ENHANCING DATA LOCALITY BY USING TERMINAL PROPAGATION*

BRUCE HENDRICKSON†, ROBERT LELAND‡, AND RAFAEL VAN DRIESSCHE§

Abstract.
Terminal propagation is a method developed in the circuit placement community for adding constraints to graph partitioning problems. This paper adapts and expands this idea, and applies it to the problem of partitioning data structures among the processors of a parallel computer. We show how the constraints in terminal propagation can be used to encourage partitions in which messages are communicated only between architecturally near processors. We then show how these constraints can be handled in two important partitioning algorithms, spectral bisection and multilevel–KL. We compare the quality of partitions generated by these algorithms to each other and to partitions generated by more familiar techniques.

1. Introduction. To perform a computational task on a parallel computer it is first necessary to partition the task into pieces and to map the pieces to different processors. In many calculations the underlying computational structure can be conveniently modeled as a graph in which vertices correspond to computational tasks and edges reflect data dependencies. The partitioning and mapping problems can then be then be addressed by assigning processor labels to vertices of the graph so that the corresponding assignment of tasks to processors leads to efficient execution.

Graph partitioning in this context has been an active area of research recently, and many new and effective strategies have been developed. Much less attention, however, has been paid to the mapping problem. When the mapping problem has been considered, it has typically been addressed as a post-processing problem in which the pieces of a a given partitioning must be assigned to processors in an intelligent fashion.

This primary emphasis on partitioning is justified by the impact a partition has on communication within a parallel computer. The number of graph edges cut in a partition typically corresponds to the volume of communication in the parallel application, and, since communication is an expensive operation, minimizing this volume is extremely important in achieving high performance. Mapping, in contrast, does not affect communication volume. Furthermore, with current parallel hardware technology, the cost of an isolated message between architecturally distant processors is only marginally greater than that of a message between nearest neighbors.

Nevertheless, mapping quality is still very important. A message between distant processors must traverse many wires, which are therefore rendered unavailable to transmit other messages. Conversely, if each message consumes only a small number of wires, more messages can be sent at once. It is in this competition for wires that a good mapping can be distinguished from a bad one. More formally we say that a

* To appear in Proc. 29th Hawaii Intl. Conf. System Sciences.

† Sandia National Labs, Albuquerque, NM 87185-1110. Email: bah@cs.sandia.gov.
‡ Sandia National Labs, Albuquerque, NM 87185-1109. Email: leland@cs.sandia.gov
§ Dept. Computer Sciences, Katholieke Universiteit Leuven, Belgium.
Email: Rafael.VanDriessche@cs.kuleuven.ac.be.

good mapping is one that reduces message *congestion* and thereby preserves communication *bandwidth.* Many scientific computing applications of interest, for example those employing an iterative sparse solver kernel, have a structure in which many messages simultaneously compete for limited communication bandwidth, and good mappings are especially important in these cases.

In such problems the simple, two-phased approach in which the mapping is decoupled from the partitioning may be effective. But this is intuitively not optimal because it does not allow for trading–off between partition and mapping quality. Ideally, the partitioning and mapping should be generated together in such a way that some aggregate cost metric is minimized.

This paper describes a general framework for coupling recursive partitioning schemes[1] to the mapping problem and shows how to apply it to two important algorithms, multilevel–KL and spectral bisection. Our approach is based upon an idea taken from the circuit placement community known as *terminal propagation* in which the result of one partitioning step in the recursion is used to constrain subsequent steps. The constraints effectively transmit mapping information between partitioning problems.

As a simple illustration consider the mesh depicted in left side of Fig. 1, and to the right its partition into four sets using the popular spectral bisection algorithm [22]. The mesh was first sliced horizontally, and then the two halves were divided independently. Although the interfaces between the regions are quite small, the region just above the horizontal cut is adjacent to all the others. Consequently, this decomposition can not be mapped to a hypercube or mesh topology in such a way that all communication is between neighboring processors.

