AN IMPROVED SPECTRAL GRAPH PARTITIONING ALGORITHM FOR MAPPING PARALLEL COMPUTATIONS*

BRUCE HENDRICKSON† AND ROBERT LELAND†

Abstract. Efficient use of a distributed memory parallel computer requires that the computational load be balanced across processors in a way that minimizes interprocessor communication. We present a new domain mapping algorithm that extends recent work in which ideas from spectral graph theory have been applied to this problem. Our generalization of spectral graph bisection involves a novel use of multiple eigenvectors to allow for division of a computation into four or eight parts at each stage of a recursive decomposition. The resulting method is suitable for scientific computations like irregular finite elements or differences performed on hypercube or mesh architecture machines. Experimental results confirm that the new method provides better decompositions arrived at more economically and robustly than with previous spectral methods. This algorithm allows for arbitrary non–negative weights on both vertices and edges to model inhomogeneous computation and communication. We also present a new spectral lower bound for graph bisection.

Key words. graph partitioning, parallel computation, load balancing, graph spectrum, eigenvector

AMS subject classifications. 05C50, 68R10, 65Y05

1. Introduction.

Efficient use of a distributed memory parallel computer requires that the computational load be balanced across processors in a way that minimizes interprocessor communication. This mapping requirement can be abstracted to a graph problem in which nodes represent computation, edges represent communication, and the objective is to assign an equal number of vertices to each processor in a way that, in some metric, minimizes the number of edges crossing between processors. Extensive practical experience has shown that the quality of this mapping has a substantial impact on performance, hence there is considerable interest in effective mapping algorithms.

Finding a mapping that actually minimizes communication between balanced sets is an NP–hard problem [9], so it is unlikely that an efficient, general algorithm exists. The practical importance of this problem has, however, motivated a variety of heuristic approaches. A thorough review of these methods and the extensive literature associated with them is beyond the scope of this paper. We simply note that the established methods range from quick, linear time algorithms based on geometric assumptions [17, 19] or local graph information [5, 15] to very slow algorithms which approximate a global search for the minimum using genetic operators or simulated annealing [14, 16]. The faster heuristics often do not provide mappings of adequate quality for bench-marking purposes or for performance-critical codes which will be used many times, while the more expensive mapping techniques are generally impractical for large problems. This paper describes a method designed to provide high quality partitionings at moderate cost.

Most existing load balancing methods are based on recursive graph bisection — the graph is broken in half, the halves are halved independently, and so on, until

* This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract number DE–AC04–76DP00789.

† Sandia National Laboratories, Albuquerque, NM, 87185. Electronic mail addresses bah@cs.sandia.gov and leland@cs.sandia.gov respectively.

‡ Appeared in SIAM J. Sci. Comput., 16(2):452–469, 1995

there are as many pieces as processors in the parallel machine. Bisection techniques have several inherent shortcomings. First, bisection algorithms are unable to accept a less attractive initial cut which would allow net savings in later cuts, i.e. they have no *look ahead* capability. In the VLSI community, for example, it has been observed that recursive quadrisection leads to better circuit layouts than twice as many steps of bisection [25]. Second, in bisection the task of splitting the graph into sets of vertices (the *decomposition* problem) is largely decoupled from that of assigning a set of vertices to a specific processor (the *assignment* problem). The communication overhead of an application program, however, depends on both the decomposition and the assignment, hence it is generally preferable to consider these aspects of the problem together. We might for example choose to accept a higher volume of communication between two sets in order to place them topologically closer on a given architecture.

Our approach to the graph partitioning problem addresses these shortcomings. It is based upon results from spectral graph theory, in which eigenvectors of a matrix are used to bisect a graph. The idea of using eigenvectors to partition graphs dates back to work in the early 70's by Donath and Hoffman [3, 4], and Fiedler [6, 7], but it has recently generated renewed interest [2, 18, 21, 22, 23]. Simon [24] and Williams [27] have applied spectral bisection to the load balancing problem and found it to have a number of attractive features in this context. Unlike some other techniques, spectral methods are invariant under geometric transformations of the computational domain, as well as under renumbering of the computational graph. They also seem to generate good partitions in practice, albeit at a fairly high cost compared with some weaker heuristics.

Our method generalizes spectral graph bisection in several important ways. First and most importantly, we use multiple eigenvectors in a new way to divide a problem into four or eight pieces at once rather than just two. On an intuitive level, the first eigenvector defines a surface which bisects the graph, the second defines an intersecting surface which bisects these two pieces, and so on. This allows us to perform fewer recursive steps while dividing a problem into a given number of pieces. By trading off the combined effects of several cuts, we can reduce the look ahead problem associated with bisection. In addition, by using multiple eigenvectors we often achieve a substantial economy in the net cost of the eigenvector calculations, the dominant expense. Second, our model allows for inhomogeneous computation and communication requirements of an application, substantially broadening the class of problems for which it is appropriate. Third, our method does not ignore machine architecture, but rather minimizes a function that explicitly accounts for hypercube topology in the communication cost. Recent empirical evidence confirms that this should lead to significantly better partitions in practice [10]. Our method can also be applied to meshes since d-dimensional meshes can be recursively decomposed as d-dimensional hypercubes. (We note that the function we minimize may not be appropriate for other applications of graph partitioning.) Fourth, unlike most other approaches, our method solves the assignment problem simultaneously with the decomposition.

Other methods for using multiple eigenvectors to partition graphs into multiple sets have been developed by Donath and Hoffman [3] and Rendl and Wolkowicz [23]. However, these algorithms require 2^k eigenvectors to produce 2^k sets rather than the k eigenvectors we need. Furthermore, they do not account for hypercube or mesh topology as we do. These methods do have the advantage of allowing for partitioning into sets of arbitrary size, whereas our method is largely restricted to producing balanced partitions.

Our partitioning algorithm is designed for mapping computations across message passing multiprocessors, and is most appropriate for applications in which the computational requirements are static so that a good decomposition can be determined *a-priori*. It is particularly well suited to problems in which many different messages are simultaneously competing for use of the communication network. Many problems in scientific communication fit this description because they involve alternate phases of computation and communication in which the same calculation is repeated in each computation phase and many messages are transmitted in the communication phase. Examples include typical unstructured finite difference and finite element calculations.

The structure of this paper is as follows. In §2, we review hypercube multiprocessors, describe our graph model of computation and develop an associated metric of communication cost. This allows us to construct a discrete optimization problem which describes the optimal mapping in §3. Since this optimization problem is NP-hard, we derive a continuous problem approximating it in §4. In §5 and §6 we describe how the solution to this continuous problem reduces to an eigenvector calculation, and how the eigenvectors can be used to generate an approximate solution to the discrete optimization problem. Some new *a-posteriori* lower bounds on partition quality are presented in §7. Results of some sample calculations are given in §8, and conclusions are presented in §9. A less formal, more intuitive presentation of much of this material can be found in [11, 12].

2. Preliminaries.

2.1. Hypercube multiprocessors. A d-dimensional hypercube multiprocessor consists of a set of 2^d processors, identified by distinct binary numbers from 0 to 2^d-1. Information is transmitted between them by passing messages through a network in which wires connect processors whose binary values differ in a single bit. We will assume there is no global memory, and that wires can simultaneously transmit data in either direction.

Hypercube multiprocessors enjoy popularity because they have attractive theoretical and practical properties: The network is very regular and can be described elegantly in a recursive fashion. Each processor is connected to d communication wires, and a message can be routed between any two processors by traversing at most d wires. Furthermore, a message route can be devised simply; to travel between two processors, a message merely uses one wire from each bit in which the two processor's binary representations differ.

2.2. A graph model of computation. As mentioned in §1, our approach to the partitioning problem is targeted mainly towards scientific computing applications. Most of these problems involve repeated iterations of the same cycle of computations. Although it is sometimes possible to achieve parallelism by effectively overlapping multiple iterations [28], the more common approach is to exploit parallelism within each iteration. Within an iteration, a processor performs a set of computations followed by a set of communication operations, and since each iteration involves the same set of operations, it is sufficient to distribute the task among processors based upon the requirements of a single iteration.

