A Nested Dissection Partitioning Method for Parallel Sparse Matrix-Vector Multiplication

Erik G. Boman and Michael M. Wolf*
Sandia National Laboratories†
Albuquerque, NM 87185-1318
Email: egboman@sandia.gov, michael.wolf@ll.mit.edu

***Abstract*—We consider how to map sparse matrices across processes to reduce communication costs in parallel sparse matrix-vector multiplication, an ubiquitous kernel in high performance computing. 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 that approximately solves this model.**

We have implemented our algorithm using hypergraph partitioning software to enable a fair comparison with existing methods. We present partitioning results for sparse structurally symmetric matrices from several application areas. Our new method is competitive with the best 2D algorithm (fine-grain hypergraph model) in terms of communication volume, but requires fewer messages. The nested dissection method is almost as fast to compute as 1D methods and the communication volume is significantly reduced (up to 97%) compared to 1D layout. Further improvements in quality may be possible by small modifications to existing nested dissection ordering software.

I. Introduction and Background

Sparse matrix-vector multiplication (SpMV) is a ubiquitous kernel in high performance computing and is of particular importance for iterative linear solvers and graph analytics related to network data (e.g., Signal Processing for Graphs algorithms [1]). An important combinatorial problem in parallel computing is how to map or distribute the matrix and the vectors across the processes to minimize the communication cost. A good distribution for these matrices and vectors is crucial to obtaining good parallel performance for SpMV, especially for matrices derived from network data that are particularly computationally challenging. Our work is relevant to parallel computing on both distributed-memory and shared-memory systems. Typically, multicore computers have non-uniform memory access so data layout is important. The phrase "communication" correspond to memory access on those systems.

The SpMV $y = Ax$ is usually parallelized such that the process that owns element a_{ij} computes $a_{ij}x_j$ (a local operation if x_j, y_i, and a_{ij} reside on the same process; otherwise communication is required). In general, there are two phases of communication: sending x_j to processes with a nonzero a_{ij} (expand) and sending the partial inner product y_i values to relevant processes (fold). In this paper, we address sparse matrix-vector partitioning (Definition 1), 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. *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, (patterns of A_i are disjoint)*

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

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

For SpMV, the total *volume* is the most common communication cost metric (see e.g., [2]). However, several other communication metrics are also relevant, including the maximum volume for any process, the total number of messages, and the maximum number of messages for any process [3]. Any single metric is insufficient to predict performance. We choose to focus on both total volume and messages in this paper since we believe that to consider only volume is too simplistic.

Stated above is a very general form. We first consider a restricted version for symmetric matrices where the input and output vectors must have the same distribution. It has been observed [2], [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. 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. The simplest 1D method is to assign approximately 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. However, it is often possible to reduce the communication by partitioning the rows in a better (non-contiguous) way, using graphs or hypergraphs to model this problem [2]. The complication is that solving these

*Currently at MIT Lincoln Laboratory.

†Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

graph [5] or hypergraph [6] partitioning problems optimally (maintaining load-balance while minimizing communication) is known to be NP-hard. Multilevel methods, however, have been effective in obtaining high quality partitions of graphs at a low cost (linear time in the graph size) [7]–[9].

Although the simplicity of 1D distributions may be desirable, the communication volume can often be reduced by using 2D distributions. Figure 1 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 1 (right).

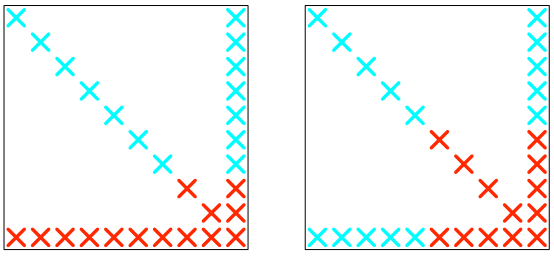


Fig. 1. Arrowhead matrix with 1D (left) and 2D (right) distribution, for two parts (colors). The communication volumes in this example are eight and two words, respectively.

