# A NESTED DISSECTION APPROACH TO SPARSE MATRIX PARTITIONING FOR PARALLEL COMPUTATIONS

ERIK G. BOMAN* AND MICHAEL M. WOLF†

**Abstract.** We consider how to distribute sparse matrices among processes to reduce communication costs in parallel sparse matrix computations, specifically, sparse matrix-vector multiplication. Our main contributions are: (i) an exact graph model for communication with general (two-dimensional) matrix distribution, and (ii) a recursive partitioning algorithm based on nested dissection (substructuring). We show that the communication volume is closely linked to vertex separators.

We have implemented our algorithm using hypergraph partitioning software to enable a fair comparison with existing methods. We present numerical results for sparse matrices from several application areas, with up to 9 million nonzeros. The results show that our new approach is superior to traditional 1d partitioning and comparable to a current leading partitioning method, the fine-grain hypergraph method, in terms of communication volume. Our nested dissection method has two advantages over the fine-grain method: it is faster to compute, and the resulting distribution requires fewer communication messages.

**Keywords**: Parallel data distributions, sparse matrix computations, graph partitioning, nested dissection, matrix-vector multiplication.

**AMS subject classifications.** 05C50, 05C85, 65F50, 65Y05, 68R10

**1. Introduction.** Sparse matrix-vector multiplication (SpMV) is a common kernel in many computations, e.g., iterative solvers for linear systems of equations and PageRank computation (power method) for ranking web pages. Often the same matrix is used for many iterations. An important combinatorial problem in parallel computing is how to distribute the matrix and the vectors among processes to minimize the communication cost. Such "communication" is also important on serial computers with deep memory hierarchies, where slow memory is typically much slower than fast memory. Since CPU performance increases more rapidly than memory, we expect memory latency and bandwith to grow in importance. Our present work is relevant to both parallel computation on distributed memory computers and serial computation on machines with hierarchical memory, but we phrase our work in the context of parallel computing. Our work also applies to Jacobi and Gauss-Seidel iterations and the more general computation $y = F(x)$, where $x$ is an input vector, $y$ is an output vector, and $F$ is a sparse (possibly nonlinear) operator that is "decomposable" such that partial evaluations can be computed independently and then combined.

Sparse matrix-vector multiplication $y = Ax$ is usually parallelized such that the process that owns element $a_{ij}$ computes the contribution $a_{ij}x_j$. This is a local operation if $x_j$, $y_i$ and $a_{ij}$ all reside on the same process; otherwise communication is required. In general, the following four steps are performed [7, 21]:

1. **Expand:** Send entries $x_j$ to processes with a nonzero $a_{ij}$ for some $i$.
2. **Local multiply-add:** $y_i := y_i + a_{ij}x_j$.
3. **Fold:** Send partial $y$ values to relevant processes.

*Scalable Algorithms Dept, Sandia National Labs, Albuquerque, NM 87185-1318, egboman@sandia.gov. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the DOE's National Nuclear Security Administration under contract number DE-AC-94AL85000.

†Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, mmwolf@illinois.edu, and Scalable Algorithms Dept, Sandia National Labs.

4. **Sum:** Sum up the partial $y$ values.

In this paper, we address sparse matrix-vector partitioning, where both the matrix nonzeros and vector elements are partitioned into different parts. For parallel computing, this data is mapped to different processes based on this part assignment.

DEFINITION 1.1. Sparse matrix-vector partitioning: *Given a sparse matrix $A$, an integer $k > 1$, and $\epsilon > 0$, compute*

*(i) a matrix partition $A = \sum_{i=1}^{k} A_i$ where each $A_i$ contains a subset of the nonzeros of $A$, such that $nnz(A_i) \leq (1+\epsilon)nnz(A)/k$, $\forall i$, where $nnz$ denotes the number of nonzeros,*

*(ii) partitions of the input and output vectors,*

*such that when the data is distributed across processes based on these partitions, the communication volume in sparse matrix-vector multiply, $y = Ax$, is minimized.*

This problem is NP-hard since it contains as a special case hypergraph partitioning. Stated above is a very general form. We first consider a special version for symmetric matrices where the input and output vectors must have the same distribution. It has been observed [7, 4] that the matrix and vector partitioning problems can be separated. For any given matrix distribution (partition), it is easy to find a "compatible" vector partition and these together give a solution to the combined matrix-vector problem. Additional objectives can be minimized in the vector partitioning phase [4, 5]. We focus on the matrix partitioning step but simultaneously obtain a compatible vector partitioning as well.

By far, the most common way to partition a sparse matrix is to use a 1d scheme where each part is assigned the nonzeros for a set of rows or columns. This approach has two advantages: simplicity for the user and only one communication phase (not two). The simplest 1d method is to assign $\approx n/k$ consecutive rows (or columns) to each part, where $n$ denotes the number of rows and $k$ the number of parts in a partition (Figure 1.1). However, it is often possible to reduce the communication by partitioning the rows in a better (non-contiguous) way, using graphs, bipartite graphs, or hypergraphs to model this problem (Subsections 2.1 - 2.3) [16, 7].

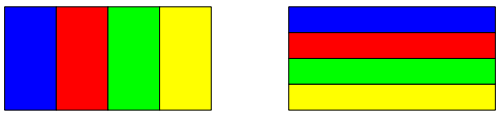


FIG. 1.1. *1d row and column partitioning of a matrix. Each color denotes a part that is assigned a different process.*

Recently, several 2d decompositions have been proposed [8, 21]. The idea is to reduce the communication volume further by giving up the simplicity of the 1d structure. The fine-grain distribution [8] is of particular interest since it is the most general and is theoretically optimal (though not in practice). We introduce a graph model that also accurately describes communication in fine-grain distribution. This leads to a new graph-based algorithm, a "nested dissection partitioning algorithm," which we describe in Section 4. (A preliminary version, without analysis, appeared in [6].) This nested dissection partitioning algorithm is related to previous nested dissection work for parallel Cholesky factorization [14, 15]. An important aspect to both our partitioning method and the previous parallel Cholesky factorization work is that communication is limited to separator vertices in the corresponding graph.

The rest of this paper is organized as follows. In Section 2 we discuss 1d and

2d data distribution, while in Section 3 we present a new graph model for symmetric partitioning. In Section 4 we present an algorithm based on this model, and in Section 5 we discuss the nonsymmetric case. We present numerical results in Section 6 that validate our approach.

## 2. Background: 1d and 2d Distributions.

**2.1. 1d Graph Model.** The standard graph model is limited to structurally symmetric matrices. In this case, the graph $G$ is defined such that the adjacency matrix of $G$ has the same nonzero pattern as the matrix $A$. Each row (or column) in $A$ corresponds to a vertex in $G$. A partitioning of the vertices in $G$ gives a partitioning of the rows (columns) in $A$. The standard objective is to minimize the number of cut edges, but this does not accurately reflect communication volume. Figure 2.1 illustrates this. Twice the number of cut edges (highlighted in magenta) yields a communication volume of 6 words, which overcounts the correct volume of 4 words. The problem is that vertices 1 and 8 are counted twice in the metric but each only contributes one word to the volume. The communication required is associated with the boundary vertices, so a better metric is to minimize the boundary vertices (4 words for Figure 2.1, which is correct). This is an exact metric for bisection, while for $k > 2$ one should also take into account the number of adjacent parts.

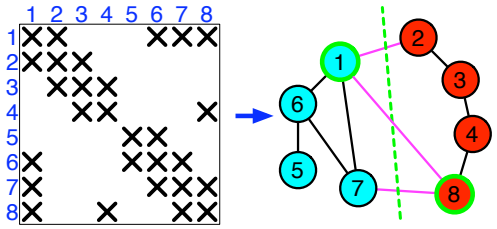



FIG. 2.1. *1d graph partitioning of matrix into two parts. Correct communication volume is 4 words. Communication of highlighted vertices is overcounted in edge metric.*

**2.2. 1d Bipartite Graph Model.** The graph model works poorly on nonsymmetric square matrices because they need be symmetrized, and does not apply to rectangular matrices. The bipartite graph model was designed to rectify this [16]. The bipartite graph $G = (R, C, E)$ is defined such that vertices $R$ correspond to rows, vertices $C$ correspond to columns, and edges $E$ correspond to nonzeros. The standard (simplest) objective is to partition both $R$ and $C$ such that the number of cut edges is minimized. Only one of the vertex partitionings (either $R$ for rows, or $C$ for columns) is used to obtain a 1d matrix partitioning. Again, the cut edges do not correctly count communication volume, and boundary vertices should be used instead.

