volume of communication in each phase separately may be advantageous [53]. This problem can be formulated as a multiobjective hypergraph partitioning problem [1, 53, 55] on the composite hypergraphs.

As discussed above, our models can be applied to the multiphase, multiphysics, and multimesh computations but with certain limitations. The dependencies must remain the same throughout the computations; our methods cannot be used, for example, in adaptive mesh refinement. The weights assigned to elements, for load balancing issues, should be static and available prior to the partitioning; our methods cannot be used for applications whose computational requirements vary in time [35]. If, however, the computational requirements change gradually in time, then our methods can be used to repartition the load at certain time intervals. Some problems are more suitable to geometric partitioning methods; contact detection without a priori knowledge of the contacting surfaces, for example, should be performed on geometrically close elements [14, 40, 50]. Our methods, in their current forms, will probably be of little help in those problems. To be useful, the models should be enriched with some geometric constructs, as is done in [40].

6. Experiments. We chose the right preconditioned BiCGStab method to evaluate the effectiveness of the proposed composite hypergraph partitioning approach. We used a set of unsymmetric sparse matrices from the University of Florida Sparse Matrix Collection [27]. Approximate inverse preconditioners were obtained using SPAI version 3.0 [32]. Factored approximate inverses were obtained using AINV [11]. These two programs have parameters that affect the quality of the preconditioner matrices. However, we set the parameters in such a way that the number of nonzeros of the approximate inverse or the total number of nonzeros of the factors of the approximate inverse is at most twice and at least half the number of nonzeros of the coefficient matrix. We adjusted the tolerance parameter *eps*, the number of nonzero entries allowed per step *mn*, and the number of steps *ns* in SPAI. In AINV, we adjusted the drop tolerance parameter τ. The properties of the matrices, approximate inverses, and factors of the approximate inverses are given in Table 6.1. In the table, the coefficient matrices are listed with a suffix of A; the approximate inverse matrices are listed with a suffix of M; the factors of the approximate inverse matrices are listed with suffixes of Z and W, where the approximate inverse is equivalent to ZW. The hypergraphs were partitioned using PaToH [19] with default parameters. The imbalance among processor loads is kept below 10% in all partitioning instances. Throughout this section, we use the term "SPAI-matrices" to refer to a pair consisting of a coefficient matrix and its approximate inverse preconditioner. Similarly, we use "AINV-matrices" to refer to a triplet of coefficient matrix and the factors of its approximate inverse preconditioner.

Since the partitioning tool PaToH incorporates randomized algorithms, it was run 20 times starting from different random seeds for partitioning the hypergraphs. Averages of the resulting communication volumes of these runs are displayed in the following tables. PaToH is a fairly stable toolkit; the standard deviation of the total communication volumes of the 20 runs is less than 4% of the mean for all hypergraphs

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.1
Properties of test matrices.

Matrix	Number of rows/cols	Number of nonzeros					
		Total	Average	Row		Column	
			row/col	min	max	min	max
Zhao1-A	33861	166453	4.9	3	6	2	7
big-A	13209	91465	6.9	3	12	3	12
cage11-A	39082	559722	14.3	3	31	3	31
cage12-A	130228	2032536	15.6	5	33	5	33
epb2-A	25228	175027	6.9	3	87	3	87
epb3-A	84617	463625	5.5	3	6	3	7
mark3jac060-A	27449	170695	6.2	2	44	2	47
olafu-A	16146	1015156	62.9	24	89	24	89
stomach-A	213360	3021648	14.2	7	19	6	22
xenon1-A	48600	1181120	24.3	1	27	1	27
SPAI							
Zhao1-M	33861	180988	5.3	1	11	1	16
big-M	13209	109088	8.3	2	22	1	21
cage11-M	39082	424708	10.9	2	51	2	21
cage12-M	130228	1444650	11.1	1	62	2	21
epb2-M	25228	244453	9.7	2	177	2	21
epb3-M	84617	532851	6.3	2	20	2	20
mark3jac060-M	27449	276586	10.1	1	37	1	21
olafu-M	16146	719873	44.6	5	114	4	46
stomach-M	213360	2910283	13.6	2	120	2	46
xenon1-M	48600	878143	18.1	1	35	1	21
AINV; $M = ZW$							
Zhao1-Z	33861	179803	5.3	1	13	1	28
Zhao1-W	33861	57832	1.7	1	5	1	6
big-Z	13209	56302	4.3	1	11	1	13
big-W	13209	56314	4.3	1	13	1	11
cage11-Z	39082	302775	7.7	1	26	1	110
cage11-W	39082	299939	7.7	1	26	1	32
epb2-Z	25228	116161	4.6	1	13	1	22
epb2-W	25228	107620	4.3	1	36	1	19

(except `Zhao1-W` in Table 6.3, which has a negligible total communication volume). Details of the communication patterns can be found in the accompanying report [60].

