SIAM J. SCI. COMPUT.
Vol. 25, No. 6, pp. 1837–1859

ENCAPSULATING MULTIPLE COMMUNICATION-COST METRICS IN PARTITIONING SPARSE RECTANGULAR MATRICES FOR PARALLEL MATRIX-VECTOR MULTIPLIES*

BORA UÇAR† AND CEVDET AYKANAT†

Abstract. This paper addresses the problem of one-dimensional partitioning of structurally unsymmetric square and rectangular sparse matrices for parallel matrix-vector and matrix-transpose-vector multiplies. The objective is to minimize the communication cost while maintaining the balance on computational loads of processors. Most of the existing partitioning models consider only the total message volume hoping that minimizing this communication-cost metric is likely to reduce other metrics. However, the total message latency (start-up time) may be more important than the total message volume. Furthermore, the maximum message volume and latency handled by a single processor are also important metrics. We propose a two-phase approach that encapsulates all these four communication-cost metrics. The objective in the first phase is to minimize the total message volume while maintaining the computational-load balance. The objective in the second phase is to encapsulate the remaining three communication-cost metrics. We propose communication-hypergraph and partitioning models for the second phase. We then present several methods for partitioning communication hypergraphs. Experiments on a wide range of test matrices show that the proposed approach yields very effective partitioning results. A parallel implementation on a PC cluster verifies that the theoretical improvements shown by partitioning results hold in practice.

Key words. matrix partitioning, structurally unsymmetric matrix, rectangular matrix, iterative method, matrix-vector multiply, parallel computing, message volume, message latency, communication hypergraph, hypergraph partitioning

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

DOI. 10.1137/S1064827502410463

1. Introduction.

Repeated matrix-vector and matrix-transpose-vector multiplies that involve the same large, sparse, structurally unsymmetric square or rectangular matrix are the kernel operations in various iterative algorithms. For example, iterative methods such as the conjugate gradient normal equation error and residual methods (CGNE and CGNR) [15, 34] and the standard quasi-minimal residual method (QMR) [14], used for solving unsymmetric linear systems, require computations of the form $y = Ax$ and $w = A^T z$ in each iteration, where A is an unsymmetric square coefficient matrix. The least squares (LSQR) method [31], used for solving the least squares problem, and the Lanczos method [15], used for computing the singular value decomposition, require frequent computations of the form $y = Ax$ and $w = A^T z$, where A is a rectangular matrix. Iterative methods used in solving the normal equations that arise in interior point methods for linear programming require repeated computations of the form $y = AD^2A^T z$, where A is a rectangular constraint matrix and D is a diagonal matrix. Rather than forming the coefficient matrix AD^2A^T, which may be quite dense, the above computation is performed as $w = A^T z$, $x = D^2 w$, and $y = Ax$. The surrogate constraint method [29, 30, 39, 40], which is used for solving the linear feasibility problem, requires decoupled matrix-vector and matrix-transpose-vector multiplies involving the same rectangular matrix.

*Received by the editors July 1, 2002; accepted for publication (in revised form) October 13, 2003; published electronically May 25, 2004. This work was partially supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) under grant 199E013.
http://www.siam.org/journals/sisc/25-6/41046.html
†Computer Engineering Department, Bilkent University, Ankara, 06800, Turkey (ubora@cs.bilkent.edu.tr, aykanat@cs.bilkent.edu.tr).

In the framework of this paper, we assume that no computational dependency exists between the input and output vectors x and y of the $y = Ax$ multiply. The same assumption applies to the input and output vectors z and w of the $w = A^T z$ multiply. In some of the above applications, the input vector of the second multiply is obtained from the output vector of the first one—and vice versa—through linear vector operations because of intra- and interiteration dependencies. In other words, linear operations may occur only between the vectors that belong to the same space. In this setting, w and x are x-space vectors, whereas z and y are y-space vectors. These assumptions hold naturally in some of the above applications that involve a rectangular matrix. Since x- and y-space vectors are of different dimensions, they cannot undergo linear vector operations. In the remaining applications, which involve a square matrix, a computational dependency does not exist between x-space and y-space vectors because of the nature of the underlying method. Our goal is the parallelization of the computations in the above iterative algorithms through rowwise or columnwise partitioning of matrix A in such a way that the communication overhead is minimized and the computational-load balance is maintained.

In the framework of this paper, we do not address the efficient parallelization of matrix-vector multiplies involving more than one matrix with different sparsity patterns. Handling such cases requires simultaneous partitioning of the participating matrices in a method that considers the complicated interaction among the efficient parallelizations of the respective matrix-vector multiplies (see [21] for such a method). The most notable cases are the preconditioned iterative methods that use an explicit preconditioner such as an approximate inverse [3, 4, 16] $M \approx A^{-1}$. These methods involve matrix-vector multiplies with M and A. The present work can be used in such cases by partitioning matrices independently. However, this approach would suffer from communication required for reordering the vector entries between the two matrix-vector multiplies.

The standard graph-partitioning model has been widely used for one-dimensional (1D) partitioning of square matrices. This approach models matrix-vector multiply $y = Ax$ as a weighted undirected graph and partitions the vertices such that the parts are equally weighted and the total weight of the edges crossing between the parts is minimized. The partitioning constraint and objective correspond to, respectively, maintaining the computational-load balance and minimizing the total message volume. In recent works, Çatalyürek [6], Çatalyürek and Aykanat [7], and Hendrickson [19] mentioned the limitations of this standard approach. First, it tries to minimize a wrong objective function since the edge-cut metric does not model the actual communication volume. Second, it can only express square matrices and produce symmetric partitioning by enforcing identical partitions on the input and output vectors x and y. Symmetric partitioning is desirable for parallel iterative solvers on symmetric matrices because it avoids the communication of vector entries during the linear vector operations between the x-space and y-space vectors. However, this symmetric partitioning is a limitation for iterative solvers on unsymmetric square or rectangular matrices when the x-space and y-space vectors do not undergo linear vector operations.

Recently, Aykanat, Pinar, and Çatalyürek [2], Çatalyürek and Aykanat [7, 8], and Pinar, Çatalyürek, Aykanat, and Pinar [32] proposed hypergraph models for partitioning unsymmetric square and rectangular matrices with the flexibility of producing unsymmetric partitions on the input and output vectors. Hendrickson and Kolda [21] proposed a bipartite graph model for partitioning rectangular matrices with the same flexibility. A distinct advantage of the hypergraph model over the bipartite graph model is that the hypergraph model correctly encodes the total message volume into

its partitioning objective. Several recently proposed alternative partitioning models for parallel computing were discussed in the excellent survey by Hendrickson and Kolda [20]. As noted in the survey, most of the partitioning models mainly consider minimizing the total message volume. However, the communication overhead is a function of the message latency (start-up time) as well as the message volume. Depending on the machine architecture and problem size, the communication overhead due to the message latency may be much higher than the overhead due to the message volume [12]. None of the works listed in the survey address minimizing the total message latency. Furthermore, the maximum message volume and latency handled by a single processor are also crucial cost metrics to be considered in partitionings. As also noted in the survey [20], new approaches that encapsulate these four communication-cost metrics are needed.

In this work, we propose a two-phase approach for minimizing multiple communication-cost metrics. The objective in the first phase is to minimize the total message volume while maintaining the computational-load balance. This objective is achieved through partitioning matrix A within the framework of the existing 1D matrix partitioning methods. The partitioning obtained in the first phase is an input to the second phase so that it determines the computational loads of processors while setting a lower bound on the total message volume. The objective in the second phase is to encapsulate the remaining three communication-cost metrics while trying to attain the total message volume bound as much as possible. The metrics minimized in the second phase are not simple functions of the cut edges or hyperedges or vertex weights defined in the existing graph and hypergraph models even in the multiobjective [37] and multiconstraint [25] frameworks. Besides, these metrics cannot be assessed before a partition is defined. Hence, we anticipate a two-phase approach. Pinar and Hendrickson [33] also adopt a multiphase approach for handling complex partitioning objectives. Here, we focus on the second phase and do not go back and forth between the phases. Therefore, our contribution can be seen as a postprocess to the existing partitioning methods. For the second phase, we propose a *communication-hypergraph* partitioning model. A hypergraph is a generalization of a graph in which hyperedges (nets) can connect more than two vertices. The vertices of the communication hypergraph, with proper weighting, represent primitive communication operations, and the nets represent processors. By partitioning the communication hypergraph into equally weighted parts such that nets are split among as few vertex parts as possible, the model maintains the balance on message-volume loads of processors and minimizes the total message count. The model also enables incorporating the minimization of the maximum message-count metric.

The organization of the paper is as follows. Section 2 gives background material on the parallel matrix-vector multiplies and hypergraph partitioning problem. The proposed communication-hypergraph and partitioning models are discussed in section 3. Section 4 presents three methods for partitioning communication hypergraphs. Finally, experimental results are presented and discussed in section 5.

2. Background.

2.1. Parallel multiplies. Suppose that rows and columns of an $M \times N$ matrix A are permuted into a $K \times K$ block structure

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

for *rowwise* or *columnwise* partitioning, where K is the number of processors. Block $A_{k\ell}$ is of size $M_k \times N_\ell$, where $\sum_k M_k = M$ and $\sum_\ell N_\ell = N$. In rowwise partitioning, each processor P_k holds the kth row stripe $[A_{k1} \cdots A_{kK}]$ of size $M_k \times N$. In columnwise partitioning, P_k holds the kth column stripe $[A_{1k}^T \cdots A_{Kk}^T]^T$ of size $M \times N_k$. We assume that either row stripes or column stripes have a nearly equal number of nonzeros for having computational-load balance. We assume rowwise partitioning of A throughout the following discussion.