Recently, several 2D decompositions have been proposed [10]–[12]. The idea is to reduce the communication volume further by giving up the simplicity of the 1D structure. The fine-grain hypergraph method [10] is of particular interest since it is the most general method with the matrix nonzeros being assigned to parts (processes) individually. In the fine-grain hypergraph model, each nonzero is a vertex while rows and columns correspond to hyperedges. This gives an exact model of the communication volume and the corresponding method typically yields low communication volume. However, this method is mostly of theoretical interest since it is quite slow and not supported in any standard software library.

In this paper, we introduce a graph model that also accurately describes communication in general (fine-grain) distribution. This leads to a new graph-based algorithm, a "nested dissection partitioning algorithm," which is related to previous nested dissection work for parallel Cholesky factorization [13]. (A brief preliminary version, without analysis, appeared in [14].) 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. One of our objectives in developing new 2D methods is to produce similar quality partitions to fine-grain hypergraph in shorter runtime.

II. An Exact Graph Model for Structurally Symmetric Matrices

We present an accurate graph model for communication volume in SpMV for structurally symmetric matrices, which can be extended to nonsymmetric matrices by using a bipartite graph model. We restrict our attention here to symmetric partitioning schemes, where a_{ij} and a_{ji} are assigned the same part, and further, the input and output vectors have the same distribution. This allows us to work with the undirected graph $G(V, E)$, where the vertices correspond to the vector indices (and diagonal nonzeros) and the edges correspond to the off-diagonal 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.

Theorem 1. *Let $G(V, E)$ be the graph of a symmetric sparse matrix A such that A has the same nonzero structure as the adjacency matrix of $G(V, E)$. Let $E(v)$ denote the set of edges incident to vertex v. Let $\pi(v)$ denote the part to which v belongs. Let $\pi(e)$ denote the part to which edge e belongs. Then the communication volume in SpMV is $2\sum_{v\in V}(|\pi(v) \cup \pi(E(v))| - 1)$.*

Proof: A vertex v_i incurs communication if and only if there are incident edges that belong to a different part. The volume (for one phase of communication) is equivalent to the number of parts assigned to the incident edges that differ from the part of v_i since these correspond to the processes that will receive x_i from and send their portion of the inner product for y_i to the process owning v_i during the sparse matrix-vector product operation. 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 SpMV when optimally solved.

Figure 2 illustrates the exact graph model for 2D symmetric partitioning of matrices. The graph corresponds to the symmetric matrix (showed partitioned on the right). The edges and vertices in the graph are partitioned into two parts (colors). The vertices that have incident edges belonging to a different part (and thus incur communication) are highlighted in green. The matrix on the right shows the 2D symmetric matrix partition obtained from the partitioned graph. The partition of the diagonal 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.

III. A Vertex Separator Partitioning Algorithm

In Section II, 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

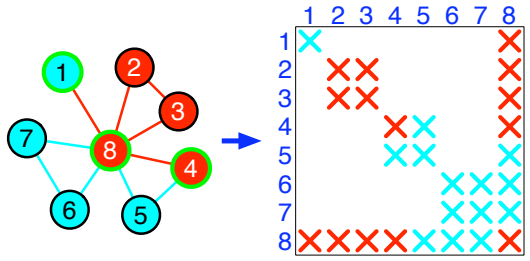


Fig. 2. 2D graph bisection for symmetrically partitioned matrix. Part (color) of graph edge corresponds to symmetric pair of off-diagonal nonzeros.

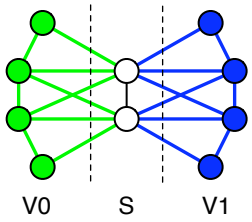


Fig. 3. Bisection. Vertex separator (uncolored vertices) used to partition vertices into three disjoint subsets (V_0, V_1, S).

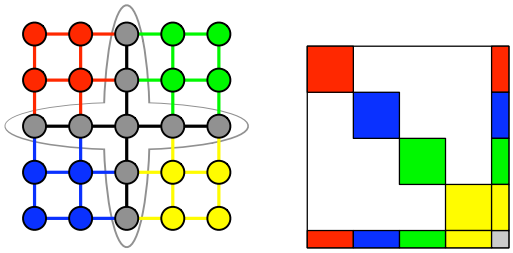
from a vertex separator for the graph. One constraint that we impose on our algorithm is that the vertex and edge partitions are *compatible*. A vertex partition is *compatible* with an edge partition if every vertex belongs to the same part as one of its incident edges. Similarly, an edge partition 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.

