Partitioning Sparse Rectangular Matrices for Parallel Processing*

Tamara G. Kolda

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6367. kolda@msr.epm.ornl.gov.

Abstract. We are interested in partitioning sparse rectangular matrices for parallel processing. The partitioning problem has been well-studied in the square symmetric case, but the rectangular problem has received very little attention. We will formalize the rectangular matrix partitioning problem and discuss several methods for solving it. We will extend the spectral partitioning method for symmetric matrices to the rectangular case and compare this method to three new methods — the alternating partitioning method and two hybrid methods. The hybrid methods will be shown to be best.

1 Introduction

Organizing the nonzero elements of a sparse matrix into a desirable pattern is a key problem in many scientific computing applications, particularly load balancing for parallel computation. In this paper we are interested in ordering the nonzeros of a given matrix into *approximate block diagonal form* via permutations. This problem corresponds directly to the partitioning problem in graph theory, and so is often referred to as matrix partitioning.

The partitioning problem has been well-studied in the symmetric case [1, 2, 4, 11, 12, 13, 14, 15, 16, 21, 22, 23]. The rectangular partitioning problem, however, has received very little attention; the primary reference in this area is Berry, Hendrickson, and Raghavan [3] on envelope reduction for hypertext matrices.

Let A denote a sparse rectangular $m \times n$ matrix. We will assume throughout that we are working with pattern (0-1) matrices, but the results and methods can easily be extended to nonnegative weighted matrices. Our goal is to partition A into a block 2×2 matrix so that most of the nonzeros are on the block diagonal and so that each block diagonal has about the same number of nonzeros. In other words, we wish to find permutation matrices P and Q such that

$$B \equiv PAQ = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} ,$$

* This work was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation.

where B_{12} and B_{21} are as sparse as possible and the block rows or block columns each have about the same number of nonzeros. In order to avoid a trivial solution (e.g., $B_{11} = A$), we require that B_{11} have p rows and q columns where p is some integer between 1 and $m-1$ and q is some integer between 1 and $n-1$. The values p and q may or may not have been chosen in advance; typically we will want $p \approx m/2$ and $q \approx n/2$ to maintain load balance. If there exists P and Q such that B_{12} and B_{21} are identically zero, then we say that A is *block diagonalizable.* If we wish to partition A in a block $2^k \times 2^k$ matrix, we can recursively partition the block diagonals.

The matrix partitioning problem is equivalent to the *edge-weighted graph partitioning problem*: Given an undirected edge-weighted graph, partition the nodes into two sets of given sizes such that the sum of the weights of the edges that pass between the two sets is minimized. The graph partitioning problem is a well-known NP-complete problem (see problem ND14 on p. 209 in Garey and Johnson [6]). A rectangular $m \times n$ matrix corresponds to a bipartite graph [3] with m left nodes and n right nodes. There is an edge between left node i and right node j if a_{ij} is nonzero, and the weight of the edge is one. See Fig. 1 for an illustration. Suppose that we partition A so that the union of the first p rows and the first q columns form one partition and the remaining rows and columns form the other partition. The edges passing between the two partitions correspond to the nonzeros in the off-diagonal blocks of the partitioned matrix.

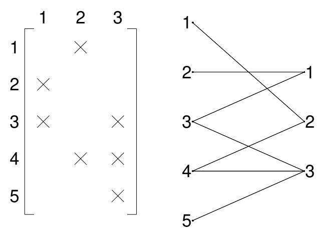

Fig. 1. The bipartite graph of a rectangular matrix.

Observe that the graph of A is disconnected if and only if the matrix A is block diagonalizable. Throughout we will assume that the graph of A is connected. If it is not, we will re-order the matrix so that it is block diagonalized with the blocks in decreasing order of size. We will only work with components that cross the boundary of the desired partition. In the discussion of the theory, we will assume that the graph of A is connected.

Many iterative methods, e.g., LSQR[20] require matrix-vector and matrix-vector-transpose multiplies with rectangular matrices. In Sect. 2 we will describe how to implement these kernels to take advantage of the partitioned matrix.