2.1.1. Matrix-vector multiply. Consider an iterative algorithm involving repeated matrix-vector multiplies of the form $y = Ax$, where y and x are column vectors of size M and N, respectively. The rowwise partition of matrix A defines a partition on the output vector y, whereas the input vector x is partitioned conformably with the column permutation of A. That is, y and x vectors are partitioned as $y = [y_1^T \cdots y_K^T]^T$ and $x = [x_1^T \cdots x_K^T]^T$, where processor P_k computes subvector y_k of size M_k while holding subvector x_k of size N_k. In order to avoid communication during linear vector operations, all other x-space and y-space vectors are partitioned conformably with the partitions on x and y vectors, respectively.

Row-parallel $y = Ax$ executes the following steps at each processor P_k:

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 the respective column of A. This multicast-like operation is referred to here as an *expand* operation.

2.1.2. Matrix-vector and matrix-transpose-vector multiplies. Consider an iterative algorithm involving repeated matrix-vector and matrix-transpose-vector multiplies of the form $y = Ax$ and $w = A^T z$. A rowwise partition of A induces a columnwise partition of A^T. Therefore, the partition on the z vector defined by the columnwise partition of A^T will be conformable with that on the y vector. That is, $z = [z_1^T \cdots z_K^T]^T$ and $y = [y_1^T \cdots y_K^T]^T$, where z_k and y_k are both of size M_k. In a dual manner, the columnwise permutation of A induces a rowwise permutation of A^T. Therefore, the partition on the w vector induced by the rowwise permutation of A^T will be conformable with that on the x vector. That is, $w = [w_1^T \cdots w_K^T]^T$ and $x = [x_1^T \cdots x_K^T]^T$, where w_k and x_k are both of size N_k.

We use a *column-parallel* algorithm for $w = A^T z$ and the row-parallel algorithm for $y = Ax$ and thus obtain a *row-column-parallel* algorithm. In $y = Ax$, processor P_k holds x_k and computes y_k. In $w = A^T z$, P_k holds z_k and computes w_k.

Column-parallel $w = A^T z$ executes the following steps at processor P_k:

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

The multinode accumulation on w_k-vector entries is referred to here as the *fold* operation.

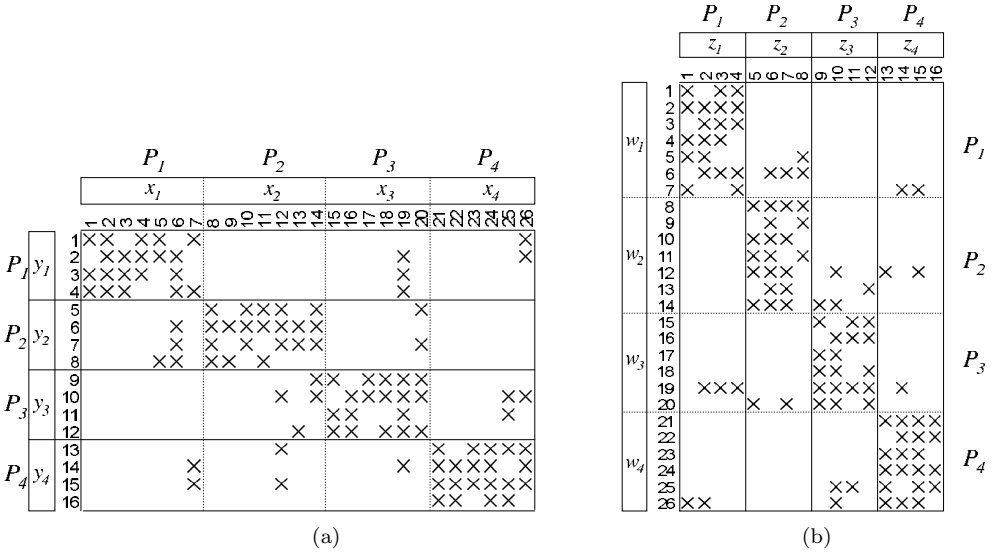


FIG. 2.1. 4×4 *block structures of a sample matrix* A: (a) A_{BL} *for row-parallel* $y = Ax$ *and* (b) $(A^T)_{BL}$ *for column-parallel* $w = A^T z$.

2.1.3. Analysis. Here, we restate and summarize the facts given in [7, 21] for the communication requirement in the row-parallel $y = Ax$ and column-parallel $w = A^T z$. We will use Figure 2.1 for a better understanding of these facts. Figure 2.1 displays 4×4 block structures of a 16×26 sample matrix A and its transpose. In Figure 2.1(a), horizontal solid lines identify a partition on the rows of A and on vector y, whereas vertical dashed lines identify *virtual column stripes* inducing a partition on vector x. In Figure 2.1(b), vertical solid lines identify a partition on the columns of A^T and on vector z, whereas horizontal dashed lines identify *virtual row stripes* inducing a partition on vector w. The computational-load balance is maintained by assigning $25, 26, 25$, and 25 nonzeros to processors P_1, P_2, P_3, and P_4, respectively.

FACT 1. *The number of messages sent by processor* P_k *in row-parallel* $y = Ax$ *is equal to the number of nonzero off-diagonal blocks in the* k*th virtual column stripe of* A. *The volume of messages sent by* P_k *is equal to the sum of the number of nonzero columns in each off-diagonal block in the* k*th virtual column stripe.*

In Figure 2.1(a), P_2, holding x-vector block $x_2 = x[8:14]$, sends vector $\hat{x}_2^3 = x[12:14]$ to P_3 because of nonzero columns 12, 13, and 14 in A_{32}. P_3 needs those entries to compute $y[9]$, $y[10]$, and $y[12]$. Similarly, P_2 sends $\hat{x}_2^4 = x[12]$ to P_4 because of the nonzero column 12 in A_{42}. Hence, the number of messages sent by P_2 is 2 with a total volume of four words. Note that P_2 effectively expands $x[12]$ to P_3 and P_4.

FACT 2. *The number of messages sent by processor* P_k *in column-parallel* $w = A^T z$ *is equal to the number of nonzero off-diagonal blocks in the* k*th column stripe of* A^T. *The volume of messages sent by* P_k *is equal to the sum of the number of nonzero rows in each off-diagonal block in the* k*th column stripe of* A^T.

In Figure 2.1(b), P_3, holding z-vector block $z_3 = z[9:12]$, computes the off-diagonal block products $w_2^3 = (A^T)_{23} \times z_3$ and $w_4^3 = (A^T)_{43} \times z_3$. It then forms vectors $\hat{w}_2^3$ and $\hat{w}_4^3$ to be sent to P_2 and P_4, respectively. $\hat{w}_2^3$ contains its contribution to $w[12:14]$ due to the nonzero rows 12, 13, and 14 in $(A^T)_{23}$. Accordingly, $\hat{w}_4^3$ contains its contribution to $w[25:26]$ due to the nonzero rows 25 and 26 in $(A^T)_{43}$. Hence, P_3 sends two messages with a total volume of five words.

FACT 3. *Communication patterns of $y=Ax$ and $w=A^Tz$ multiplies are duals of each other. If a processor P_k sends a message to P_ℓ containing some x_k entries in $y=Ax$, then P_ℓ sends a message to P_k containing its contributions to the corresponding w_k entries in $w=A^Tz$.*

Consider the communication between processors P_2 and P_3. In $y=Ax$, P_2 sends a message to P_3 containing $x[12:14]$, whereas in $w=A^Tz$, P_3 sends a dual message to P_2 containing its contributions to $w[12:14]$.

FACT 4. *The total number of messages in the $y=Ax$ or $w=A^Tz$ multiply is equal to the number of nonzero off-diagonal blocks in A or A^T. The total volume of messages is equal to the sum of the number of nonzero columns or rows in each off-diagonal block in A or A^T, respectively.*

In Figure 2.1, there are nine nonzero off-diagonal blocks, containing a total of 13 nonzero columns or rows in A or A^T. Hence, the total number of messages in $y=Ax$ or $w=A^Tz$ is nine, and the total volume of messages is 13 words.

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 n_i 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 as $Nets(v_j)$, which is also extended to a set of vertices appropriately. The degree of a vertex v_j is denoted by $d_j=|Nets(v_j)|$. Weights can be associated with vertices.

$\Pi=\{\mathcal{V}_1,\ldots,\mathcal{V}_K\}$ is a K-way vertex partition of $\mathcal{H}=(\mathcal{V},\mathcal{N})$ if each part $\mathcal{V}_k$ is nonempty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. In Π, a net is said to *connect* a part if it has at least one pin 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. In Π, 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*:

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

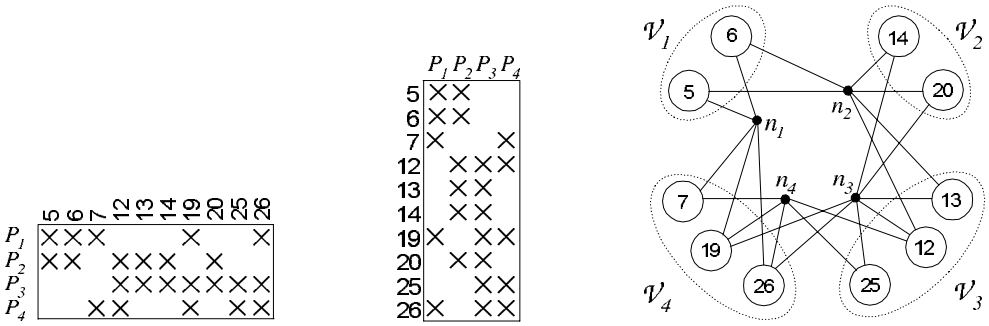
This objective function is widely used in the VLSI community, and it is referred to as the *connectivity*–1 metric [28]. The partitioning constraint is to maintain a balance on part weights, i.e.,

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

where W_{max} is the weight of the part with the maximum weight, W_{avg} is the average part weight, and ϵ is a predetermined imbalance ratio. This problem is NP-hard [28].

