Optimizing Parallel Sparse Matrix-Vector Multiplication by Corner Partitioning

Michael M. Wolf[1,2], Erik G. Boman[2], and Bruce A. Hendrickson[3]

[1] Dept. of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
[2] Scalable Algorithms Dept, Sandia National Laboratories, NM, USA **
[3] Computer Science and Informatics Dept, Sandia National Laboratories, NM, USA
mmwolf@illinois.edu,egboman@sandia.gov,bahendr@sandia.gov

Abstract. The multiplication of a vector by a sparse matrix is an important kernel in scientific computing. We study how to optimize the performance of this operation in parallel by reducing communication. We review existing approaches and present a new two-dimensional partitioning method for symmetric matrices, called *corner* partitioning. Our method is simple and can be implemented using existing software for hypergraph partitioning. Experimental results show our method often produces better quality than traditional one-dimensional partitioning methods and is competitive with two-dimensional methods. It is also fast to compute. Finally, we propose a graph model for an ordering problem to further optimize our approach. This leads to a graph algorithm based on vertex cover or vertex separator.

Key words: parallel algorithms, combinatorial scientific computing, partitioning, sparse matrix-vector multiplication

1 Parallel Matrix-Vector Multiplication

Sparse matrix-vector multiplication is a common kernel in many computations, e.g., iterative solvers for linear systems of equations, eigensolvers, 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 compute nodes 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 processor speeds increase much more rapidly than memory, we expect memory latency and bandwidth to grow in importance.

We focus on minimizing the total communication volume while keeping the computation balanced across compute nodes. Sparse matrix-vector multiplication $y = Ax$ is usually parallelized such that the compute node that owns element

** Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin company, for the US Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

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 compute node; otherwise communication is required. In general, the following four steps are performed [4, 9]:

1) **Expand:** Send entries x_j to compute nodes 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 compute nodes.
4) **Sum:** Sum up the partial y values.

Usually steps 3 and 4 are combined to reduce communication. Clearly, the communication will depend on the data distribution, and thus partitioning to obtain this distribution is important for obtaining high parallel efficiency.

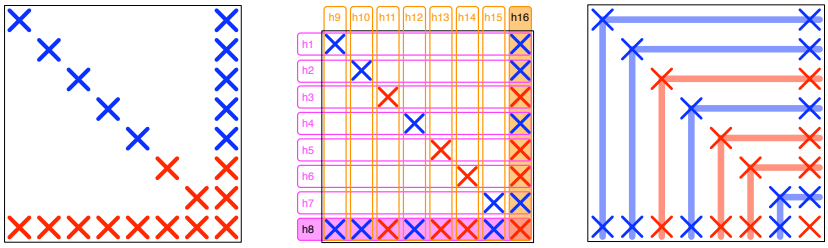
In this paper, we review current one- and two-dimensional data distributions (section 2), and introduce a new two-dimensional partitioning method (section 3). This "corner" method focuses on partitioning matrices with symmetric nonzero structures although in section 6 we describe an extension for partitioning square matrices with nonsymmetric nonzero structures. Our objective in developing the "corner" method is to obtain a method that yields partitionings of better quality than 1-D partitioning methods and similar quality to the best 2-D methods, while being faster to compute than the 2-D methods.

2 One- and Two-Dimensional Partitioning

In one-dimensional sparse matrix partitioning, each compute node (partition) is assigned nonzeros belonging to a set of rows (1-D row partitioning) or a set of columns (1-D column partitioning). The rows (or columns) in a partition do not have to be consecutive. Catalyurek and Aykanat [4] showed that partitioning matrix rows to minimize communication can be accurately modeled as *hypergraph partitioning*. In their row-net model, each column corresponds to a vertex and each row to a hyperedge. One-dimensional partitioning works well for many problems, and many parallel applications distribute matrices in a one-dimensional manner. However, for particular problems, one-dimensional partitioning is potentially disastrous in terms of the communication volume. The "arrowhead" matrix shown in Figure 1(a) is an example for which one-dimensional partitioning is inadequate. For the bisection (number of partitions, k, is 2) case shown in the figure, any load-balanced one-dimensional partitioning will yield a communication volume of approximately $\frac{3}{4}n$ for the matrix-vector product. This is far from the minimum communication volume of 2 for this problem and it is unacceptable for the communication volume to scale as n for this matrix. Thus, we need more flexible partitioning than traditional one-dimensional partitioning.