**2.3. 1d Hypergraph Model.** Aykanat and Catalyurek introduced the hypergraph model to count communication volume accurately [7]. A hypergraph generalizes a graph. Whereas a graph edge contains exactly two vertices, a hyperedge can contain an arbitrary set of vertices [2, 3]. There are two 1d hypergraph models. In the row-net model, each column is a vertex and each row a hyperedge, while in the column-net model, each row is a vertex and each column a hyperedge. The objective is to find a balanced vertex partitioning and minimize the number of cut hyperedges. The communication volume is $\sum_i (\lambda_i - 1)$, where $\lambda_i$ is the number of parts that touch

hyperedge $i$. Finding the optimal balanced vertex partitioning is NP-hard but in practice good partitions can be found in (near-linear) polynomial time [7].

**2.4. 2d Matrix Distributions.** Although the simplicity of 1d distributions may be desirable, the communication volume can often be reduced by using 2d distributions. Figure 2.2 shows an example where 1d partitioning will always be poor. Consider the arrowhead matrix of dimension $n$, and bisection ($k = 2$). Due to a single dense row and column, any load balanced 1d partitioning will have a communication volume of approximately $(3/4)n$ words. The optimal volume is actually 2 words as demonstrated in the 2d partitioning of Figure 2.2 (right).

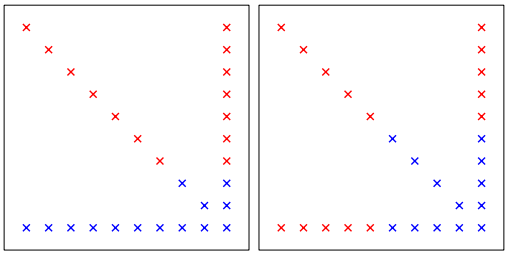


FIG. 2.2. *Arrowhead matrix with 1d (left) and 2d (right) distribution, for two parts (red and blue). The communication volumes in this example are eight and two words, respectively.*

**2.5. Current 2d Methods.** Two-dimensional partitioning is a more flexible alternative to one-dimensional partitioning. For dense matrices, it was realized that a 2d block (checkerboard) distribution reduces communication since communication is limited to process rows and columns. For sparse matrices, the situation is more complex. Several algorithms have been proposed to take advantage of the flexibility afforded by a two-dimensional partitioning but none have become dominant. Catalyurek and Aykanat proposed both a fine-grain [8] and a coarse-grain [9] decomposition, while Bisseling and Vastenhouw later developed the Mondriaan method [21]. The coarse-grain method is similar to the 2d block (checkerboard) decomposition in the dense case, but is difficult to compute (requires multiconstraint hypergraph partitioning) and often gives relatively high communication volume. The fine-grain method gives low communication volume but is also expensive to compute. The Mondriaan method tries to find a middle ground. It is based on recursive 1d hypergraph partitioning and thus is relatively fast but still produces partitions with low communication cost.

The most flexible approach to matrix partitioning is to allow any nonzero to be assigned any part. This is the idea underlying the fine-grain method [8]. The authors propose a hypergraph model that exactly represents communication volume. In the fine-grain hypergraph model, each nonzero corresponds to a vertex, and each row and each column corresponds to a hyperedge. A matrix $A$ with $m$ rows, $n$ colums, and $z$ nonzeros yields a hypergraph with $z$ vertices and $m + n$ hyperedges. Catalyurek and Aykanat proved that this fine-grain hypergraph model yields a minimum volume partition when optimally solved [8]. As with the 1d hypergraph model, finding the optimal partition of the fine-grain model is NP-hard. This hypergraph can be partitioned into $k$ approximately equal parts cutting few hyperedges using standard one-dimensional partitioning algorithms and software. This usually takes significantly longer than a one-dimensional partitioning of a typical matrix since the fine-grain hypergraph is larger than a 1d hypergraph model of the original matrix. However, there is special structure in this hypergraph (each vertex is incident to exactly two hyperedges) that

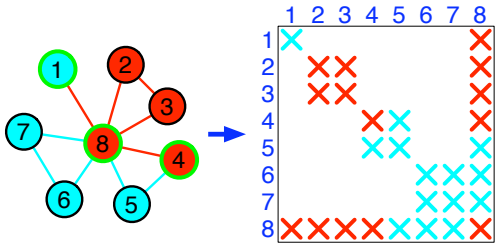



FIG. 3.1. *2d graph bisection for symmetrically partitioned matrix and resulting matrix partitioning. Part (color) of graph edge corresponds to symmetric pair of off-diagonal nonzeros.*

potentially can be exploited to reduce runtime. In general, the partitioning of the fine-grain method with the one-dimensional algorithms produces good quality partitions but can be slow. Thus, our goal in developing new two-dimensional methods is to produce similar quality partitions to fine-grain in a shorter runtime.

**3. An Exact Graph Model for Structurally Symmetric Matrices.** We present an accurate graph model for communication volume in matrix-vector multiplication. In this section we study structurally symmetric matrices, while the nonsymmetric case is analyzed in Section 5. We restrict our attention here to symmetric partitioning schemes, where $a_{ij}$ and $a_{ji}$ are assigned the same part. One advantage of this is that we can save storage (e.g., only store the lower triangular half for numerically symmetric matrices). A second advantage is that we can work with the undirected graph $G(V, E)$, where the vertices correspond to the vectors and the edges correspond to the nonzeros. We partition both the vertices and edges, that is, assign vector elements and matrix nonzeros to parts. We allow arbitrary assignment of both vertices and edges, which distinguishes our approach from the 1d graph model and allows for 2d partitioning. A vertex incurs communication iff there are incident edges that belong to a different part. The volume depends on how many parts are represented among the incident edges.

THEOREM 3.1. *Let $G(V, E)$ be the graph of a symmetric sparse matrix. Let $E(v)$ denote the set of edges incident to vertex $v$. Let $\pi(v)$ and $\pi(e)$ denote the parts to which $v$ and $e$ belong, respectively. Then the communication volume in matrix-vector multiplication is $2\sum_{v\in V}(|\pi(v) \cup \pi(E(v))| - 1)$.*

The factor two arises because any communication occurs in both phases (expand and fold). This exact graph model yields a minimum volume balanced partition for sparse symmetric matrix-vector multiplication when optimally solved.

Figure 3.1 shows an example of the exact graph model for 2d symmetric partitioning of matrices. The graph on the left corresponds to the symmetric matrix (showed partitioned on the right). The edges and vertices in the graph are partitioned into two parts (represented by cyan and red). The vertices that have incident edges belonging to a different part (and thus incur communication) are highlighted in green. Each contributes two words to the communication volume in the resulting matrix-vector multiplication. The matrix on the right shows the 2d symmetric matrix partition obtained from the partitioned graph. The partition of the diagonal entries (as well as the vector entries) corresponds to the partition of the graph vertices. The partition of the off-diagonal entries corresponds to the partition of the edges in the graph.

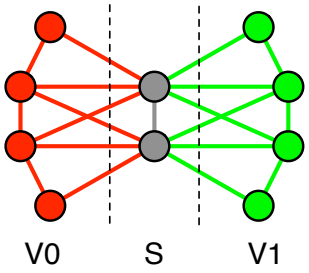



FIG. 4.1. *Bisection. Vertex separator (gray vertices) used to partition vertices into three disjoint subsets* $(V_0, V_1, S)$.

**4. A Vertex Separator Partitioning Algorithm.** In Section 3, we introduced an exact graph model for 2d partitioning of symmetric matrices. If we solved this model optimally, we would obtain a balanced partition to minimize communication volume for resulting matrix-vector multiplication. However, this problem is NP-hard. In this section, we introduce an algorithm for solving this exact graph model suboptimally in polynomial time (assuming the vertex separator is found in polynomial time). An edge separator in the fine-grain hypergraph model corresponds to a vertex separator in the graph. Thus, we can derive a fine-grain decomposition from a vertex separator for the graph. One constraint that we impose on our algorithm is that the vertex and edge partitions are *compatible*. A vertex partition is *compatible* with an edge partition if every vertex belongs to the same part as one of its incident edges. Similarly, an edge partition is *compatible* with a vertex partition if every edge belongs to the same part as one of its two vertices. There is no reason to violate this constraint since it will only increase the communication volume.