3. Models for minimizing communication cost. In this section, we present our hypergraph partitioning models for the second phase of the proposed two-phase approach. We assume that a K-way rowwise partition of matrix A is obtained in the first phase with the objective of minimizing the total message volume while maintaining the computational-load balance.

3.1. Row-parallel $y=Ax$. Let A_{BL} denote a block-structured form (see (2.1)) of A for the given rowwise partition.



(a) (b) (c)

FIG. 3.1. *Communication matrices* (a) C *for row-parallel* $y = Ax$, (b) C^T *for column-parallel* $w = A^T z$, *and* (c) *the associated communication hypergraph and its four-way partition.*

3.1.1. Communication-hypergraph model. We identify two sets of columns in A_{BL}: *internal* and *coupling*. Internal columns have nonzeros only in one row stripe. The x-vector entries that are associated with these columns should be assigned to the respective processors to avoid unnecessary communication. Coupling columns have nonzeros in more than one row stripe. The x-vector entries associated with the coupling columns, referred to as x_C, necessitate communication. The proposed approach considers partitioning these x_C-vector entries to reduce the total message count and the maximum message volume. Consequences of this partitioning on the total message volume will be addressed in section 3.4.

We propose a rowwise compression of A_{BL} to construct a matrix C, referred to here as the *communication matrix*, which summarizes the communication requirement of row-parallel $y = Ax$. First, for each $k = 1, \ldots, K$, we compress the kth row stripe into a single row with the sparsity pattern being equal to the union of the sparsities of all rows in that row stripe. Then, we discard the internal columns of A_{BL} from the column set of C. Note that a nonzero entry c_{kj} remains in C if coupling column j has at least one nonzero in the kth row stripe. Therefore, rows of C correspond to processors in such a way that the nonzeros in row k identify the subset of x_C-vector entries needed by processor P_k. In other words, nonzeros in column j of C identify the set of processors that need $x_C[j]$. Since the columns of C correspond to the coupling columns of A_{BL}, C has $N_C = |x_C|$ columns, each of which has at least two nonzeros. Figure 3.1(a) illustrates communication matrix C obtained from A_{BL} shown in Figure 2.1(a). For example, the fourth row of matrix C has nonzeros in columns 7, 12, 19, 25, and 26 corresponding to the nonzero coupling columns in the fourth row stripe of A_{BL}. These nonzeros summarize the need of processor P_4 for x_C-vector entries $x[7], x[12], x[19], x[25]$, and $x[26]$ in row-parallel $y = Ax$.

Here, we exploit the *row-net* hypergraph model for sparse matrix representation [7, 8] to construct a *communication hypergraph* from matrix C. In this model, communication matrix C is represented as a hypergraph $\mathcal{H}_C = (\mathcal{V}, \mathcal{N})$ on N_C vertices and K nets. Vertex and net sets $\mathcal{V}$ and $\mathcal{N}$ correspond to the columns and rows of matrix C, respectively. There exist one vertex v_j for each column j and one net n_k for each row k. Consequently, vertex v_j represents $x_C[j]$, and net n_k represents processor P_k. Net n_k contains vertices corresponding to the columns that have a nonzero in row k, i.e., $v_j \in n_k$ if and only if $c_{kj} \neq 0$. $Nets(v_j)$ contains the set of nets corresponding to the rows that have a nonzero in column j. In the proposed model, each vertex v_j corresponds to the atomic task of expanding $x_C[j]$. Figure 3.1(c) shows

the communication hypergraph obtained from the communication matrix C. In this figure, white and black circles represent, respectively, vertices and nets, and straight lines show the pins of nets.

3.1.2. Minimizing total latency and maximum volume. Here, we will show that minimizing the total latency and maintaining the balance on message-volume loads of processors can be modeled as a hypergraph partitioning problem on the communication hypergraph. Consider a K-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of communication hypergraph $\mathcal{H}_C$. Without loss of generality, we assume that part $\mathcal{V}_k$ is assigned to processor P_k for $k = 1, \ldots, K$. The consistency of the proposed model for accurate representation of the total latency requirement depends on the condition that each net n_k connects part $\mathcal{V}_k$ in Π, i.e., $\mathcal{V}_k \in \Lambda_k$. We first assume that this condition holds and discuss the appropriateness of the assumption later in section 3.4.

Since Π is defined as a partition on the vertex set of $\mathcal{H}_C$, it induces a processor assignment for the atomic expand operations. Assigning vertex v_j to part $\mathcal{V}_\ell$ is decoded as assigning the responsibility of expanding $x_C[j]$ to processor P_ℓ. The destination set $\mathcal{E}_j$ in this expand operation is the set of processors corresponding to the nets that contain v_j except P_ℓ, i.e., $\mathcal{E}_j = Nets(v_j) - \{P_\ell\}$. If $v_j \in n_\ell$, then $|\mathcal{E}_j| = d_j - 1$; otherwise, $|\mathcal{E}_j| = d_j$. That is, the message-volume requirement of expanding $x_C[j]$ will be $d_j - 1$ or d_j words in the former and latter cases. Here, we prefer to associate a weight of $d_j - 1$ with each vertex v_j because the latter case is expected to be rare in partitionings. In this way, satisfying the partitioning constraint in (2.3) relates to maintaining the balance on message-volume loads of processors. Here, the message-volume load of a processor refers to the volume of outgoing messages. We prefer to omit the incoming volume in considering the message-volume load of a processor with the assumption that each processor has enough local computation that overlaps with incoming messages in the network.

Consider a net n_k with the connectivity set Λ_k in partition Π. Let $\mathcal{V}_\ell$ be a part in Λ_k other than $\mathcal{V}_k$. Also, let v_j be a vertex of net n_k in $\mathcal{V}_\ell$. Since $v_j \in \mathcal{V}_\ell$ and $v_j \in n_k$, processor P_ℓ will be sending $x_C[j]$ to processor P_k due to the associated expand assignment. A similar *send* requirement is incurred by all other vertices of net n_k in $\mathcal{V}_\ell$. That is, the vertices of net n_k that lie in $\mathcal{V}_\ell$ show that P_ℓ must gather all x_C-vector entries corresponding to vertices in $n_k \cap \mathcal{V}_\ell$ into a single message to be sent to P_k. The size of this message will be $|n_k \cap \mathcal{V}_\ell|$ words. Hence, a net n_k with the connectivity set Λ_k shows that P_k will be receiving a message from each processor in Λ_k except itself. Hence, a net n_k with the connectivity λ_k shows $\lambda_k - 1$ messages to be received by P_k because $\mathcal{V}_k \in \Lambda_k$ (due to the consistency condition). The sum of the connectivity–1 values of all K nets, i.e., $\sum_{n_k}(\lambda_k - 1)$, will give the total number of messages received. As the total number of incoming messages is equal to the total number of outgoing messages, minimizing the objective function in (2.2) corresponds to minimizing the total message latency.

Figure 3.2(a) shows a partition of a generic communication hypergraph to clarify the above concepts. The main purpose of the figure is to show the number rather than the volume of messages, so multiple pins of a net in a part are contracted into a single pin. Arrows along the pins show the directions of the communication in the underlying expand operations. Figure 3.2(a) shows processor P_k receiving messages from processors P_ℓ and P_m because net n_k connects parts $\mathcal{V}_k$, $\mathcal{V}_\ell$, and $\mathcal{V}_m$. The figure also shows P_k sending messages to three different processors P_h, P_i, and P_j due to nets n_h, n_i, and n_j connecting part $\mathcal{V}_k$. Hence, the number of messages sent by P_k is equal to $|Nets(\mathcal{V}_k)| - 1$.

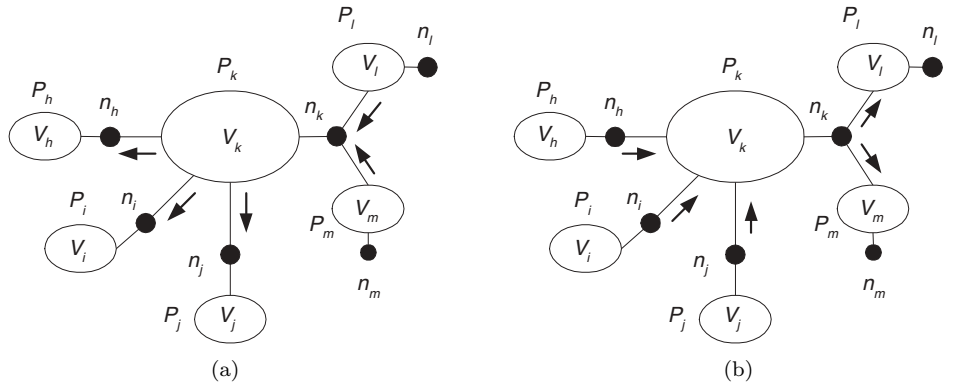


FIG. 3.2. *Generic communication-hypergraph partitions for showing incoming and outgoing messages of processor P_k in* (a) *row-parallel $y = Ax$ and* (b) *column-parallel $w = A^T z$.*

3.2. Column-parallel $w = A^T z$. Let $(A^T)_{BL}$ denote a block-structured form (see (2.1)) of A^T for the given rowwise partition of A.