We represent a computation as an undirected, weighted graph $G = (V, E)$, using n to denote the size of the vertex set V, and m the size of the edge set E. Each vertex $v_i \in V$ corresponds to a computational task to be performed on a single processor, and the time required to execute that task is represented by a positive weight $w_v(v_i)$. We denote by W_v the sum of the weights of all the vertices in the graph. An undirected edge $e_{ij} \in E$ connects two vertices v_i and v_j if the computational task represented by one of the vertices requires input from the other. The edge has an associated positive weight $w_e(e_{ij})$ proportional to the amount of data that must be transmitted between the tasks. If each task requires data from the other, then this weight is the sum of the two amounts of information. The sum of the weights of all the edges is denoted by W_e. For technical reasons that will become clear in §5, we will assume that G is connected. The graphs associated with most scientific computations are at least largely connected, and our implementation employs a scheme which adds a minimal number of edges to connect a disconnected graph, so the assumption of connectivity is not restrictive in practice.

Partitioning a computational task among the processors corresponds to assigning each vertex of the graph to a processor. The sum of the weights of the vertices assigned to a processor represents the amount of computational effort that processor must expend, and the sum of the weights of all the edges connecting vertices assigned to two different processors represents the total amount of information that must be communicated between the two.

2.3. A communication metric. Most modern parallel computers have some form of hardware cut-through routing, hence the time required to transmit a message is nearly independent of the number of wires traversed provided there is no significant network congestion. For applications like those we are targeting in which most messages are lengthy, this implies a model in which the communication cost of a message is proportional to the message length, independent of the identity of the sending and receiving processors. The cost of a set of messages can then be modeled as the sum of their individual costs. Within the constructs of our graph model, we define the *cut-weight* of a partitioning scheme to be the sum of the weights of all the edges whose vertices are assigned to different processors. Most previous approaches to domain mapping for parallel computing have tried to minimize this cut-weight communication measure.

However, since it treats messages in isolation, the cut-weight metric fails to consider any effects of message congestion. The applications we are considering typically have a communication phase in which there are many messages simultaneously competing for wires. In this case, each wire a message uses is unavailable for other tasks, so the load a message places on the network is proportional to the number of wires it consumes. Consequently we define the *hop-weight* of a message to be the length of the message multiplied by the number of wires it requires, and the hop-weight of a collection of messages to be the sum of their individual hop-weights. We will use hop-weight as our measure of the communication cost of a mapping. Recent experimental work has indicated that this is the most accurate communication metric for scientific computing [10].

With the intent of making this discussion more formal, we let $\mathcal{M} : V \rightarrow P$ be an assignment scheme that maps vertices to processors. We denote by $\mathcal{V}(q)$ the set of vertices assigned to a processor q, so $\mathcal{V}(q) = \{v \in V : \mathcal{M}(v) = q\}$, and we use ρ_i to indicate the processor to which vertex v_i is assigned. The number of wires that a message must traverse to get from ρ_i to ρ_j we denote by h_{ij}, and observe that it is a function of the machine topology, not the mapping. With this notation, we can formally define the hop-weight of an assignment as

$$hop\text{-}weight(\mathcal{M}) = \sum_{e_{ij} \in E} w_e(e_{ij}) h_{ij}. \tag{1}$$

Next we map the binary digits designating a processor to ± 1 by

$$(2) \qquad c^{(k)}(q) = \begin{cases} 1 & \text{if bit } k \text{ of } q = 1, \\ -1 & \text{if bit } k \text{ of } q = 0. \end{cases}$$

This transformation is convenient because the simple function $(1 - c^{(k)}(q)c^{(k)}(r))/2$ is zero if processors q and r have the same kth bit and one if they differ. For a hypercube the number of differing bits between processors q and r is equal to h_{rq} and can now be expressed as $\sum_{k=1}^{d}(1 - c^{(k)}(q)c^{(k)}(r))/2$. Hence the total communication cost on a hypercube under an assignment scheme $\mathcal{M}$ can be represented as

$$(3) \qquad Cost(\mathcal{M}) = hop\text{-}weight(\mathcal{M}) = \frac{1}{2}\sum_{e_{ij}\in E} w_e(e_{ij}) \sum_{k=1}^{d}(1 - c^{(k)}(\rho_i)c^{(k)}(\rho_j)).$$

We would like to find an assignment that minimizes this communication cost, while keeping the computational load balanced. We note that when $d = 1$, hop-weight reduces to cut-weight, so in this case, the minimal cost is the *bisection width* of the graph.

3. A discrete optimization problem. When using a spectral method to solve a combinatorial problem, the general strategy is to formulate the combinatorial problem as a discrete optimization and then relax the discreteness constraint to obtain a continuous optimization problem. This continuous version may have some special structure making it tractable, even if the original discrete problem is NP-hard. After the continuous problem is solved, the result is mapped back to a nearby discrete point, which often provides a good approximation to the discrete optimum. A survey of results obtained using this general approach is given by Mohar in [18].

To follow this strategy we need to express our problem as a discrete optimization problem. The communication cost we wish to minimize is given in (3), but it will prove useful to add an additional term and interchange the order of summation to obtain

$$(4) Cost(\mathcal{M}) = \frac{1}{2}\sum_{k=1}^{d} \{\sum_{e_{ij}\in E} w_e(e_{ij})(1 - c^{(k)}(\rho_i)c^{(k)}(\rho_j)) + \frac{1}{2}\sum_{i=1}^{n} t_i(c^{(k)}(\rho_i)^2 - 1)\}.$$

Since $c^{(k)}(q) = \pm 1$, this last term is zero, and does not change the value of $Cost(\mathcal{M})$. However, when the discreteness constraint on $c^{(k)}$ is relaxed in §4, this term will become important. Appropriate values for t_i will also be considered in §4.

Equation (4) describes the communication cost to be minimized, but it must be constrained to ensure load balance. The computational load is balanced if the sums of the weights of the vertices assigned to each processor are equal. Strict balance is not always feasible, *e.g.* when bisecting a graph with an odd number of equally weighted vertices. Hence it should be understood that the balance constraint may have some associated discretization error. With this in mind, we can write the load balance constraint

$$(5) \qquad \mathcal{W}(q) = W_v/2^d, \qquad \forall q \in \{0, \ldots, 2^d - 1\},$$

where $\mathcal{W}(q)$ denotes the sum of the weights of all vertices assigned to processor q, so that $\mathcal{W}(q) = \sum_{v\in\mathcal{V}(q)} w_v(v)$. It will prove convenient to use a different form of the balance constraint expressed in terms of the $c^{(k)}$ notation introduced in (2). In particular, the conditions

$$
\begin{aligned}
(6) \qquad & \text{(a)} \qquad \sum_{q=0}^{2^d-1} \mathcal{W}(q) = W_v \\
& \text{(b)} \qquad \sum_{q=0}^{2^d-1} \mathcal{W}(q) \prod_{j\in\mathcal{S}} c^{(j)}(q) = 0, \qquad \forall \mathcal{S} : \emptyset \neq \mathcal{S} \subseteq \{1,\ldots,d\}.
\end{aligned}
$$

ensure balance, as demonstrated by Theorem 3.1.

THEOREM 3.1. *Equations (5) and (6) are equivalent.*

Proof. Since it excludes the null subset, condition (6b) provides $2^d - 1$ constraint equations. Combining (6b) with (6a) yields 2^d equations for the 2^d unknown values of $\mathcal{W}(q)$. We will show these equations to be linearly independent and hence that a unique solution exists. Then, since the solution to (5) is easily seen to be a solution to (6), we conclude that the two constraint formulations are equivalent.

The proof that the equations (6) are linearly independent is inductive. First we consider the $d = 1$ base case, and note that equation (6a) becomes $\mathcal{W}(1)+\mathcal{W}(2) = W_v$. Similarly, equation (6b) becomes $\mathcal{W}(1)\cdot 1 + \mathcal{W}(2)\cdot -1 = 0$ since $\mathcal{S} = \{1\}$ means $j = 1$ and hence the $c^{(j)}$ term associated with one vertex set is $+1$, and that associated with the other vertex set is -1. These equations are clearly linearly independent.