**4.1. Bisection.** For simplicity, we consider bisection first. In the next subsection we generalize to $k$-way partitioning for $k > 2$ using recursive bisection. First we compute a small balanced vertex separator $S$ for the graph using any vertex separator algorithm. This partitions the vertices into three disjoint subsets $(V_0, V_1, S)$. Let $E_j := \{e \in E | e \cap V_j \neq \emptyset\}$ for $j = 0, 1$, that is, $E_j$ is the set of edges with at least one endpoint in $V_j$. $V_j$ and $E_j$ are assigned to part $P_j$ for $j = 0, 1$. An example of a graph partitioned using this algorithm is shown in Figure 4.1.

The procedure above intentionally does not specify how to distribute the vertices in $S$ and the edges therein. The partitioning of these vertices and edges does not affect the communication volume as long as the partitions are compatible. There are several ways to exploit this flexibility, yielding several variations on our basic algorithm.

1. If load balance in the matrix is of primary concern, distribute the vertices in $S$ (and edges therein) in such a way to obtain balance.
2. To improve balance in the vector distribution, assign more vertices in $S$ to the process with the fewest vector elements.
3. One can also try to minimize a secondary objective, such as minimizing the maximum communication volume for any process. This is similar to the *vector partitioning* problem posed in [4, 5].

Since a primary goal is load balance in the matrix nonzeros, we seek to balance the edges in the graph. A balanced vertex separator normally balances the number of vertices; to achieve edge balance we can weight each vertex by its degree.

**4.1.1. Analysis of Separator Assignment.** For a minimal separator $S$, each vertex in $S$ must have at least one non-separator neighbor in $V_0$ and one in $V_1$. We assume this to be the case for the (possibly non-minimal) separator used in our algorithm. This is a reasonable assumption since otherwise a smaller separator could be trivially obtained by moving the problematic vertex to the set of its non-separator neighbor vertex, and most vertex separator implementations would do this. Communication in SpMV is limited to the vertices in $S$. This follows from the above method of assigning all edges that have at least one non-separator vertex to the part of this non-separator vertex, such that each non-separator vertex is only incident to edges of its part. Thus, non-separator vertices do not incur communication.

LEMMA 4.1. *Suppose $S$ is a separator where each vertex in $S$ has at least one non-separator neighbor in $V_0$ and one in $V_1$. Then the communication in SpMV is limited to the vertices in $S$, and the volume is $2|S|$. Furthermore, the assignment of vertices in $S$ and edges therein does not matter as long as compatibility is maintained.*

*Proof.* Each vertex in $S$ is incident to at least one edge assigned to $P_0$ and one edge assigned to $P_1$. Thus, the part assignment of the edges connecting vertices in $S$ does not affect the communication volume. Each separator vertex will incur 1 word of communication for each phase whether it is assigned to $P_0$ or $P_1$. Thus, the communication volume is exactly $|S|$ in each phase or $2|S|$ total. □

THEOREM 4.2. *For bisection, the vertex separator partitioning algorithm with a minimal balanced vertex separator minimizes communication volume in sparse matrix-vector multiplication (for a balanced partition).*

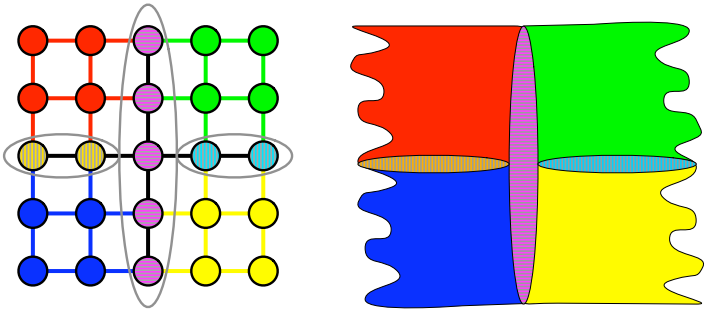
*Proof.* Since each vertex incurring communication (in bisection) incurs 1 word of communication (in each phase), finding the minimum set of vertices that incur communication will minimize the communication in matrix-vector multiplication. This minimum set of vertices is the minimum vertex separator. □

This shows that the vertex separator method is optimal for $k = 2$, just like the fine-grain hypergraph method.

**4.2. Nested Dissection Partitioning Algorithm.** In practice, one wishes to partition into $k > 2$ parts. If we knew a method to compute a balanced $k$-separator, a set $S$ such that the removal of $S$ breaks $G$ into $k$ disjoint subgraphs, we could assign each subgraph to a different part. However, we do not know efficient methods to compute a $k$-separator and do not consider this option any further. A more practical approach is to use recursive bisection. In fact, the procedure to compute a $k-$separator via recursive bisection is known as "nested dissection" and well studied [13, 19] since it is important for sparse matrix factorization.

The procedure is illustrated in Figures 4.2 and 4.3. In this example there are four parts. We show the recursive procedure on a mesh, a generic graph, and the corresponding matrix. The striped and gray areas correspond to separators and separator-separator edges, respectively. We have not specified how to partition this data. It is important to note that it is not necessary to use the nested dissection ordering to permute the matrix, as shown in Figure 4.3. We only do this in illustrations to make the partitioning method more clear. Figure 4.4(a) shows the actual partitioning of the cage5 matrix [11] with the corresponding nested dissection ordered partitioning in Figure 4.4(b) for easier visualization of our method.

Algorithm 1 summarizes our recursive algorithm. Note that if $k$ is a power of



(a) 5-by-5 grid (b) Generic graph

FIG. 4.2. *Graphs partitioned using nested dissection. Striped areas are separators.*

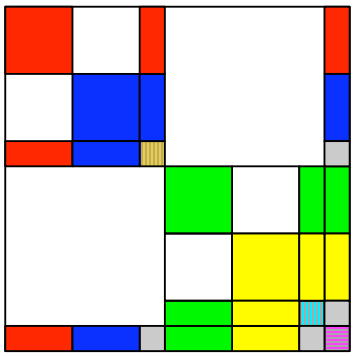
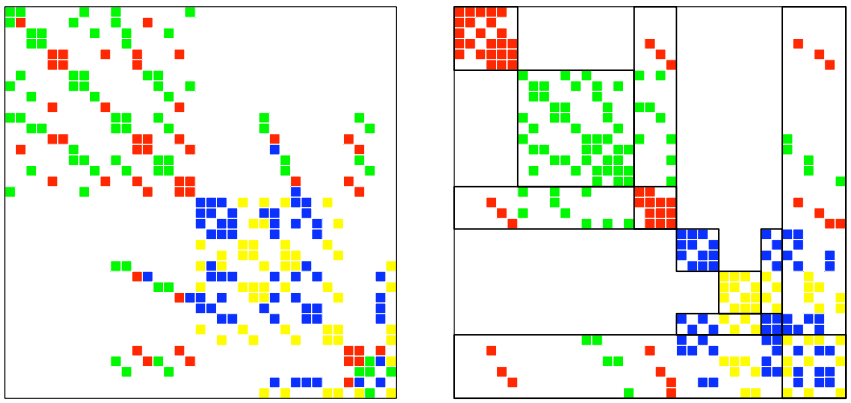


FIG. 4.3. *Matrix reordered and partitioned with nested dissection. Striped areas represent nonzeros corresponding to separators in Figure 4.2 where we have some flexibility in assignment. Gray blocks of nonzeros correspond to separator-separator edges in the the graph for which we also have flexibility in assignment.*



(a) Actual partitioning. (b) Permuted partitioning.

