Partitioning Sparse Rectangular Matrices for Parallel Computations of Ax and $A^T v$[*]

Bruce Hendrickson[1] and Tamara G. Kolda[2]

[1] Parallel Computing Sciences Department, Sandia National Labs, Albuquerque, NM 87185–1110. `bah@cs.sandia.gov`.
[2] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831–6367. `kolda@msr.epm.ornl.gov`.

Abstract. This paper addresses the problem of partitioning the nonzeros of sparse nonsymmetric and nonsquare matrices in order to efficiently compute parallel matrix-vector and matrix-transpose-vector multiplies. Our goal is to balance the work per processor while keeping communications costs low. Although the symmetric partitioning problem has been well-studied, the nonsymmetric and rectangular cases have received scant attention. We show that this problem can be described as a partitioning problem on a bipartite graph. We then describe how to use (modified) multilevel methods to partition these graphs and how to implement the matrix multiplies in parallel to take advantage of the partitioning. Finally, we compare various multilevel and other partitioning strategies on matrices from different applications. The multilevel methods are shown to be best.

1 Introduction

In many parallel algorithms, we require numerous matrix-vector and matrix-transpose-vector multiplies with sparse matrices. Partitioning is used to distribute the nonzeros of the sparse matrix so that the work per processor is balanced and the communication costs are low. Of particular interest to us is the case when the matrix is square nonsymmetric or rectangular; we refer to both cases as *rectangular*. Specifically, given a rectangular matrix A, we find permutations P and Q so that the nonzero values of PAQ are clustered in the diagonal blocks as illustrated in Figure 1. As we will show in Sect. 2, a nearly block diagonal structure helps to reduce the memory requirements and communication cost in the matrix-vector products. Furthermore, we require the block rows and block columns to each have about the same number of nonzeros; this corresponds to balancing the floating point operations per processor.

The need to perform repeated matrix-vector multiplies with the same rectangular matrix (and its transpose) arises in numerous linear algebra algorithms.

[*] This work was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy, under contracts DE-AC05-96OR22464 and DE-AL04-94AL85000 with Lockheed Martin Energy Research Corporation.

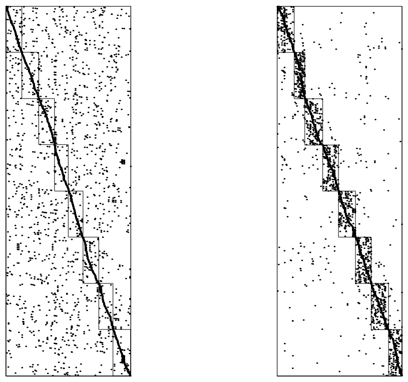

Fig. 1. Before and after partitioning.

Important examples include applying QMR to nonsymmetric linear systems [3], using LSQR to solve least squares problems [14], solving the normal equations that arise in interior point methods using CG [15], or computing the truncated SVD of hypertext matrices in information retrieval [2].

Although matrix partitioning has been well-studied in the symmetric case, little work has been done in the rectangular case [13]. The symmetric problem is commonly phrased in terms of partitioning graphs. We will show that the rectangular problem can be conveniently phrased in terms of partitioning *bipartite* graphs. We will extend the work of Berry, Hendrickson, and Raghavan [2] and Kolda [13] for partitioning rectangular matrices by incorporating multilevel schemes which are already popular in the symmetric case [1, 8, 10, 11]. Multilevel methods, described in Sect. 3, have three phases: coarsening, base-level partitioning, and un-coarsening with refinement. In Sect. 4, we will compare the (modified) multilevel methods with various refinements to non-multilevel methods on a set of test matrices.

Further information can be found in Hendrickson and Kolda [6], an extension of this research.

2 Parallel Multiplies

We propose the following parallel implementations for the matrix-vector and matrix-transpose-vector multiplications. Suppose that we have p processors. We partition A into a block $p \times p$ matrix,

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix} ,$$

so that most of the nonzeros are in the diagonal blocks. Here block (i, j) is of size $m_i \times n_j$ where $\sum_i m_i = m$ and $\sum_j n_j = n$.