Suppose now that for some value of d we know that a system of this form, denoted $G(d\times d)\bar{\mathcal{W}}(d) = f(d)$, is linearly independent. We can demonstrate that the system $G((d+1)\times(d+1))\bar{\mathcal{W}}(d+1) = f(d+1)$ is linearly independent as follows. Note that the set of subsets $\emptyset \neq \mathcal{S}' \subseteq \{1,\ldots,d+1\}$ consists of $\mathcal{S}$, $\{d+1\}$ and a remainder comprised of each set in $\mathcal{S}$ with $d+1$ included. We can therefore rewrite (6) for the $d+1$ case:

$$
\begin{aligned}
(7) \qquad & \text{(a)} \quad \sum_{q=0}^{2^d-1} \mathcal{W}(q) + \sum_{q=2^d}^{2^{d+1}-1} \mathcal{W}(q) = W_v \\
& \text{(b)} \quad \sum_{q=0}^{2^d-1} \mathcal{W}(q) \prod_{j\in\mathcal{S}} c^{(j)}(q) + \sum_{q=2^d}^{2^{d+1}-1} \mathcal{W}(q) \prod_{j\in\mathcal{S}} c^{(j)}(q) = 0 \\
& \text{(c)} \quad \sum_{q=0}^{2^d-1} \mathcal{W}(q) c^{(d+1)}(q) + \sum_{q=2^d}^{2^{d+1}-1} \mathcal{W}(q) c^{(d+1)}(q) = 0 \\
& \text{(d)} \quad \sum_{q=0}^{2^d-1} \mathcal{W}(q) \prod_{j\in\mathcal{S}} c^{(j)}(q) c^{(d+1)}(q) + \sum_{q=2^d}^{2^{d+1}-1} \mathcal{W}(q) \prod_{j\in\mathcal{S}} c^{(j)}(q) c^{(d+1)}(q) = 0
\end{aligned}
$$

Now, letting $d+1$ be the position of the most significant bit in the binary representation of q, we have $c^{(d+1)}(q) = -1$ for values of q in the lower range $0 \leq q \leq 2^d - 1$, and $c^{(d+1)}(q) = 1$ for values in the upper range $2^d \leq q \leq 2^{d+1} - 1$. Upon substituting these values we arrive at the identity

$$
(8) \qquad G((d+1)\times(d+1)) = \begin{bmatrix} G(d\times d) & G(d\times d) \\ -G(d\times d) & G(d\times d) \end{bmatrix}.
$$

This system is clearly linearly independent since any linear combination of rows which zeros the left hand portion of a row will double the right hand portion, and by our inductive hypothesis, the individual matrix blocks are linearly independent. $\square$

Constraint (6b) can now be reformulated to involve the $w_v(v_i)$ values.

$$(9)\qquad \begin{aligned}\sum_{q=0}^{2^d-1} \mathcal{W}(q)\prod_{j\in\mathcal{S}} c^{(j)}(q) &= \sum_{q=0}^{2^d-1}\{\sum_{v_i\in q} w_v(v_i)\}\prod_{j\in\mathcal{S}} c^{(j)}(q)\\ &= \sum_{q=0}^{2^d-1}\sum_{v_i\in q}\{w_v(v_i)\prod_{j\in\mathcal{S}} c^{(j)}(\rho_i)\}\\ &= \sum_{i=1}^{n}\{w_v(v_i)\prod_{j\in\mathcal{S}} c^{(j)}(\rho_i)\}\end{aligned}$$

Equation (6a) is automatically satisfied by the $w_v(v_i)$ values and need not be explicitly included. We can therefore combine (4) with (9) to obtain a formal statement of the problem of minimizing communication subject to the load balance constraint.

Problem P1

$$(10)\quad \text{Minimize } \frac{1}{2}\sum_{k=1}^{d}\{\sum_{e_{ij}\in E} w_e(e_{ij})(1-c^{(k)}(\rho_i)c^{(k)}(\rho_j))+\frac{1}{2}\sum_{i=1}^{n} t_i(c^{(k)}(\rho_i)^2-1)\}$$

Subject to :

(a) $c^{(k)}(q)=\pm 1, \quad \forall k\in\{1,\ldots,d\},\ \forall q\in\{0,\ldots,2^d-1\}$

(b) $\displaystyle\sum_{i=1}^{n} w_v(v_i)\prod_{j\in\mathcal{S}} c^{(j)}(\rho_i)=0, \quad \forall\mathcal{S}:\emptyset\neq\mathcal{S}\subseteq\{1,\ldots,d\}.$

We will call this discrete optimization problem (**P1**). It is NP-hard since it generalizes the problem of graph bisection [9]. A general, efficient algorithm for solving it is therefore unlikely to exist, and we are forced to resort to heuristics.

4. A continuous approximation. Since solving (**P1**) is difficult, we approximate it by an easier problem. In particular, we relax the constraint that $c^{(k)}(q)=\pm 1$, which changes the discrete problem into a tractable continuous optimization problem. Unfortunately the solution to the continuous problem does not give us a valid partitioning since the $c^{(k)}(q)$'s will no longer have discrete values corresponding to the bit patterns of the target processors. We can, however, use the solution of the continuous optimization to find a nearby point satisfying the ± 1 condition. This nearby point will not generally be the absolute minimizer of (**P1**), but the hope is that it will provide a good answer in practice.

It will be convenient to reformulate (**P1**) in matrix terms. For a fixed k, consider the n values of $c^{(k)}(\rho_i)\ \forall i\in\{1,\ldots,n\}$ as an n-vector denoted by $x^{(k)}$. Introduce the weighted adjacency matrix A such that

$$(11)\qquad A(i,j)=\begin{cases} w_e(e_{ij}) & \text{if } e_{ij}\in E\\ 0 & \text{Otherwise.}\end{cases}$$

Letting $D=\mathrm{Diag}(t_i)$ and $\tau=\sum_{i=1}^{n} t_i$, the objective function in (**P1**) can be rewritten in matrix notation as

$$(12)\qquad \frac{1}{2}d(W_e-\frac{\tau}{2})+\frac{1}{4}\sum_{k=1}^{d}(x^{(k)})^T Bx^{(k)},$$

where $(x^{(k)})^T$ denotes the transpose of $x^{(k)}$, and $B = D - A$. We note that the leading constant term has no effect on the minimizer, just on the minimum value.

We set the diagonal values t_i to make each row sum of B zero. There is no compelling reason for this choice, but it is convenient for several reasons. First, since $t_i = \sum_{e_{ij} \in E} w_e(e_{ij})$ implies that $\tau = 2W_e$, the initial term in the cost is identically zero. Second, the matrix B is positive semidefinite. Furthermore, we will show that if the graph is connected then B has only a single null vector consisting of all 1's. Third, if the edge weights are all 1, then B reduces to the familiar *Laplacian* matrix of the graph — our matrix is a weighted Laplacian. We expect this to be advantageous because unweighted Laplacians have proved useful in a number of combinatorial optimization problems [18]. In particular, when used to partition graphs into two sets (a special case of what we will describe below), the Laplacian facilitates several theoretical results [2, 6, 7]. Fourth, this choice is convenient for solving the eigenvector problem arising below.

Now we relax the constraint that each of the elements of the $x^{(k)}$ vectors must be ± 1. Instead, we impose the norm constraint $||x^{(k)}||_2 = \sqrt{n}$. Combining this continuous constraint with (10), (12) and the expression for τ yields the following continuous approximation to (**P1**).

Problem P2

$$(13) \qquad \text{Minimize } \frac{1}{4}\sum_{k=1}^{d}(x^{(k)})^T B x^{(k)}$$

$$\begin{aligned} &\text{Subject to :} \\ &\text{(a1)} \quad x^{(k)} \in \mathbb{R}^n, \quad \forall k \in \{1, \ldots, d\} \\ &\text{(a2)} \quad (x^{(k)})^T x^{(k)} = n, \quad \forall k \in \{1, \ldots, d\} \\ &\text{(b)} \quad \sum_{i=1}^{n} w_v(v_i) \prod_{j \in \mathcal{S}} x^{(j)}(i) = 0, \quad \forall \mathcal{S} : \emptyset \neq \mathcal{S} \subseteq \{1, \ldots, d\}. \end{aligned}$$

We call this problem (**P2**) and note that its solution provides a lower bound for the solution of (**P1**). The advantage of approximating (**P1**) by (**P2**) is that the latter can be solved efficiently, as the next section will demonstrate.