6.1. Composite versus simple hypergraph partitioning. Here we evaluate two alternative approaches to partitioning composite hypergraphs. These alternative approaches are based on partitioning simple hypergraphs, i.e., partitioning hypergraphs of a single matrix. The first alternative is to obtain independent partitions on the matrices by partitioning the simple hypergraph models of the coefficient and the preconditioner matrices independently. This approach requires vector reordering in between the two matrix-vector multiplies. We discuss this alternative in section 6.1.1. The second alternative is to obtain the same partition for the coefficient and preconditioner matrices. For this purpose, we partition the coefficient matrices symmetrically by rows or columns using hypergraph models and apply the resulting partitions to the preconditioner matrices as well. This alternative avoids the vector reordering op-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.2

Total communication volume for 64-way simple hypergraph (C/R and R/C) and composite hypergraph (CR and RC) partitionings for SPAI-matrices. In simple hypergraph partitionings, A and M are partitioned independently, and therefore vector entries are reordered in between the two multiplies to meet the partitioning requirement.

	C/R (simple)		CR (composite)	R/C (simple)		RC (composite)
	Volume		Volume	Volume		Volume
Matrix	SpMxV	Reorder	SpMxV	SpMxV	Reorder	SpMxV
Zhao1-A	11421	134307	13026	10811	135348	13734
Zhao1-M	9857		10809	11756		14551
big-A	3210	52150	3666	3215	52804	5453
big-M	3054		3562	5447		7610
cage11-A	58177	153366	63779	58272	151840	79052
cage11-M	32937		38714	42512		61304
cage12-A	185531	504816	207077	185191	510824	248253
cage12-M	98493		119287	122513		180128
epb2-A	5984	98523	6731	5527	100912	10610
epb2-M	5967		7215	7411		11694
epb3-A	5713	332064	8167	7250	333320	17911
epb3-M	6846		9759	7057		18512
mark3jac060-A	13331	108209	17447	12676	109780	19503
mark3jac060-M	15567		18970	16563		21096
olafu-A	14012	63743	16743	13912	57372	23870
olafu-M	25137		29348	24735		36427
stomach-A	36800	837087	47689	37219	853440	77080
stomach-M	44232		57755	47965		89479
xenon1-A	26710	189847	29644	26669	178388	36044
xenon1-M	33597		40270	33241		46970

eration by partitioning all vectors conformally. Note that since we partition a single matrix, a graph model could also be used. We discuss this alternative in section 6.1.2.

6.1.1. Simple hypergraph partitioning: Independent partitions on the matrices. We have conducted experiments with the CR and RC partitionings of SPAI-matrices where the first partition dimension corresponds to the matrix A and the second to the matrix M. In the independent partitioning approach, we partition the simple hypergraph models of the matrices A and M independently.

Table 6.2 displays the average communication volumes in the sparse matrix-vector multiply (SpMxV) operations resulting from the composite and simple hypergraph partitionings for 64-way partitioning of SPAI-matrices. The table also shows the volume of communication required to reorder the vector entries—in an iteration of the BiCGStab method—when the matrices are partitioned independently. Suppose that symmetric partitions PAP^T and QMQ^T were obtained on the A and M matrices. Then at each iteration we have to reorder $\hat{p}$ and $\hat{s}$ from Q to P after the matrix-vector multiplies at lines 12 and 17 of the BiCGStab method (see Figure 3.1), respectively. We also have to reorder v and t from P to Q before the vector update at line 15 and the inner product at line 19, respectively. The volume of communication in the reordering operation is given as the average of 20 different partitions of SPAI-matrices. In all of the partitioning instances, the volume of communication in the reordering operation

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.3

Total communication volume for 64-way simple hypergraph (C/R/C and R/C/R) and composite hypergraph (CRC and RCR) partitionings for AINV-matrices. In simple hypergraph partitionings, A, Z, and W are partitioned independently, and therefore vector entries are reordered in between the successive multiplies to meet the partitioning requirement.

	C/R/C (simple)		CRC (composite)	R/C/R (simple)		RCR (composite)
	Volume		Volume	Volume		Volume
Matrix	SpMxV	Reorder	SpMxV	SpMxV	Reorder	SpMxV
Zhao1-A	11421	199423	15258	10811	199881	14977
Zhao1-Z	10808		12811	13580		17376
Zhao1-W	170		2494	377		4380
big-A	3210	77975	3730	3215	78077	3633
big-Z	1861		2262	1947		2314
big-W	1859		2305	1948		2325
cage11-A	58177	225147	65430	58272	226743	65052
cage11-Z	22023		28640	40428		48783
cage11-W	21676		27397	39731		48453
epb2-A	5984	148944	10313	5527	148830	10490
epb2-Z	2058		3967	2478		5672
epb2-W	1431		3786	1174		4935

itself is higher than that obtained by the simultaneous partitioning method. These high volumes of communication and the associated message start-up overheads prohibit the use of the independent partitioning method. For example, the independent partitioning method incurs higher total communication volume than the proposed simultaneous partitioning method by an average ratio of 8.4 for 32-way CR partitioning (see [60]). The average ratio in 64-way CR partitioning is 6.5. For RC partitioning, the average ratios are 5.6 and 4.2 for 32- and 64-way partitionings, respectively.