In Sect. 3, we will present several algorithms for the rectangular partitioning problem. We will discuss the well-known spectral partitioning method and show

how it can be applied to this problem. We will also introduce a new alternating partitioning method as well as two hybrid strategies.

In Sect. 4, we will compare the various partitioning methods, and show that the hybrid methods are the best.

2 Parallel Matrix-Vector Multiplication

We propose the following parallel implementations for the matrix-vector and matrix-transpose-vector multiplications. Suppose that we have $r = 2^k$ processors. We partition A into a block $r \times r$ matrix,

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1r} \\ A_{21} & A_{22} & \cdots & A_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ A_{r1} & A_{r2} & \cdots & A_{rr} \end{bmatrix} ,$$

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.

[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. 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 [9] for more detail on the algorithm and for more details on potential applications.

3 Algorithms for the Rectangular Partitioning Problem

Here we will discuss how the well-known spectral method can be applied to the rectangular problem and introduce a new method that can be used on its own or in combination with other methods. The spectral method will be used as a basis for comparison to the new methods in Sect. 4.

3.1 Spectral Partitioning

In the symmetric problem, spectral partitioning based on the Fiedler vector is a well-known technique; see, for example, Pothen, Simon, and Liou [21]. Many people have studied the effectiveness of spectral graph partitioning; for example, Guattery and Miller [8] show that spectral partitioning can be bad, while Spielman and Teng [23] show that it can be good.

One natural way to approach the rectangular problem is to *symmetrize* the matrix A, yielding the $(m+n) \times (m+n)$ matrix

$$\tilde{A} = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} ,$$

and apply spectral partitioning to the symmetrized matrix. This approach is used by Berry et al. [3]. Note that the graphs of $\tilde{A}$ and A are the same.

In order to apply spectral partitioning, we compute the Laplacian of $\tilde{A}$,

$$L = D - \tilde{A} ,$$

where $D = \operatorname{diag}\{d_1, d_2, \ldots, d_{m+n}\}$ and $d_i = \sum_j \tilde{a}_{ij}$. The matrix L is symmetric and semi-positive definite; furthermore, the multiplicity of the zero eigenvalue must be one since we are assuming that the graph of A, and hence of $\tilde{A}$, is connected [5]. Let w denote the Fiedler vector of L, that is, the eigenvector corresponding to the smallest positive eigenvalue of L. Let u denote the first m and v the last n elements of w, and sort the elements of u and v so that

$$\begin{aligned} u_{i_1} \geq u_{i_2} \geq \cdots \geq u_{i_m} \quad , \\ v_{j_1} \geq v_{j_2} \geq \cdots \geq v_{j_n} \quad . \end{aligned}$$

Then $\{i_1, i_2, \ldots, i_m\}$ and $\{j_1, j_2, \ldots, j_n\}$ define the row and column partitions respectively. In other words, assign rows $i_1, i_2, \ldots, i_p$ and columns $j_1, j_2, \ldots, j_q$ to the first partition and the remaining rows and columns to the second partition. Note that the ordering is independent of p and q. This means that the values of p and q may be fixed in advance as something like $\lceil m/2 \rceil$ and $\lceil n/2 \rceil$ respectively, or they may be chosen after the ordering has been computed to ensure good load balancing.

In Sect. 4 we will use this method as a basis for comparison for our new methods.

3.2 The Alternating Partitioning Method

Rather than trying to compute both the row and column partitions simultaneously as is done in the spectral method, the new method proposed in this section focuses on one partition at a time, switching back and forth. This method is derived from the Semi-Discrete Matrix Decomposition, a decomposition that has been used for image compression [19] and information retrieval [17, 18].