5. Solving the continuous approximation. To solve (**P2**) we begin by focusing on a subset of the constraints. Instead of considering all of the terms of (13b), we will concentrate on only those terms involving 2 or fewer elements in the products. These terms are

$$\begin{aligned} (14) \qquad &\text{(b1)} \quad \sum_{i=1}^{n} w_v(v_i) x^{(k)}(i) = 0, \quad \forall k \in \{1, \ldots, d\} \\ &\text{(b2)} \quad \sum_{i=1}^{n} w_v(v_i) x^{(k)}(i) x^{(j)}(i) = 0, \quad \forall k, j \in \{1, \ldots, d\} : k \neq j. \end{aligned}$$

We make the important observation that these are the only constraints contained in (13b) when $d \leq 2$. In particular, if $d = 1$ (bisection), then only (b1) is relevant, and if $d = 2$ (quadrisection) (b1) and (b2) are the only terms in (13b).

To simplify, we change variables to a set of vectors $y^{(k)} \in \mathbb{R}^n$ defined by $y^{(k)}(i) = \sqrt{w_v(v_i)}x^{(k)}(i)$. Since the $x^{(k)}$ values are relaxations of ± 1, the appropriate normalization for the y vectors is $(y^{(k)})^T y^{(k)} = W_v$. Letting $s, \bar{s} \in \mathbb{R}^n$ be vectors in which

$s_i = \sqrt{w_v(v_i)}$, and $\bar{s}_i = 1/\sqrt{w_v(v_i)}$, (14) is transformed to

$$\text{(15)} \qquad \begin{array}{lll} \text{(b1)} & s^T y^{(k)} = 0, & \forall k \in \{1, \ldots, d\} \\ \text{(b2)} & (y^{(k)})^T y^{(j)} = 0, & \forall k, j \in \{1, \ldots, d\} : k \neq j. \end{array}$$

Combining (15) with (13), and letting $C = \text{Diag}(\bar{s})^T B \ \text{Diag}(\bar{s})$, we can rewrite (**P2**) as

Problem P3

$$\text{(16)} \qquad \text{Minimize} \ \frac{1}{4}\sum_{k=1}^{d}(y^{(k)})^T C y^{(k)}$$

Subject to :

$$\begin{array}{ll} \text{(a1)} & y^{(k)} \in \mathbb{R}^n, \quad \forall k \in \{1, \ldots, d\} \\ \text{(a2)} & (y^{(k)})^T y^{(k)} = W_v, \quad \forall k \in \{1, \ldots, d\} \\ \text{(b1)} & s^T y^{(k)} = 0, \quad \forall k \in \{1, \ldots, d\} \\ \text{(b2)} & (y^{(k)})^T y^{(j)} = 0, \quad \forall k, j \in \{1, \ldots, d\} : k \neq j \\ \text{(b3)} & \displaystyle\sum_{i=1}^{n} w_v(v_i)^{1-|\mathcal{S}|/2} \prod_{j \in \mathcal{S}} y^{(j)}(i) = 0, \quad \forall \mathcal{S} : \mathcal{S} \subseteq \{1, \ldots, d\}, |\mathcal{S}| > 2, \end{array}$$

which we denote by (**P3**). Next we collect a number of well known but useful observations regarding the matrix C. We note that these observations use our assumption that the weights have positive values.

THEOREM 5.1. *The matrix C has the following properties.*
(I) C is symmetric positive semidefinite.
(II) The eigenvectors of C can always be chosen to be pairwise orthogonal.
(III) The vector s is an eigenvector of C with eigenvalue zero.
(IV) If the graph is connected, s is the only eigenvector of C with eigenvalue zero.

Proof. Orient each edge in the graph arbitrarily and define the standard incidence matrix of a graph $F \in \mathbb{R}^{n \times m}$ such that

$$\text{(17)} \qquad F(i,l) = \begin{cases} 1 & \text{if } i \text{ is the initial vertex of edge } l, \\ -1 & \text{if } i \text{ is the terminal vertex of edge } l, \\ 0 & \text{if } i \text{ is not incident to edge } l. \end{cases}$$

Now define a weighted incidence matrix $G \in \mathbb{R}^{n \times m}$ as $G = \text{Diag}(\bar{s})F \ \text{Diag}(\sqrt{w_e(e_l)})$. Property (I) follows from the observation that C can be written as GG^T. Property (II) is a consequence of the symmetry of C and acknowledges that C may have multiple eigenvalues. The observation that the vector of all ones is a zero eigenvector of B yields (III). Property (IV) is a trivial generalization of Theorem 2.1c of [18], which is the same result applied to unweighted graphs. □

We now define one further minimization problem, denoted by (**P4**), to be the same as (**P3**) but with constraint (b3) removed. This is useful because (**P4**) can be solved easily, and its solution can then be used to solve (**P3**). (In fact, (**P3**) and (**P4**) are equivalent if $d \leq 2$). We define $u^{(i)}$ to be the normalized eigenvectors of C with corresponding nondecreasing eigenvalues λ_i. The solution to (**P4**) is easily expressed in terms of these eigenvectors, as the following lemmas and theorem demonstrate.

LEMMA 5.2. *Let $y^{(1)}, \ldots, y^{(d)}$ be a set of vectors that solves* (**P4**)*, and denote the span of the $y^{(k)}$ vectors by $\mathcal{Y}$. Then any set of orthogonal vectors $z^{(k)}$ that spans $\mathcal{Y}$ and satisfies $||z^{(k)}||_2 = \sqrt{W_v}$, $\forall k \in \{1, \ldots, d\}$ also solves* (**P4**).

Proof. The objective function in (**P4**) can be written as $trace(Y^TCY)$, where Y is the $n \times d$ matrix whose kth column is the vector $y^{(k)}$. Since the columns of Y are orthogonal, any orthogonal basis Z for $\mathcal{Y}$ that satisfies the normalization constraint can be written as $Z = YR$, where R is an orthonormal matrix. We now observe that $trace(Z^TCZ) = trace(R^TY^TCYR) = trace(Y^TCY)$, and the lemma follows. □

LEMMA 5.3. *If $y^{(1)}, \ldots, y^{(d)}$ solves (**P4**), then there is another set of vectors $\bar{z}^{(1)}, \ldots, \bar{z}^{(d)}$ in which $(\bar{z}^{(k)})^T u^{(i)} = 0$, if $k \geq i$ that also solves (**P4**)*

Proof. Theorem 5.1(III) coupled with constraint (b1) ensures that $(y^{(k)})u^{(1)} = 0$ for all k. Starting with the $y^{(k)}$ vectors, we can apply rotations to satisfy the remaining orthogonality constraints. Lemma 5.2 ensures that the value of the objective function is invariant with respect to these rotations. □

THEOREM 5.4. *Any set of orthogonal vectors $y^{(1)}, \ldots, y^{(d)}$ that spans $\{u^{(2)}, \ldots, u^{(d+1)}\}$ and satisfies $||y^{(k)}||_2 = \sqrt{W_v}$, $\forall k \in \{1, \ldots, d\}$ solves (**P4**). Furthermore, if $\lambda_{d+1} < \lambda_{d+2}$, then these are the only solutions to (**P4**).*

Proof. By Lemma 5.3, there is a solution $\bar{z}^{(k)}$ to (**P4**) in which $(\bar{z}^{(k)})^T u^{(i)} = 0$ if $k \geq i$. Since C is symmetric, we can rewrite $\bar{z}^{(k)}$ as a linear sum of eigenvectors of C: $\bar{z}^{(k)} = \sum_{i=1}^{n} \alpha_i^{(k)} u^{(i)}$. Now the objective function can be written

$$4\ Cost = \sum_{k=1}^{d} (\bar{z}^{(k)})^T C \bar{z}^{(k)} = \sum_{k=1}^{d} \sum_{i=1}^{n} (\alpha_i^{(k)})^2 \lambda_i. \tag{18}$$