Table 6.3 displays the averages of the communication volumes of the 64-way simultaneous and independent partitionings for AINV-matrices. In this table CRC corresponds to the case where the A, Z, and W matrices are partitioned columnwise, rowwise, and columnwise, respectively. Similarly, RCR corresponds to the case where those matrices are partitioned rowwise, columnwise, and rowwise. With AINV preconditioning, the independent partitioning method requires two additional vector-reordering operations (due to the chains of matrix-vector multiplies at lines 12 and 17 of the BiCGStab method). For 64-way partitioning of AINV-matrices, the average ratios of the communication volumes in the independent partitionings (including the reordering cost) to those in the simultaneous partitionings is 7.2 for the CRC case and 6.5 for the RCR case. The average ratios for 32-way partitionings are 10.1 and 9.0 for the CRC and RCR cases, respectively (see [60]). As is clear from these numbers, the independent partitioning approach is not feasible for the AINV-matrices as well.

Consider the difference between the total communication volumes of the composite and simple hypergraph partitionings (without the reordering cost). The increases in the total communication volumes in the composite partitionings remain below 26% of those in the simple partitionings, on average, for the 32-way CR partitioning instances. The minimum and the maximum of these increases are 13% (`Zhao1`) and 61% (`epb3`). The 64-way CR partitionings give better ratios. The average increase is 20% with the minimum and the maximum being 12% and 43%, which are obtained for the same matrices. In fact, for each matrix the increase in the 64-way CR partitioning

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.4
Relation between the sparsity patterns of the coefficient matrices and the approximate inverses. We use A and M to represent the set of the positions of the nonzeros in the corresponding matrices.

Matrix	Number of nonzeros			
	$A \cup M$	$A \setminus M$	$M \setminus A$	$\frac{A \cap M}{M}$
Zhao1	234205	67752	53217	0.63
big	147632	56167	38544	0.49
cage11	780776	221054	356068	0.48
cage12	2784199	751663	1339549	0.48
epb2	333794	158767	89341	0.35
epb3	773107	309482	240256	0.42
mark3jac060	397706	227011	121120	0.18
olafu	1357370	342214	637497	0.52
stomach	5182305	2160657	2272022	0.26
xenon1	1520936	339816	642793	0.61

is smaller than the increase in the 32-way CR partitioning. We investigated the 8- and 16-way CR partitionings as well [60] and observed that for each matrix in our data set the larger the number of parts, the smaller the increases. The same relation holds for most of the RC partitioning cases and for the CRC and RCR partitioning of AINV-matrices [60]. The reason behind this may be the following. The cutsize function almost always increases monotonically with the increasing K. In other words, the flexibility of finding better partitions reduces with the increasing K. At the limit, where $K = |\mathcal{V}|$ and all the nets are in cut, the cutsize of a partition on the composite hypergraph will be equivalent to the sum of the cutsizes of partitions on the simple hypergraphs (i.e., $nnz(A) + nnz(M) - 2m$) that forms it. Therefore, the difference between the total communication volumes should decrease as K increases and has to be zero at the limit.

6.1.2. Simple hypergraph partitioning: The same partition on the matrices. Obtaining a symmetric partition on A and then applying the resulting partition to M results in CC or RR partitionings on the A and M matrices. Recall that in CC and RR partitioning schemes, there is a communication phase in between the two matrix-vector multiplies. Since the A and M matrices have a comparable number of nonzeros (see Table 6.1), processor loads for the two matrix-vector multiplies should be balanced separately, i.e., a two-constraint formulation is necessary.