Before we describe the method, we will re-examine the problem. If we let $\mathcal{I}$ denote the set of row indices that are permuted to a value less than or equal to p and correspondingly let $\mathcal{I}^c$ denote the set of row indices permuted to a value greater than p and define the set $\mathcal{J}$ in an analogous way for the columns, then we can write the rectangular partitioning problem as a maximization problem,

$$\begin{aligned} \max_{\mathcal{I},\mathcal{J}} \; & \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}}} a_{ij} + \sum_{\substack{i\in\mathcal{I}^c\\ j\in\mathcal{J}^c}} a_{ij} - \sum_{\substack{i\in\mathcal{I}\\ j\in\mathcal{J}^c}} a_{ij} - \sum_{\substack{i\in\mathcal{I}^c\\ j\in\mathcal{J}}} a_{ij} \;\;, \\ \text{s.t. } & \mathcal{I} \subseteq \{1,2,\cdots,m\} \;\;,\;\; \mathcal{J} \subseteq \{1,2,\cdots,n\} \;\;, \\ & |\mathcal{I}| = p, \;\; |\mathcal{J}| = q \;\; . \end{aligned} \tag{1}$$

Here the objective function is the sum of the nonzeros on the block diagonal minus the sum of the elements off the block diagonal.

We can then rewrite this problem as an integer programming problem. Let x be a vector that defines the set membership for each row index; that is, $x_i = 1$ if row i is in $\mathcal{I}$, and $x_i = -1$ if row i is in $\mathcal{I}^c$, and let the vector y be defined in an analogous way for the columns. Then we can rewrite problem (1) as

$$\begin{aligned} & \max x^T A y \;\;, \\ & \text{s.t. } x_i = \pm 1 \;\;,\;\; y_j = \pm 1 \;\;, \\ & \qquad x^T e = 2p - m \;\;,\;\; y^T e = 2q - n \;\;, \end{aligned} \tag{2}$$

where e denotes the ones vector whose length is implied by the context.

Although we cannot solve (2) exactly, we can use an *alternating method* to get an approximate solution. We fix the partition for, say, the right nodes (y), and then compute the best possible partition for the left nodes (x). Conversely, we then fix the partition for the left nodes, and compute the best possible partition for the right nodes, and so on.

Suppose that we have fixed the partition for the right nodes. To determine the best partition of the left nodes, we need to solve

$$\begin{aligned} &\max x^T s \ , \\ &\text{s.t. } x_i = \pm 1 \ , \\ &\quad x^T e = 2p - m \ , \end{aligned} \tag{3}$$

where $s = Ay$ is fixed. The solution to this problem can be computed exactly. If we sort the entries of s so that

$$s_{i_1} \geq s_{i_2} \geq \cdots \geq s_{i_m} \ ,$$

then x defined by $x_{i_1} = x_{i_2} = \cdots = x_{i_p} = +1$ and $x_{i_{p+1}} = x_{i_{p+2}} = \cdots = x_{i_m} = -1$ is the exact solution to (3). Observe that the ordering of the elements of s does not depend on the value for p. If p has not been specified ahead of time, we would choose p to ensure load balancing. However, note that then p may be changing every iteration. An analogous procedure would be employed to find y when x is fixed.

Assuming p and q are fixed, each time we fix one side's partition and then compute the other, we are guaranteed that the value of the objective will never decrease. In other words, let $x^{(k)}$ and $y^{(k)}$ denote the partitions at the kth iteration of the method, and let f_k denote the objective value, ${x^{(k)}}^T A y^{(k)}$; then $f_{k+1} \geq f_k$ for all k. In the experiments presented in this paper, the method terminates when the objective value stops increasing. Alternatively, the method could terminate after at most some fixed number of iterations.

This method is called the alternating partitioning (AP) method and is specific to the rectangular problem since we are dealing with both row and column partitions and so we can alternate between working with one and then the other. In the symmetric case, we are only dealing with one partition.

We have not yet specified how to choose the first partition when we start the iterations, but that choice is important. In the standard method, we simply use the identity partition; however the next subsection will present two *hybrid* methods that use other techniques to generate a starting partition.

3.3 Hybrid Alternating Partitioning Methods

Since the alternating partitioning method is a greedy method, its key to success is having a good starting partition. Here we propose two possibilities.

Hybrid Spectral – Alternating Partitioning Method. This method uses the partition generated by the spectral method described in Sect. 3.1 as the starting partition for the alternating partitioning method.

Hybrid RCM – Alternating Partitioning Method. The Reverse Cuthill-McKee (RCM) method is not generally used as a partitioning method, but it generates a good inexpensive starting partition for the alternating partitioning method.