Constraint (a2) implies $\sum_{i=1}^{n} (\alpha_i^{(k)})^2 = W_v$ for all k, and the construction of the $\bar{z}^{(k)}$ vectors guarantees that $\alpha_i^{(k)} = 0$ if $k \geq i$. Using these identities we can rewrite the cost function

$$4\ Cost = \sum_{k=1}^{d} \sum_{i=k+1}^{n} (\alpha_i^{(k)})^2 \lambda_i \tag{19}$$

$$\geq \sum_{k=1}^{d} \lambda_{k+1} \sum_{i=k+1}^{n} (\alpha_i^{(k)})^2 \tag{20}$$

$$= \sum_{k=1}^{d} \lambda_{k+1} W_v .$$

It is easy to verify that this lower bound can be achieved by letting $\bar{z}^{(k)} = \sqrt{W_v} u^{(k+1)}$. By applying Lemma 5.2, we conclude that all orthogonal bases $y^{(1)}, \ldots, y^{(d)}$ for the space spanned by $\{u^{(2)}, \ldots, u^{(d+1)}\}$ in which $||y^{(k)}||_2 = \sqrt{W_v}$ solve (**P4**).

To see that these are the only solutions to (**P4**) it is sufficient to observe that if $\lambda_{d+1} < \lambda_{d+2}$, the inequality between (19) and (20) is strict when $i > d+1$ unless $\alpha_i^{(k)} = 0$. But this implies the $\bar{z}^{(k)}$ vectors lie in the space spanned by $\{u^{(2)}, \ldots, u^{(d+1)}\}$, and the theorem follows. □

Theorem 5.4 indicates that the solution to (**P4**) is invariant under rotations of the d-dimensional space, and for $d > 1$ the space of these rotations has dimension $\binom{d}{2}$. This multiplicity of solutions is quite convenient since the continuous solution is only an approximation to the the discrete problem (**P1**). If the continuous optimization had only a single minimizer and that minimizer was far from any of the discrete points then the continuous problem could be a poor model of the discrete one. Since we have a $\binom{d}{2}$-dimensional subspace of minimizers, we have a better chance of finding a good discrete solution. These degrees of freedom also allow us to satisfy the additional constraints of (**P3**).

5.1. Spectral bisection. If we wish to divide our graph into two pieces, then (**P4**) reduces to (**P3**) since constraints (b2) and (b3) have no effect. We therefore take $y^{(1)}$ to be $\sqrt{W_v}u^{(2)}$, and let $x^{(1)}(i) = y^{(1)}(i)/\sqrt{w_v(v_i)}$. The vector $x^{(1)}$ is the continuous approximation to ± 1 values, so we need to map it to a nearby discrete point with an equal weight of $+1$ and -1 values. We do this by finding the median weighted value among all the $x^{(1)}(i)$'s and mapping values above the median to $+1$ and values below to -1. This gives a balanced decomposition, and in practice typically a low cut-weight.

Once the graph is divided into two pieces, each piece can be divided again by applying this technique recursively. For unweighted graphs, this is the partitioning procedure described by Pothen, Simon and Liou in [21] and first applied to the load balancing problem by Simon [24]. Simon found this approach to produce better partitions than coordinate bisection or graph bisection, two methods in common use in the parallel computing community.

5.2. Spectral quadrisection. Dividing the graph into four pieces requires two eigenvectors. With two eigenvectors the constraint (16b3) is unnecessary, so (**P4**) is again equivalent to (**P3**). The solutions of (**P4**) are any appropriately normalized orthogonal basis for the space spanned by $y^{(1)} = \sqrt{W_v}u^{(2)}$ and $y^{(2)} = \sqrt{W_v}u^{(3)}$. This multiplicity of solutions allows us a single rotational degree of freedom, which yields vectors of the form $\overline{y}^{(1)} = y^{(1)}\cos\theta + y^{(2)}\sin\theta$, and $\overline{y}^{(2)} = -y^{(1)}\sin\theta + y^{(2)}\cos\theta$. From the $\overline{y}$ vectors we generate x vectors whose values approximate ± 1 by $x^{(k)}(i) = \overline{y}^{(k)}(i)/\sqrt{w_v(v_i)}$. Ideally, we would like to find $\overline{y}$ vectors in which the corresponding x values are near to points with values ± 1 to help ensure that the cost of the discrete solution is not too different from the continuous optimum.

The distance from $x^{(k)}(i)$ to ± 1 can be expressed as $(1 - x^{(k)}(i)^2)^2$. Summing over each element of both k vectors, we find that we must solve

$$\text{Minimize} \sum_{i=1}^{n}\sum_{k=1}^{2}(1 - x^{(k)}(i)^2)^2. \tag{21}$$

Expanding $x^{(k)}(i)$ in terms of θ, (21) reduces to minimizing a constant coefficient quartic equation in sines and cosines of θ. The construction of the coefficients in this equation requires $O(n)$ work, but the cost of the resulting minimization problem is independent of n. Although this is a global optimization problem, in our experience the number of local minimizers is small, so a solution can be found by a sequence of local minimizations from random starting points [13].

Once $x^{(1)}$ and $x^{(2)}$ have been determined, a nearby discrete point must be found that balances the partition sizes. Our solution to this problem is described in §6.

5.3. Spectral octasection. Dividing the graph into eight pieces requires three eigenvectors. In this case, the constraints (16b1) and (16b2) are insufficient, since (16b3) generates an additional cubic constraint of the form

$$\sum_{i=1}^{n} y^{(1)}(i)y^{(2)}(i)y^{(3)}(i)/\sqrt{w_v(v_i)} = 0. \tag{22}$$

As before, the solutions of (**P4**) are any appropriately normalized orthogonal basis for the space spanned by $y^{(1)} = \sqrt{W_v}u^{(2)}$, $y^{(2)} = \sqrt{W_v}u^{(3)}$ and $y^{(3)} = \sqrt{W_v}u^{(4)}$, but these are not necessarily solutions of (**P3**). The additional constraint (16b3) removes one degree of freedom from the three-dimensional solution space for (**P3**), leaving a two-dimensional parameter space to explore.

As in §5.2, we use these remaining degrees of freedom to look for $\overline{y}$ vectors that generate x values as near as possible to ± 1. The bases for the eigenspace $\overline{y}$ can be described in terms of three rotational parameters. The $\overline{y}^{(k)}$ vectors are mapped to $x^{(k)}$ vectors by $x^{(k)}(i) = \overline{y}^{(k)}(i)/\sqrt{w_v(v_i)}$. This generates a constrained optimization problem

$$\begin{aligned} &\text{Minimize } \sum_{i=1}^{n}\sum_{k=1}^{3}(1 - x^{(k)}(i)^2)^2 \\ &\text{Subject to :} \\ &\quad \sum_{i=1}^{n} w_v(v_i)x^{(1)}(i)x^{(2)}(i)x^{(3)}(i) = 0. \end{aligned} \tag{23}$$

in which the objective function is a constant coefficient polynomial in sines and cosines of three angular parameters. The coefficients can be generated in $O(n)$ time, after which the cost of the optimization problem is independent of n. As before this is a global optimization problem, but in our experience the number of local minimizers is small, so a solution can be found by a sequence of constrained local minimizations from random starting points [8].

As in §5.2, once $x^{(1)}$, $x^{(2)}$ and $x^{(3)}$ have been determined, a nearby discrete point must be found that balances the partition sizes. Our method for solving this problem is described in §6.

5.4. Higher order partitionings. When $d > 3$ the partitioning problem becomes more difficult. The subspace defined by the set of eigenvectors of C will allow $\binom{d}{2}$ degrees of rotational freedom. However, there will be a set of $\binom{d}{3} + \cdots + \binom{d}{d}$ constraints due to (16b3). When $d > 4$, there are more constraints than degrees of freedom, so it will not generally be possible to construct a balanced solution from the $d+1$ lowest eigenvectors of C. When $d = 4$ there are six variables and five constraints, so it should be possible to satisfy all the balance conditions. However, these constraints consist of three cubic equations and one quartic, so the computational complexity of satisfying them is daunting. For this reason we have chosen not to implement any partitioning above octasection, and we suggest recursive application of one of the above schemes to divide a problem across a larger number of processors.