Observe that the approach evaluated here disregards the sparsity pattern of the preconditioner matrices. However, the sparsity patterns of the approximate inverse preconditioners are usually related to the sparsity patterns of the coefficient matrices [23, 39]. Therefore, the partitions on the coefficient matrices are expected to be effective for the preconditioners. To justify this reasoning, we show the relation between the sparsity patterns of the coefficient matrices and the approximate inverses in Table 6.4. As seen in the table, the relation between the sparsity patterns of the coefficient and preconditioner matrices varies; 63% of the nonzeros of `Zhao1-M` are covered by the nonzeros of `Zhao1-A`, and only 18% of the nonzeros of `mark3jac060-M` are covered by the nonzeros of `mark3jac060-A`. Another reason for using the same partition on the coefficient and preconditioner matrices is the following. Parallel construction of the approximate inverse preconditioners produces preconditioners in such a way that the initial partitions on the coefficient matrices become partitions on the preconditioner matrices. For example, the left approximate inverse preconditioners

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.5

Total communication volume for 64*-way simple hypergraph and composite hypergraph partitionings for CC and RR partitioning of SPAI-matrices. In simple hypergraph partitionings, A and M have the same symmetric partition which is obtained by partitioning A.*

	CC			RR		
	Volume		%	Volume		%
Matrix	Simple	Composite	gain	Simple	Composite	gain
Zhao1-A	12982	12892	8	12502	12131	8
Zhao1-M	15406	13116		12523	10896	
big-A	4089	3849	18	4068	3783	20
big-M	8082	6114		5548	3904	
cage11-A	67198	64928	18	68374	63612	19
cage11-M	72710	49361		60385	41030	
cage12-A	213360	208209	18	216501	203197	18
cage12-M	218227	146726		183548	125836	
epb2-A	9820	9357	16	9611	8545	19
epb2-M	12913	9766		11649	8590	
epb3-A	11764	11293	26	12441	11607	22
epb3-M	21010	13063		18036	12201	
mark3jac060-A	15676	18090	34	18107	15891	36
mark3jac060-M	42608	20183		34929	18005	
olafu-A	19802	16733	26	20273	16173	26
olafu-M	38318	26470		39026	27911	
stomach-A	50980	57777	19	52582	58230	13
stomach-M	107817	71060		90605	66561	
xenon1-A	30510	27683	21	29597	27615	17
xenon1-M	49568	35520		47909	36437	

can be efficiently constructed rowwise when the coefficient matrix A is partitioned rowwise [24]. The construction yields the same rowwise partition on the approximate inverse M. Equivalently, a right approximate inverse preconditioner can be efficiently constructed columnwise when the coefficient matrix A is partitioned columnwise.

The row-net hypergraph model of A can be used to obtain a CC partition on A and M. In order to obtain load balance for the two multiplies, the vertices of A are assigned two weights which correspond to the number of nonzeros in the respective columns of A and M matrices. That is, v_i has weights $\langle |c_i(A)|, |c_i(M)| \rangle$. Similarly, the column-net hypergraph model of A, with two weights on the vertices, can be used to obtain an RR partition on A and M.

The composite hypergraph model for the CC partitioning scheme is built in three steps as follows. First, the enhanced row-net hypergraph models of $y \leftarrow Ax$ and $w \leftarrow Mz$ are created. Second, vertex amalgamation operations are applied to the vertices y_i and $c_i(M)/z_i$, and also to the vertices $c_i(A)/x_i$ and w_i for each i. Third, vertex weighting operations are applied to the vertices in such a way that the vertex $y_i/c_i(M)/z_i$ has weights $\langle 0, |c_i(M)| \rangle$, and the vertex $w_i/c_i(A)/x_i$ has weights $\langle |c_i(A), 0| \rangle$ for each i. Theoretically, a three-constraint formulation is necessary to balance the vector operations; however, we use a two-constraint formulation in order to ease the job of the hypergraph partitioning tool. The composite hypergraph model for the RR partitioning scheme is built similarly.

Table 6.5 displays the averages of the communication volumes of the 64-way partitioning of the SPAI-matrices with the composite hypergraphs and simple hypergraph models of the coefficient matrices. The "% gain" columns in this table show the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

improvements achieved by the composite hypergraph partitioning as the percentage of the total communication volumes found by partitioning the simple hypergraph of A. The minimum improvements are obtained for the `Zhao1` matrices in both CC and RR cases. The maximum improvements are obtained for the `mark3jac060` matrices in both CC and RR cases. As seen in Table 6.4, the `Zhao1-A` and `Zhao1-M` matrix pair have the highest ratio of common nonzeros, and the `mark3jac060-A` and `mark3jac060-M` matrix pair have the lowest ratio of common nonzeros. Although `xenon1-A` covers 61% of `xenon1-M` (second maximum), the improvements achieved for these matrices are quite satisfactory. The average of the improvements is 20% in the 32- and 64-way CC and RR partitioning choices (see also [60]).

We have also experimented with the 32- and 64-way CC and RR partitionings using single constraint formulation. In the single constraint formulation, the weight of a vertex v_i is set to the sum of the number of nonzeros in the ith columns (rows) of A and M for the CC (RR) partitioning. Both the composite hypergraph and the simple hypergraph formulations were able to obtain balance on the total loads of the processors. Since these formulations ignore the fact that there is a local synchronization, both formulations could not obtain balance on the loads of the processors for the individual matrix-vector multiplies. The composite hypergraph partitioning approach obtained (on average 17%) better solutions than the simple hypergraph partitioning [60]. The best and worst improvements are again obtained for the `mark3jac060` and `Zhao1` matrix pairs, respectively.