FIG. 4.4. *cage5 matrix partitioned using nested dissection. (a) shows how the matrix actually looks after being partitioned. (b) is a symmetric permutation of (a) for visualization purposes.*

**Algorithm 1** Nested Dissection Graph Partitioning

```
1: procedure NDPartition(G, k, ε, i, part)
2:     // Input: G = (V, E), the graph of a structurally symmetric matrix
3:     // Input: k, the desired number of parts
4:     // Input: ε, the allowed imbalance
5:     // Input: i, the label for the first part (in this bisection)
6:     // Output: part, a map of vertices and edges to parts (processes)
7:     if k > 1 then
8:         k_0 := ⌈k/2⌉
9:         k_1 := ⌊k/2⌋
10:        [V_0, V_1, S] := sep(G, k_0/k, ε/(log k)))        ▷ Find balanced separator.
11:        G_0 := G|V_0                                      ▷ G restricted to V_0
12:        G_1 := G|V_1                                      ▷ G restricted to V_1
13:        NDPartition(G_0, k_0, ε, i, part(G_0))
14:        NDPartition(G_1, k_1, ε, i + k_0, part(G_1))
15:        for each edge e with one endpoint, v, in V_0 or V_1 but not the other do
16:            part(e) := part(v)
17:        end for
18:        Assign vertices in S to compatible parts
19:        Assign edges with both endpoints in S to compatible parts
20:    else          ▷ Base case; simply assign part number to vertices and edges.
21:        part(V) := i
22:        part(E) := i
23:    end if
24: end procedure
```

two, then balanced separators are sufficient, but for general $k$, $\alpha$-balanced separators are required, where $\alpha < 1$ indicates the fraction of vertices in the larger subset. The subroutine *sep* finds an $\alpha-$balanced separator. After the separator $S$ is found, the recursive subproblems for $G_0$ and $G_1$ can be solved independently, possibly in parallel. This is an advantage of using recursive bisection.

One could optionally dynamically adjust the imbalance tolerance in the recursion, following the strategy in [21], but we did not explore this.

**4.2.1. Vertex Separator Algorithms.** Computing a minimal (balanced) vertex separator is NP-hard. We do not propose any new algorithms but rather leverage existing methods. Many application graphs are known to have good separators. For example, well-shaped meshes in $d$ dimensions have separators of size $O(n^{1-1/d})$. The most effective separator heuristics for large, irregular graphs are multilevel algorithms such as those implemented in METIS [18] and Scotch [10]. It is also possible to construct vertex separators from edge separators. This allows the use of graph partitioning software, but the quality is often not as good as a more direct method. A third option is to derive a vertex separator from hypergraph partitioning.

**4.3. Variations.** In nested dissection algorithms, there is a choice how to handle the separator at each level. Say $V$ has been partitioned into $V_0$, $V_1$, and $S$, where $S$ is the separator. The question is whether $S$ should be included in the subproblems or not. In the original nested dissection by George [13] and also the generalized nested dissection method [19], it was included in the recursion, but in many implementations it is not. We have chosen not to include the separator vertices in the subproblems

in the recursion since it simplified our implementation. A complication for us is that if the separator is not included, additional rules are needed to decide how to assign vertices and edge adjacent to the separators. However, this can be advantageous if this flexibility is used properly. We also found it useful to formulate a slightly more general non-recursive algorithm, which has the following steps:

1. Compute non-overlapping separators. This yields $k$ disjoint subdomains divided by a hierarchy of $k-1$ separators.
2. Let $V_i$ be the vertices in the subdomain $i$. Assign the vertices in $V_i$ to part $P_i$.
3. Let $E_i$ be the edges that contain a vertex in $V_i$. Assign the edges in $E_i$ to $P_i$.
4. Assign separator vertices to parts.
5. Assign edges connecting vertices of the same separator to parts.
6. Assign edges connecting vertices of two different separators to parts. By design these separators will be at different levels.

The algorithm does not depend on any particular method for calculating the vertex separators in step 1. However, in general, smaller separators will tend to yield lower communication volumes. Steps 2 and 3 are fully expounded in the description above. However, there are many different ways to do the part assignment in steps 4-6 and we leave this decision to the particular implementation.

In our initial implementation, we assign all the vertices in a given separator (step 4) to a part in the range of parts belonging to one half of the subdomain. The half is chosen to keep the vertex partitioning as balanced as possible. We assigned each separator vertex (step 4) to the part of the first traversed neighbor vertex in the correct range that had already been assigned a part. This greedy heuristic can be improved but had the advantage of being simple to implement and yielding better results than some more complicated heuristics. For the part assignment of edges interior to a separator (step 5), we assigned these edges to the part of the lowered numbered of the two vertices. We assigned edges connecting vertices from two different separators (step 6) to the part of the vertex of the lower-level separator. There were two reasons for this choice: It is consistent with Algorithm 1, and empirically it yielded better results. The rules in steps 4-6 are reasonable but not always optimal, so there may be room for improvement.

**4.4. Communication Volume Analysis.** For our analysis of our implementation, we assume that the number of parts is a power of two ($k = 2^i$). It follows that there are $k-1$ separators. We number the separators $S_1, S_2, \ldots, S_{k-1}$ in order of the level of the separators. We define $S$ to be the union of the separators ($S = \bigcup_{i=1}^{k-1} S_i$) and know that $\bigcap_{i=1}^{k-1} S_i = \emptyset$ since the separators are non-overlapping. We show lower and upper bounds on the communication volume, Vol.

THEOREM 4.3. *The communication volume in SpMV with partitioning produced by Algorithm 1 satisfies* $\mathrm{Vol}(G,k) \geq 2|S|$.

*Proof.* Each separator vertex $v_s \in S$ is connected to edges of at least two different parts. Thus, for any partitioning of $v_s$, $v_s$ will incur at least one word of communication for each of the two communication phases. □

It is important to note that this bound is valid for the general algorithm as well as our particular implementation.

THEOREM 4.4. *The communication volume in SpMV with partitioning produced*

*by our implementation outlined in Subsection 4.3 satisfies*

$$\mathrm{Vol}(G,k) \leq 2\sum_{i=1}^{k-1} |S_i| \left( \frac{k}{2^{\lfloor \log_2 i \rfloor}} - 1 \right).$$

*Proof.* With our implementation choice of assigning edges connecting vertices from two different separators to the part of the lower level separator, a separator vertex does not incur communication from its connection with higher level separator vertices. Thus, a given separator vertex $v_s$ of separator $S_i$ can be incident to edges of at most $\frac{k}{2^{\lfloor \log_2 i \rfloor}}$ different parts since the edge partition is compatible with the vertex partition. Thus, $v_s$ incurs at most $\frac{k}{2^{\lfloor \log_2 i \rfloor}} - 1$ words of communication for each communication phase. □

It is important to note that this bound does not apply to the general nested dissection partitioning algorithm (step steps above) but for our slightly more specific algorithm (Algorithm 1) that assigns edges connecting vertices from two different separators the part of the lower level separator. Without this edge partitioning rule, our bound would not be as tight ($\mathrm{Vol}(G,k) \leq 2\sum_{i=1}^{k-1} |S_i| (k-1)$). This justifies our implementation choice.

**4.5. Relation to Nested Dissection for Parallel Cholesky Factorization.** As previously mentioned, the dissection partitioning algorithm presented in this section is related to previous nested dissection work for parallel Cholesky factorization [14, 15]. The elimination tree used in Cholesky factorization is equivalent to a tree that can be formed from our hierarchy of separators and with the non-separator vertices as leaves of the tree. Also in both usages of nested dissection, communication is limited to separator vertices in the corresponding graph.

However, there are a few important distinctions between the partitioning for parallel Cholesky factorization and our partitioning for parallel matrix-vector multiplication. In parallel Cholesky, the elimination tree represents dependency constraints in the elimination order, but there are no such dependency constraints in matrix-vector multiplication. Also in the parallel Cholesky factorization work, there are additional fill nonzeros to take into account (and to partition) but no such additional nonzeros occur in our partitioning for matrix-vector multiplication. Related to this, permuting the matrix is important to Cholesky in order to reduce the fill. For our objective of reducing communication volume in matrix-vector multiplication, it is not important to impose the nested dissection ordering on the matrix since vertex orderings will have no effect on the communication volume. Also, the parallel Cholesky factorization assumes that the nonzeros in each column of lower triangular $L$ will be assigned to one process (1d partitioning). This column partitioning with subtree task assignment will be a special case of the partitioning allowed in our general matrix partitioning algorithm. We do not specify that the separator-separator blocks (gray blocks in Figure 4.3) are partitioned in this column-wise manner and thus allow for a 2d partitioning of the lower triangular portion of the matrix.