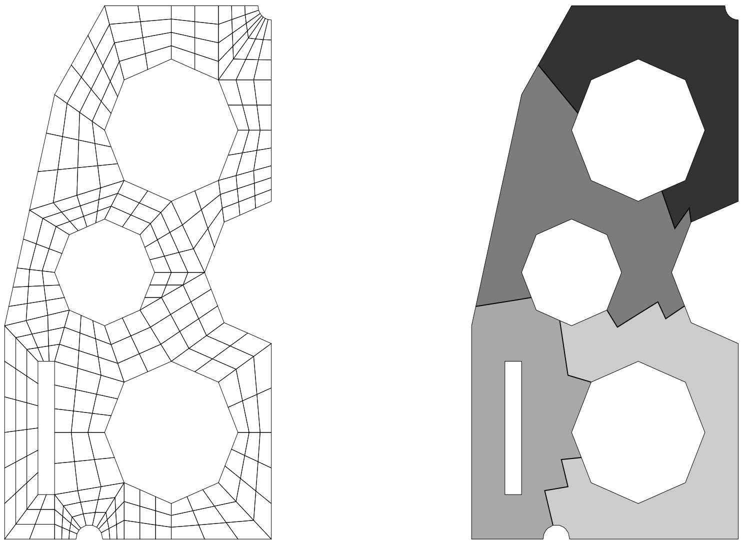

Fig. 1. Simple mesh (left) and its spectral bisection decomposition (right).

However, if we partition the same mesh using the terminal propagation variant of spectral bisection which we describe in §5.2, we obtain one of the two decompositions

[1] Most partitioning methods are recursive, but some, *e.g.* the greedy method described in [6], are not. The method described in this paper does not apply to these non–recursive methods.

depicted in Fig. 2. Here we perform two cuts exactly as before, but in the third cut we include constraints to encourage a partition in which only neighboring processors need communicate. In both cases, the interfaces remain small, but the resulting decomposition can now be mapped optimally to a hypercube or mesh.

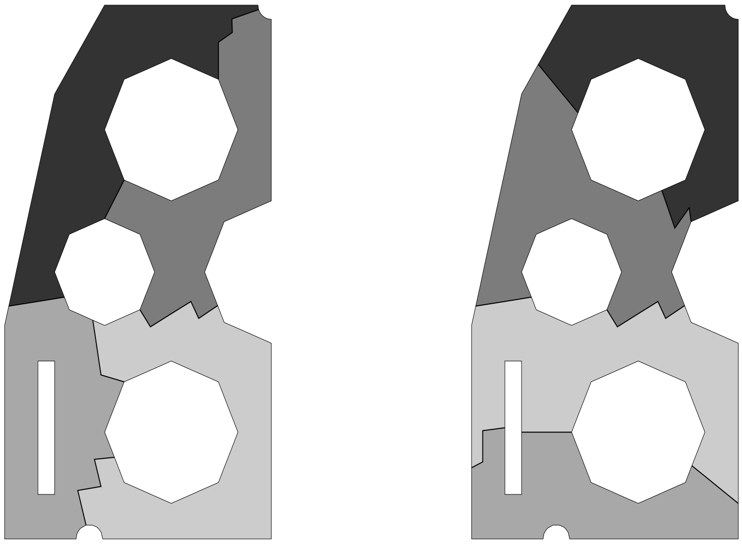

Fig. 2. Two decompositions of the simple mesh produced by spectral terminal propagation.

In the next section we describe the terminal propagation idea, showing how it couples recursive partitioning and mapping. In §3 we review an important partitioning algorithm from the circuit community and show how it has been extended to include terminal propagation. In §4 we extend this technique to incorporate it in a multilevel partitioning approach. An enhanced spectral partitioning algorithm including terminal propagation is described in §5. We present experimental results obtained with these new methods in §6.

2. Terminal propagation. Most of the graph partitioning algorithms being used today were developed by researchers in the circuit placement community. When placing circuit elements on a chip, it is important to keep the wire lengths as short as possible. This saves valuable space on the chip and helps to keep transmission delays small. One important methodology for positioning circuit elements involves partitioning the graph which describes the circuit. Typically, the circuit is partitioned into two pieces of approximately equal size with few wires crossing between them. The chip area is similarly divided, and the two circuit halves are placed in the two chip halves. This process is now repeated recursively on each half–problem. Since few wires cross between the two halves, most wires are localized and so kept short.