6. Generating a partition from real values. The procedures described in §5.2 and §5.3 generate a point in $\mathbb{R}^d$ for each vertex in the graph. These continuous points need to be mapped to points with coordinates ± 1 to determine a partition. This mapping must ensure that equal weights of vertices are assigned to each partition, and each continuous value should be mapped to a nearby discrete point.

It is useful to describe this mapping problem in terms of a complete, weighted bipartite graph $\mathcal{B} = \{V_1, V_2, \mathcal{E}\}$. The first set of vertices V_1 consists of the n vertices of our original graph, while the second set V_2 corresponds to the 2^d sets. A weighted edge $e \in \mathcal{E}$ connects each vertex $x \in V_1$ to each vertex $y \in V_2$, with weight equal to the distance between the continuous point corresponding to x and the discrete point associated with the set y. Any distance function can be used, but we chose the square of the Euclidean distance for computational convenience. There is also a vertex weight associated with each vertex $x \in V_1$, equal to the weight of the corresponding vertex in the original graph.

The optimal mapping can now be described in terms of a minimum cost assignment from V_1 to V_2 with the constraint that the sums of the vertex weights of the elements of V_1 mapped to each element of V_2 are equal. This is a generalization of a class of assignment problems considered by Tokuyama and Nakano [26], who develop an assortment of algorithms that generalize in a straightforward manner to our problem. Their best algorithm is randomized and requires $O(2^d n)$ time. We chose instead to implement one of their simpler, deterministic algorithms that runs in $O(2^{2d-1} n \log n)$ time. By exploiting the geometric structure of our particular application it is possible to reduce this time bound to $O(3^d n \log n)$.

7. Lower bounds on partitions. A known bound on the edge count for bisection of an unweighted graph is $\frac{1}{4} n\lambda_2$, where λ_2 is the second lowest eigenvalue of the Laplacian matrix of the graph (see for example [2, 18]). A simple consequence of the results in §5 is a generalization of this bound with respect to both weighting and dimensionality.

THEOREM 7.1. *The communication cost induced by cutting a graph into 2^d pieces is always at least $\frac{1}{4} W_v \sum_{i=2}^{d+1} \lambda_i$.*

Proof. Since (**P4**) is derived from (**P1**) by relaxing constraints, the minimum of (**P4**) will never be larger than that of (**P1**). Substitution of the solution in Theorem 5.4 into the cost function of (16) leads to the result. □

A better bisection bound can be determined by considering the difference between the continuous and discrete solution vectors. The continuous solution is the vector $y^{(1)} = \sqrt{W_v} u^{(2)}$ from §5.1. We let $b \in \mathbb{R}^n$ be the vector with the smallest 2-norm among all vectors such that $y^{(1)}(i) + b(i) = \pm\sqrt{w_v(v_i)}$, and let $\beta = ||b||_2^2$. We note that b (and consequently β) is easy to compute using

$$b(i) = \min\{y^{(1)}(i) - \sqrt{w_v(v_i)},\ y^{(1)}(i) + \sqrt{w_v(v_i)}\}. \tag{24}$$

THEOREM 7.2. *The bisection width of a graph is bounded by*

$$Cost \geq \frac{1}{4}\{W_v \lambda_2 + (\lambda_3 - \lambda_2)\beta(1 - \frac{\beta}{4W_v})\}. \tag{25}$$

Proof. If $c \in \{\pm 1\}^n$ is the discrete solution to (**P1**), define z to be its weighted counterpart, $z(i) = \sqrt{w_v(v_i)} c(i)$. We note that if z defines a partition, then $-z$ defines the same partition, so without loss of generality we can assume that $z^T y^{(1)} \geq 0$. We define $a \in \mathbb{R}^n$ to be the difference between z and $y^{(1)}$, so $a(i) = z(i) - y^{(1)}(i)$. We can expand a in terms of the eigenvectors of C so that $a = \sum_{j=2}^n \alpha_j u_j$, where this expansion begins at 2 since a is orthogonal to u_1. It follows from the definition of β that $\beta \leq a^T a = \sum_{j=2}^n \alpha_j^2$. Now

$$\begin{aligned} W_v = z^T z &= (y^{(1)} + a)^T (y^{(1)} + a) \\ &= (y^{(1)})^T y^{(1)} + 2a^T y^{(1)} + a^T a \\ &= W_v + 2\sqrt{W_v}\alpha_2 + a^T a, \end{aligned} \tag{26}$$

so $\alpha_2 = -a^T a/(2\sqrt{W_v})$. Since $0 \leq z^T y^{(1)} = (y^{(1)} + a)^T y^{(1)} = W_v + \sqrt{W_v}\alpha_2$, it follows that $\alpha_2 \geq -\sqrt{W_v}$, which implies that $a^T a \leq 2W_v$.

The bisection width of the graph can be expressed as

$$4\ Cost = z^T C z \tag{27}$$

$$
\begin{aligned}
&= (y^{(1)} + a)^T C (y^{(1)} + a) \\
&= (y^{(1)})^T C y^{(1)} + 2a^T C y^{(1)} + a^T C a \\
&= W_v \lambda_2 + 2\sqrt{W_v} \lambda_2 \alpha_2 + \sum_{j=2}^{n} \lambda_j \alpha_j^2 \\
&= W_v \lambda_2 - \lambda_2 a^T a + \sum_{j=2}^{n} \lambda_j \alpha_j^2 \\
&= W_v \lambda_2 + \sum_{j=2}^{n} (\lambda_j - \lambda_2) \alpha_j^2 \qquad (28)
\end{aligned}
$$

The sum in the second term of (28) is minimized when $\alpha_j = 0$ for all $j > 3$, in which case $\alpha_3^2 = a^T a - \alpha_2^2 = a^T a(1 - a^T a/(4W_v))$. This implies that

$$
(29) \qquad 4\ Cost \quad \geq \quad W_v \lambda_2 + (\lambda_3 - \lambda_2) a^T a (1 - a^T a/(4W_v)).
$$

This last term comprises a concave function in $a^T a$, so its minimum value occurs when $a^T a$ is either maximized or minimized. Using the above observations that $\beta \leq a^T a \leq 2W_v$, we obtain

$$
(30)\ 4\ Cost \quad \geq \quad \min\{W_v \lambda_2 + (\lambda_3 - \lambda_2) W_v,\ W_v \lambda_2 + (\lambda_3 - \lambda_2) \beta (1 - \beta/(4W_v))\}.
$$

But since $\beta(1 - \beta/(4W_v))$ has a maximum value of W_v, the second term of (30) always dominates and the theorem follows. □

Although the bound in Theorem 7.2 is better than previously known spectral bounds, it is still rather loose in practice and its practical value is therefore not clear. It may help in identifying classes of graphs for which the spectral method achieves near optimal results, or for proving that some particular graphs have large bisection widths.

8. Results. We have compared the quality of partitions produced by our algorithm with those generated by several other graph partitioning methods which are in common use or have been recently advocated. Our conclusion is that the improved spectral partitioning algorithm we have proposed generates significantly better partitions than these other methods, which are themselves considered to be quite good. This is based on direct experimental comparison using a variety of meshes. We have selected one representative test for this paper, a finite element meshing of a multi-element airfoil provided by Barth [1].

The airfoil mesh is shown in Fig. 1 and its *dual* is shown in Fig. 2. The dual has a vertex representing each element in the mesh (triangular faces in this case) and an edge connecting vertices representing elements which share an edge in the mesh. There are 8034 vertices and 11813 edges in this dual graph. The dual is relevant because in many parallel finite element codes, data is organized by assigning collections of individual elements to each processor. The iterative solution of the resulting equations then involves some computation associated with each element and some communication between elements sharing an edge or vertex. The dual graph therefore provides a better model for the iterative solution than the original mesh does. For ease of comparison with other methods, we chose to partition an instance of the dual in which all vertex and edge weights are equal to 1.