Two-dimensional partitioning is a more flexible alternative to one-dimensional partitioning, in which no specific partition is assigned to a given row or column. Rather, we have to specify the partition for particular sets of nonzeros. In the most flexible method each nonzero is assigned a partition independently. This

problem can be modeled as a *fine-grain* hypergraph[5], where each nonzero is represented by a vertex in the hypergraph. Each row is represented by a hyperedge in the hypergraph (h1-h8 in Figure 1(b)). Likewise, each column is represented by a hyperedge in the hypergraph (h9-h16 in Figure 1(b)). The vertices are partitioned into k equal sets ($k = 2$ in Figure 1(b)) such that the sum of the costs over the cut hyperedges is minimized. A cut hyperedge contains vertices assigned to different partitions. The cost associated with a cut hyperedge is the number of different partitions, minus one, assigned to its vertices. The communication volume is equivalent to this hyperedge cut metric. Catalyurek and Aykanat [5] proved that this fine-grain hypergraph model yields a minimum volume partitioning when optimally solved. In Figure 1(b), we see the fine-graph hypergraph partitioning of the 8×8 arrowhead matrix. The resulting communication volume is 2, a significant improvement over the communication volume of $\frac{3}{4}n = 6$ from the optimal one-dimensional partitioning.



(a) 1-D, volume=6 (b) 2-D, volume=2 (c) "Corners"

Fig. 1. Arrowhead matrix. (a) 1-D row-wise bisection. (b) 2-D fine-grain hypergraph bisection. Cut hyperedges are shaded. (c) "Corners" in 2-D fine-grain partitioning.

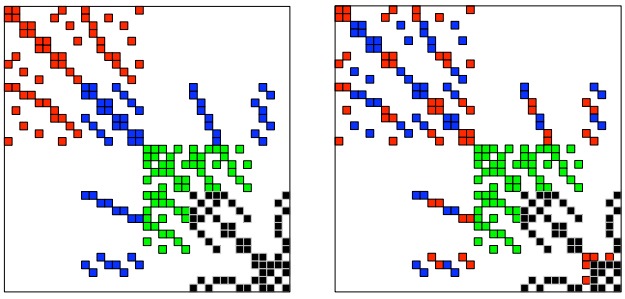
Like 1-D hypergraph partitioning, solving the fine-grain hypergraph model optimally is NP-hard but there are heuristics that work well in practice in near-linear time. Unfortunately, the resulting fine-grain hypergraph problem is a larger NP-hard problem than 1-D partitioning and thus, may be too expensive to solve for large matrices. However, the fine-grain hypergraph method, in general, yields low communication volume partitionings. Thus, when developing new two-dimensional methods, our goal is to obtain similar quality partitionings to the fine-grain method with methods that are similar to one-dimensional partitioning methods in runtime. The recent 2-D Mondriaan method [9] is based on recursive 1-D partitioning. It is more restrictive than the fine-grain hypergraph method but is fast in comparison. A symmetric variation of Mondriaan is described in [9] but the limited results were inconclusive. We propose a new method that combines the simplicity of 1-D partitioning with the symmetric partitioning approach.

3 Corner Partitioning for Symmetric Matrices

An examination of the minimum cut fine-grain hypergraph partitioning in Figure 1(b) suggests a new partitioning method. We see that each partition consists of a set of "corners" (more easily seen in Figure 1(c)), which are basically one-dimensional partitions reflected across the diagonal. Using these "corners", the hope is that we could reproduce an optimal fine-grain partitioning using a less costly one-dimensional partitioning method for certain matrices. Our corner partitioning algorithm for a symmetric matrix A has the following steps:

1. Let L be the lower triangular part of A.
2. Partition L along columns (or rows). This assigns lower triangular nonzeros to compute nodes. A_{ij} is assigned to the partition of L_{ij} for $i \geq j$.
3. Partition the remaining nonzeros in A (those above the diagonal) to obtain a symmetric partition. A_{ij} is assigned to the partition of L_{ji} for $i < j$.