6.2. Effects of partitioning dimensions on the composite hypergraph partitioning. Comparing the left and right halves of Table 6.2, we see that the CR partitioning yields, on average, 26% better total communication volume than the RC one. This ratio remains the same for 8-, 16-, and 32-way partitionings (see Table 6.6 and [60]). The matrices in our data set do not have dense rows or dense columns. Therefore, the rowwise and columnwise partitionings of the matrices are expected to be comparable in terms of the total communication volume. This theoretical expectation is verified by the communication volume values listed in the SpMxV columns in Table 6.2 for the simple hypergraph partitioning. In light of this observation, we can deduce that the performance difference between the CR and RC partitioning schemes is due to the two-constraint formulation in the RC scheme. This degradation in the multiconstraint formulation is in concordance with the previously reported results [41, 64]. The degradation in our case stems from two facts. First, the additional balance constraints shrink the search space. Second, the heuristics in PaToH are not very well tailored toward handling the multiple vertex weights.

6.3. Parallelization results. It is important to see whether the theoretical improvements obtained by the proposed simultaneous partitioning method hold in practice. For this purpose, we have implemented a parallel program for the BiCGStab method. The program uses the LAM/MPI 6.5.6 [15] message passing library. The tests were carried out on a Beowulf class [56] PC cluster with 24 nodes. Each node has a 400MHz Pentium-II processor and 128 MB memory. The interconnection network comprises a 3COM Superstack II 3900 managed switch connected to Intel Ethernet Pro 100 Fast Ethernet network interface cards at each node. The system runs Linux kernel 2.4.20 and the Debian GNU/Linux 3.0 distribution.

We are not concerned with the numerics of the preconditioners and the BiCGStab method. Therefore, for each matrix, we let the BiCGStab run for 100 iterations and measure the average running time of a single iteration. In order to guarantee 100 iterations, we set ρ and ω of the BiCGStab method (see Figure 3.1) to 1.0 after computing

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 6.6
Communication patterns for 8- and 16-way composite hypergraph partitionings for SPAI-matrices and the respective speedup values. Matrix name `mark3jac060` *is shortened to* `mark3_060`.

	8-way					16-way				
	Volume		Message		Sp.	Volume		Message		Sp.
Matrix	tot	max	tot	max	up	tot	max	tot	max	up
	CR									
Zhao1-A	4098	746	32.2	5.7	6.2	6444	586	96.7	9.3	8.7
Zhao1-M	3514	694	32.2	5.6		5478	551	97.0	9.3	
big-A	1032	201	31.4	5.7	5.7	1581	156	73.2	7.5	7.3
big-M	989	191	31.9	5.6		1527	150	75.3	7.5	
cage11-A	24424	4144	54.6	7.0	5.5	34835	3314	201.2	14.8	8.1
cage11-M	14663	2439	55.1	7.0		21010	1917	208.9	15.0	
cage12-A	87542	14306	56.0	7.0	5.9	122878	11925	230.3	15.0	9.4
cage12-M	50962	7839	56.0	7.0		71066	6136	233.1	15.0	
epb2-A	2326	429	39.0	6.4	6.4	3357	371	102.8	9.6	8.6
epb2-M	2242	438	35.0	6.5		3335	335	84.7	8.4	
epb3-A	2354	442	23.9	4.3	7.3	3971	393	66.0	6.5	12.4
epb3-M	3003	536	23.9	4.3		5023	496	66.3	6.5	
mark3_060-A	5249	960	35.2	6.3	5.8	9370	786	115.0	11.3	8.7
mark3_060-M	6323	1182	32.2	6.0		10287	964	105.3	11.0	
olafu-A	3908	960	25.8	5.0	6.7	6489	781	66.2	6.8	10.6
olafu-M	6749	1449	28.0	5.4		11258	1285	77.8	7.8	
stomach-A	14614	2815	21.1	4.0	7.1	24436	2351	67.2	7.0	14.1
stomach-M	16193	3206	21.4	4.0		28014	2652	67.8	7.1	
xenon1-A	10848	2037	36.2	6.5	6.7	15998	1496	113.2	11.3	11.2
xenon1-M	14437	2523	37.7	6.7		21459	2032	117.8	11.8	
	RC									
Zhao1-A	4146	905	33.2	5.8	5.9	6728	734	95.5	9.3	8.1
Zhao1-M	4362	771	33.1	5.6		7141	716	95.5	9.3	
big-A	1467	320	33.9	5.8	5.2	2408	280	84.7	8.8	6.4
big-M	2084	433	34.0	5.8		3326	366	85.9	8.4	
cage11-A	30373	6054	55.9	7.0	4.2	43335	4599	227.0	15.0	5.5
cage11-M	24447	4302	55.9	7.0		34339	3151	227.4	15.0	
cage12-A	107187	17621	56.0	7.0	4.4	147504	12516	239.8	15.0	6.2
cage12-M	80949	15047	56.0	7.0		109472	10885	239.7	15.0	
epb2-A	3074	646	46.7	6.8	6.1	4555	560	132.1	12.9	8.6
epb2-M	3789	814	43.8	6.5		5388	633	109.5	10.6	
epb3-A	6347	1907	39.8	6.8	7.2	8490	1244	108.3	11.7	11.7
epb3-M	6766	1937	39.5	6.8		8991	1480	108.0	11.8	
mark3_060-A	5568	981	40.7	6.7	5.7	9792	906	137.4	13.0	7.4
mark3_060-M	6417	1213	38.6	6.8		11099	1027	126.3	12.2	
olafu-A	5605	1088	30.5	5.8	6.0	9325	1012	86.0	9.1	8.6
olafu-M	9075	1947	33.5	6.2		14636	1636	99.8	10.2	
stomach-A	21139	4354	27.4	5.3	7.1	35856	4255	81.5	8.8	12.7
stomach-M	24538	5221	27.6	5.7		41254	4660	82.8	8.9	
xenon1-A	12654	2322	39.6	6.9	6.1	18800	1797	126.2	12.2	9.2
xenon1-M	16486	2974	40.8	7.0		24477	2286	131.6	12.9	