This simple approach has an important shortcoming. Since the two halves are completely decoupled, there is no longer any mechanism to minimize the length of the wires which cross between them For instance, consider dividing the circuit and chip area into quarters as shown in Fig. 3. In the first step, we divide the circuit in half, assigning one part to the left half of the chip and the other to the right half. Next we divide the left half circuit again, assigning the resulting pieces to the upper and lower left quadrants. Now consider a wire that was cut in the first partition, and assume its left endpoint is located in the lower left quadrant (at, for example, point 1). Clearly, it would be preferable from the point of view of minimizing wire length if its right endpoint were assigned to the lower right quadrant at point 2 rather than the upper right quadrant at point 3, but simple partitioning algorithms are too shortsighted to recognize this.

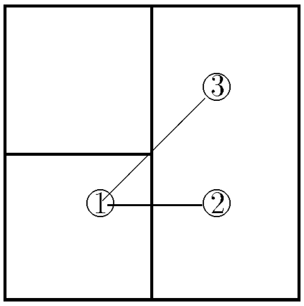

Fig. 3. The basic motivation for terminal propagation in the circuit layout context. We are about to partition the right half of the chip and, in order to minimize wire length, and would prefer the end of the edge connected to point 1 to land in the lower quadrant at point 2 rather than the upper quadrant at point 3. Ordinary recursive partitioning schemes cannot distinguish the two locations.

It was to address this myopia that introduced the concept of terminal propagation in [5]. Their approach was intimately coupled with the popular partitioning algorithm due to Kernighan and Lin [15], but we describe it here more generally to allow adaptation of the underlying idea to other recursive partitioning algorithms. The basic idea of terminal propagation is to associate with each vertex in the subgraph being partitioned a value which reflects its net desire or *preference* to be in the top quadrant instead of the bottom quadrant. Note that this preference is a function only of edges that connect the vertex to vertices which are not in the current subgraph. The name *terminal propagation* comes from circuit layout applications in which there are additional constraints of this type which come from the wires which connect to the boundary of the chip at specified locations. These *terminals* impose preferences upon how subgraphs should be partitioned, and these preferences are *propagated* through the recursive partitioning process.

An analogous problem arises in parallel computing. Consider a graph describing a computation which we want to partition among processors. The usual manner for addressing this problem involves dividing the graph into two pieces, and assigning the two pieces to halves of the parallel machine. We can apply this approach recursively until each processor is assigned a unique piece of the graph. Unfortunately, this approach does not include any consideration of architectural distance between processors. Since edges between subproblems are ignored in the recursion, messages may end up traversing many wires. In the language of the previous section, the mapping is decoupled from the partitioning. This is essentially the same problem that occurs in circuit placement, which motivates the use of terminal propagation.

In the parallel computing context we need a slightly different but closely related interpretation of the terminal propagation which we depict in Fig. 4. The quadrants now represent processors or sets of processors of (for example) a hypercube or a 2–dimensional mesh architecture. The (sets of) processors can identified by a 2 bit code and the number of wires necessary to traverse between two processors is the number of bits in which their processor identifiers differ. Assume we have already partitioned the graph into two pieces and assigned them to left and right halves of the computer, and that we have similarly divided the left half–graph into top and bottom quadrants. When partitioning the right half–graph between processors 10 and 11 we would like messages to travel short distances. The mapping shown in the left hand figure is better since it results in a total message distance of $2 \times 1 + 2 = 4$ whereas the mapping in the right hand figure induces a total cost of $2 \times 2 + 1 = 5$.