3.2.1. Communication-hypergraph model. A communication hypergraph for column-parallel $w = A^T z$ can be obtained from $(A^T)_{BL}$ as follows. We first determine the internal and coupling *rows* to form w_C, i.e., the w-vector entries that necessitate communication. We then apply a columnwise compression, similar to that in section 3.1.1, to obtain communication matrix C^T. Figure 3.1(b) illustrates communication matrix C^T obtained from the block structure of $(A^T)_{BL}$ shown in Figure 2.1(b). Finally, we exploit the *column-net* hypergraph model for sparse matrix representation [7, 8] to construct a communication hypergraph from matrix C^T. The row-net and column-net hypergraph models are duals of each other. The column-net representation of a matrix is equivalent to the row-net representation of its transpose and vice versa. Therefore, the resulting communication hypergraph derived from C^T will be topologically identical to that of the row-parallel $y = Ax$ with dual communication-requirement association. For example, the communication hypergraph shown in Figure 3.1(c) represents communication matrix C^T as well. In this hypergraph, net n_k represents processor P_k as before. However, vertices of net n_k denote the set of w_C-vector entries for which processor P_k generates partial results. Each vertex v_j corresponds to the atomic task of folding on $w_C[j]$. Hence, $Nets(v_j)$ denotes the set of processors that generates a partial result for $w_C[j]$.

3.2.2. Minimizing total latency and maximum volume. Consider a K-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of communication hypergraph $\mathcal{H}_C$ with the same part-to-processor assignment and consistency condition as in section 3.1.2. Since the vertices of $\mathcal{H}_C$ correspond to fold operations, assigning a vertex v_j to part $\mathcal{V}_\ell$ in Π is decoded as assigning the responsibility of folding on $w_C[j]$ to processor P_ℓ. Consider a net n_k with the connectivity set Λ_k. Let $\mathcal{V}_\ell$ be a part in Λ_k other than $\mathcal{V}_k$. Also, let v_j be a vertex of net n_k in $\mathcal{V}_\ell$. Since $v_j \in \mathcal{V}_\ell$ and $v_j \in n_k$, processor P_k will be sending its partial result for $w_C[j]$ to P_ℓ because of the associated fold assignment to P_ℓ. A similar *send* requirement is incurred to P_k by all other vertices of net n_k in $\mathcal{V}_\ell$. That is, the vertices of net n_k that lie in $\mathcal{V}_\ell$ show that P_k must gather all partial w_C results corresponding to vertices in $n_k \cap \mathcal{V}_\ell$ into a single message to be sent to P_ℓ. The size of this message will be $|n_k \cap \mathcal{V}_\ell|$ words. Hence, a net n_k with connectivity set Λ_k shows that P_k will be sending a message to each processor in Λ_k except itself.

Hence, a net n_k with the connectivity λ_k shows $\lambda_k - 1$ messages to be sent by P_k because $\mathcal{V}_k \in \Lambda_k$ (due to the consistency condition). The sum of the connectivity-1 values of all K nets, i.e., $\sum_{n_k}(\lambda_k - 1)$, will give the total number of messages sent. Therefore, minimizing the objective function in (2.2) corresponds to minimizing the total message latency.

As vertices of $\mathcal{H}_C$ represent atomic fold operations, the weighted sum of vertices in a part will relate to the volume of incoming messages of the respective processor with vertex degree weighting. However, as mentioned earlier, we prefer to define the message-volume load of a processor as the volume of outgoing messages. Each vertex v_j of net n_k that lies in a part other than $\mathcal{V}_k$ incurs one word of message-volume load to processor P_k. In other words, each vertex of net n_k that lies in part $\mathcal{V}_k$ relieves P_k from sending a word. Thus, the message-volume load of P_k can be computed in terms of the vertices in part $\mathcal{V}_k$ as $|n_k| - |n_k \cap \mathcal{V}_k|$. Here, we prefer to associate unit weights with vertices so that maintaining the partitioning constraint in (2.3) corresponds to an approximate message-volume load balancing. This approximation will prove to be a reasonable one if the net sizes are close to each other.

Figure 3.2(b) shows a partition of a generic communication hypergraph to illustrate the number of messages. Arrows along the pins of nets show the directions of messages for fold operations. Figure 3.2(b) shows processor P_k sending messages to processors P_ℓ and P_m because net n_k connects parts $\mathcal{V}_k$, $\mathcal{V}_\ell$, and $\mathcal{V}_m$. Hence, the number of messages sent by P_k is equal to $\lambda_k - 1$.

3.3. Row-column-parallel $y = Ax$ and $w = A^T z$. To minimize the total message count in $y = Ax$ and $w = A^T z$, we use the same communication hypergraph $\mathcal{H}_C$ with different vertex weightings. As in sections 3.1.2 and 3.2.2, the cutsize of a partition of $\mathcal{H}_C$ quantifies the total number of messages sent both in $y = Ax$ and $w = A^T z$. This property is in accordance with Facts 3 and 4 given in section 2.1.3. Therefore, minimizing the objective function in (2.2) corresponds to minimizing the total message count in row-column-parallel $y = Ax$ and $w = A^T z$ multiplies.

Vertex weighting for maintaining the message-volume balance needs special attention. If there is a synchronization point between $w = A^T z$ and $y = Ax$, the *multi-constraint* partitioning [25] should be adopted with two different weightings to impose a communication-volume balance in both multiply phases. If there is no synchronization point between the two multiplies (e.g., $y = AA^T z$), we recommend imposing a balance on aggregate message-volume loads of processors by associating an aggregate weight of $(d_j - 1) + 1 = d_j$ with each vertex v_j.

3.4. Remarks on partitioning models. Consider a net n_k which does not satisfy the consistency condition in a partition Π of $\mathcal{H}_C$. Since $\mathcal{V}_k \notin \Lambda_k$, processor P_k will be receiving a message from each processor in Λ_k in row-parallel $y = Ax$. Recall that P_k needs the x_C-vector entries represented by the vertices in net n_k independent of the connectivity between part $\mathcal{V}_k$ and net n_k. In a dual manner, P_k will be sending a message to each processor in Λ_k in column-parallel $w = A^T z$. Hence, net n_k with the connectivity λ_k will incur λ_k incoming or outgoing messages instead of $\lambda_k - 1$ messages determined by the cutsize of Π. That is, our model undercounts the actual number of messages by one for each net dissatisfying the consistency condition. In the worst case, this deviation may be as high as K messages in total. This deficiency of the proposed model may be overcome by enforcing the consistency condition through exploiting the *partitioning with fixed vertices* feature, which exists in some of the hypergraph-partitioning tools [1, 9]. We discuss such a method in section 4.1.

Partitioning x_C-vector entries affects the message-volume requirement determined in the first phase. The message-volume requirement induced by the partitioning in the first phase is equal to $nnz(C)-N_C$ for row-parallel $y=Ax$. Here, $nnz(C)$ and N_C denote, respectively, the number of nonzeros and the number of columns in communication matrix C. Consider $x_C[j]$ corresponding to column j of C. Assigning $x_C[j]$ to any one of the processors corresponding to the rows of C that have a nonzero in column j will not change the message-volume requirement. However, assigning it to some other processor will increase the message-volume requirement for expanding $x_C[j]$ by one word. In a partition Π of communication hypergraph $\mathcal{H}_C$, this case corresponds to having a vertex $v_j \in \mathcal{V}_k$ while $v_j \notin n_k$. In other words, processor P_k holds and expands $x_C[j]$ although it does not need it for local computations. A dual discussion holds for column-parallel $w=A^Tz$, where such a vertex-to-part assignment corresponds to assigning the responsibility of folding on a particular w_C-vector entry to a processor which does not generate a partial result for that entry. In the worst case, the increase in the message volume may be as high as N_C words in total for both types of multiplies. In hypergraph-theoretic view, the total message volume will be in between $\sum_k |n_k|-|\mathcal{V}|$ and $\sum_k |n_k|$, where $\sum_k |n_k| = nnz(C)$ and $|\mathcal{V}|=N_C$.

The proposed communication-hypergraph partitioning models exactly encode the total number of messages and the maximum message volume per processor metrics into the hypergraph partitioning objective and constraint, respectively, under the above conditions. The models do not directly encapsulate the metric of the maximum number of messages per processor; however, it is possible to address this metric within the partitioning framework. We give a method in section 4.3 to address this issue.

The allowed imbalance ratio (ϵ) is an important parameter in the proposed models. Choosing a large value for ϵ relaxes the partitioning constraint. Thus, large ϵ values enable the associated partitioning methods to achieve better partitioning objectives through enlarging the feasible search space. Hence, large ϵ values favor the total message-count metric. On the other hand, small ϵ values favor the maximum message-volume metric by imposing a tighter constraint on the part weights. Thus, ϵ should be chosen according to the target machine architecture and problem characteristics to trade the total latency for the maximum volume.

3.5. Illustration on the sample matrix. Figure 3.1(c) displays a four-way partition of the communication hypergraph, where closed dashed curves denote parts. Nets and their associated parts are kept close to each other. Note that the consistency condition is satisfied for the given partition. In the figure, net n_2 with the connectivity set $\Lambda_2=\{\mathcal{V}_1,\mathcal{V}_2,\mathcal{V}_3\}$ shows processor P_2 receiving messages from processors P_1 and P_3 in row-parallel $y=Ax$. In a dual manner, net n_2 shows P_2 sending messages to P_1 and P_3 in column-parallel $w=A^Tz$. Since the connectivities of nets n_1, n_2, n_3, and n_4 are, respectively, 2, 3, 3, and 2, the total message count is equal to $(2-1)+(3-1)+(3-1)+(2-1)=6$ in both types of multiplies. Hence, the proposed approach reduces the number of messages from nine (see section 2.1.3) to six by yielding the given partition on x_C-vector (w_C-vector) entries.