RCM is typically used for envelope reduction on symmetric matrices and is based on the graph of the matrix. Essentially, the method chooses a starting node and labels it 1. It then consecutively labels the nodes adjacent to it, then labels the nodes adjacent to them, and so forth. Once all the nodes are labelled, the ordering is reversed (hence the name). See George and Liu [7] for further discussion. In the nonsquare or nonsymmetric case, we apply RCM to the symmetrized matrix as was done by Berry et al. [3].

4 Experimental Results

In this section we compare the various partitioning methods presented in Sect. 3 on a collection of matrices listed in Table 1. These matrices were obtained from Matrix Market [2] with the exception of `ccealink`, `man1`, `man2`, and `nhse400` which were provided by Michael Berry and are those used in Berry et al. [3]. These matrices range in size from 100×100 to 4000×400. All of the square matrices are structurally nonsymmetric.

Table 1. Rectangular test matrices.

Matrix	Rows	Columns	Nonzeros
bfw782a	782	782	7514
ccealink	1778	850	2388
gre__115	115	115	421
illc1033	1033	320	4732
impcol_a	207	207	572
impcol_c	137	137	411
impcol_d	425	425	1339
impcol_e	225	225	1308
man1	1853	625	3706
man2	1426	850	2388
nhse400	4233	400	5119
nnc261	261	261	1500
watson3	124	124	780
west0132	132	132	414
west0156	156	156	371
west0479	479	479	1910
utm300	300	300	3155

Tables 2, 3, and 4 show the results of partitioning the rectangular matrices into block 2×2, 8×8, and 16×16 matrices. In these experiments, all the matrices were converted to pattern (0-1) matrices, but we could have converted them to nonnegatively weighted matrices instead. When partitioning into a block 2×2

[2] `http://math.nist.gov/MatrixMarket/`

matrix, we choose $p = \lceil m/2 \rceil$ and $q = \lceil n/2 \rceil$. (Alternatively, we could compute p and q on the fly.) When partitioning into more blocks, we first partition into a block 2×2 matrix and the recursively partition the diagonal blocks into block 2×2 matrices until the desired number of blocks is reached.

The tables are formatted as follows. Each row corresponds to a given matrix whose name is specified in the first column. The second column lists the number of nonzeros outside of the block diagonal in the original matrix. The number of nonzeros outside of the block diagonal is equivalent to the number of edge cuts in the graph partitioning problem. Columns 3 – 6 list the number of nonzeros outside the block diagonal after applying the method listed in the column header to the original matrix. The number in parentheses is the time in seconds it took to compute the ordering. All timings were done in MATLAB. Spectral partitioning requires the second eigenvector corresponding to the second smallest eigenvalue, and this was computed using the MATLAB EIGS routine. (Note that EIGS has a random element to it, so the results for the spectral and hybrid spectral – alternating partitioning method cannot be repeated exactly.) The alternating partitioning method was implemented using our own code. The MATLAB SYMRCM routine was used to compute the RCM ordering.

Table 2. Approximate block diagonalization of rectangular matrices into block 2×2 matrices.

	Nonzeros outside of Block 2×2 Diagonal				
Matrix	**Orig**	**Spectral**	**AP**	**RCM-AP**	**Spec-AP**
`bfw782a`	2438	229 (27.52)	232 (2.04)	383 (2.38)	**147** (27.74)
`ccealink`	324	384 (14.92)	349 (1.99)	144 (2.50)	**106** (15.33)
`gre__115`	104	45 (1.14)	77 (0.12)	48 (0.18)	**40** (1.44)
`illc1033`	2336	588 (16.71)	**345** (1.28)	524 (1.57)	385 (16.61)
`impcol_a`	59	9 (1.90)	25 (0.19)	24 (0.26)	**8** (1.94)
`impcol_c`	46	25 (1.25)	26 (0.14)	31 (0.16)	**22** (1.24)
`impcol_d`	48	35 (6.74)	38 (0.45)	**20** (0.57)	25 (6.79)
`impcol_e`	152	26 (2.53)	46 (0.26)	49 (0.32)	**24** (2.58)
`man1`	803	202 (46.68)	703 (1.99)	211 (2.75)	**169** (46.13)
`man2`	759	189 (33.12)	382 (1.62)	204 (2.04)	**150** (33.31)
`nhse400`	213	186 (86.05)	169 (3.07)	74 (4.35)	**60** (83.67)
`nnc261`	75	73 (3.66)	**71** (0.31)	73 (0.38)	**71** (4.70)
`watson3`	157	87 (1.24)	119 (0.17)	**77** (0.20)	**77** (1.26)
`west0132`	87	33 (1.16)	36 (0.14)	30 (0.19)	**29** (1.17)
`west0156`	231	**7** (1.25)	37 (0.16)	18 (0.18)	**7** (1.27)
`west0479`	755	117 (8.64)	191 (0.58)	142 (0.73)	**112** (8.70)
`utm300`	266	**215** (5.38)	266 (0.61)	240 (0.75)	**215** (5.31)