This partitioning algorithm is a 1-D partitioning of the triangular part which is then reflected around the diagonal, giving the corner structure. We remark that this is a special case of *symmetric partitioning* as described in [1]. Figure 3 illustrates the method on a test matrix with 36 rows (cage5 matrix [7]). Version (a) partitions rows/columns consecutively, while (b) is the result of hypergraph partitioning. For the hypergraph model used in (b), there exists a vertex v_i for every column i in the lower triangular part of A. There is a hyperedge for every row in the lower triangular part of A, containing vertices that correspond to columns for which there are nonzeros in this row. Variation (b) will always be at least as good as (a) since (a) is contained as a special case. Thus we only show (a) as an illustration and do not consider the method any further.



(a) Consecutive (b) Hypergraph

Fig. 2. cage5 matrix partitioned with corner method.

We now show that the communication volume for corner partitioning is closely related to that of 1-D partitioning.

Lemma 1. *The communication volume for matrix-vector multiplication with a corner distribution is exactly twice that of the corresponding 1-D distribution in the lower (upper) half.*

In this proof, we assume the lower triangular part is partitioned by columns. In the expand phase, vector entries are communicated to processors that need them based on the columns in the matrix. Similarly, the communication in the fold phase is "along rows". In the symmetric case, the communication volumes are the same in each phase. Consider the row-wise phase of communication. By design, there is no communication caused by the upper triangular part of the matrix since rows to the right of the diagonal are wholly owned by a single compute node (partition). Thus, the only communication is caused by the lower triangular part, which has a 1-D distribution. Similarly all the communication in the column-wise phase of communication is caused by the upper triangular part. It follows that the upper and lower triangular parts have the same communication volumes although they result from different phases.

Theorem 1. *Hypergraph partitioning is the optimal partitioning method for corner partitioning.*

This result follows directly from the above lemma and the optimality of the hypergraph model for 1-D distribution.

4 Results

We used PaToH [4] for hypergraph partitioning in order to implement one-dimensional column, two-dimensional corner, and two-dimensional fine-grain partitioning. We studied the partitioning of five symmetric matrices [7, 8] (first 5 matrices in Table 1) used as a benchmark set in [9] for these three methods. We partitioned these matrices into 4, 16, 64, and 256 partitions. The resulting communication volumes are reported in Table 2. In general, we see the corner method yielded significantly higher quality partitionings than the one-dimensional method. With the exception of the **lap200** matrix and a few $k = 256$ and **bcsstk32** data points, the corner method yielded similar or slightly better partitionings than the fine-grain hypergraph method. In Table 2 (columns 4, 7, 10), we see that the corner method was much faster to compute than the fine-grain hypergraph method, and also faster than 1-D.

4.1 Results for matrices where 1-D partitioning is inadequate

We have observed that 1-D partitioning performs relatively poorly on some matrices, so these are of particular interest. We selected three matrices where fine-grain partitioning gave substantially lower communication volume than 1-D, and studied corner partitioning. The corner results varied wildly, and for one problem (c-73) were in fact worse than standard 1-D partitioning. A closer look revealed this was due to the orientation of the "corner" being poorly aligned with the

Table 1. Matrix Info.

Name	rows/cols	nonzeros	application
cage10	11397	150645	DNA electrophoresis
lap200	40000	200000	2-D Laplacian
finan512	74752	596992	portfolio optimization
bcsstk32	44609	2014701	structural engineering
bcsstk30	28924	2043492	structural engineering
asic680ks	682712	2329176	circuit simulation
c-73	169422	1279274	non-linear optimization
Dubcova2	65025	1030225	PDE Solver
c-big	345241	2341011	non-linear optimization

matrix structure. A simple alternative was to partition the lower triangular part by rows, not by columns. The results are shown in Table 3. We see that corner partitioning based on either rows or columns always outperforms 1-D; thus we recommend to try both and pick the best. In the next section, we introduce an extension to this method that can improve upon this.

5 Reordering Strategies

The observation that corner partitioning based on partitioning along rows or columns can give very different results, has led us to consider reordering the matrix. Let P be a permutation matrix. We can then form $\hat{A} = PAP^T$ and partition $\hat{A}$. To compute $y = Ax$, we can compute $y = P^T\hat{A}Px$. Simpler yet, let $\hat{y} = Py$ and $\hat{x} = Px$. Then $\hat{y} = \hat{A}\hat{x}$. In many applications, repeated multiplications are needed, and then x and y could be permuted (redistributed) only once, so we ignore the cost of doing that step. The problem then is to find a good permutation P.