**5. Nonsymmetric Case.** The undirected graph model is limited to structurally symmetric graphs. In the nonsymmetric case, we can use a directed graph or a bipartitite graph. First we show how we can apply our nested dissection partitioning algorithm to bipartite graphs to partition nonsymmetric matrices. Then we discuss previous related work of the doubly bordered block diagonal (DBBD) form.

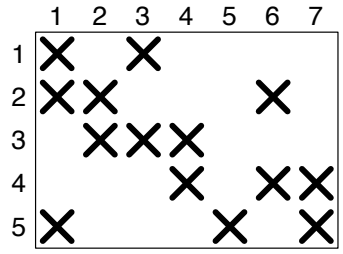



FIG. 5.1. *Rectangular matrix.*

**5.1. Bipartite Graph Model.** We generalize our symmetric communication (graph) model to the nonsymmetric case. This generalization is equivalent to a model recently proposed by Trifunovic [20]. We start with the bipartite graph $G = (R, C, E)$ of the matrix, where $R$ and $C$ correspond to rows and columns, respectively. In 1d distribution, we partition either the rows ($R$) or the columns ($C$). For fine-grain distribution, we partition both ($R$, $C$) and $E$ into $k$ sets. Note that we explicitly partition the edges $E$, which distinguishes our approach from previous work. To balance computation and memory, our primary objective is to balance the edges (matrix nonzeros). Vertex balance is a secondary objective.

We wish to analyze the communication requirements, so suppose that the vertices and edges have already been partitioned. Again, we assign communication cost to vertices such that a vertex incurs communication if and only if it has at least one incident edge in a different part. The communication volume will depend on the number of different parts to which these edges belong. Similar to the symmetric case, we have:

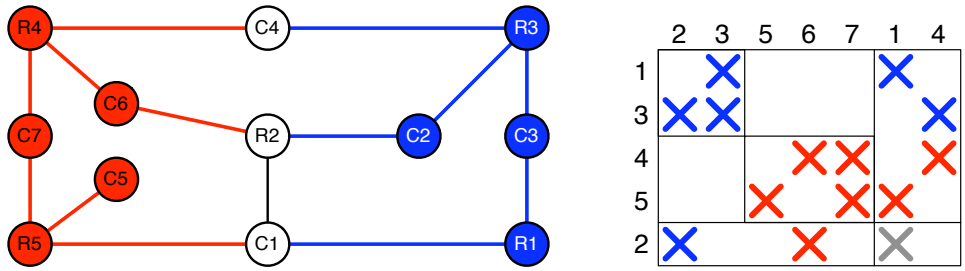
THEOREM 5.1. *Let $G(R, C, E)$ be the bipartite graph of a sparse matrix. Let $E(v)$ denote the set of edges incident to vertex $v$. Let $\pi(v)$ and $\pi(e)$ denote the parts to which $v$ and $e$ belong, respectively. Then the communication volume in matrix-vector multiplication is $\sum_{v \in R \cup C} (|\pi(v) \cup \pi(E(v))| - 1)$.*

In the bisection case, the volume is simply equal to the number of vertices that have at least one incident edge in a different part (boundary vertices).

Our model is a variation of the bipartite graph model proposed in [20]. One difference is that while [20] only partitions edges, we partition both edges and vertices. Second, [20] calls the resulting graph problem *edge coloring*, which we find confusing since "coloring" has a specific (and different) meaning in graph theory. Indeed the problem at hand is a partitioning problem.

Once we have built the bipartite graph for our nonsymmetric matrix, we can apply our nested dissection algorithm directly to this bipartite graph to partition the matrix. This procedure is outlined in Figures 5.1 and 5.2. Figure 5.1 shows a nonsymmetric matrix. The corresponding bipartite graph is shown in Figure 5.2(a). This bipartite graph is partitioned using our nested dissection partitioning algorithm. The uncolored vertices form a vertex separator for this bipartite graph. They and the one separator-separator edge are left for the particular implementation to partition. Figure 5.2(b) shows the partitioned nonsymmetric matrix corresponding to the partitioned bipartite graph of Figure 5.2(a).

**5.2. Equivalence to Fine-Grain Hypergraph Model.** Since our bipartite graph model is exact, it must be equivalent to the fine-grain hypergraph model. In




(a) Bipartite graph. (b) Partitioned matrix.

FIG. 5.2. *Bisection of bipartite graph and resulting partitioned/reordered nonsymmetric matrix (in DBBD form). (b) is reordered for visualization purposes.*

fact, there is a simple and elegant relation between the two models, independently discovered by Trifunovic [20]. Let the dual of a hypergraph $H = (V, E)$ be another hypergraph $H^* = (V^*, E^*)$, where $|V^*| = |E|$, $|E^*| = |V|$, and hyperedge $i$ in $E^*$ contains vertex $j$ iff hyperedge $j$ in $E$ contains vertex $i$. Now let $H$ be the hypergraph for the fine-grain model.

THEOREM 5.2 (Trifunovic (2006)). *The dual hypergraph $H^*$ of the fine-grain hypergraph is the bipartite graph.*

In other words, vertices in the fine-grain hypergraph $H$ are edges in the bipartite graph $G$ and vice versa. Hence partitioning the vertices of $H$ corresponds to partitioning the edges in $G$. In our bipartite graph model, we also explicitly partition the vertices in $G$, while the hyperedges in $H$ are partitioned only implicitly in the hypergraph algorithm.

Given a matrix with $z$ nonzeros, the bipartite graph has $z$ edges while the fine-grain hypergraph has $2z$ pins. Thus, algorithms based on the bipartite graph model may potentially use less memory, though using the standard adjacency list data structure each edge is stored twice so there is no savings.

**5.3. Vector Constraints.** For symmetric matrices we use a symmetric partitioning scheme, and this also gives a symmetric vector layout (i.e., the input and output vectors have the same distribution). For rectangular matrices, the input and out vectors clearly have different distributions. In the nonsymmetric square case, vectors layouts may be the same or different; both variations may be useful in applications. Different layouts is simplest to handle since no special consideration is required. A symmetric vector layout is a constraint that must be explicitly enforced. We chose the method used in [7, 21] that artificially adds nonzeros to the diagonal of $A$ before partitioning. After partitioning, we assign both $x_i$ and $y_i$ to the part that owns $a_{ii}$.

**5.4. Doubly Bordered Block Form.** The nested dissection ordering method for Cholesky produces a reordered matrix in doubly bordered block diagonal (DBBD) form, that is, the matrix is block diagonal except for a row and column block "border".

For example, with two diagonal blocks we have the structure

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

A key insight is that communication is limited to these borders, both for Cholesky and SpMV. Aykanat et al. [1] showed DBBD form for a nonsymmetric matrix can be obtained by computing a vertex separator in the bipartite graph. We can use this in the following way. Given a nonsymmetric possibly rectangular matrix (see Figure 5.1), form the bipartite graph (Figure 5.2(a)). Perform a nonsymmetric nested dissection ordering where the row and column vertices in the separator are ordered last. This gives a permuted matrix (Figure 5.2(b)) that can be partitioned according to the same rules as in the symmetric case. Note that as before, in practice it is not necessary to do the permutation (but is helpful in the illustration).

**6. Results.** We compare the partitionings of different methods for a set of 19 sparse matrices. These matrices were derived from different application areas (structural analysis, information retrieval, linear programming, circuit simulation, chemical engineering, etc.); 14 (of 19) were used and described in [21]. We replaced the **west0381** matrix (too small to partition into 256 parts) with the **Stanford_Berkeley** web matrix, obtained from [11]. Since the Mondriaan test set is fairly small by today's standards, we complemented the test set with four additional matrices from [11]. We performed separate experiments for symmetric and nonsymmetric matrices. We summarize the matrix properties in Tables 6.1-6.4. The first five matrices are symmetric (Table 6.1), the next four are larger symmetric matrices (Table 6.2), the next five are rectangular (Table 6.3), and the final five are square but structurally nonsymmetric (Table 6.4). Note that we treat explicit zeros in the sparse matrix storage as nonzeros so our number of nonzeros may differ slightly (but not significantly) from [11].