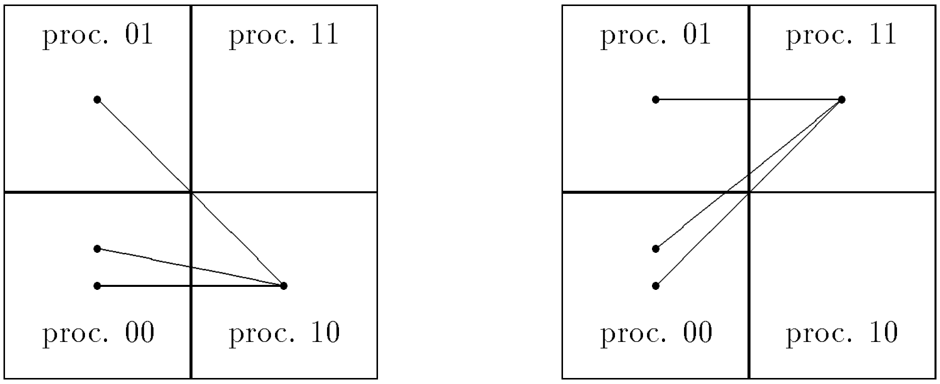

Fig. 4. The terminal propagation idea in the parallel computing context. Here we are defining the partition between processors 10 and 11. The mapping on the left is preferable because it induces a total hypercube (or mesh) communication cost of 4 hops, whereas that on the right induces a cost of 5 hops.

Phrasing this argument in terms of preference values, since the vertex in question has two neighbors in the lower left quadrant and one in the upper left, its preference to be in the lower right quadrant will be 1. If the edges to these external vertices had weights, the preference values would be scaled appropriately. Now, instead of partitioning to minimize just the number of edges crossing between the upper and lower right quadrants, we minimize the sum of the number of cross edges and the unsatisfied preferences.

This objective function can be phrased algebraically. When working on a subproblem, we construct a preference value for each vertex based upon the edges with connect it to vertices in other subproblems. Say we are deciding whether to place vertex i in one partition or the other, and i is connected to a vertex j which is not in the current subproblem. The edge between i and j contributes a value to the preference equal to

$w_e(e_{ij})(D_2 - D_1)$, where $w_e(e_{ij})$ is the weight of the edge, D_1 is the architectural distance between i and j if i is placed in the first partition and D_2 is the architectural distance between i and j if i is placed in the second partition. If desired, we can also scale the vector of preferences to adjust in our metric the relative importance of architectural locality versus communication volume.

With this setup we can now state the problem formally. Let $G = (V, E)$ be a graph with vertices $v \in V$ and edges $e_{ij} \in E$. We allow either edges or vertices to have positive weights associated with them, which we denote by $w_v(v)$ and $w_e(e_{ij})$ respectively. We will use n (and m) to denote the number of vertices (and edges) in the graph. Assume we want to divide V into two subsets V_1 and V_2, and that we have a a vector d of preferences for the vertices of V to be in V_1. The cost associated with a partition now has two components. First, every edge $e_{ij} \in E_1$ crossing between V_1 and V_2 contributes a value of $w_e(e_{ij})$. Second, for each vertex in V_1 with a negative preference we add the magnitude of the preference to the cost, and similarly for each vertex in V_2 with a positive preference. Our goal is to find a partition of the vertices into two sets of nearly equal size in which this combined cost function is minimized.

Unfortunately, this problem is NP–hard, so an efficient, general algorithm is unlikely to exist. In the next two sections we will describe two heuristics for this problem that generalize popular techniques for the unconstrained partitioning problem.

3. The algorithm of Kernighan/Lin and Fiduccia/Mattheyses.

3.1. Standard KL/FM. In 1970, Kernighan and Lin proposed a heuristic for partitioning graphs based upon greedy exchange of vertices to reduce the number of edges cut by a partition [15]. Their basic approach has been enhanced and improved through the years, most significantly by Fiduccia and Mattheyses who devised a linear time variant [7]. This approach to partitioning is often referred to as KL/FM after these four authors. Most of the work on this algorithm, including the two papers cited above, was motivated by the circuit placement problem.

The KL/FM algorithm is a technique for improving an initial, perhaps random partition. The key notion is that of the *gain* of a vertex, the net reduction in cuts which would ensue if the vertex were moved to the other partition. The basic step is selecting a vertex with the highest gain value and moving it to the other partition.

There are two details which add complexity and considerable power to this very simple idea. First, in order to keep sets from becoming unbalanced, only moves between equal sized sets or from the larger to the smaller are allowed. Second, the algorithm continues trying to move vertices even if doing so makes the partition temporarily worse. The hope is that this reduction in quality will be compensated for by a larger improvement later on. This was the key insight of Kernighan and Lin's paper and makes the approach superior to a simple greedy algorithm.