their actual values. The speedup values corresponding to these running times are given in Table 6.6 under the column *Sp./up*. The given speedup values are the averages of 20 runs corresponding to different partitions. In order to show how the improvements obtained by the proposed method relate to parallel running times, we give the average communication patterns induced by the partitions in the same table as well.

As seen from Table 6.6, CR gives better speedup values than RC for all matrices (the reasons are discussed in section 6.2). On average, CR obtains speedup values of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

6.3 and 9.9 for 8- and 16-way partitionings, respectively, where the highest speedups are 7.3 (`epb3`) and 14.1 (`stomach`). Meanwhile, RC obtains speedups of 5.8 and 8.4, on average, for 8-way and 16-way partitionings, respectively, where the highest speedups are 7.2 (`epb3`) and 12.7 (`stomach`). The lowest speedups for 8-way partitioning are obtained for the `cage11` matrix by both of the partitioning schemes. The lowest speedups for 16-way partitioning are obtained for the `big` and `cage11` matrices by the CR and RC schemes, respectively. As seen from Table 6.6, the `cage11` matrix pair has a communication pattern inferior to all but the `cage12` matrix pair in terms of the total and maximum number of message metrics. Therefore, we were already expecting to have the lowest speedups with the `cage` matrices. The `big` matrix has the smallest number of nonzeros. The low granularity of computations may be the reason behind having the lowest speedup with 16-way CR partitioning of the `big` matrix. The same reasoning may also explain why we obtain better speedups in `cage12` than those in `cage11`.

We have also experimented with the CRC and RCR partitioning schemes for AINV-matrices (see [60]). As expected, the speedup values are not as good as those given in Table 6.6 because of an additional load balancing constraint for the third matrix-vector multiply, and because of an additional communication phase. The highest speedups for 8- and 16-way CRC partitionings are 6.7 and 8.9, respectively. The highest speedups for 8- and 16-way RCR partitionings are 6.4 and 8.9, respectively.

6.4. Partitioning timings. Finally, we comment on the additional partitioning overhead introduced by simultaneous partitioning instead of individual partitionings. Let the sum of the times elapsed in individual partitionings (of the SPAI- and AINV-matrices) be 1.0. Then the average running times of the simultaneous partitioning of the SPAI-matrices with the CR and RC schemes are 1.4 and 1.2, respectively, for all $K = 8$, 16, 32, and 64. The average running times of the simultaneous partitioning of the AINV-matrices with both the CRC and RCR schemes are close to 1.4. These increases are acceptable because the simultaneous partitioning method obtains much smaller total communication volume than the individual partitioning method combined with the reordering cost. See [60] for a list of the partitioning times.

7. Conclusion. We demonstrated that hypergraph models are able to capture the application of multiple matrices. In particular, we developed models that allow simultaneous partitioning of a matrix and its explicit preconditioner or factors of the preconditioner. These points were raised by Hendrickson and Kolda [37].