In the proposed two-phase approach, partitioning x_C-vector entries in the second phase can also be regarded as repermuting coupling columns of A_{BL} obtained in the first phase. In a dual manner, partitioning w_C-vector entries can be regarded as repermuting coupling rows of $(A^T)_{BL}$. Figure 3.3 shows the repermuted A_{BL} and $(A^T)_{BL}$ matrices induced by the sample communication-hypergraph partition shown in Figure 3.1(c). The total message count is 6 as enumerated by the total number

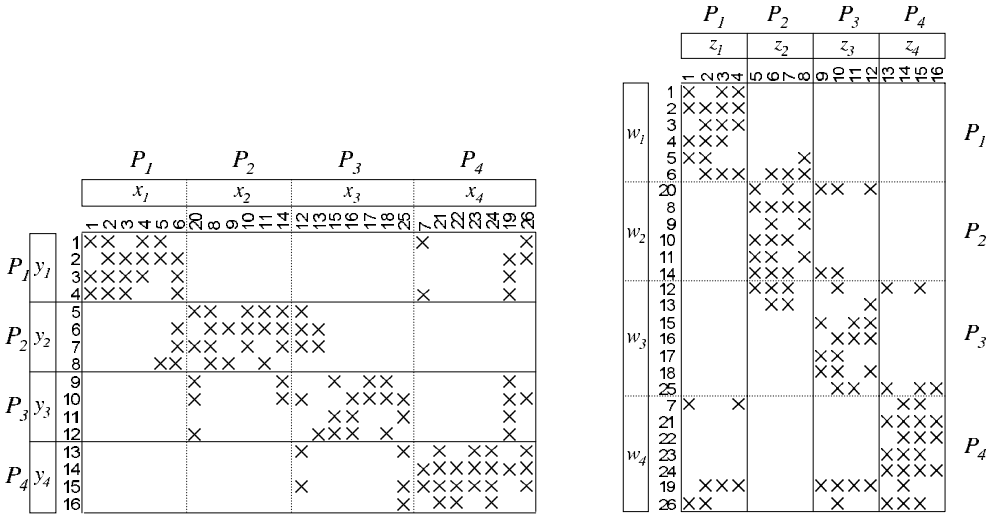


FIG. 3.3. *Final* 4×4 *block structures:* (a) A_{BL} *for row-parallel* $y = Ax$, *and* (b) $(A^T)_{BL}$ *for column-parallel* $w = A^T z$, *induced by* 4*-way communication-hypergraph partition in Figure* 3.1(c).

of nonzero off-diagonal blocks according to Fact 4, thus matching the cutsize of the partition given in Figure 3.1(c).

As seen in Figure 3.1(c), each vertex in each part is a pin of the net associated with that part. Therefore, for both types of multiplies, the sample partitioning does not increase the total message volume, and it remains at its lower bound which is $\sum_k |n_k| - |\mathcal{V}| = (5+6+7+5) - 10 = 13$ words. This value can also be verified from the repermuted matrices given in Figure 3.3 by enumerating the total number of nonzero columns in the off-diagonal blocks according to Fact 4.

For row-parallel $y = Ax$, the message-volume load estimates of processors are 2, 2, 4, and 5 words according to the vertex weighting proposed in section 3.1.2. These estimates are expected to be exact since each vertex in each part is a pin of the net associated with that part. This expectation can be verified from the repermuted A_{BL} matrix given in Figure 3.3(a) by counting the number of nonzero columns in the off-diagonal blocks of the virtual column stripes according to Fact 1.

For column-parallel $w = A^T z$, the message-volume load estimates of processors are 2, 2, 3, and 3 words according to the unit vertex weighting proposed in section 3.2.2. However, the actual message-volume loads of processors are 3, 4, 4, and 2 words. These values can be obtained from Figure 3.3(b) by counting the number of nonzero rows in the off-diagonal blocks of the virtual row stripes according to Fact 2. The above values yield an estimated imbalance ratio of 20% and an actual imbalance ratio of 23%. The discrepancy between the actual and estimated imbalance ratios is because of the differences in net sizes.

4. Algorithms for communication-hypergraph partitioning. We present the following three methods for partitioning communication hypergraphs. In these methods, minimizing the cutsize while maintaining the partitioning constraint corresponds to minimizing the total number of messages while maintaining the balance on communication-volume loads of processors according to models proposed in sections 3.1.2 and 3.2.2. Method *PaToH-fix* is presented to show the feasibility of using

a publicly available tool to partition communication hypergraphs. Method *MSN* involves some tailoring toward partitioning communication hypergraphs. Method *MSNmax* tries to incorporate the minimization of the maximum message count per processor into the *MSN* method.

4.1. PaToH-fix: Recursive bipartitioning with fixed vertices. The *multilevel* paradigm has been successfully used in graph and hypergraph partitioning leading to successful tools [9, 17, 22, 24, 26]. The multilevel heuristics consist of three steps: *coarsening*, *initial partitioning*, and *uncoarsening*. In the first step, a multilevel clustering is applied starting from the original graph/hypergraph by adopting various matching/clustering heuristics until the number of vertices in the coarsened graph/hypergraph falls below a predetermined threshold. Clustering corresponds to coalescing highly interacting vertices of a level to supervertices of the next level. In the second step, a partition is obtained on the coarsest graph/hypergraph using various heuristics. In the third step, the partition found in the second step is successively projected back toward the original graph/hypergraph by refining the projected partitions on the intermediate level uncoarser graphs/hypergraphs using various heuristics. A common refinement heuristic is FM, which is an iterative improvement method proposed for graph/hypergraph bipartitioning by Fiduccia and Mattheyses [13] as a faster implementation of the KL algorithm proposed by Kernighan and Lin [27].

In this work, we use the multilevel hypergraph-partitioning tool PaToH [9] for partitioning communication hypergraphs. Recall that the communication-hypergraph partitioning differs from the conventional hypergraph partitioning because of the net-to-part association needed to satisfy the consistency condition mentioned in sections 3.1.2 and 3.2.2. We exploit the *partitioning with fixed vertices* feature supported by PaToH to achieve this net-to-part association as follows. The communication hypergraph is augmented with K zero-weighted artificial vertices of degree one. Each artificial vertex v_k^* is added to a unique net n_k as a new pin and marked as fixed to part $\mathcal{V}_k$. This augmented hypergraph is fed to PaToH for K-way partitioning. PaToH generates K-way partitions with these K labeled vertices lying in their fixed parts thus establishing the required net-to-part association. A K-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ generated by PaToH is decoded as follows. The atomic communication tasks associated with the actual vertices assigned to part $\mathcal{V}_k$ are assigned to processor P_k, whereas v_k^* does not incur any communication task.

4.2. MSN: Direct K-way partitioning. Most of the partitioning tools, including PaToH, achieve K-way partitioning through recursive bisection. In this scheme, first a two-way partition is obtained, and then this two-way partition is further bipartitioned recursively. The connectivity-1 cutsize metric (see (2.2)) is easily handled through net splitting [8] during recursive bisection steps. Although the recursive-bisection paradigm is successful in K-way partitioning in general, its performance degrades for hypergraphs with large net sizes. Since communication hypergraphs have nets with large sizes, this degradation is also expected to be notable with *PaToH-fix*. In order to alleviate this problem, we have developed a multilevel direct K-way hypergraph partitioner (*MSN*) by integrating Sanchis's direct K-way refinement (SN) algorithm [35] to the uncoarsening step of the multilevel framework.

The coarsening step of *MSN* is essentially the same as that of PaToH. In the initial partitioning step, a K-way partition on the coarsest hypergraph is obtained by using a simple constructive approach which mainly aims to satisfy the balance constraint. In *MSN*, the net-to-part association is handled implicitly rather than by introducing artificial vertices. This association is established in the initial partitioning step through

associating each part with a distinct net which connects that part, and it is maintained later in the uncoarsening step. In the uncoarsening step, the SN algorithm, which is a generalization of the two-way FM paradigm to K-way refinement [11, 36], is used. SN, starting from a K-way initial partition, performs a number of passes until it finds a locally optimum partition, where each pass consists of a sequence of vertex moves. The fundamental idea is the notion of *gain*, which is the decrease in the cutsize of a partition due to a vertex moving from a part to another. The local search strategy adopted in the SN approach repeatedly moves a vertex with the maximum gain even if that gain is negative and records the best partition encountered during a pass. Allowing tentative moves with negative gains brings restricted "hill-climbing ability" to the approach.

In the SN algorithm, there are $K-1$ possible moves for each vertex. The algorithm stores the gains of the moves from a source part in $K-1$ associated priority queues—one for each possible destination part. Hence, the algorithm uses $K(K-1)$ priority queues with a space complexity of $O(N_C K)$, which may become a memory problem for large K. The moves with the maximum gain are selected from each of these $K(K-1)$ priority queues, and the one that maintains the balance criteria is performed. After the move, only the move gains of the vertices that share a net with the moved vertex may need to be updated. This may lead to updates on at most $4K-6$ priority queues. Within a pass, a vertex is allowed to move at most once.

4.3. MSNmax: Considering maximum message latency. The proposed models do not encapsulate the minimization of the maximum message latency per processor. By similar reasoning in defining the message-volume load of a processor as the volume of outgoing messages, we prefer to define the message-latency load of a processor in terms of the number of outgoing messages. Here, we propose a practical way of incorporating the minimization of the maximum message-count metric into the *MSN* method. The resulting method is referred to here as *MSNmax*. *MSNmax* differs from *MSN* only in the SN refinement scheme used in the uncoarsening phase. *MSNmax* still relies on the same gain notion and maintains updated move gains in $K(K-1)$ priority queues. The difference lies in the move selection policy, which favors the moves that reduce the message counts of overloaded processors. Here, a processor is said to be overloaded if its message count is above the average by a prescribed percentage (e.g., 25% is used in this work). For this purpose, message counts of processors are maintained during the course of the SN refinement algorithm.