A simple case is when P is the reverse permutation operator, that is, Px is the same as x but with the entries in reverse order. In this case, performing a column-based corner partition of PAP^T is the same as a row-based corner partition of A. We observed in section 4 that this could make a big difference. This indicates that ordering is important and that reordering can be potentially exploited to decrease the communication volume for a corner partitioning.

Note that reordering is irrelevant for one-dimensional partitioning since the related graph and hypergraph models are invariant to ordering.

5.1 Graph model for optimal ordering for corner partitioning

Although we initially used a hypergraph model for corner partitioning, in this section we found a graph model to be more useful. We can model the communication volume for the corner partitioning/ordering problem using a graph $G(V, E)$, where vector entries are represented by a set of vertices V and the off-diagonal nonzeros are represented by edges in the graph such that if $a_{i,j} \neq 0$ for $i \neq j$

Table 2. Communication volume and runtimes (in s) for k-way partitioning of symmetric matrices using different partitioning methods. Improvements are relative to 1-D.

		1-D hypergraph		fine-grain hypergraph			corner column		
Name	k	volume	runtime	volume	% impr.	runtime	volume	% impr.	runtime
cage10	4	5379.0	13.1	**4063.7**	24.5	28.3	4089.3	24.0	**5.8**
	16	12874.5	25.0	**8865.5**	31.1	46.7	8920.9	30.7	**12.0**
	64	23463.3	41.3	**16334.7**	30.4	68.9	17164.0	26.8	**20.8**
	256	40830.9	66.7	**29239.0**	28.4	101.9	32138.0	21.3	**38.2**
lap200	4	**1535.1**	7.9	1538.5	-0.2	19.0	1640.0	-6.8	**6.0**
	16	**3013.9**	15.2	3017.9	-0.1	30.2	3336.5	-10.7	**10.2**
	64	5813.0	25.2	**5786.4**	0.5	44.4	6656.4	-14.5	**18.4**
	256	11271.8	51.4	**11061.4**	1.9	71.9	13342.8	-18.4	**33.6**
finan512	4	295.7	23.8	261.2	11.7	82.8	**215.0**	27.3	**13.7**
	16	1216.7	48.6	1027.4	15.6	128.3	**845.0**	30.5	**26.0**
	64	9986.0	90.9	8624.6	13.6	185.4	**8135.2**	18.5	**50.1**
	256	38985.4	142.5	**26471.6**	32.1	253.0	42248.6	-8.4	**91.4**
bcsstk32	4	2111.9	58.5	**1611.4**	23.7	470.7	1751.3	17.1	**25.1**
	16	7893.1	102.0	**6330.8**	19.8	718.4	7220.0	8.5	**42.7**
	64	19905.4	152.7	**18673.1**	6.2	922.6	19616.4	1.5	**65.4**
	256	**46399.0**	215.2	46469.7	-0.2	1133.1	47695.8	-2.8	**101.1**
bcsstk30	4	1794.4	76.0	1935.7	-7.9	688.6	**1531.0**	14.7	**30.4**
	16	8624.7	139.7	9774.8	-13.3	1076.5	**7232.2**	16.1	**53.2**
	64	23308.0	205.7	25677.2	-10.2	1381.1	**20351.4**	12.7	**83.1**
	256	56100.4	262.4	57844.8	-3.1	1639.8	**50689.4**	9.6	**110.7**

then $(i, j) \in E$. Each vertex is given both a position, which refers to the position of the corresponding diagonal elements after reordering, and a set number, corresponding to the compute node that owns the matrix diagonal entry (and also the corresponding vector entry). Let π_i be the position of vertex v_i and let s_i be the set of v_i. Again, we analyze the column-based version of the corner method. Consider row (column) i. Clearly, no communication is required if all nonzeros in row i are in the same set. By design, all nonzeros a_{ij} with $j > i$ will be in the same set, so we only need to examine the nonzeros where $j < i$. Thus, it follows that a vertex v_i is involved in communication if and only if it is connected to at least one additional vertex v_j such that the v_j is assigned a position π_j less than π_i. As a consequence, we have the following theorem for the bisection case:

Theorem 2. *The communication volume of the matrix-vector product resulting from corner partitioning and bisection ($k = 2$) is equal to* $\mathrm{Vol} = 2\sum_{v_i \in V} c_i$, *where*

$$c_i = \begin{cases} 1, \exists v_j : \pi_j < \pi_i, (v_i, v_j) \in E, s_i \neq s_j \\ 0, \quad \text{otherwise} \end{cases}.$$

First, we focus on a simplified version of the partitioning/ordering problem in which we will partition first and then reorder the rows/columns symmetrically to obtain minimal communication volume for the particular partitioning of the nonzeros. With a fixed partitioning we know that the s_i in the graph model do

Table 3. Communication volume for k-way partitioning of symmetric matrices using different partitioning methods. Asterisks designate runs that did not achieve the desired 3% load-balance tolerance.

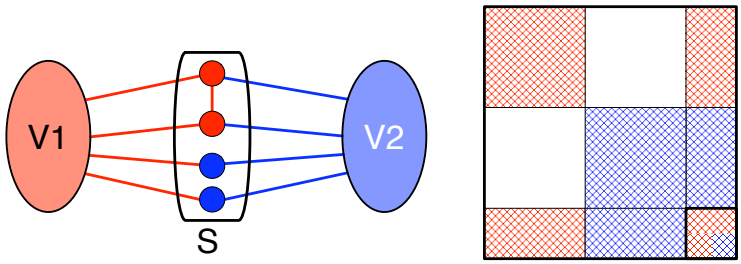
		1-D hyp.	fine-grain hyp.		corner col		corner row	
Name	k	volume	volume	% impr.	volume	% impr.	volume	% impr.
asic680ks	4	3560.4	**1813.3**	49.1	2214.2	37.8	4015.4	-12.8
	16	9998.5	**4634.0**	53.7	5562.8	44.4	11343.9	-13.5
	64	21785.8	**9554.9**	56.1	11147.3	48.8	26379.8	-21.1
	256	38869.4	**19128.1**	50.8	21024.0	45.9	47509.9	-22.2
c-73	4	42363.0	**1611.8**	96.2	90410.5	-113.4	1656.4	96.1
	16	98617.7	5058.1	94.9	255484.8	-159.1	**5009.3**	94.9
	64	219429.0*	15067.8	93.1	450166.1*	-105.2	**14596.6**	93.3
	256	168176.2*	38566.7	77.1	453705.1*	-169.8	**38219.3**	77.3
Dubcova2	4	1825.3	**1460.7**	20.0	3091.1	-69.3	1724.3	5.5
	16	5613.7	**4508.0**	19.7	9517.7	-69.5	5456.3	2.8
	64	13492.7	**10846.1**	19.6	23225.2	-72.1	13208.1	2.1
	256	30110.9	**23847.0**	20.8	26140.9	13.2	30354.7	-0.8
c-big	4	34360.8	**14903.3**	56.6	25443.9	26.0	40034.2	-16.5
	16	63139.9	**32514.5**	48.5	55718.1	11.8	66792.1	-5.8
	64	95836.7	**54998.6**	42.6	94232.3	1.7	95169.6	0.7
	256	135631.3*	**84249.8**	37.9	142318.1*	-4.9	121532.8	10.4

not change during the reordering process. As explained above, a vertex v_i in the graph model contributes to the communication volume only if $\exists v_j : \pi_j < \pi_i, (v_i, v_j) \in E, s_i \neq s_j$. Thus, we want to minimize the number of vertices with this property. This is equivalent to finding a minimum vertex cover of the edges that contain vertices in different partitions (edges in the bipartite graph on the boundary between the two partitions) and order these cover vertices last. We can find this minimum vertex cover of the bipartite graph in low polynomial time and thus efficiently find an optimal ordering for our fixed corner partitioning.

It is important to note that the cover vertices for the bipartite graph form a vertex separator for the original graph. This indicates that an alternative algorithm is to find a small balanced vertex separator directly. A more general partitioning method based on that idea was proposed in [2] and we are pursuing this method further [3]. However, this separator approach can also be used in a more specific manner with the corner method to further reduce communication resulting from the method with an improved partitioning/ordering.