Matrix-Vector Multiply (Block Row). We do the following on each processor to compute $y = Ax$:

1. Let i denote the processor id. This processor owns the ith block row of A, that is, $\begin{bmatrix} A_{i1} \; A_{i2} \cdots A_{ip} \end{bmatrix}$, and x_i, the ith block of x of length n_i.
2. Send a message to each processor $j \neq i$ for which $A_{ji} \neq 0$. This message contains only those elements of x_i corresponding to nonzero *columns* in A_{ji}.
3. While waiting to receive messages, the processor computes the contribution from the diagonal matrix block, $y_i^{(i)} = A_{ii} x_i$. The block A_{ii}, while still sparse, may be dense enough to improve data locality.
4. Then, for each $j \neq i$ such that A_{ij} is nonzero, a message is received containing a sparse vector $\bar{x}_j$ that only has the elements of x_j corresponding to nonzero columns in A_{ij}, and $y_i^{(j)} = A_{ij} \bar{x}_i$, is computed. (We assume that processor i already knows which elements to expect from processor j.)
5. Finally, the ith block of the product y is computed via the sum $y_i = \sum_j y_i^{(j)}$. Block y_i is of size m_i.

Matrix-Transpose-Vector Multiply (Block Row). To compute $z = A^T v$, each processor does the following:

1. Let i denote the processor id. This processor owns v_i, the ith block of v of size m_i, and the ith block row of A.
2. Compute $z_j^{(i)} = A_{ij}^T v_i$, for each $j \neq i$ for which $A_{ij} \neq 0$. Observe that the number of nonzeros in $z_j^{(i)}$ is equal to the number of nonzero rows in A_{ij}^T, i.e., the number of nonzero columns in A_{ij}. Send the nonzero[1] elements of $z_j^{(i)}$ to processor j.
3. While waiting to receive messages from the other processors, compute the diagonal block contribution $z_i^{(i)} = A_{ii}^T v_i$.
4. From each processor j such that $A_{ji} \neq 0$, receive $\bar{z}_i^{(j)}$ which contains only the nonzero elements of $z_i^{(j)}$. (Again, we assume that processor i already knows which elements to expect from processor j.)
5. Compute the ith component of the product, $z_i = z_i^{(i)} + \sum_{j \neq i} \bar{z}_i^{(j)}$. Block z_i is of size n_i.

Block column algorithms are analogous to those given for the block row layout. Observe that sparse off-diagonal blocks result in less message volume. See Hendrickson and Kolda [6] for more details on these algorithms.

[1] Here we mean any elements that are guaranteed to be zero by the structure of A_{ij}. Elements that are zero by cancellation are still communicated.

3 Multilevel Partitioning of Rectangular Matrices

A rectangular $m \times n$ matrix $A = [a_{ij}]$ corresponds to an undirected *bipartite* graph $G = (R, C, E)$ with $R = \{r_1, \ldots, r_m\}$, $C = \{c_1, \ldots, c_n\}$ and $(r_i, c_j) \in E$ iff $a_{ij} \neq 0$ (see Fig. 2). The weight of each row vertex is the number of nonzeros in

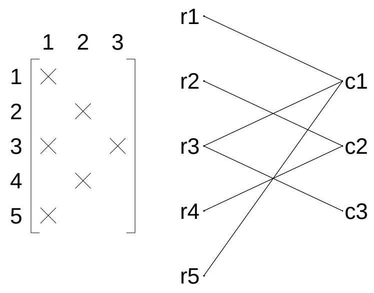

Fig. 2. Bipartite graph representation of a matrix.

the corresponding row; the column vertices are unweighted. The weight of each edge is initially set to one. We now wish to divide $R \cup C$ into p sets in such a way that the total row weight per set is balanced, and the total number of edges crossing between sets is kept small. The first criterion ensures load balance in the multiplies, while the third limits the communication volume. This partitioning problem is known to be NP-hard [4]. We will present methods to divide into $p = 2$ partitions. In order to divide into $p = 2^k$ sets, we recursively partition the block diagonals.

We will focus on multilevel methods to approximately solve this problem. This type of method has three phases. In phase 1, the graph is successively coarsened by merging vertex pairs. Row vertices merge only with row vertices, likewise for column vertices. The rows are merged as follows. A random eligible row is chosen, say i. It is merged with a randomly chosen, unmerged row node, say i', that is a path of length two away. That is, two row nodes can be merged if they are currently unmerged and some column has nonzero values in both row locations. (If no such row node exists, the i is ineligible for merging.) We choose another eligible row at random and continue the process until all row vertices have had the chance to be paired. We pair the column vertices via an analogous procedure. The vertex weight of a node in the coarse graph is the sum of the weights of its constituent pair. There is an edge between two nodes in the coarse graph if any pair of their constituent vertices had an edge. The weight of the edge is the sum of all the edge weights of the edges between the constituent vertices. The coarse graph maintains the bipartite structure of the original graph and has about half as many vertices as the original graph. To further coarsen, we repeat the process until we have reached the desired number of vertices.

In phase 2, the coarsest graph is partitioned randomly.