The computational structure of preconditioned iterative methods abounds in scientific computing applications and data aggregation/reduction applications in large distributed data sets. We discussed the applicability of the proposed work in certain scientific computations. We think that the proposed work is applicable in distributed data-set applications where hypergraph models have already been used [22].

Acknowledgments. We thank Bruce Hendrickson for introducing us to the simultaneous partitioning problem; Michele Benzi, Miroslav Tůma, and Marcus Grote for their research software; Levent Gürel and his computational electromagnetics research group at Bilkent University for granting us access to their computing machinery. We are indebted to anonymous referees for insightful comments.

REFERENCES

[1] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, *Multi-objective circuit partitioning for cutsize and path-based delay minimization*, in Proceedings of the 2002

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

IEEE/ACM International Conference on Computer-aided Design (ICCAD '02), San Jose, CA, 2002, pp. 181–185.

[2] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput., 25 (2004), pp. 1860–1879.

[3] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. Van Der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[5] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.

[6] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.

[7] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.

[8] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[9] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.

[10] M. Benzi and M. Tůma, *A parallel solver for large-scale Markov chains*, Appl. Numer. Math., 41 (2002), pp. 135–153.

[11] M. Benzi and M. Tůma, *Private communication*, 2003.

[12] L. Bergamaschi, G. Pini, and F. Sartoretto, *Approximate inverse preconditioning in the solution of sparse eigenproblems*, Numer. Linear Algebra Appl., 7 (2000), pp. 99–116.

[13] R. H. Bisseling and W. Meesen, *Communication balancing in parallel sparse matrix-vector multiplication*, Electron. Trans. Numer. Anal., 21 (2005), pp. 47–65.

[14] K. Brown, S. Attaway, S. J. Plimpton, and B. Hendrickson, *Parallel strategies for crash and impact simulations*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 373–390.

[15] G. Burns, R. Daoud, and J. Vaigl, *LAM: An open cluster environment for MPI*, in Proceedings of Supercomputing Symposium '94, J. W. Ross, ed., University of Toronto, 1994, pp. 379–386.

[16] Ü. V. Çatalyürek, *Hypergraph Models for Sparse Matrix Partitioning and Reordering*, Ph.D. thesis, Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, 1999.

[17] Ü. V. Çatalyürek and C. Aykanat, *Decomposing irregularly sparse matrices for parallel matrix-vector multiplications*, in Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Comput. Sci. 1117, Springer, Berlin, 1996, pp. 75–86.

[18] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Systems, 10 (1999), pp. 673–693.

[19] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool*, Version 3.0, Tech. Report BU-CE-9915, Computer Engineering Department, Bilkent University, Ankara, Turkey, 1999.

[20] Ü. V. Çatalyürek and C. Aykanat, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proceedings of Scientific Computing 2001 (SC2001), Denver, CO, 2001, pp. 10–16.

[21] T. F. Chan, E. Chow, Y. Saad, and M. C. Yeung, *Preserving symmetry in preconditioned Krylov subspace methods*, SIAM J. Sci. Comput., 20 (1998), pp. 568–581.

[22] C. Chang, T. Kurc, A. Sussman, Ü. V. Çatalyürek, and J. Saltz, *A hypergraph-based workload partitioning strategy for parallel data aggregation*, in Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, VA, CD-ROM, SIAM, Philadelphia, 2001.

[23] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[24] E. Chow, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, Internat. J. High Perform. Comput. Appl., 15 (2001), pp. 56–74.