TABLE 6.1
*Symmetric Matrix Info*

| Name | N | nnz | nnz/N | application |
|---|---|---|---|---|
| cage10 | 11,397 | 150,645 | 13.2 | DNA electrophoresis |
| lap200 | 40,000 | 200,000 | 5.0 | 2-D Laplacian |
| finan512 | 74,752 | 596,992 | 8.0 | portfolio optimization |
| bcsstk32 | 44,609 | 2,014,701 | 45.2 | structural engineering |
| bcsstk30 | 28,924 | 2,043,492 | 70.7 | structural engineering |

TABLE 6.2
*Large Symmetric Matrix Info*

| Name | N | nnz | nnz/N | application |
|---|---|---|---|---|
| c-73 | 169,422 | 1,279,274 | 7.6 | non-linear optimization |
| asic680ks | 682,712 | 2,329,176 | 3.4 | circuit simulation |
| pkustk04 | 55,590 | 4,218,660 | 75.9 | structural engineering |
| gupta3 | 16,783 | 9,323,427 | 555.5 | linear programming |

In Subsections 6.1 - 6.2, we compare the communication volume of the resulting parallel sparse matrix-vector multiplication for these matrix partitionings. We

TABLE 6.3
*Nonsymmetric Rectangular Matrix Info*

| Name | rows | cols | nnz | application |
|---|---|---|---|---|
| dfl001 | 6,071 | 12,230 | 35,632 | linear programming |
| cre_b | 9,648 | 77,137 | 260,785 | linear programming |
| tbdmatlab | 19,859 | 5,979 | 430,171 | information retrieval |
| nug30 | 52,260 | 379,350 | 1,567,800 | linear programming |
| tbdlinux | 112,757 | 20,167 | 2,157,675 | information retrieval |

TABLE 6.4
*Nonsymmetric Square Matrix Info*

| Name | N | nnz | nnz/N | application |
|---|---|---|---|---|
| gemat11 | 4,929 | 33,185 | 6.7 | power flow optimization |
| memplus | 17,758 | 99,147 | 5.6 | circuit simulation |
| onetone2 | 36,057 | 227,628 | 6.3 | circuit simulation |
| lhr34 | 35,152 | 764,014 | 21.7 | chemical engineering |
| Stanford_Berkeley | 683,446 | 7,583,376 | 11.1 | information retrieval |

compare the implementation of our nested dissection algorithm with 1d hypergraph partitioning and fine-grain hypergraph partitioning. Though NP-hard problems, several good codes for graph and hypergraph partitioning are available, all based on the multilevel method. We used PaToH 3.0 [7] (called via Zoltan 3.0 [12]) as our hypergraph partitioner. The imbalance tolerance was set to 3%. Metis and ParMetis are often used to find nested dissection orderings, but were not suitable for us because (i) Metis does not return the separators, and (ii) ParMetis runs only in parallel and quality deteriorates with increasing numbers of processors. Instead we derive our vertex separators from edge separators produced by hypergraph partitioning. This choice also enables a fair comparison across methods since the code base is the same. In Subsection 6.3, we compare the communication volume of our nested dissection partitioning implementation with that of the Mondriaan implementation. In Subsection 6.4, we compare the messages sent for our nested dissection implementation with 1d hypergraph partitioning and fine-grain hypergraph partitioning for the symmetric matrices.

All our experiments were run on a Dell Precision 64-bit workstation with four dual-core Intel Xeon processors (though only one was used per run) and 16 Gb RAM. The operating system was Red Hat Enterprise Linux and the compiler gcc 4.1.1.

**6.1. Symmetric Matrices.** We partition the 5 symmetric matrices shown in Table 6.1 using 1d, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, and 256 parts (Note that 256 is larger than previous papers used, reflecting the greater number of processors and cores in current computers.). We use the nested dissection implementation outlined in Section 4 to partition the matrices directly. The average communication volumes are shown in Table 6.5. For 1d partitioning, we list the total communication volume. For the fine-grain and nested dissection methods, we list a scaled volume relative to the 1d volumes (as well as in subsequent Tables 6.6 - 6.8) such that scaled volumes less than 1 indicate an improvement over the 1d method. We see that our nested dissection method performs consistently better than 1d (scaled volumes less than 1). When compared to the fine-graph method, we see for

most partitionings that the nested dissection method yielded similar or better results for four of the five matrices. The nested dissection only performed significantly worse for the **cage10** matrix and the 256 part partitioning of the **finan512** matrix. Another important point is that the nested dissection method runtimes were significantly lower than that of fine-grain (up to a 89% decrease) and only slightly higher than 1d.

TABLE 6.5
*Average (20 runs) communication volume (in words) and runtimes (in seconds) for k-way partitioning of symmetric matrices using different partitioning methods.*

| | | 1d | | fine-grain | | nested dissection | |
|---|---|---|---|---|---|---|---|
| Name | k | total volume | runtime | scaled volume | runtime | scaled volume | runtime |
| cage10 | 4 | 5379.0 | 13.1 | **0.755** | 28.3 | 0.822 | 13.8 |
| | 16 | 12874.5 | 25.0 | **0.689** | 46.7 | 0.887 | 26.4 |
| | 64 | 23463.3 | 41.3 | **0.696** | 68.9 | 0.980 | 43.7 |
| | 256 | 40830.9 | 66.7 | **0.716** | 101.9 | 1.030 | 69.9 |
| lap200 | 4 | 1535.1 | 7.9 | 1.002 | 19.0 | **0.997** | 8.5 |
| | 16 | 3013.9 | 15.2 | 1.001 | 30.2 | **0.999** | 16.7 |
| | 64 | 5813.0 | 25.2 | **0.995** | 44.4 | 1.002 | 27.8 |
| | 256 | 11271.8 | 51.4 | **0.981** | 71.9 | 1.002 | 56.5 |
| finan512 | 4 | 295.7 | 23.8 | 0.883 | 82.8 | **0.775** | 24.5 |
| | 16 | 1216.7 | 48.6 | 0.844 | 128.3 | **0.770** | 50.6 |
| | 64 | 9986.0 | 90.9 | 0.864 | 185.4 | **0.807** | 95.4 |
| | 256 | 38985.4 | 142.5 | **0.679** | 253.0 | 0.770 | 152.1 |
| bcsstk32 | 4 | 2111.9 | 58.5 | **0.763** | 470.7 | 0.840 | 61.5 |
| | 16 | 7893.1 | 102.0 | **0.802** | 718.4 | 0.861 | 110.3 |
| | 64 | 19905.4 | 152.7 | 0.938 | 922.6 | **0.910** | 168.5 |
| | 256 | 46399.0 | 215.2 | 1.002 | 1133.1 | **0.944** | 239.5 |
| bcsstk30 | 4 | 1794.4 | 76.0 | 1.079 | 688.6 | **0.781** | 78.5 |
| | 16 | 8624.7 | 139.7 | 1.133 | 1076.5 | **0.827** | 148.4 |
| | 64 | 23308.0 | 205.7 | 1.102 | 1381.1 | **0.902** | 222.0 |
| | 256 | 56100.4 | 262.4 | 1.031 | 1639.8 | **0.982** | 287.5 |

We partition the 4 larger symmetric matrices shown in Table 6.2 using 1d column, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, 256, and 1024 parts. These matrices were chosen to demonstrate instances in which 1d partitioning is insufficient. We picked these examples based on a comparison of the 1d and fine-grain partitioning methods. The resulting communication volumes for the partitioning of these matrices is shown in Table 6.6. As expected, our nested dissection method performs much better than 1d (scaled volumes significantly less than 1). When compared to the fine-graph method, we see similar but in general not quite as good results. Again, we see a great improvement over fine-grain in terms of runtime (up to 96% decrease in the runtime).