A. Bisection

For simplicity, we consider bisection first. First we compute a small balanced vertex separator S for the graph where the separator is balanced by edges (since the primary goal is load balance of the matrix non zeros). 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 $j = 0, 1$ (example shown in Figure 3).

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].



(a) Graph (b) Matrix

Fig. 4. k-way partition of graph and corresponding block matrix.

B. Extension to k-way Partitioning

In practice, one wishes to partition into $k > 2$ parts. A natural extension is to compute a balanced k-separator, a set S such that the removal of S breaks G into k disjoint subgraphs. We then assign each subgraph to a different part. Again, we need to decide how to assign the vertices and edges in the separator. Edges with one endpoint in subgraph G_i should be assigned to part i. Such a partition is illustrated in Figure 4.

The lower right matrix block corresponds to the separator. We left it gray because it is not obvious how best to partition it. Unlike the bisection case, the communication volume can change depending on how the separator vertices/edges are assigned. From Theorem 1 it follows that:

Theorem 2. *The communication volume in SpMV for a matrix A and a partition π from a k-separator S, is*

$$Vol(A, k, \pi) = 2 \sum_{v \in S} (\lambda(v) - 1),$$

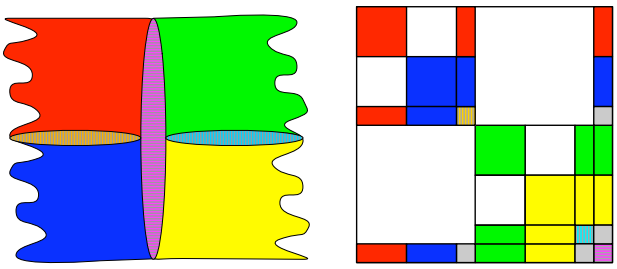
where $\lambda(v)$ is the number of parts assigned to edges incident to v.

This theorem tells us that not only do we want to find small separators, but also the vertices in the separators should be connected to as few parts as possible.

C. Nested Dissection Partitioning Algorithm

The communication volume depends not only on the separator sizes, but also on the structure of the separators. We seek separators that are connected to as few parts as possible. We propose to use recursive bisection. At each bisection, the graph is split into two subgraphs and a small separator. At the next level, the separators for the right and left subgraphs are not connected, so the overall connectivity of separators is limited. This recursive bisection technique has been effectively used in many graph and hypergraph partitioning methods [2], [15].

Coincidentally, recursive bisection is the most common method to produce a k-separator. This procedure is known as "nested dissection" and has been well studied [16], [17] since it is important for sparse matrix ordering and factorization. Our idea is that nested dissection not only produces a small



(a) Generic graph (b) Corresponding matrix

Fig. 5. Graph partitioned using nested dissection and corresponding matrix. Striped areas in matrix represent nonzeros corresponding to separators in graph 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.

k-separator but also implicitly reduces the communication volume by imposing helpful structure in the separators.

The procedure is illustrated in Figure 5 for four parts. We show the recursive procedure on 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 5. We do this in illustrations to make the partitioning method more clear.

Algorithm 1 summarizes our recursive algorithm. The subroutine SEPARATOR finds an α-balanced separator with respect to the edges (where $\alpha = \hat{k}/k$). At each level of recursion, the α-balanced separators are found within a load-imbalance tolerance of $\epsilon/(\log k)$, allowing the overall k-separator to have the same amount of work (edges) in each subdomain within the tolerance ϵ. After the separator S is found, the two subproblems can be solved independently, possibly in parallel. This is an advantage of using recursive bisection.

Computing a minimal (balanced) vertex separator is NP-hard. We do not propose any new algorithms but rather leverage existing methods. The most effective separator heuristics for large, irregular graphs are multilevel algorithms such as those implemented in METIS [18] and Scotch [19]. 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.

D. 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. 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 edges adjacent to the separators. However, this can be advantageous if this flexibility is used properly.