[25] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

[26] J. K. Cullum, K. Johnson, and M. Tůma, *Effects of problem decomposition on the convergence behavior of parallel numerical algorithms*, Numer. Linear Algebra Appl., 10 (2003), pp. 445–465.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[27] T. Davis, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices/.
[28] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.
[29] R. W. Freund and N. M. Nachtigal, *A New Krylov-Subspace Method for Symmetric Indefinite Linear Systems*, Tech. Report ORNL/TM-12754, Oak Ridge National Laboratories, 1994.
[30] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[31] N. I. M. Gould and J. A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., 19 (1998), pp. 605–625.
[32] M. J. Grote, *SPAI: SParse Approximate Inverse Preconditioner*, http://www.sam.math.ethz.ch/~grote/spai/.
[33] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[34] B. Hendrickson, *Graph partitioning and parallel solvers: Has the emperor no clothes?*, in Workshop on Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Comput. Sci. 1457, Springer, Berlin, 1998, pp. 218–225.
[35] B. Hendrickson and K. Devine, *Dynamic load balancing in computational mechanics*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 485–500.
[36] B. Hendrickson and T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Comput., 26 (2000), pp. 1519–1534.
[37] B. Hendrickson and T. G. Kolda, *Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing*, SIAM J. Sci. Comput., 21 (2000), pp. 2048–2072.
[38] C. Hoover, A. DeGroot, J. Maltby, and R. Procassini, *Paradyn: Dyna3d for Massively Parallel Computers*, Presentation at Tri-Laboratory Engineering Conference on Computational Modeling, Pleasanton, CA, 1995.
[39] T. Huckle, *Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning*, Tech. Report 342/12/98 A, Technische Universität München, Germany, 1998.
[40] G. Karypis, *Multi-constraint Mesh Partitioning for Contact/Impact Computations*, Tech. Report 03-022, Department of Computer Science and Engineering/Army HPC Research Center, University of Minnesota, Minneapolis, MN, 2003.
[41] G. Karypis and V. Kumar, *Multilevel Algorithms for Multi-constraint Graph Partitioning*, Tech. Report 98-019, Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.
[42] G. Karypis and V. Kumar, *Multilevel Algorithms for Multi-constraint Hypergraph Partitioning*, Tech. Report 99-034, Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.
[43] T. G. Kolda, *Partitioning sparse rectangular matrices for parallel processing*, in Solving Irregularly Structured Problems in Parallel, Lecture Notes in Comput. Sci. 1457, Springer, Berlin, 1998, pp. 68–79.
[44] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
[45] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditioning* II: *Solution of* 3*D FE systems on massively parallel computers*, Internat. J. High Speed Comput., 7 (1995), pp. 191–216.
[46] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley–Teubner, Chichester, UK, 1990.
[47] K. McManus, M. Cross, C. Walshaw, S. Johnson, and P. Leggett, *A scalable strategy for the parallelization of multiphysics unstructured mesh-iterative codes on distributed-memory systems*, Internat. J. High Perform. Comput. Appl., 14 (2000), pp. 137–174.
[48] G. Meurant, *Computer Solution of Large Linear Systems*, Elsevier, Amsterdam, The Netherlands, 1999.
[49] A. Pinar, Ü. V. Çatalyürek, C. Aykanat, and M. Pinar, *Decomposing linear programs for parallel solution*, in Proceedings of the Second International Workshop on Applied Parallel Computing in Physics, Chemistry and Engineering Science, Lecture Notes in Comput. Sci. 1041, Springer, Berlin, 1996, pp. 473–482.
[50] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner, *Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics*, J. Parallel Distrib. Comput., 50 (1998), pp. 104–122.
[51] S. Plimpton, B. Hendrickson, and J. Stewart, *A parallel rendezvous algorithm for interpolation between multiple grids*, in Proceedings of the 1998 ACM/IEEE Conference on

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Supercomputing, San Jose, CA, 1998, CD-ROM, pp. 1–8.
[52] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, MA, 1996.
[53] K. Schloegel, G. Karypis, and V. Kumar, *A New Algorithm for Multi-objective Graph Partitioning*, Tech. Report 99-033, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, 1999.
[54] K. Schloegel, G. Karypis, and V. Kumar, *Graph partitioning for high-performance scientific simulations*, in CRPC Parallel Computing Handbook, J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, eds., Morgan Kaufmann, San Francisco, CA, 2002, Chap. 18, pp. 491–541.
[55] N. Selvakkumaran and G. Karypis, *Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization*, in Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '03), San Jose, CA, 2003, pp. 726–733.
[56] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranaweke, and C. V. Packer, *BEOWULF: A parallel workstation for scientific computation*, in Proceedings of the 24th International Conference on Parallel Processing, Oconomowoc, WI, 1995, pp. 11–34.
[57] R. S. Tuminaro, J. N. Shadid, and S. A. Hutchinson, *Parallel sparse matrix vector multiply software for matrices with data locality*, Concurrency: Practice and Experience, 10 (1998), pp. 229–247.
[58] B. Uçar and C. Aykanat, *Minimizing communication cost in fine-grain partitioning of sparse matrices*, in Proceedings of the 18th International Symposium on Computer and Information Sciences (ISCIS XVIII), A. Yazıcı and C. Şener, eds., Antalya, Turkey, 2003.
[59] B. Uçar and C. Aykanat, *Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies*, SIAM J. Sci. Comput., 25 (2004), pp. 1837–1859.
[60] B. Uçar and C. Aykanat, *Experiments on Hypergraph Models for Parallelizing Preconditioned Iterative Methods*, Tech. Report BU-CE-0413, Department of Computer Engineering, Bilkent University, Ankara, Turkey, 2004.
[61] B. Uçar and C. Aykanat, *A Library for Parallel Sparse Matrix-Vector Multiplies*, Tech. Report BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey, 2005.
[62] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[63] B. Vastenhouw and R. H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, SIAM Rev., 47 (2005), pp. 67–95.
[64] C. Walshaw, M. Cross, and K. McManus, *Multiphase mesh partitioning*, Appl. Math. Modeling, 25 (2000), pp. 123–140.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.