Table 1 shows the results obtained by applying various partitioning methods to the dual of the multi-element airfoil graph. The methods are listed in rank order

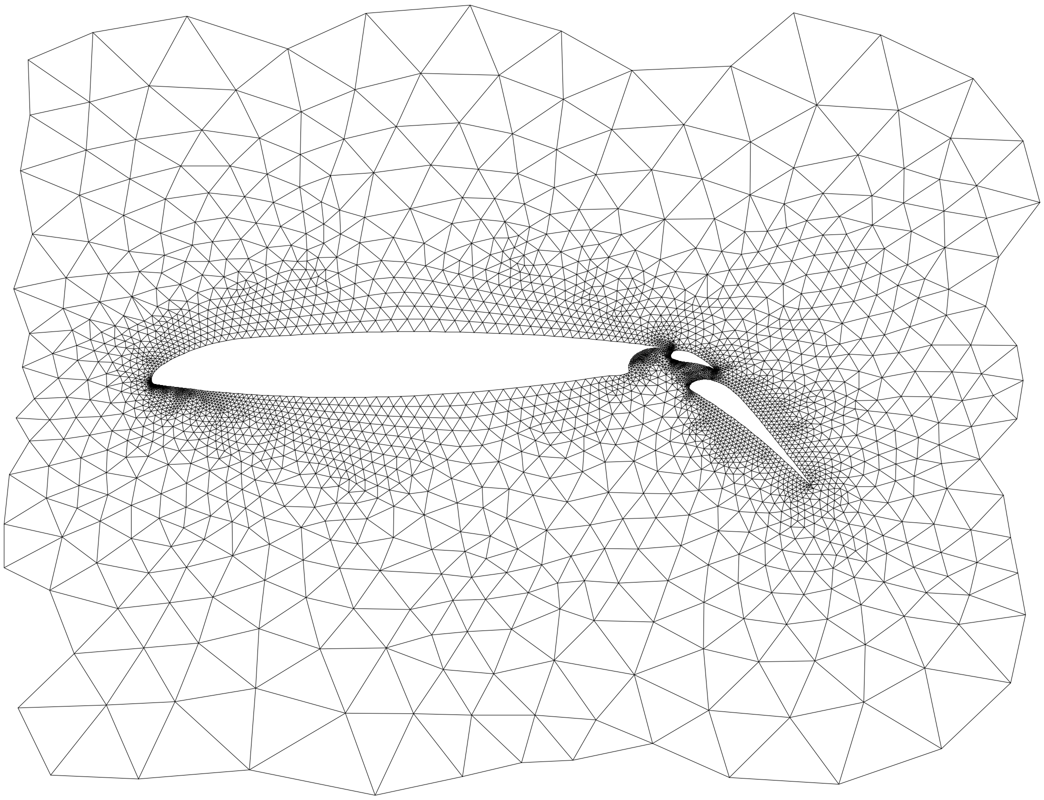

FIG. 1. *Multi-element airfoil mesh.*

by hop-weight, which has been shown to closely correlate with the overhead due to communication for the applications we are targeting [10]. A brief discussion of some important aspects of the partitioning algorithms follows.

Method	*8 Processors*		*64 Processors*	
	cuts	hops	cuts	hops
KL	300	458	1158	2183
Inertial	317	396	1166	1855
RSB	212	286	997	1661
RSQ			1030	1626
RSO	221	224	1018	1463
RSOKL	197	200	911	1287

TABLE 1
Performance of different partitioning algorithms on dual of the multi-element airfoil mesh.[1]

KL refers to a recursive application of a version of the classic graph bisection heuristic devised by Kernighan and Lin [15]. KL must be supplied with an initial partition which is then improved by a greedy local strategy. We used an x-coordinate

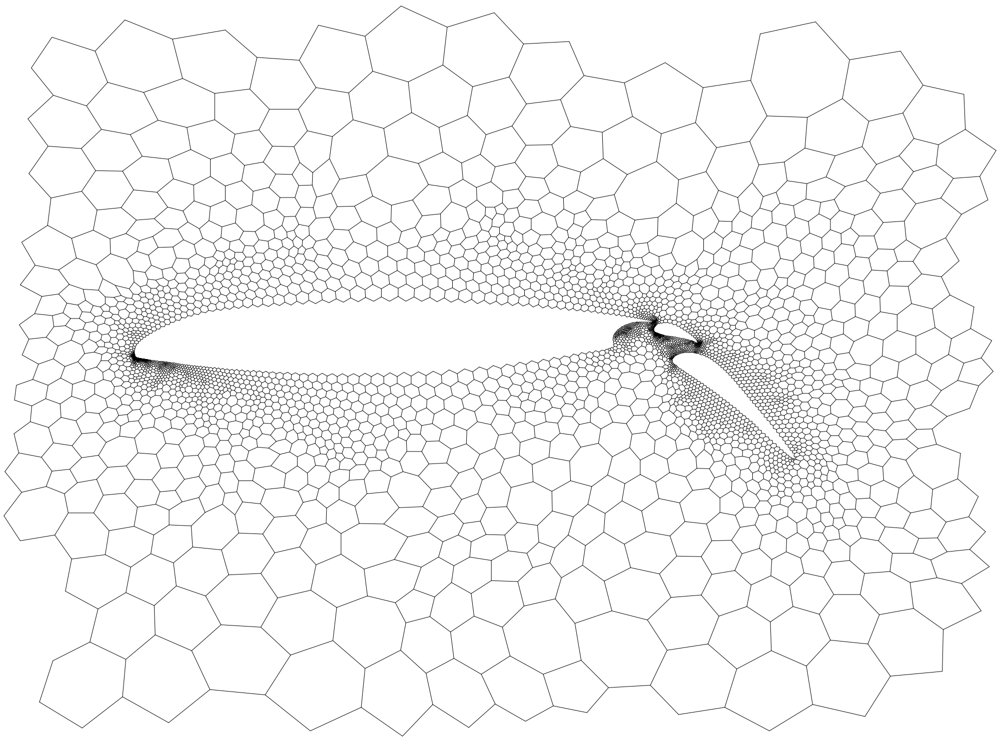

FIG. 2. *Dual of multi-element airfoil mesh.*

bisection of the vertices of the dual as an initial guess since this produced better partitions than any of the random initial guesses we tried. KL is a quick, linear time algorithm but is sensitive to the numbering of the vertices, and tends to do poorly on large problems because it only considers very local information about the graph. As with all bisection algorithms, one bit in the final processor assignment of a given vertex is determined at a time, so this algorithm makes no effort to minimize hops. It is clearly possible to add a phase to a bisection algorithm or any recursive partitioning algorithm which does try to further minimize hops by choosing an advantageous permutation of the set assignments of subgraphs. We have not used any such strategy in our experiments.

The inertial method recently proposed by Nour-Omid [24] is also a recursive bisection method. It treats the mesh as a rigid structure and makes cuts orthogonal to the principle axis of the structure. This is also a fast algorithm which can be implemented to run in linear time, but requires geometric information which may be unavailable and, as the table indicates, it produces partitions of only moderate quality.

[1] There is no 8 processor entry for RSQ because it is not possible to partition into 8 sets with an integral number of quadrisection steps.

Recursive Spectral Bisection (RSB) is the name given by Simon to the $d = 1$ spectral partitioning algorithm studied by him and others [24]. It requires no geometric information, is order insensitive and makes more sophisticated use of global information than the inertial method or KL. It produces significantly better partitions of large graphs than KL or inertial, but has an $O(n\sqrt{n})$ runtime dominated by the Lanczos iteration used to find the bisecting eigenvectors. Simon [24] and Williams [27] have both concluded that RSB is preferable to several partitioning strategies not considered here.