Algorithm 1 Nested Dissection Graph Partitioning

```
1:  procedure NDPARTITION(G, k, ε, i, part)
2:      // Input: G = (V, E) (graph of a symmetric matrix)
3:      // Input: k (number of parts), ε (allowed imbalance)
4:      // Input: i (label for the first part in this bisection)
5:      // Output: part (a map of vertices and edges to parts)
6:      if k > 1 then                                  ▷ Bisect and recurse
7:          k̂ := ⌈k/2⌉
8:          α := k̂/k
9:          [V_0, V_1, S] := SEPARATOR(G, α, ε/(log k))   ▷
                Find α-balanced separator
10:         NDPARTITION(G(V_0), k̂, ε, i, part(V_0))
11:         NDPARTITION(G(V_1), k − k̂, ε, i + k̂, part(V_1))
12:         for each edge e with one vertex, v, in V_0 or V_1 do
13:             part(e) := part(v)
14:         end for
15:         Assign vertices in S to compatible parts
16:         Assign edges with both v_i ∈ S to compatible parts
17:     else                          ▷ Base case: assign V, E this part.
18:         part(V) := i
19:         part(E) := i
20:     end if
21: end procedure
```

Algorithm 1 does not depend on any particular method for calculating the vertex separators in line 10. In general, smaller separators will tend to yield lower communication volumes. The assignment rule in lines 13-15 means that edges between separator vertices and non-separator (at this level of recursion) vertices are assigned to the part of the latter, corresponding to the partitioning shown in Figure 5. However, this rule is not essential and other partitionings of these edges may give lower communication cost.

In our current implementation, we assign all the vertices in a given separator (line 16) 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 to the part of the first traversed neighbor vertex in the correct range that had already been assigned a part. This greedy heuristic is not optimal but has the advantage of being simple to implement and yields better results than some more complicated heuristics we tried. For the assignment of edges interior to a separator (line 17), we assigned these to the part of the lower numbered incident vertex (a reasonable heuristic but there are other options).

IV. RESULTS

We compare the partitionings of different methods for a set of eight sparse matrices (summarized in Table I). These matrices were derived from different application areas with the first four being used and described in [12] and the last four obtained from [20].

Below, we compare the communication volume and the messages sent in the resulting parallel SpMV as well as the runtimes for several partitioning methods. In particular, we

TABLE I
MATRIX INFO

Name	N	nnz	nnz/N	application
cage10	11,397	150,645	13.2	DNA electrophoresis
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
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

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 [2] (called via Zoltan [21]) as our hypergraph partitioner with an imbalance tolerance of 3%. For our nested dissection method, we derived our vertex separators via hypergraph partitioning. We first apply hypergraph partitioning, then compute a vertex cover of the resulting boundary graph. This gives a small and balanced vertex separator for the original graph. This choice also enables a fair comparison across methods since the code base is the same for both graph and hypergraph-oriented algorithms.

All our experiments were run on a linux 64-bit workstation with four dual-core Intel Xeon processors (though only one core was used per run) and 16 GB RAM. Additional experiments can be found in [22].

A. Communication Metrics

We partition the eight symmetric matrices shown in Table I using 1D, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, and 256 parts. The average communication volumes are shown in Table II. 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 (scaled volume less than 1 indicates an improvement over the 1D method). The 1D column and nested dissection partition methods had problems obtaining a balanced partition for some of the partitionings, unlike the fine-grain method, which by means of its flexibility can always realize load balance.

We see that our nested dissection method performs consistently better than 1D. When compared to the fine-graph method, we see for most partitionings that the nested dissection method yielded similar or better results for six of the eight matrices. The nested dissection only performed significantly worse for **cage10** and **asic680ks**.

Communication volume is not the only important metric when evaluating the quality of a sparse matrix partitioning in terms of parallel SpMV. The number of messages communicated can be as important or more important if the volume is low or the latency is high. 1D partitioning of the sparse matrix yields a parallel matrix-vector algorithm with only one phase

TABLE II
AVERAGE (20 RUNS) COMMUNICATION VOLUME (IN WORDS) AND AVERAGE MESSAGES SENT (SAME AS RECEIVED) PER PROCESS FOR K-WAY PARTITIONING OF SYMMETRIC MATRICES. * - DESIGNATES PARTITIONS THAT DID NOT MEET THE 3% LOAD BALANCE TOLERANCE.