For row-parallel $y = Ax$, the message count of a processor can be reduced by moving vertices out of the associated part. Recall that moving a vertex from a part corresponds to relieving the associated processor of the respective atomic expand task. For this reason, only the priority queues of the overloaded parts are considered for selecting the move with the maximum gain. For column-parallel $w = A^T z$, the message count of a processor P_k can be reduced by reducing the connectivity of the associated net n_k through moves from the parts in $\Lambda_k - \{P_k\}$. Therefore, only the priority queues of the parts that are in the connectivity sets of the nets associated with the overloaded parts are considered. For both types of parallel multiplies, moves selected from the restricted set of priority queues are likely to decrease the message counts of overloaded processors besides decreasing the total message count.

5. Experimental results. We have tested the performance of the proposed models and associated partitioning methods on a wide range of large unsymmetric square and rectangular sparse matrices. Properties of these matrices are listed in Table 5.1. The first four matrices, which are obtained from University of Florida

TABLE 5.1
Properties of unsymmetric square and rectangular test matrices.

$M \times N$ matrix A				$K \times N_C$ communication matrix C					
				$K = 24$		$K = 64$		$K = 128$	
Name	M	N	NNZ	N_C	NNZ	N_C	NNZ	N_C	NNZ
lhr14	14270	14270	321988	11174	25188	12966	31799	13508	36039
lhr17	17576	17576	399500	13144	29416	16070	38571	16764	46182
onetone1	36057	36057	368055	8137	20431	11458	30976	13911	39936
onetone2	36057	36057	254595	3720	9155	6463	16259	11407	27264
pig-large	28254	17264	75018	1265	3347	1522	4803	1735	6193
pig-very	174193	105882	463303	4986	12015	6466	16185	7632	20121
CO9	10789	14851	101578	4458	9226	7431	21816	7887	25070
fxm4-6	22400	30732	248989	769	1650	2010	4208	4223	8924
kent	31300	16620	184710	5200	10691	11540	28832	14852	49976
mod2	34774	31728	165129	4760	9870	8634	18876	10972	24095
pltexpA4	26894	70364	143059	1961	4218	3259	7858	5035	13397
world	34506	32734	164470	5116	10405	9569	20570	13610	30881

Sparse Matrix Collection,[1] are from the unsymmetric linear system application. The `pig-large` and `pig-very` matrices [18] are from the least squares problem. The remaining six matrices, which are obtained from Hungarian Academy of Sciences OR Lab,[2] are from miscellaneous and stochastic linear programming problems. In this table, the *NNZ* column lists the number of nonzeros of the matrices.

We have tested $K = 24$-, 64-, and 128-way rowwise partitionings of each test matrix. For each K value, K-way partitioning of a test matrix forms a partitioning instance. Recall that the objective in the first phase of our two-phase approach is minimizing the total message volume while maintaining the computational-load balance. This objective is achieved by exploiting the recently proposed computational-hypergraph model [8]. The hypergraph-partitioning tool PaToH [9] was used with default parameters to obtain K-way rowwise partitions. The computational-load imbalance values of all partitions were measured to be below 6%.

For the second phase, communication matrix C was constructed for every partitioning instance as described in sections 3.1.1 and 3.2.1. Table 5.1 displays properties of these communication matrices. Then, the communication hypergraph was constructed from each communication matrix as described in sections 3.1.1 and 3.2.1. Note that communication-matrix properties listed in Table 5.1 also show communication-hypergraph properties. That is, for each K value, the table effectively shows a communication hypergraph on K nets, N_C vertices, and *NNZ* pins.

The communication hypergraphs are partitioned using the proposed methods discussed in section 4. In order to verify the validity of the communication hypergraph model, we compare the performance of these methods with a method called *Naive*. This method mimics the current state of the art by minimizing the communication overhead due to the message volume without spending any explicit effort toward minimizing the total message count. The *Naive* method tries to obtain a balance on message-volume loads of processors while attaining the total message-volume requirement determined by the partitioning in the first phase. The method adopts a constructive approach, which is similar to the best-fit-decreasing heuristic used in solving the NP-hard K-feasible bin packing problem [23]. Vertices of the communication hypergraph are assigned to parts in the decreasing order of vertex weights. Each

[1] http://www.cise.ufl.edu/~davis/sparse/
[2] ftp://ftp.sztaki.hu/pub/oplab

TABLE 5.2
Performance of the methods with varying imbalance ratios in 64*-way partitionings.*

Matrix	Partition method	Total msg				Max vol			
		ϵ=0.1	ϵ=0.3	ϵ=0.5	ϵ=1.0	ϵ=0.1	ϵ=0.3	ϵ=0.5	ϵ=1.0
lhr17	Naive	1412	—	—	—	373	—	—	—
	PaToHfix	817	726	724	700	643	755	858	1042
	MSN	745	662	625	592	678	793	895	1177
	MSNmax	731	684	649	638	676	799	920	1119
pig-very	Naive	2241	—	—	—	161	—	—	—
	PaToHfix	1333	1176	1151	1097	272	316	361	448
	MSN	1407	1199	1137	1019	284	343	398	526
	MSNmax	1293	1142	1040	967	298	354	411	530
fxm4-6	Naive	—	—	—	312	—	—	—	67
	PaToHfix	212	193	193	188	70	75	81	105
	MSN	244	205	199	172	72	83	96	114
	MSNmax	247	213	208	165	70	85	94	103

vertex v_j is allowed to be assigned only to the parts in $Nets(v_j)$ to avoid increases in the message volume. Here, the best-fit criterion corresponds to assigning v_j to a part in $Nets(v_j)$ with the minimum weight thus trying to obtain a balance on the message-volume loads.

The partitioning methods, *PaToH-fix, MSN,* and *MSNmax*, incorporate randomized algorithms. Therefore, they were run 20 times starting from different random seeds for K-way partitioning of every communication hypergraph. Randomization in the *Naive* method were realized by random permutation of the vertices before sorting. Averages of the resulting communication patterns of these runs are displayed in the following tables. In these tables, the *Total msg* and *Total vol* columns list, respectively, the total number and total volume of messages sent. The *Max msg* and *Max vol* columns list, respectively, the maximum number and maximum volume of messages sent by a single processor.

The following parameters and options are used in the proposed partitioning methods. *PaToH-fix* were run with the coarsening option of absorption clustering using pins (`ABS_HPC`), and the refinement option of Fiduccia–Mattheyses (`FM`). The scaled heavy-connectivity matching (`SHCM`) of PaToH was used in the coarsening step of the multilevel partitioning methods *MSN* and *MSNmax*. `ABS_HPC` is the default coarsening option in *PaToH-fix*. It is a quite powerful coarsening method that absorbs nets into supervertices, which helps FM-based recursive-bisection heuristics. However, we do not want nets being absorbed in *MSN* and *MSNmax* to be able to establish net-to-part association in the initial partitioning phase. Therefore, `SHCM`, which does not aim to absorb nets, was selected.

Table 5.2 shows the performance of the proposed methods with varying ϵ in 64-way partitioning of three matrices, each of which is the largest (in terms of the number of nonzeros) in its application domain. The performance variation is displayed in terms of the total message-count and maximum message-volume metrics because these two metrics are exactly encoded in the proposed models. Recall that *Naive* is a constructive method and its performance does not depend on ϵ. Therefore, the performance values for *Naive* are listed under the columns corresponding to the attained imbalance ratios. As seen in Table 5.2, by relaxing ϵ, each method can find partitions with smaller total message counts and larger maximum message-volume values. It is also observed that imbalance values of the partitions obtained by all of the proposed methods are usually very close to the given ϵ. These outcomes are in accordance

with the discussion in section 3.4. As seen in the table, all of the proposed methods perform significantly better than the *Naive* method even with the tightest constraint of $\epsilon = 0.1$. However, the detailed performance results are displayed for $\epsilon = 1.0$ (i.e., $W_{max} \leq 2W_{avg}$ in (2.3)) in the following tables. We chose such a relaxed partitioning constraint in order to discriminate among the proposed methods. It should be noted here that imbalance ratios for the message-volume loads of processors might be greater than the chosen ϵ value because of the approximation in the proposed vertex weighting scheme. For example, with $\epsilon = 1.0$, the methods *PaToH-fix, MSN,* and *MSNmax* produce partitions with actual imbalance ratios of 0.94, 1.26, and 1.35 for matrix `lhr17`, respectively.

Table 5.3 displays the communication patterns for $K = 64$- and 128-way partitions in row-parallel $y = Ax$. The bottom of the table shows the average performance of the proposed methods compared with the *Naive* method. These values are obtained by first normalizing the performance results of the proposed methods with respect to those of the *Naive* method for every partitioning instance and then averaging these normalized values over the individual methods.

In terms of the total message-volume metric, *Naive* achieves the lowest values as seen in Table 5.3. This is expected since *Naive* attains the total message volume determined by the partitioning in the first phase. The increase in the total message-volume values for the proposed methods remain below 66% for all partitioning instances. As seen in the bottom of the table, these increases are below 41% on the average. Note that the total message-volume values for *Naive* are equal to the differences of the *NNZ* and N_C values of the respective communication matrix (see Table 5.1). Also note that the *NNZ* values of the communication matrices listed in Table 5.1 show the upper bounds on the total message-volume values for the proposed partitioning methods.

In terms of the maximum message-volume metric, the proposed partitioning methods yield worse results than the *Naive* method by a factor between 2.0 and 2.4 on the average as seen in the bottom of Table 5.3. This performance difference stems from three factors. First, *Naive* is likely to achieve small maximum message-volume values since it achieves the lowest total message-volume values. Second, the best-fit-decreasing heuristic adopted in *Naive* is an explicit effort toward achieving a balance on the message volume. Third, the relaxed partitioning constraint ($\epsilon = 1.0$) used in the proposed partitioning methods leads to higher imbalance ratios among the message-volume loads of processors.