Recursive Spectral Quadrisection (RSQ) is our $d = 2$ spectral partitioning algorithm. Here two bits in the final processor assignment are determined concurrently to approximately minimize hops in the corresponding two hypercube dimensions. This can be at the expense of a slight increase in the cut-weight, as the table indicates. Generally only a marginal number of additional Lanczos iterations are required to compute the second eigenvector, so RSQ is often actually cheaper than two levels of RSB. If we assume that the cost of the eigenvector calculation is proportional to $n\sqrt{n}$ (which is appropriate in an idealized sense for the Lanczos procedure [20]), then a single step of RSQ is faster than two steps of RSB by a constant factor of $1 + \sqrt{2}/2$. In practice the time savings is less than this and does not always occur.

Recursive Spectral Octasection (RSO) is our $d = 3$ spectral partitioning algorithm which approximately minimizes hops in three hypercube dimensions at a time. In general it produces partitionings with fewer hops and perhaps slightly more cuts than RSQ and RSB, although it happens to do better on cuts than RSQ in this case. Assuming again that the eigenvector calculations cost $O(n\sqrt{n})$, one step of RSO would ideally be faster than three steps of RSB by a constant factor of $(3 + \sqrt{2})/2$. While this factor is not achieved in practice, the octasection algorithm is generally faster than either quadrisection or bisection.

The last algorithm, RSOKL, is a composite algorithm in which the output of RSO at each stage of recursion is fed into a generalized KL algorithm capable of minimizing hops over an 8 way initial partitioning. The motivation for this strategy was to combine the global strength of RSO with the local finesse of KL. The resulting partition is clearly the best with respect to *both* cuts and hops. The KL phase of the algorithm accounts for only a small portion of the run time, so the net cost of RSOKL is often *less* than that of RSB. Notice that for the 8 processor case the cut and hop totals are nearly equal, indicating that almost all communication occurs between adjacent processors. KL can be appended to the other algorithms as well, but we have found RSOKL to be the best combination given our communication metric.

RSO and RSQ can also be more robust than RSB when the graph exhibits symmetry. For example, the three-fold symmetry of the cubic grid-graph causes λ_2 of its Laplacian to have a multiplicity of three, and the corresponding eigenspace to be three-dimensional. Since RSB chooses a single vector from this subspace essentially at random, it may fail badly. It will, for example, make a diagonal cut through the grid for some Lanczos starting vectors. In contrast, RSO works within the entire subspace, rotating the basis vectors in such a way that it returns a gray-coding of blocks of the grid. This is the optimal result in which cuts and hops are as small as possible and all communication is between adjacent processors.

To demonstrate that this discrepancy does arise in practice, we ran both methods on a simple $4 \times 4 \times 4$ grid graph. In RSB we used the Lanczos starting vector recommended by Pothen, Simon and Liou [21], namely $r_i = i - (n+1)/2$, and iterated until the eigen-residual $Au - \lambda u$ was below 10^{-6}. The resulting decomposition had 72 cuts and 78 hops. In RSO we solved to the same accuracy for this starting vector and several random starting vectors. In each case we obtained 48 cuts and 48 hops, the optimal partitioning. Similar results may be observed with other symmetric graphs.

9. Conclusions. We have presented a method for mapping large problems onto the nodes of a hypercube multiprocessor in such a way that the computational load is balanced and the communication overhead is kept small. For the problems we have investigated, this approach generates mappings that have lower communication requirements than other partitioning techniques. Because the 2nd and 3rd eigenvectors are relatively inexpensive to calculate, the net cost of spectral quadrisection or octasection is generally less than that of spectral bisection. In addition, our method yields computable lower bounds for the communication cost of any balanced partitioning scheme which are tighter than those previously known.

Although our method was developed with a hypercube communication network in mind, this approach should work well for other machine topologies. For example, a two-dimensional mesh can be defined as a collection of two-dimensional hypercubes, so a recursive application of our quadrisection approach is immediately applicable. Similarly, a three-dimensional mesh is composed of three-dimensional hypercubes, so our octasection algorithm can be applied. For other architectures we expect our approach to be useful as a heuristic. Although the method tries to minimize a communication function that counts hypercube hops, in practice the spectral quadrisection and octasection algorithms divide a domain into pieces that require a small communication volume. This should lead to low communication overhead on most parallel machines.

Graph partitioning also finds application in network design, circuit layout, sparse matrix computations and a number of other disciplines. Consequently, the partitioning algorithm we have described may find uses far afield from parallel computing. More broadly, the way we have made use of multiple eigenvectors is, to our knowledge, unlike any previous work in spectral graph theory. It is our hope that these ideas can be applied to other spectral graph theoretic problems.

Acknowledgements. The ideas in this paper have evolved through discussions with many people including Horst Simon, Alex Pothen, Louis Romero, Ray Tuminaro, John Shadid and Steve Plimpton.

REFERENCES

[1] T. Barth. Personal Communication, December 1991.

[2] R. Boppana, *Eigenvalues and graph bisection: An average case analysis*, in Proc. 28th Annual Symposium on Foundations of Computer Science, IEEE, 1987, pp. 280–285.

[3] W. Donath and A. Hoffman, *Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices*, IBM Technical Disclosure Bulletin, 15 (1972), pp. 938–944.

[4] ———, *Lower bounds for the partitioning of graphs*, IBM J. Res. Develop., 17 (1973), pp. 420–425.

[5] C. M. Fiduccia and R. M. Mattheyses, *A linear time heuristic for improving network partitions*, in Proc. 19th IEEE Design Automation Conference, IEEE, 1982, pp. 175–181.

[6] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Math. J., 23 (1973), pp. 298–305.

[7] ———, *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory*, Czechoslovak Math. J., 25 (1975), pp. 619–633.

[8] R. Fletcher, *Practical Methods of Optimization, Volume 2, Constrained Optimization*, John Wiley & Sons, New York, 1986.

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

[10] S. Hammond, *Mapping unstructured grid computations to massively parallel computers*, PhD thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY, 1992.

[11] B. Hendrickson and R. Leland, *An improved spectral load balancing method*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 953–961.

[12] ———, *Multidimensional spectral load balancing*, Tech. Rep. SAND 93-0074, Sandia National Laboratories, Albuquerque, NM, January 1993.

[13] J. J. Dennis and R. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[14] L.-M. Jin and S.-P. Chan, *A genetic approach to network partitioning*, Intl. J. Computer Math., 42 (1992), pp. 47–60.

[15] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal, 29 (1970), pp. 291–307.

[16] N. Mansour, *Physical optimization algorithms for mapping data to distributed memory multiprocessors*, PhD thesis, Syracuse University, School of Computer Science, Syracuse, NY, 1992.

[17] G. L. Miller, S.-H. Teng, and S. A. Vavasis, *A unified geometric approach to graph separators*, in Proc. 32nd Symp. Foundations of Computer Science, IEEE, October 1991, pp. 538–547.

[18] B. Mohar, *The Laplacian spectrum of graphs*, in 6th Intl. Conf. Theory and Applications of Graphs, Kalamazoo, MI, 1988.

[19] B. Nour-Omid, A. Raefsky, and G. Lyzenga, *Solving finite element equations on concurrent computers*, in Parallel computations and their impact on mechanics, A. K. Noor, ed., American Soc. Mech. Eng., New York, 1986, pp. 209–227.

[20] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[21] A. Pothen, H. Simon, and K. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal., 11 (1990), pp. 430–452.

[22] D. Powers, *Graph partitioning by eigenvectors*, Lin. Alg. Appl., 101 (1988), pp. 121–133.

[23] F. Rendl and H. Wolkowicz, *A projection technique for partitioning the nodes of a graph*, Tech. Rep. CORR 90-20, University of Waterloo, Faculty of Mathematics, Waterloo, Ontario, November 1990.

[24] H. D. Simon, *Partitioning of unstructured problems for parallel processing*, in Proc. Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergammon Press, 1991.

[25] P. Suaris and G. Kedem, *An algorithm for quadrisection and its application to standard cell placement*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 294–303.

[26] T. Tokuyama and J. Nakano, *Geometric algorithms for a minimum cost assignment problem*, in Proc. 7th Annual Symposium on Computational Geometry, ACM, 1991, pp. 262–271.

[27] R. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency, 3 (1991), pp. 457–481.

[28] D. Womble, *A time–stepping algorithm for parallel computers*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 824–837.