**6.2. Nonsymmetric Matrices.** We partitioned the 10 nonsymmetric matrices shown in Tables 6.3 and 6.4 using 1d column, 1d row, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, and 256 parts. However, in order to use the nested dissection partition method with the nonsymmetric matrices, we have to form bipartite graphs as described in Section 5. We can then apply the nested dissection implementation outlined in Section 4 to partition the bipartite graph, which

TABLE 6.6

*Average (20 runs) communication volume (in words) and runtimes (in seconds) for k-way partitioning of symmetric matrices using different partitioning methods. * - designates partitions that did not meet the 3% load balance tolerance.*

| | | 1d | | fine-grain | | nested dissection | |
|---|---|---|---|---|---|---|---|
| | | total | | scaled | | scaled | |
| Name | k | volume | runtime | vol. | runtime | vol. | runtime |
| c-73 | 4 | 42314.2 | 162.2 | 0.0381 | 280.7 | **0.033** | 171.1 |
| | 16 | 99862.8 | 358.1 | 0.051 | 418.7 | **0.050** | 381.8 |
| | 64 | 206273.2* | 343.1* | 0.0730 | 604.9 | **0.063** | 361.8 |
| | 256 | 171079.3* | 545.0* | **0.225** | 783.9 | 0.246* | 559.6* |
| | 1024 | 216157.1* | 430.9* | **0.424** | 894.6 | 0.427* | 444.2* |
| asic680ks | 4 | 3560.4 | 78.7 | **0.509** | 270.8 | 0.612 | 78.7 |
| | 16 | 9998.5 | 107.4 | **0.463** | 384.8 | 0.605 | 127.7 |
| | 64 | 21785.8 | 146.7 | **0.439** | 489.7 | 0.588 | 166.5 |
| | 256 | 38869.4 | 193.9 | **0.492** | 598.1 | 0.613 | 233.8 |
| | 1024 | 62482.8 | 301.2 | **0.631** | 854.6 | 0.678 | 381.8 |
| pkustk04 | 4 | 6610.8 | 133.4 | 0.626 | 1356.4 | **0.526** | 141.9 |
| | 16 | 27565.4 | 237.9 | **0.492** | 2465.5 | 0.602 | 258.7 |
| | 64 | 75329.7 | 346.4 | **0.416** | 3471.2 | 0.623 | 380.1 |
| | 256 | 162105.5 | 461.6 | **0.428** | 3731.1 | 0.558 | 485.5 |
| | 1024 | 372343.2* | 549.2* | **0.410** | 3871.3 | 0.502* | 585.8 |
| gupta3 | 4 | 30066.9 | 379.8 | 0.285 | 5291.0 | **0.186** | 429.0 |
| | 16 | 103475.9 | 585.5 | 0.313 | 14299.8 | **0.206** | 653.4 |
| | 64 | 332559.5 | 796.8 | 0.267 | 24856.8 | **0.191** | 872.2 |
| | 256 | 1105498.9* | 924.7* | **0.165** | 27766.7 | 0.181* | 1028.9* |
| | 1024 | 3125684.9* | 1030.8* | **0.107** | 29565.7 | 0.198* | 1106.8* |

gives us a partitioning of the nonsymmetric matrix. In this subsection, we report the communication volumes of the SpMV resulting from these partitionings.

Table 6.7 shows communication volumes averaged over 20 runs for the 5 rectangular matrices from Table 6.3. The nested dissection method results were consistently worse than the fine-grain results for these rectangular results and often worse than one of the 1d methods. Only for the **tbdlinux** matrix did the nested dissection method yield significantly lower communication volumes than both 1d methods.

Table 6.8 shows communication volumes for the 5 square matrices from Table 6.4 when the input and output vectors are required to have the same partitioning. The nested dissection results are very similar to the fine-grain results and the 1d column results for the **gemat11**, **lhr34**, and **Stanford_Berkeley** matrices. For the **memplus** and **onetone2** matrices, the nested dissection method yields better results than 1d column (and 1d row for **memplus**). When compared to the fine-graph method for these two, we see similar but in general not quite as good results.

**6.3. Mondriaan Comparisons.** Mondriaan [21] is also a relevant method to compare against since it produces a 2d matrix partitioning, and as our method, is quite fast compared to fine-grain (being based on 1d hypergraph partitioning). Here we compare the results of our nested dissection partitioning implementation with Mondriaan 1.0.2. It is important to note that this is not a direct comparison of algorithms but also of implementations. Mondriaan uses its own hypergraph partitioner,

TABLE 6.7

*Average (20 runs) communication volume (in words) for k-way partitioning of rectangular nonsymmetric matrices using different partitioning methods. ** - for one run, hypergraph partitioner failed to produce partition after several hours, averaging 19 runs.*

| Name | k | 1d col. total vol. | 1d row scaled vol. | fine-grain scaled vol. | nested diss. scaled vol. |
|---|---|---|---|---|---|
| dfl001 | 4 | 1388.4 | 2.141 | **0.996** | 1.181 |
| | 16 | 3575.5 | 1.631 | **0.997** | 1.155 |
| | 64 | 6040.2 | 1.391 | **0.995** | 1.119 |
| | 256 | 8897.0 | 1.377 | **0.990** | 1.097 |
| cre_b | 4 | **1119.6** | 29.194 | 1.027 | 2.312 |
| | 16 | **3509.3** | 15.970 | 1.011 | 1.848 |
| | 64 | **7952.3** | 9.315 | 1.024 | 1.605 |
| | 256 | 17077.8 | 6.048 | **0.997** | 1.409 |
| tbdmatlab | 4 | 14991.2 | 0.937 | 0.718 | **0.681** |
| | 16 | 40562.8 | 1.343 | **0.778** | 0.888 |
| | 64 | 81468.6 | 1.661 | **0.797** | 1.041 |
| | 256 | 144098.2 | 1.673 | **0.757** | 1.093 |
| nug30 | 4 | **56796.5** | 4.746 | 1.100 | 1.307 |
| | 16 | **115539.4** | 3.320 | 1.157 | 1.507 |
| | 64 | **199977.0** | 2.674 | 1.172 | 1.530** |
| | 256 | **307627.1** | 2.090 | 1.166 | 1.494 |
| tbdlinux | 4 | 52021.1 | 0.813 | 0.471 | **0.429** |
| | 16 | 146980.9 | 1.136 | **0.565** | 0.594 |
| | 64 | 307829.8 | 1.449 | **0.610** | 0.733 |
| | 256 | 569152.5 | 1.600 | **0.611** | 0.854 |

which generally produces worse cut results than PaToH. Comparisons of the symmetric test cases are given in Table 6.9, where a ratio less than one indicates our method has lower communication volume. We observe that our method was best for all but two matrices. Looking at only the first five test cases it appears the two methods are roughly similar in quality. However, Mondriaan suffers some of the same problems as 1d partitioning methods since it is uses 1d partitioning recursively. Therefore, the first bisection will sometimes incur a large communication volume. We see our method produces much better results than Mondriaan on the four large test matrices (Table 6.2), which are difficult for 1d methods.

**6.4. Message Metric.** Communication volume is not the only metric that is important when evaluating the quality of a sparse matrix partitioning in terms of parallel matrix-vector multiplication. The number of messages communicated can also be as important or more important if the communication volume is low or the latency is high. 1d partitioning of the sparse matrix yields a parallel matrix-vector algorithm with only one phase of communication. This tends to result in a low number of messages in comparison to 2d partitioning methods such as the fine-grain method. Table 6.10 shows the average number of messages sent (same as average number of messages received) per part for the 1d column, fine-grain, and nested dissection partitionings of the five symmetric matrices shown in Table 6.1 into 16, 64, and 256 parts. As expected, the 1d method has consistently the lowest average number of messages sent per part of the three methods. The nested dissection results