We are using the spectral method as a basis for comparison for our new methods. Let us first consider the alternating partitioning method. In the 2×2 and 8×8 tests, the spectral method does better than the alternating partitioning

Table 3. Approximate block diagonalization of rectangular matrices into block 8×8 matrices.

	Nonzeros outside of Block 8×8 Diagonal								
Matrix	**Orig**	**Spectral**		**AP**		**RCM-AP**		**Spec-AP**	
bfw782a	3872	832	(55.35)	1236	(5.02)	963	(5.72)	**804**	(55.03)
ccealink	684	991	(36.90)	813	(5.61)	286	(6.37)	**216**	(32.05)
gre__115	265	131	(3.92)	146	(0.53)	128	(0.64)	**118**	(4.26)
illc1033	4066	2474	(31.35)	1640	(3.17)	**1398**	(3.83)	1399	(32.72)
impcol_a	269	56	(5.66)	67	(0.73)	83	(0.88)	**48**	(5.70)
impcol_c	170	108	(4.19)	106	(0.57)	94	(0.66)	**85**	(4.14)
impcol_d	260	171	(15.02)	167	(1.37)	163	(1.82)	**128**	(15.43)
impcol_e	724	303	(6.70)	173	(0.91)	153	(1.10)	**137**	(6.81)
man1	1492	518	(82.38)	1249	(4.91)	578	(6.49)	**433**	(83.09)
man2	2207	449	(59.49)	904	(3.90)	524	(4.95)	**326**	(60.79)
nhse400	1001	1493	(143.98)	1142	(7.99)	**227**	(10.25)	234	(148.47)
nnc261	596	**339**	(9.23)	398	(1.04)	353	(1.27)	341	(9.99)
watson3	350	285	(4.02)	236	(0.56)	234	(0.71)	**232**	(4.40)
west0132	308	97	(4.01)	111	(0.57)	104	(0.67)	**82**	(4.14)
west0156	337	63	(4.13)	73	(0.59)	53	(0.76)	**42**	(4.17)
west0479	1309	427	(18.58)	420	(1.65)	413	(2.08)	**274**	(18.75)
utm300	1359	991	(12.19)	1211	(1.59)	**830**	(1.98)	888	(12.96)

Table 4. Approximate block diagonalization of rectangular matrices into block 16×16 matrices.