		1D		fine-grain		nested diss.	
		total		scaled		scaled	
Name	k	volume	msgs	vol.	msgs	vol.	msgs
cage10	4	5.38e3	**3.0**	**0.76**	6.0	0.82	3.5
	16	1.29e4	**11.6**	**0.69**	21.4	0.89	15.4
	64	2.35e4	**20.0**	**0.70**	35.1	0.98	27.7
	256	4.08e4	**24.2**	**0.72**	36.4	1.03	33.6
finan512	4	2.96e2	**2.0**	0.88	4.0	**0.78**	**2.0**
	16	1.21e3	**2.0**	0.84	4.0	**0.77**	**2.0**
	64	9.99e3	**2.1**	0.86	9.3	**0.81**	2.3
	256	3.90e4	**6.0**	**0.68**	17.4	0.77	7.2
bcsstk32	4	2.11e3	**2.7**	**0.76**	4.7	0.84	3.1
	16	7.89e3	**4.5**	**0.80**	7.7	0.86	5.3
	64	1.99e4	**6.1**	0.94	10.9	**0.91**	7.6
	256	4.64e4	**6.7**	1.00	11.7	**0.94**	8.6
bcsstk30	4	1.79e3	**1.6**	1.08	3.3	**0.78**	**1.6**
	16	8.62e3	**3.6**	1.13	7.6	**0.83**	4.2
	64	2.33e4	**5.9**	1.10	12.2	**0.90**	7.6
	256	5.61e4	**7.3**	1.03	16.0	**0.98**	9.9
c-73	4	4.23e4	**2.6**	0.04	2.8	**0.03**	3.0
	16	9.99e4	**8.3**	**0.05**	11.4	**0.05**	9.0
	64	2.06e5*	**21.5**	0.07	26.1	**0.06**	22.9
	256	1.71e5*	**24.1***	**0.23**	48.0	0.25*	28.4*
asic680ks	4	3.56e3	**3.0**	**0.51**	4.6	0.61	3.5
	16	1.00e4	**10.8**	**0.46**	19.2	0.61	13.7
	64	2.18e4	**19.9**	**0.44**	33.5	0.59	25.4
	256	3.89e4	**21.3**	**0.49**	31.5	0.61	26.7
pkustk04	4	6.61e3	**1.6**	0.63	4.6	**0.53**	1.7
	16	2.76e4	**3.5**	**0.49**	13.5	0.60	4.4
	64	7.53e4	**8.3**	**0.42**	23.0	0.62	11.5
	256	1.62e5	**15.3**	**0.43**	26.0	0.56	20.6
gupta3	4	3.01e4	**2.9**	0.29	5.9	**0.19**	3.5
	16	1.03e5	**12.0**	0.31	23.9	**0.21**	15.8
	64	3.33e5	**39.4**	0.27	63.2	**0.19**	46.5
	256	1.11e6*	127.7*	**0.17**	**92.6**	0.18*	132.0*

of communication. This results in a low number of messages in comparison to 2D partitioning methods such as the fine-grain method. Table II shows the average number of messages sent (same as average number of messages received) per part during the resulting SpMV for the 1D column, fine-grain, and nested dissection partitionings of the eight symmetric matrices. As expected, the 1D method has consistently the lowest average number of messages sent per part of the three methods. The number of messages resulting from the nested dissection partitions is significantly lower than that of the fine-grain method for most of the partitionings of the matrices.

B. Partitioning Time

Another important factor in the effectiveness of a partitioning method is how much runtime is required to obtain the partition. The runtimes of partitioning the matrices in Table I with the different methods are shown in Figure 6. We first scale the runtimes for each matrix relative to the one-dimensional column partitioning and $k = 4$ runtime. Each line in the figures is the geometric average of the scaled runtimes across the matrices in the set.