In terms of the total message-count metric, all of the proposed methods yield significantly better results than the *Naive* method in all partitioning instances. They reduce the total message count by a factor between 1.3 and 3.0 in 64-way, and between 1.2 and 2.9 in 128-way partitionings. As seen in the bottom of Table 5.3, the reduction factor is approximately 2 on the average. Comparing the performance of the proposed methods, both *MSN* and *MSNmax* perform better than *PaToH-fix* in all partitioning instances, except 64-way partitioning of `plexpA_4` and 128-way partitioning of `onetone2`, leading to a considerable performance difference on the average. This experimental finding confirms the superiority of the direct K-way partitioning approach over the recursive-bisection approach. There is no clear winner between *MSN* and *MSNmax*. *MSN* performs better than *MSNmax* in 14 out of 24 partitioning instances, leading to a slight performance difference on the average.

In terms of the maximum message-count metric, all of the proposed methods again yield considerably better results than the *Naive* method in all instances, except 64- and 128-way partitionings of `pig` matrices. However, the performance difference

TABLE 5.3
Communication patterns for K-way row-parallel $y = Ax$.

Matrix	Part. method	$K = 64$				$K = 128$			
		Total		Max		Total		Max	
		msg	vol	msg	vol	msg	vol	msg	vol
lhr14	Naive	1318	18833	43.9	308	2900	22531	47.6	204
	PaToH-fix	676	28313	34.0	813	1417	32661	47.8	627
	MSN	561	26842	24.4	975	1247	30796	31.6	577
	MSNmax	640	24475	19.2	897	1348	28758	22.7	535
lhr17	Naive	1412	22501	45.6	373	3675	29418	58.9	265
	PaToH-fix	700	34515	36.5	1042	1867	42623	54.3	750
	MSN	592	32530	26.3	1177	1453	40009	34.0	736
	MSNmax	638	31149	22.0	1119	1599	38557	26.8	689
onetone1	Naive	1651	19518	39.9	332	4112	26025	47.4	231
	PaToH-fix	663	26789	27.2	714	1639	35741	39.1	580
	MSN	545	27109	24.1	1008	1384	35129	31.1	688
	MSNmax	610	24012	20.9	950	1507	31345	26.4	642
onetone2	Naive	995	9796	30.4	186	2049	15857	28.6	139
	PaToH-fix	429	12940	17.8	381	804	20983	25.1	423
	MSN	406	13236	17.1	510	787	20649	22.1	422
	MSNmax	420	12389	15.1	485	807	18850	20.4	381
pig-large	Naive	1220	3281	39.4	60	2723	4458	39.6	47
	PaToH-fix	759	4363	40.5	144	1764	5805	52.5	142
	MSN	619	4108	34.5	153	1551	5752	43.0	115
	MSNmax	682	3812	35.6	138	1678	5185	35.0	100
pig-very	Naive	2241	9719	56.5	161	4574	12489	78.7	117
	PaToH-fix	1097	14725	59.8	448	2533	18567	97.8	398
	MSN	1019	14349	54.5	526	2389	17317	77.3	320
	MSNmax	967	14008	55.4	530	2501	15729	80.5	317
CO9	Naive	1283	14385	41.0	369	1645	17183	48.9	289
	PaToH-fix	622	19221	34.6	567	1191	23575	35.8	434
	MSN	521	18352	27.1	687	904	20727	28.9	412
	MSNmax	513	17736	23.1	684	800	21281	25.6	492
fxm4-6	Naive	312	2198	13.6	67	562	4701	15.9	64
	PaToH-fix	188	2856	11.8	105	361	5746	13.8	129
	MSN	172	2746	10.1	114	338	5647	12.2	129
	MSNmax	165	2543	8.9	103	322	5386	11.7	124
kent	Naive	342	17292	14.1	547	1020	35124	21.9	602
	PaToH-fix	235	21200	9.2	621	740	42328	15.8	631
	MSN	190	21539	8.9	905	596	39774	19.6	866
	MSNmax	201	19666	7.0	773	614	40012	13.0	830
mod2	Naive	376	10242	22.4	366	811	13123	33.8	240
	PaToH-fix	294	16683	19.8	606	658	21409	22.6	431
	MSN	254	13353	15.2	604	575	17329	18.7	391
	MSNmax	231	14400	12.5	639	548	19009	14.4	408
pltexpA4	Naive	507	4599	21.9	116	1013	8362	25.6	99
	PaToH-fix	257	5553	17.7	243	579	10163	22.8	208
	MSN	245	5828	15.3	241	556	9705	21.7	213
	MSNmax	264	5321	13.2	214	546	9582	19.4	206
world	Naive	534	11001	27.4	387	1785	17271	44.1	222
	PaToH-fix	362	18355	21.2	603	1036	26514	35.6	488
	MSN	315	14765	16.8	595	902	23927	24.8	476
	MSNmax	287	16243	14.8	680	886	23762	20.6	476
		Normalized averages over Naive							
	Naive	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	PaToH-fix	0.56	1.41	0.81	2.03	0.59	1.38	0.91	2.38
	MSN	0.48	1.34	0.68	2.37	0.51	1.29	0.75	2.30
	MSNmax	0.49	1.28	0.60	2.26	0.52	1.24	0.63	2.21

between the proposed methods and the *Naive* method is not as large as that in the total message-count metric. Comparing the performance of the proposed methods, both *MSN* and *MSNmax* perform better than *PaToH-fix* in all partitioning instances, except 128-way partitioning of `kent`, leading to a considerable performance difference on the average. *MSNmax* is the clear winner in the maximum message-count metric as expected. As seen in the bottom of the table, *MSNmax* yields, respectively, 40% and 37% fewer maximum message counts than *Naive*, for 64 and 128-way partitionings, on the average.

We have also experimented with the performance of the proposed methods for 64-way and 128-way partitionings for column-parallel $w = A^T z$ and row-column-parallel $y = AA^T z$ on the test matrices. Since very similar relative performance results were obtained in these experiments, we omit presentation and discussion of these experimental results due to the lack of space.

It is important to see whether the theoretical improvements obtained by our methods in the given performance metrics hold in practice. For this purpose, we have implemented row-parallel $y = Ax$ and row-column-parallel $y = AA^T z$ multiplies using the LAM/MPI 6.5.6 [5] message passing library. The parallel multiply programs were run on a Beowulf class [38] PC cluster with 24 nodes. Each node has a 400Mhz Pentium-II processor and 128MB memory. The interconnection network is comprised of a 3COM SuperStack II 3900 managed switch connected to Intel Ethernet Pro 100 Fast Ethernet network interface cards at each node. The system runs the Linux kernel 2.4.14 and the Debian GNU/Linux 3.0 distribution.

Within the current experimental framework, *MSNmax* seems to be the best choice for communication-hypergraph partitioning. For this reason, in Table 5.4, only the parallel running times of the multiply programs for *MSNmax* partitionings are given in comparison with those for *Naive* partitionings. Communication patterns for the resulting partitions are also listed in the table in order to show how improvements in performance metrics relate to improvements in parallel running times.

As seen in Table 5.4, the partitions obtained by *MSNmax* lead to considerable improvements in parallel running times compared with those of *Naive* for all matrices. The improvements in parallel running times are in between 4% and 40% in $y = Ax$, and between 5% and 31% in $y = AA^T z$. In row-parallel $y = Ax$, the lowest percent improvement of 4% occurs for matrix `kent` despite the modest improvement of 28% achieved by *MSNmax* over *Naive* in total message count. The reason seems to be the equal maximum message counts obtained by these partitioning methods. The highest percent improvement of 40% occurs for matrix `fxm4-6` for which *MSNmax* achieves significant improvements of 49% and 36% in the total and maximum message counts, respectively. However, the higher percent improvements obtained by *MSNmax* for matrix `lhr14` in message-count metrics do not lead to higher percent improvements in parallel running time. This might be attributed to *MSNmax* achieving lower percent improvements for `lhr14` in message-volume metrics compared with those for `fxm4-6`. These experimental findings confirm the difficulty of the target problem.

Table 5.5 displays partitioning times for the three largest matrices selected from different application domains. The *Phase* 1 *Time* and *Phase* 2 *Time* columns list, respectively, the computational-hypergraph and communication-hypergraph partitioning times. Sequential matrix-vector multiply times are also displayed to show the relative preprocessing overhead introduced by the partitioning methods. All communication-hypergraph partitionings take significantly less time than computational-hypergraph partitionings except partitioning communication hypergraph of

TABLE 5.4
Communication patterns and parallel running times in msecs for 24-way row-parallel $y = Ax$ and row-column-parallel $y = AA^Tz$.

Matrix	Part.	$y = Ax$					$y = AA^Tz$				
		Total		Max		Parl	Total		Max		Parl
	method	msg	vol	msg	vol	time	msg	vol	msg	vol	time
lhr14	Naive	414	14014	23	603	2.57	838	28028	46	1177	5.07
	MSNmax	176	19580	12	1601	1.90	342	42456	27	1960	3.95
lhr17	Naive	393	16272	22	691	2.79	792	32544	45	1159	5.71
	MSNmax	168	24510	17	2229	2.20	334	48554	23	2112	4.38
onetone1	Naive	362	12294	19	546	2.52	728	24588	41	788	5.49
	MSNmax	152	15153	16	1403	1.85	262	34304	24	1586	4.37
onetone2	Naive	205	5435	12	297	1.60	412	10870	24	419	3.24
	MSNmax	102	6294	9	690	1.31	186	15234	16	715	2.44
pig-large	Naive	325	2082	23	108	2.06	650	4164	42	162	3.41
	MSNmax	151	2872	20	276	1.28	312	5554	26	271	2.35
pig-very	Naive	497	7029	23	354	3.51	994	14058	46	456	7.33
	MSNmax	228	10214	23	937	2.74	428	20538	29	963	5.95
CO9	Naive	122	4768	11	437	1.74	244	9536	22	1184	3.34
	MSNmax	68	6834	9	750	1.35	152	13700	16	1430	2.99
fxm4-6	Naive	113	881	11	44	1.57	226	1762	27	108	3.18
	MSNmax	58	1005	7	96	0.95	120	2038	15	124	2.31
kent	Naive	57	5491	5	488	1.12	114	10982	9	972	2.27
	MSNmax	41	5783	5	541	1.08	86	12596	7	1025	2.12
mod2	Naive	79	5110	11	617	1.74	158	10220	22	1586	3.67
	MSNmax	59	7764	7	779	1.53	130	15890	14	2148	3.50
pltexpA4	Naive	106	2257	9	146	1.25	212	4514	20	225	2.46
	MSNmax	60	2543	8	256	0.93	120	5410	14	314	2.08
world	Naive	79	5289	9	667	1.89	158	10578	19	2204	3.73
	MSNmax	65	8316	7	836	1.66	134	13638	16	2442	3.38
Normalized averages over Naive											
	Naive	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	MSNmax	0.55	1.33	0.79	2.10	0.78	0.56	1.37	0.65	1.52	0.82

TABLE 5.5
24-way partitioning and sequential matrix-vector multiply times in msecs.