	Nonzeros outside of Block 16×16 Diagonal								
Matrix	**Orig**	**Spectral**		**AP**		**RCM-AP**		**Spec-AP**	
bfw782a	4320	1387	(61.83)	1951	(6.28)	1429	(7.17)	**1361**	(62.18)
ccealink	904	1195	(38.56)	1053	(7.27)	358	(7.96)	**266**	(32.63)
gre__115	291	171	(6.45)	195	(0.92)	162	(1.06)	**153**	(6.35)
illc1033	4378	2776	(33.45)	2194	(4.12)	1985	(4.92)	**1920**	(37.83)
impcol_a	413	125	(8.49)	115	(1.18)	127	(1.46)	**100**	(8.45)
impcol_c	252	160	(8.15)	157	(1.04)	143	(1.16)	**135**	(7.26)
impcol_d	444	289	(18.86)	278	(1.96)	305	(2.50)	**243**	(19.83)
impcol_e	976	615	(9.95)	474	(1.55)	471	(1.86)	**450**	(9.84)
man1	1837	796	(91.35)	1479	(6.21)	699	(7.98)	**602**	(91.92)
man2	2519	596	(66.93)	1086	(5.06)	671	(6.27)	**456**	(69.61)
nhse400	1594	1751	(161.72)	1795	(10.33)	**331**	(12.64)	428	(156.84)
nnc261	890	556	(12.69)	558	(1.53)	538	(1.87)	**528**	(13.08)
watson3	415	367	(6.39)	**289**	(0.99)	297	(1.18)	300	(6.90)
west0132	376	162	(6.34)	152	(0.97)	141	(1.24)	**132**	(6.88)
west0156	351	92	(8.11)	87	(1.04)	74	(1.23)	**70**	(6.43)
west0479	1618	640	(22.55)	541	(2.33)	477	(2.84)	**414**	(22.62)
utm300	1584	1504	(15.40)	1401	(2.21)	**1268**	(2.70)	**1268**	(15.95)

method on the majority of matrices, but on the 16×16 tests, the alternating partitioning method does better than spectral partitioning 11 out of 17 times. In terms of time, the alternating partitioning method is approximately 10 times faster than the spectral method.

The hybrid RCM-AP method is overall better than the alternating partitioning method. It outperforms the spectral method in the majority of matrices in the 8×8 and 16×16 tests. This method is the best overall an average of 2.3 times for each block size. The time for the hybrid RCM-AP method is only slightly more than that for the alternating partitioning method, and still about 10 times less than that for the spectral method.

For all three block sizes, we see that the hybrid spectral-AP method is the best overall in terms of reducing the number of nonzeros outside the block diagonal. This method is only slightly more expensive than the spectral method in terms of time and yields consistently better results. Sometimes the improvement is remarkable; see, for example, `ccealink` in all four tables. Unfortunately, the spectral and hybrid spectral-AP methods are very expensive in terms of time since they require the computation of some eigenpairs.

Figure 2 shows the effect of different partitioning strategies on the `west0156` matrix in the block 8×8 case.

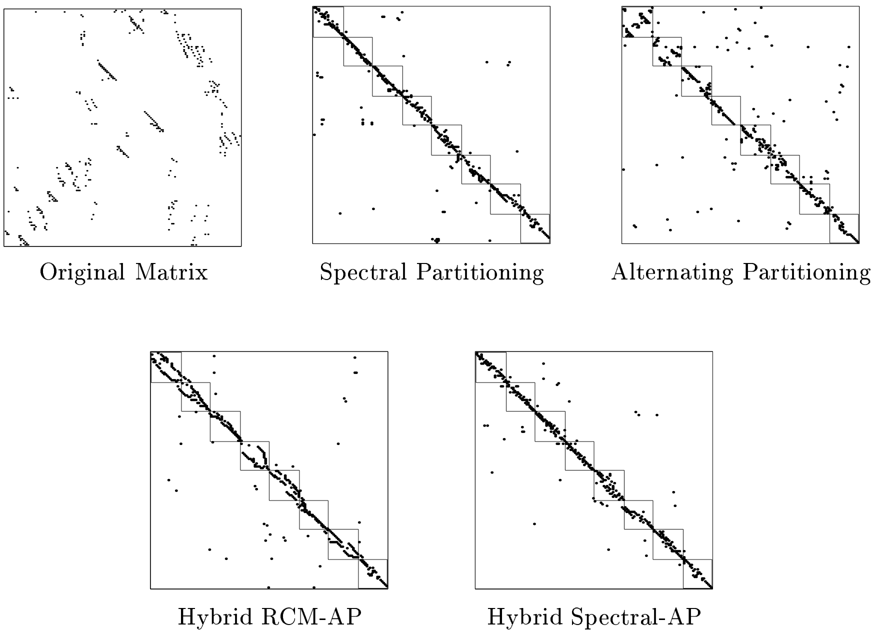

Fig. 2. Comparison of block 8×8 partitions on `west0156`.

Given these results, we recommend the hybrid spectral-AP method when quality is the top concern and the hybrid RCM-AP method when both time and quality matter.