The algorithm thus consists of two nested loops as depicted in Fig. 5. In the inner loop, vertices whose movement would maximally improve the partition are selected, subject to set size constraints. Once a vertex is moved, the gain values of all its neighbors are updated. A particular vertex is allowed to move just once during each pass through the outer loop. The best partition encountered in this sequence of moves is recorded, and the outer loop resets the current partition to this best partition.

```
Best Partition := Current Partition
Until No better partition is discovered
    Compute all initial gains
    Until Termination criteria reached
        Select vertex to move
        Perform move
        Update gains of all neighbors of moved vertex
        If Current Partition balanced and better than Best Partition Then
            Best Partition := Current Partition
    End Until
    Current Partition := Best Partition
End Until
```

Fig. 5. An algorithm for refining graph partitions.

The main contribution of Fiduccia and Mattheyses was to cast Kernighan and Lin's algorithm in the form depicted in Fig. 5, and to show how each pass through the outer loop could be performed in linear time if edge weights were integers. The key idea is to compute all gains at the beginning of the outer loop and store them in an efficient data structure. Move selection and gain value updates can then be performed in constant time. Within the inner loop, gain values are never computed from scratch, but rather are changed incrementally.

3.2. KL/FM with Terminal Propagation. The paper by Kernighan and Dunlop which introduced the concept of terminal propagation [5] described a simple enhancement to KL/FM that allows inclusion of terminal propagation considerations. There are a number of details in their paper which are relevant to circuit placement problems, but here we merely extract the essential idea.

First we add an additional, special vertex to each partition which is not allowed to switch partitions. Now for each normal vertex in the subproblem with positive preference to be in partition 1, we add an edge to the special vertex in partition 1 with a weight equal to this preference. Otherwise, we add an edge to the special vertex in partition 2 with a weight equal to the negative of this preference. Now when the KL/FM algorithm is run, the external edge information is internalized in the connections to the special vertices.

Another, more elegant approach is possible when using a Fiduccia/Mattheyses type implementation in which gain values are only computed once and are updated incrementally thereafter. The preferences are included in the initial gain calculations while the rest of the code remains unchanged. Specifically, if a vertex is in set 1 and its preference is positive, the initial gain should include a contribution equal to the negative of the preference. If that same vertex is initially in set 2, the initial gain should include a term equal to the the preference. Similar considerations apply to vertices with negative preferences. The advantage of this second approach is that the basic KL/FM loop need not be modified at all. In contrast, the first approach requires code to handle special vertices which are not allowed to move, and additional storage for all the edges incident to the special vertices.

4. Multilevel–KL.

4.1. Standard Multilevel–KL. The primary shortcoming of the KL/FM algorithm is that it enacts only local modifications to a partition. Although it is quite effective at finding local minimums, its solution may be quite far from the global optimum. This is particularly true for large graphs.

One possible remedy is to initialize KL/FM with a partition generated by another algorithm, for example the spectral bisection method discussed in §5.1. An alternate approach, suggested independently by several authors [2, 12] is to apply KL/FM on different scales. One way to think of this is as an algebraic multigrid technique in which KL/FM serves as the smoothing operator.

Such a *multilevel–KL* algorithm consists of three phases, as sketched in Fig. 6. First, a sequence of successively smaller graphs is generated from the original graph. Next, the smallest graph in the sequence is partitioned using some technique. This partition is then propagated back through the sequence of intermediate graphs, with KL/FM refinement being applied to some partitions of the intermediate graphs.

```
(1)    Until graph is small enough
           graph := coarsen(graph)

(2)    Partition graph

(3)    Until graph = original graph
           graph := uncoarsen(graph)
           partition := uncoarsen(partition)
           locally refine partition if desired.
```

Fig. 6. A multilevel algorithm for graph partitioning.