The first step in this improved partitioning/ordering algorithm is to find a small balanced vertex separator S for the graph. We can obtain this vertex separator using any vertex separator algorithm (including the method outlined above). Using this separator, we can divide the vertices into three disjoint subsets (V_1, V_2, S) as shown in Figure 3(a). The vertices in subsets V_1 and V_2 are assigned to different partitions in the bisection. We have flexibility in how we assign partitions to the separator vertices S for bisection since the communication volume is independent of this assignment. One possibility is to assign the partitions of these separator vertices to maintain load-balance. Next, we assign

positions to the vertices based on the subsets (V_1,V_2,S) in order to minimize communication. In particular, we assign the last $|S|$ positions to the vertices in S so that these separator vertices are positioned after V_1 and V_2. Although it does not affect the communication volume, we position the vertices in V_1 before the vertices in V_2 for visualization purposes. Using these positions assigned to the vertices, the matrix can be permuted. From the partitioning of the vertices, we can obtain a corner partitioning for this permuted matrix. Figure 3(b) illustrates the resulting nonzero structure from the corner partitioning/ordering of a matrix. Note that the V_1 "corners" are ordered first, followed by the V_2 "corners", and finally the S "corners." We can recursively apply this method to V_1 and V_2 to obtain a larger number of partitions.



(a) Partitioned graph (b) Resulting Matrix

Fig. 3. Graph partitioned using vertex separator. Corresponding corner partitioned matrix.

6 Nonsymmetric Case

In this paper we focus on the symmetric case, which is common in many applications. However, one can generalize the corner method in several ways. Here we describe an extension to the square but nonsymmetric case. The right model is again a hypergraph model. Let $H = (V, E)$ be a hypergraph with one vertex for each "corner." We should think of vertex i as representing the partial row to the right of and including A_{ii} and the partial column below A_{ii} (a "corner"). For simplicity, assume all diagonal entries are nonzero. Consider the potential communication requirements along row i and column i. Above the diagonal, this is determined by the row-wise decomposition in the upper triangular part, while below the diagonal, it is determined by the column-wise decomposition of the lower triangular part. In the symmetric case, the lower and upper triangular part were the same (reflected around the diagonal) but in the nonsymmetric case we need treat these separately. The solution is simple. We build a hypergraph with

one hyperedge for each row of the lower triangular part of the matrix and one for each column in the upper triangular part of the matrix. This hypergraph has twice as many hyperedges as in the symmetric case, so is more expensive to partition, but it is an accurate model of communication volume.

There is little reason to believe a symmetric partition method like the corner method would work well for highly nonsymmetric matrices, though we conjecture it will do reasonably well for near-symmetric matrices. We have not studied this empirically. An alternate approach to the nonsymmetric case is to use a bipartite graph. Then reordering can be used, as in the previous section.

7 Conclusions

We have presented a new sparse matrix data distribution, the *corner* distribution, suitable for symmetric matrices. The basic version can be computed efficiently using existing hypergraph partitioning software (even in parallel, using the Zoltan toolkit). The resulting communication volume is generally lower than for the standard 1-D partitioning, and in several cases also lower than for the 2-D fine-grain method. We also presented a graph model for optimal corner partitioning. This requires vertex cover or vertex separator, both of which can be computed efficiently. An empirical study of this modified approach is left as future work.

Acknowledgments

This work was funded by the U.S. Dept. of Energy's Office of Science through the CSCAPES SciDAC institute.

References

1. R. H. Bisseling. *Parallel Scientific Computing: A structured approach using BSP and MPI.* Oxford University Press, 2004.
2. Erik G. Boman. A nested dissection approach to sparse matrix partitioning. In *Proc. Appl. Math. and Mech.*, 2008, to appear. Presented at ICIAM'07, July 2007.
3. Erik G. Boman and Michael M. Wolf. A nested dissection approach to sparse matrix partitioning. In preparation, 2008.
4. 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.
5. 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.
6. Timothy A. Davis. The University of Florida Sparse Matrix Collection, 1994. http://www.cise.ufl.edu/research/sparse/matrices/.
7. Iain S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Mathematical Software*, 15:1–14, 1989.
8. Brendan Vastenhouw and Rob H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.