5 Conclusions

In this work we introduced the rectangular matrix partitioning problem which is an extension of the symmetric matrix partitioning problem. The rectangular partitioning problem has a number of potential uses in iterative methods such as LSQR.

We introduced the alternating partitioning method as well as two hybrid methods and compared them with a rectangular version of the spectral partitioning method. The hybrid methods compared extremely favorably, and we recommend these as the methods of choice. The hybrid spectral-AP method is the overall best when partition quality is more important than the time to compute the partition, and the hybrid RCM-AP method is recommended when time is more important.

In other work [9, 10], this author and Bruce Hendrickson explore the multilevel method which is known to work very well for the symmetric partitioning problem [2, 11, 12, 13, 15]. We develop a multilevel method specific for bipartite graphs with various refinement strategies including the alternating partitioning method and a version of Kernighan-Lin for bipartite graphs. Eventually, we would also like to examine handling four or eight diagonal blocks directly [1, 11].

Acknowledgments

The author is indebted to Bruce Hendrickson and Dianne O'Leary for many helpful discussions. The author also thanks Eduardo D'Azevedo, Chuck Romine, and the anonymous referees for their reviews, and Mike Berry for providing data.

References

[1] Charles J. Alpert and So-Zen Yao. Spectral partitioning: The more eigenvectors, the better. In *32nd ACM/IEEE Design Automation Conference*, pages 195–200, 1995.

[2] 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.

[3] 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.

[4] Julie Falkner, Franz Rendl, and Henry Wolkowicz. A computational study of graph partitioning. *Math. Prog.*, 66:211–239, 1994.

[5] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical J.*, 23:298–305, 1973.

[6] 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.

[7] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Series in Computational Mathematics. Prentice-Hall, Englewood Cliffs, 1981.

[8] Stephen Guattery and Gary L. Miller. On the quality of spectral seperators. Accepted for publication in *SIAM J. Matrix Anal. Appl.*, 1997.

[9] Bruce Hendrickson and Tamara G. Kolda. Partitioning nonsquare and nonsymmetric matrices for parallel processing. In preparation, 1998.

[10] Bruce Hendrickson and Tamara G. Kolda. Partitioning sparse rectangular matrices for parallel computations of Ax and $A^T v$. Accepted for publication in *Proc. PARA98: Workshop on Applied Parallel Computing in Large Scale Scientific and Industrial Problems*, 1998.

[11] Bruce Hendrickson and Robert Leland. Multidimensional spectral load balancing. Technical Report 93-0074, Sandia Natl. Lab., Albuquerque, NM, 87185, 1993.

[12] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Stat. Comput.*, 16:452–469, 1995.

[13] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*. ACM, 1995.

[14] Bruce Hendrickson, Robert Leland, and Rafael Van Driessche. Skewed graph partitioning. In *Proc. Eighth SIAM Conf. on Parallel Processing for Scientific Computing*. SIAM, 1997.

[15] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for paritioning irregular graphs. Technical Report 95-035, Dept. Computer Science, Univ. Minnesota, Minneapolis, MN 55455, 1995.

[16] George Karypis and Vipin Kumar. Parallel multilevel graph partitioning. Technical Report 95-036, Dept. Computer Science, Univ. Minnesota, Minneapolis, MN 55455, 1995.

[17] Tamara G. Kolda. *Limited-Memory Matrix Methods with Applications.* PhD thesis, Applied Mathematics Program, Univ. Maryland, College Park, MD 20742, 1997.

[18] Tamara G. Kolda and Dianne P. O'Leary. A semi-discrete matrix decomposition for latent semantic indexing in information retrieval. Accepted for publication in *ACM Trans. Information Systems*, 1997.

[19] Dianne P. O'Leary and Shmuel Peleg. Digital image compression by outer product expansion. *IEEE Trans. Comm.*, 31:441–444, 1983.

[20] 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.

[21] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[22] Horst D. Simon. Partitioning of unstructured problems for parallel processing. In *Computing Systems in Engineering*, number 2/3, pages 135–148. Pergammon Press, 1991.

[23] Daniel A. Speilman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. Unpublished manuscript, 1996.