As expected, the 1D method requires less time to partition the sparse matrices than the 2D methods. In particular, the 1D method is much faster than the fine-grain hypergraph method

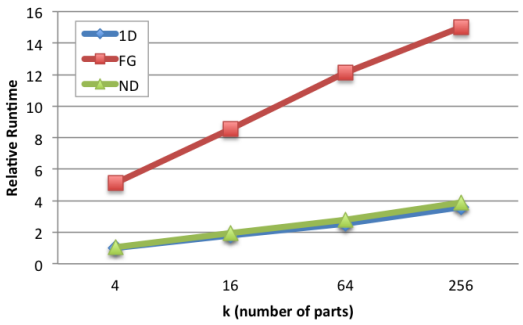


Fig. 6. Geometric average of relative runtimes for matrices in Table I.

but only slightly faster than the nested dissection method. The nested dissection method is also significantly faster than the fine-grain hypergraph method. For instance, for the nested dissection partitionings of the larger matrices, we see up to a 96% decrease in the runtime over the fine-grain partitionings.

V. Conclusions

We presented a new graph-oriented approach to sparse matrix partitioning for sparse matrix-vector multiplication. We also presented a nested dissection partitioning algorithm that approximately solves our communication graph model. Although nested dissection has been previously used in several contexts, this is the first such algorithm and analysis specifically for sparse matrix-vector multiplication. Our method is fairly simple to implement and provides a nice compromise: low communication volume, low message count, and it is relatively fast to compute. 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 at a great reduction in the runtime.

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), if good edge balance can be enforced. Second, one can likely partition the vertices and edges in and around the separators in a better way. This becomes increasingly more important as the number of parts grows.

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. As network data continues to grow in scale, partitioning algorithms such as the one presented here will be crucial to obtaining good parallel performance for the resulting graph and sparse matrix computations, helping to overcome the challenges of irregular communication and lack of data locality associated with these challenging sparse computations.

Acknowledgment

We thank Rob Bisseling, Umit Catalyurek, Michael Heath, and Bruce Hendrickson for helpful discussions. We thank Florin Dobrian and Mahantesh Halappanavar for providing the MatchBox matching code used to produce vertex separators. This work was funded by the US DOE Office of Science through the CSCAPES Institute and the SciDAC program.

References

[1] B. A. Miller, N. Arcolano, M. S. Beard, J. Kepner, M. C. Schmidt, N. T. Bliss, and P. J. Wolfe, "A scalable signal processing architecture for massive graph analysis," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2012, pp. 5329–5332.

[2] Ü. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Dist. Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[3] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. on Scientific Computing*, vol. 26, no. 6, pp. 1837–1859, 2004.

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

[5] M. Garey, D. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, pp. 237–267, 1976.

[6] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York, NY: John Wiley & Sons, 1990.

[7] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," in *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*. SIAM, 1993, pp. 711–718.

[8] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proc. Supercomputing '95*. ACM, December 1995.

[9] G. Karypis and V. Kumar, "Multilevel graph partition and sparse matrix ordering," in *Intl. Conf. Parallel Processing*, 1995, pp. 113–122.

[10] 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*, April 2001.

[11] ——, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proc. Supercomputing 2001*. ACM, 2001.

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

[13] 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*, vol. 27, pp. 129–156, 1989.

[14] E. G. Boman, "A nested dissection approach to sparse matrix partitioning," *Proc. Applied Math. and Mechanics*, vol. 7, no. 1, pp. 1 010 803–1 010 804, 2007, presented at ICIAM'07, Zürich, Switzerland, July 2007.

[15] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.

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

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

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

[19] C. Chevalier and F. Pellegrini, "PT-SCOTCH: A tool for efficient parallel graph ordering," *Parallel Comp.*, vol. 34, no. 6–8, pp. 318–331, 2007.

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

[21] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, "Zoltan data management services for parallel dynamic applications," *Computing in Science and Engineering*, vol. 4, no. 2, pp. 90–97, 2002.

[22] M. M. Wolf, "Hypergraph-Based Combinatorial Optimization of Matrix-Vector Multiplication," Ph.D. dissertation, University of Illinois at Urbana-Champaign, July 2009.