Matrix	Partitioning times				Seq.
	Phase 1		Phase 2		$y=Ax$
	Method	Time	Method	Time	time
lhr17	PaToH	6100	Naive	32	19.56
			PaToH-fix	13084	
			MSN	3988	
			MSNmax	3885	
pig-very	PaToH	20960	Naive	12	30.37
			PaToH-fix	2281	
			MSN	1086	
			MSNmax	1022	
fxm4-6	PaToH	2950	Naive	2	13.19
			PaToH-fix	58	
			MSN	112	
			MSNmax	81	

`lhr17` with *PaToH-fix*. As expected, the communication hypergraphs are smaller than the respective computational hypergraphs. However, some communication hypergraphs might have very large net sizes because of the small number of nets. Matrix `lhr17` is an example of such a case with the large average net size of $nnz(C)/K = 1225$ in the communication hypergraph versus the small average net size of $nnz(A)/N = 22$

in the computational hypergraph. This explains the above exceptional experimental outcome because running times of matching heuristics, used in the coarsening step of PaToH, increase with the sum of squares of net sizes [8] (see also Theorem 5.5 in [21]).

Comparing the running times of communication-hypergraph partitioning methods, *Naive* takes an insignificant amount of time as seen in Table 5.5. Direct K-way partitioning approaches are expected to be faster than the recursive-bisection based *PaToH-fix* because of the single coarsening step as compared with $K-1=23$ coarsening steps. As expected, *MSN* and *MSNmax* take considerably less time than *PaToH-fix* except in partitioning communication-hypergraph of `fxm4-6`, which has a moderate average net size. As seen in the table, the second-phase methods *MSN* and *MSNmax* introduce much less preprocessing overhead than the first phase. The partitionings obtained by *MSNmax* for `lhr17`, `pig-very`, and `fxm4-6` matrices lead to speedup values of 8.89, 11.1, and 13.9, respectively, in row-parallel matrix-vector multiplies on our 24-processor PC cluster.

6. Conclusion. We proposed a two-phase approach that encapsulates multiple communication-cost metrics in 1D partitioning of structurally unsymmetric square and rectangular sparse matrices. The objective of the first phase was to minimize the total message volume and maintain computational-load balance within the framework of the existing 1D matrix partitioning methods. For the second phase, communication-hypergraph models were proposed. Then, the problem of minimizing the total message latency while maintaining the balance on message-volume loads of processors was formulated as a hypergraph partitioning problem on communication hypergraphs. Several methods were proposed for partitioning communication hypergraphs. One of these methods was tailored to encapsulate the minimization of the maximum message count per processor. We tested the performance of the proposed models and the associated partitioning methods on a wide range of large unsymmetric square and rectangular sparse matrices. In these experiments, the proposed two-phase approach achieved substantial improvements in terms of communication-cost performance metrics. We also implemented parallel matrix-vector and matrix-matrix-transpose-vector multiplies using MPI to see whether the theoretical improvements achieved in the given performance metrics hold in practice. Experiments on a PC cluster showed that the proposed approach can achieve substantial improvements in parallel run times.

Parallel matrix-vector multiply $y=Ax$ is one of the basic parallel reduction algorithms. Here, the x-vector entries are the input, and the y-vector entries are the output of the reduction. The matrix A corresponds to the mapping from the input to the output vector entries. Çatalyürek and Aykanat [10] briefly list several practical problems that involve this correspondence. Hence, the proposed two-phase approach can also be used in reducing the communication overhead in such practical reduction problems.

Acknowledgment. The authors thank the anonymous referees, whose suggestions greatly improved the presentation of the paper.

REFERENCES

[1] C. J. Alpert, A. E. Caldwell, A. B. Kahng, and I. L. Markov, *Hypergraph partitioning with fixed vertices*, IEEE Trans. Computer-Aided Design, 19 (2000), pp. 267–272.

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

[3] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.
[4] M. Benzi and M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.
[5] 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, Ontario, Canada, 1994, pp. 379–386.
[6] U. V. Çatalyürek, *Hypergraph Models for Sparse Matrix Partitioning and Reordering*, Ph.D. thesis, Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, 1999.
[7] U. 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-Verlag, Berlin, 1996, pp. 75–86.
[8] U. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel and Distributed Systems, 10 (1999), pp. 673–693.
[9] U. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version* 3.0, Tech. rep. BU-CE-9915, Computer Engineering Department, Bilkent University, Ankara, Turkey, 1999.
[10] U. V. Çatalyürek and C. Aykanat, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proceedings of the IEEE Symposium on Scientific Computing, Denver, CO, 2001, pp. 10–16.
[11] A. Daşdan and C. Aykanat, *Two novel multiway circuit partitioning algorithms using relaxed locking*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 16 (1997), pp. 169–178.
[12] J. J. Dongarra and T. H. Dunigan, *Message-passing performance of various computers*, Concurrency—Practice and Experience, 9 (1997), pp. 915–926.
[13] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in Proceedings of the 19th ACM/IEEE Design Automation Conference, ACM, New York, IEEE Computer Society, Los Alamitos, CA, 1982, pp. 175–181.
[14] R. W. Freund and N. M. Nachtigal, *QMR: A quasi-minimal residual algorithm for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.
[15] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, 1996.
[16] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[17] A. Gupta, *Watson Graph Partitioning Package*, Tech. rep. RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.
[18] M. Hegland, *Description and Use of Animal Breeding Data for Large Least Squares Problems*, Tech. rep. TR-PA-93-50, CERFACS, Toulouse, France, 1993.
[19] B. Hendrickson, *Graph partitioning and parallel solvers: Has the emperor no clothes?*, in Solving Irregularly Structured Problems in Parallel, Lecture Notes in Comput. Sci. 1457, Springer-Verlag, Berlin, 1998, pp. 218–225.
[20] B. Hendrickson and T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Comput., 26 (2000), pp. 1519–1534.
[21] 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.
[22] B. Hendrickson and R. Leland, *The Chaco Users's Guide, Version* 2.0, Tech. rep. SAND95-2344, Sandia National Laboratories, Albuquerque, NM, 1995.
[23] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.
[24] G. Karypis and V. Kumar, *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version* 3.0, Tech. rep., Department of Computer Science and Engineering, Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.
[25] G. Karypis and V. Kumar, *Multilevel Algorithms for Multi-constraint Graph Partitioning*, Tech. rep. 98-019, Department of Computer Science, Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.
[26] G. Karypis, V. Kumar, R. Aggarwal, and S. Shekhar, *hMeTiS: A Hypergraph Partitioning Package Version* 1.0.1, Department of Computer Science and Engineering, Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.

[27] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 49 (1970), pp. 291–307.
[28] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley–Teubner, Chichester, UK, 1990.
[29] H. Özaktaş, *Algorithms for Linear and Convex Feasibility Problems: A Brief Study of Iterative Projection, Localization and Subgradient Methods*, Ph.D. thesis, Industrial Engineering Department, Bilkent University, Ankara, Turkey, 1998.
[30] H. Özaktas, M. Akgül, and M. Ç. Pinar, *A parallel surrogate constraint approach to the linear feasibility problem*, in Applied Parallel Computing: Computations in Physics, Chemistry, and Engineering Science, Lecture Notes in Comput. Sci. 1184, Springer-Verlag, Berlin, 1996, pp. 565–574.
[31] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
[32] A. Pinar, U. V. Çatalyürek, C. Aykanat, and M. Ç. Pinar, *Decomposing linear programs for parallel solution*, in Proceedings of the 2nd International Workshop on Parallel Computing, Lecture Notes in Comput. Sci. 1041, Springer-Verlag, Berlin, 1996, pp. 473–482.
[33] A. Pinar and B. Hendrickson, *Graph partitioning for complex objectives*, in Proceedings of Irregular 2001 Eighth International Workshop on Solving Irregularly Structured Problems in Parallel, San Francisco, CA, 2001.
[34] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
[35] L. A. Sanchis, *Multiple-way network partitioning*, IEEE Trans. Comput., 38 (1989), pp. 62–81.
[36] L. A. Sanchis, *Multiple-way network partitioning with different cost functions*, IEEE Trans. Comput., 42 (1993), pp. 1500–1504.
[37] K. Schloegel, G. Karypis, and V. Kumar, *A New Algorithm for Multi-Objective Graph Partitioning*, Tech. rep. 99-003, Department of Computer Science, Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1999.
[38] 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.
[39] E. Turna, *Parallel Algorithms for the Solution of Large Sparse Inequality Systems on Distributed Memory Architectures*, Master's thesis, Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, 1998.
[40] K. Yang and K. G. Murty, *New iterative methods for linear inequalities*, J. Optim. Theory Appl., 72 (1992), pp. 163–185.