TABLE 6.8

*Average (20 runs) communication volume (in words) for k-way partitioning of square non-symmetric matrices using different partitioning methods. Vectors are partitioned the same. ** - hypergraph partitioner failed to produce partition after several hours.*

| Name | k | 1d col. total vol. | 1d row scaled vol. | fine-grain scaled vol. | nested diss. scaled vol. |
|---|---|---|---|---|---|
| gemat11 | 4 | 2357.2 | 1.040 | 1.033 | **0.975** |
| | 16 | **4578.8** | 1.033 | 1.047 | 1.033 |
| | 64 | **6291.4** | 1.022 | 1.061 | 1.041 |
| | 256 | **8632.8** | 1.005 | 1.010 | 1.038 |
| memplus | 4 | 3615.7 | 1.002 | **0.107** | 0.227 |
| | 16 | 6120.8 | 1.004 | **0.217** | 0.399 |
| | 64 | 9696.5 | 0.999 | **0.316** | 0.513 |
| | 256 | 16741.6 | 1.001 | **0.387** | 0.547 |
| onetone2 | 4 | 1009.7 | 0.778 | **0.751** | 0.752 |
| | 16 | 4932.9 | 0.791 | **0.723** | 0.776 |
| | 64 | 11934.1 | 0.876 | **0.719** | 0.802 |
| | 256 | 26095.9 | 0.903 | **0.843** | 0.878 |
| lhr34 | 4 | 6613.6 | 1.078 | 1.005 | **0.918** |
| | 16 | 20908.6 | 1.016 | 1.053 | **0.978** |
| | 64 | **36937.8** | 1.005 | 1.073 | 1.030 |
| | 256 | **61081.2** | 1.007 | 1.036 | 1.057 |
| Stanford_Berkeley | 4 | **1383.0** | 3.536 | 1.095 | 1.300 |
| | 16 | 5948.8 | 23.001 | **0.999** | 1.283 |
| | 64 | 17725.9 | ** | **0.977** | 1.087 |
| | 256 | 43801.5 | ** | **0.971** | 1.026 |

TABLE 6.9

*Average (20 runs) communication volume ratio nested dissection/Mondriaan for k-way partitioning of symmetric matrices. An asterix indicates the desired load-balance tolerance was not achieved.*

| Name | k=4 | k=16 | k=64 | k=256 |
|---|---|---|---|---|
| cage10 | **0.876** | 1.013 | 1.194 | 1.303 |
| lap200 | 1.045 | 1.101 | 1.158 | 1.142 |
| finan512 | **0.208** | **0.524** | **0.865** | **0.792** |
| bcsstk32 | **0.757** | **0.824** | **0.930** | **0.978** |
| bcsstk30 | **0.771** | **0.836** | **0.901** | **0.837** |
| c-73 | **0.037** | **0.065** | **0.092** | **0.211*** |
| asic680ks | **0.375** | **0.329** | **0.431** | **0.459** |
| pkustk04 | **0.318** | **0.434** | **0.678** | **0.722** |
| gupta3 | **0.240** | **0.390** | **0.643** | 1.152* |

are significantly lower than the fine-grain results for most of the partitionings of the five matrices.

**7. Conclusions.** We presented a new graph-oriented approach to sparse matrix partitioning for sparse matrix-vector multiplication. Although exact hypergraph models exist, we find our graph model, which is also exact, more intuitive. We presented a nested dissection partitioning algorithm that approximately solves our communication

TABLE 6.10
*Average messages sent (same as received) per process for k-way partitioning of symmetric matrices using different partitioning methods. All methods use hypergraph partitioning.*

| Name | k | 1d | fine-grain | nested dissection |
|---|---|---|---|---|
| cage10 | 16 | **11.6** | 21.4 | 15.4 |
| | 64 | **20.0** | 35.1 | 27.7 |
| | 256 | **24.2** | 36.4 | 33.6 |
| lap200 | 16 | **6.0** | 8.1 | 6.9 |
| | 64 | **6.0** | 8.2 | 6.8 |
| | 256 | **6.0** | 8.1 | 6.7 |
| finan512 | 16 | **2.0** | 4.0 | **2.0** |
| | 64 | **2.1** | 9.3 | 2.3 |
| | 256 | **6.0** | 17.4 | 7.2 |
| bcsstk32 | 16 | **4.5** | 7.7 | 5.3 |
| | 64 | **6.1** | 10.9 | 7.6 |
| | 256 | **6.7** | 11.7 | 8.6 |
| bcsstk30 | 16 | **3.6** | 7.6 | 4.2 |
| | 64 | **5.9** | 12.2 | 7.6 |
| | 256 | **7.3** | 16.0 | 9.9 |

graph model. We showed our partitioning method is clearly superior to the traditional 1d partitioning method (up to 97% reduction in communication volume), and produces partitions of similar quality to the fine-grain method for symmetric matrices at a great reduction in the runtime. For nonsymmetric matrices, the results varied substantially among problems but for many problems the quality of our nested dissection algorithm's partitions was comparable to that of the fine-grain method. Again, there was a substantial reduction in runtime for the nested dissection algorithm. Therefore, we believe our approach is well suited as a general sparse matrix partitioning method. Although nested dissection and recursive substructuring have been previously used in several contexts, this is (to our knowledge) the first such algorithm and analysis specifically for sparse matrix-vector multiplication.

There are several directions for improvement of our algorithm and implementation. First, one could use a better implementation to find vertex separators (e.g., Metis or Scotch). We did not do this here to allow a more fair comparison of the models (not the implementation). Second, one can likely partition the vertices and edges in and around the separators in a better way. This becomes relatively more important as the number of parts grow. Third, in the nonsymmetric case, results may potentially improve by customizing the partitioner (separator routine) for bipartite graphs [16]. Furthermore, we believe our algorithm can be efficiently implemented in parallel since the core computational kernel is nested dissection (by vertex separators), so existing software like ParMetis [17] or PT-Scotch [10] may be used. We plan to study parallel performance in future work.

Although our work targeted sparse matrix-vector multiplication, the partitioning algorithm (data distribution) we presented can be used in any sparse matrix computation, and may also reduce communication in parallel graph algorithms (e.g., coloring, matching, shortest path).

**Acknowledgments.** We thank Rob Bisseling, Umit Catalyurek, Michael Heath, and Bruce Hendrickson for helpful discussions. We thank Florin Dobrian and Ma-

hantesh Halappanavar for providing a matching code used to produce vertex separators. This work was funded by the US Dept. of Energy's Office of Science through the CSCAPES Institute and the SciDAC program.

## REFERENCES

[1] C. Aykanat, A. Pinar, and U. V. Catalyurek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.

[2] C. Berge. *Graphs and Hypergraphs*, volume 6 of *North-Holland Mathematical Library*. Elsevier Science Publishing Company, 1973.

[3] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. Elsevier Science Publishing Company, 1989.

[4] R. H. Bisseling. *Parallel Scientific Computing: A structured approach using BSP and MPI*. Oxford University Press, 2004.

[5] R. H. Bisseling and W. Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, 21:47–65, 2005.

[6] E. G. Boman. A nested dissection approach to sparse matrix partitioning. In *Proc. Applied Math. and Mechanics*, volume 7, 2007. Presented at ICIAM07, Zurich, Switzerland, July 2007.

[7] Ü. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Dist. Systems*, 10(7):673–693, 1999.

[8] Ü. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001)*, April 2001.

[9] Ü. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *Proc. Supercomputing 2001*. ACM, 2001.

[10] C. Chevalier and F. Pellegrini. PT-SCOTCH: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6–8):318–331, Jul. 2007.

[11] T. A. Davis. The University of Florida Sparse Matrix Collection, 1994. Matrices found at http://www.cise.ufl.edu/research/sparse/matrices/.

[12] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[13] A. George. Nested dissection of a regular finite-element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[14] A. George, M. T. Heath, J. W.-H. Liu, and E. G.-Y. Ng. Solution of sparse positive definite systems on a hypercube. *Journal of Computational and Applied Mathematics*, 27:129–156, 1989. Also available as Technical Report ORNL/TM-10865, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[15] A. George, J. W.-H. Liu, and E. G.-Y. Ng. Communication results for parallel sparse Cholesky factorization on a hypercube. *Parallel Computing*, 10(3):287–298, May 1989.

[16] B. Hendrickson and T. G. Kolda. Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel computation. *SIAM Journal on Scientific Computing*, 21(6):2048–2072, 2000.

[17] G. Karypis and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Dept. Computer Science, University of Minnesota, 1997. `http://www.cs.umn.edu/~metis`.

[18] G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Dept. Computer Science, University of Minnesota, 1998. *http://www.cs.umn.edu/~metis*.

[19] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Ananlysis*, 16:346–358, 1979.

[20] A. Trifunovic and W. J. Knottenbelt. A general graph model for representing exact communication volume in parallel sparse matrix-vector multiplication. In *Proc. of 21st International Symposium on Computer and Information Sciences (ISCIS 2006)*, pages 813–824, 2006.

[21] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.