Lastly in phase 3, the graph is successively un-coarsened and the partition is refined each step. As we un-coarsen, the two constituent nodes of a merged vertex are initially in the same partition as the merged vertex. In the course of the refinement, one or both may switch partitions. We have experimented with three different refinement strategies. The first refinement option is a modified version of Kernighan-Lin [12] for bipartite graphs. This method moves one node at a time looking for a better partition. The second option is to use the alternating partitioning method presented in [13]. We start with a fixed column permutation that may be the result of some previous ordering. Given that the column partitioning is fixed, we must compute the best row partition. We then fix that row partition and compute the best column partition. We continue alternatingly fixing one partition and computing the other until the overall partition can no longer be improved. The final refinement option is to use alternating partitioning followed by Kernighan-Lin; this can be thought of as a rough refinement followed by a fine refinement. Note that in addition to being used as components of a multilevel approach, these refinement algorithms can improve a partition produced by any algorithm. More details on these methods can be found in Hendrickson and Kolda [6].

In Sect. 4, we will also give results for the Spectral method for bipartite graphs, described originally in Berry, *et al.* [2] and also in Hendrickson and Kolda [6].

4 Experimental Results

The software we are using is a modification of the Chaco package (written in C) developed by Hendrickson and Leland [7] for multilevel partitioning of symmetric matrices. The timings for the partitioning were done on a 300 MHz Pentium II.

We will give results for four matrices from different disciplines, each split over 16 processors. The alternating partitioning (AP) method starts with a random ordering; it has no multilevel component. The spectral method (Spectral) uses a multilevel Rayleigh Quotient Iteration/Symmlq eigensolver [1, 7]. The three multilevel methods differ in the refinement algorithm they apply after each uncoarsening step. ML-AP applies alternating partitioning, ML-KL uses Kernighan-Lin, and ML-AP+KL combines the two by applying first alternating partitioning and then Kernighan-Lin. We always coarsen to one hundred coarse vertices in the multilevel methods. In every case, the rows (or columns) of the matrix are divided among processors in such a way that there is less than 10% difference in the number of matrix nonzeros owned by different processors.

For all the tables, the format is as follows. *Edge Cuts* is the number of edges in the bipartite graph that are *cut* by the given partition. *Part Time* is the time (in seconds) to compute the partition. *Tot Msgs* and *Tot Vol* are, respectively, the total number of messages and total message volume for computing either Ax or A^Tv. *Max Msg* and *Max Vol* are, respectively, the maximum number and maximum volume of messages handled by a *single* processor in the computation of Ax or $A^T v$, incoming or outgoing.

Table 1 shows the result of applying various partitioning methods to the $17,758 \times 17,758$ nonsymmetric `memplus` matrix[2] with 99,147 nonzeros for solving linear systems. Compared to a partitioning based upon the natural ordering, the number of edge cuts is substantially reduced by each method except Spectral, and the multilevel methods reduce the value by a factor of three. The overall communication volume is reduced by more than a factor of three by the multilevel methods, but not by the AP or Spectral methods. Only the Spectral method is expensive in terms of partition computation time, and that is because it had trouble converging. In fact, the Spectral method has problems in the next two examples as well. Note that the total number of messages increase because the message passing is more distributed; i.e., observe that the maximum message volume handled by a single processor is greatly reduced by the various reorderings.

Table 1. Communication pattern for row-based partitioning of the `memplus` matrix on 16 processors.

Method	**Edge Cuts**	**Part Time**	**Total Msgs**	**Total Vol**	**Max Msgs**	**Max Vol**
Natural	69381	0.26	74	30501	15	12826
AP	49535	1.66	216	28864	15	3152
ML-KL	18138	4.28	240	8991	15	877
ML-AP	20273	4.20	239	11040	15	1073
ML-AP+KL	18762	5.54	239	10128	15	1022
Spectral	59481	84.66	137	31709	15	7312

The $28,254 \times 17,284$ `pig-large` matrix [5,9] with 75,018 nonzeros arises from a least squares problems. The multilevel methods (see Table 2) are best, reducing the edge cuts, total message volume, and the processor message volume by factors of more than three in every case.

The $6,071 \times 12,230$ `dfl001` matrix[3] with 35,632 nonzeros arises from linear programming. Here we partition the matrix *column-wise* since the matrix has some dense rows; this should yield a better partitioning. In Table 3, we see that again the number of edge cuts, the total message volume, and the maximum single processor volume are substantially reduced.

Results for the $1,853 \times 625$ `man1` matrix with 3,706 nonzeros are presented in Table 4. This is a hypertext matrix as described in Berry, *et al.* [2], and we want to compute a low-rank SVD for it. In this case, the number of messages is actually reduced, as was the total volume, and maximum processor volume.

[2] Available from MatrixMarket (`http://math.nist.gov/MatrixMarket/`).
[3] Available from NETLIB (`http://www.netlib.org/lp`).

Table 2. Communication pattern for row-based partitioning of the `pig-large` matrix on 16 processors.

Method	Edge Cuts	Part Time	Total Msgs	Total Vol	Max Msgs	Max Vol
Natural	55332	0.26	78	23203	12	4740
AP	24016	2.44	229	14069	15	1335
ML-KL	11775	4.26	216	3796	15	494
ML-AP	14966	3.53	214	6694	15	1100
ML-AP+KL	12202	6.52	216	3632	15	416
Spectral	11726	193.14	187	6475	15	810

Table 3. Communication pattern for column-based partitioning of the `dfl001` matrix on 16 processors.

Method	Edge Cuts	Part Time	Total Msgs	Total Vol	Max Msgs	Max Vol
Natural	33194	0.10	140	24636	15	8468
AP	14361	1.02	239	13804	15	1342
ML-KL	8663	2.54	238	7951	15	749
ML-AP	10388	1.80	239	9569	15	855
ML-AP+KL	8653	3.30	234	7919	15	742
Spectral	17067	44.15	228	13383	15	2056

Here we finally have a case where the Spectral method does better than all the others, although it still takes the longest time to compute. Figure 1 shows the original matrix and the result of the ML-AP+KL partitioning for 8 processors.

5 Conclusions

We presented (modified) multilevel methods for partitioning sparse nonsymmetric and nonsquare matrices. We described how this can be used to implement efficient parallel matrix-vector and matrix-transpose-vector multiplies with reduced communication. We tested our methods on four matrices from four different mathematical applications. Our results show that partitioning clearly reduces the communication volume that multilevel partitioning with alternating partitioning plus Kernighan-Lin refinement is generally the best of the partitioners.

Acknowledgements

Thanks to Iain Duff and Michael Saunders for help in finding large matrices to work with. Also thanks to Michele Benzi for many helpful conversations.

Table 4. Communication pattern for column-based partitioning of the `dfl001` matrix on 16 processors.

Method	Edge Cuts	Part Time	Total Msgs	Total Vol	Max Msgs	Max Vol
Natural	1497	0.01	236	1168	15	93
AP	1002	0.05	180	685	15	68
ML-KL	1045	0.13	137	556	15	103
ML-AP	811	0.08	151	521	14	62
ML-AP+KL	618	0.15	141	417	13	49
Spectral	556	1.75	119	306	12	39

References

1. Stephen T. Barnard and Horst D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6:101–117, 1994.
2. Michael W. Berry, Bruce Hendrickson, and Padma Raghavan. Sparse matrix reordering schemes for browsing hypertext. In James Renegar, Michael Shub, and Steve Smale, editors, *The Mathematics of Numerical Analysis*, volume 32 of *Lectures in Applied Mathematics*, pages 99–122. American Mathematical Society, 1996.
3. Roland W. Freund and Noël M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.
4. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.
5. Markus Hegland. Description and use of animal breeding data for large least squares problems. Technical Report TR-PA-93-50, CERFACS, Toulouse, France, 1993.
6. Bruce Hendrickson and Tamara G. Kolda. Partitioning nonsquare and nonsymmetric matrices for parallel processing. Technical Memorandum TM-13657, Oak Ridge National Laboratory, Oak Ridge, TN 37831, 1998. Submitted to *SIAM J. Scientific Computing.*
7. Bruce Hendrickson and Robert Leland. The Chaco user's guide, version 2.0. Technical Report SAND95-2344, Sandia Natl. Lab., Albuquerque, NM, 87185, 1995.
8. Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95.* ACM, 1995.
9. A. Hofer. *Schätzung von Zuchtwerten feldgeprüfter Schweine mit einem Mehrmerkmals-Tiermodell.* PhD thesis, ETH-Zurich, 1990. Cited in [5].
10. George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, Dept. Computer Science, Univ. Minnesota, Minneapolis, MN 55455, 1995.
11. George Karypis and Vipin Kumar. Parallel multilevel graph partitioning. Technical Report 95-036, Dept. Computer Science, Univ. Minnesota, Minneapolis, MN 55455, 1995.
12. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical J.*, 1970.
13. Tamara G. Kolda. Partitioning sparse rectangular matrices for parallel processing. In *Proc. 5th Intl. Symposium on Solving Irregularly Structured Problems in Parallel (Irregular '98)*, to appear.

14. Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Mathematical Software*, 8:43–71, 1982.
15. Weichung Wang and Dianne P. O'Leary. Adaptive use of iterative methods in interior point methods for linear programming. Technical Report CS-TR-3560, Dept. Computer Science, Univ. Maryland, College Park, MD 20742, 1995.