It is important that the small graphs represent their larger counterparts as accurately as possible. In the partitioning context, there are two basic properties we would like to preserve the in construction of the smaller graphs: the cost of a partition should be accurately preserved, and the set sizes should be properly encoded so that a balanced partition of the small graph is also a balanced partition of the larger graph. These properties are preserved by the algorithm discussed here and in [12].

The key mechanism in the construction of a small graph is an operation known as *edge contraction*. In this step, two vertices joined by an edge are merged, and the resulting vertex is given edges to the union of the neighbors of the two merged vertices. The new vertex is assigned a weight equal to the sum of the weights of its constituent vertices. Edge weights are not changed unless both merged vertices are adjacent to the same neighbor. In this case, the new edge that represents the two original edges is assigned a weight equal to the sum of the weights of the edges it replaces. So, for example, contracting one edge of a triangle with unit edges and vertex weights would yield a graph with a vertex of weight one and a vertex of weight two, joined by an edge of weight two.

The attractive feature of this contraction step is that it preserves cut and set sizes in a *weighted* sense. A partition of a small graph implies a partition of a larger graph since each vertex in the small graph is merely an amalgamation of vertices of the larger one. The total weight of small graph edges that are cut in the partition will be precisely equal to the total weight of the edges cut in the larger graph. Similarly, the total weight of vertices in each of the two small graph sets is exactly equal to the weight of the vertices in the corresponding partition of the large graph.

To construct a small graph from a larger one we need to contract a number of edges. Ideally, these edges will be well distributed throughout the large graph so the overall shape of the small graph will be similar to that of its larger counterpart. One way to do this is to select a maximal set of edges that share no vertices. Such a set is known as a maximal matching, and can be easily generated in linear time.

4.2. Multilevel–KL with Terminal Propagation. The multilevel approach can be enhanced to include terminal propagation in a fairly straightforward way. Since we are applying KL/FM on the smaller graphs, we can apply the terminal propagation variant of KL/FM. There are only two issues that need to be addressed. First, what partitioner should be used on the smallest graph? And second, how are preferences generated for small graphs?

One suitable answer to the first question is the spectral bisection algorithm with terminal propagation described in the next section. An alternate approach would be to use the original Kernignan and Lin strategy of applying KL/FM to random initial partitions. If the smallest graph is small enough, this should work well.

The second question, how to produce preferences for small graphs, is also easily answered. Consider a vertex of a small graph, which is a union of large graph vertices. The small graph vertex will generally be connected to some set of vertices not in the current subproblem. It is the total pull of these edges which determines the preference for the vertex. But this total pull is just the sum of preferences of the large graph vertices which comprise the small graph. Hence, when contracting an edge, the resulting vertex should be assigned a preference which is equal to the sum of the preferences of the two original vertices.

5. Spectral bisection with terminal propagation. An important class of partitioning algorithms known as *spectral methods* uses eigenvectors of a matrix associated with the graph to generate a partition. This surprising connection dates back to work in the early 70s by Fiedler [8, 9] and Donath and Hoffman [3, 4]. A particular spectral method that has come to be known as spectral bisection gained widespread acceptance in the parallel computing community following the work of Pothen, Simon and Liou [20] and Simon [22]. In this section we briefly rederive the spectral bisection algorithm for weighted graphs developed in [14], and then show how it can be modified to incorporate terminal propagation constraints.

5.1. Standard Spectral Bisection. One way to describe a partition is to assign a value of $+1$ to all the vertices in one set and a value of -1 to all the vertices in the other. If we denote the value assigned to vertex i by $x(i)$, then the simple function $(x(i)-x(j))^2/4$ is equal to 1 if vertices i and j are in different partitions and 0 otherwise. This allows us to write the partitioning problem as

$$
\begin{aligned}
&\text{Minimize } f(x) = \frac{1}{4}\sum_{e_{ij}\in E} w_e(e_{ij})(x(i)-x(j))^2 \\
&\text{Subject to} \\
&\text{(a)}\quad \sum_i w_v(i)x(i) \approx 0 \\
&\text{(b)}\quad x(i) = \pm 1.
\end{aligned}
\tag{1}
$$

Constraint (a) is an algebraic way of saying that each partition must have about half the total vertex weight. We do not specify it as a precise equality since it may not be possible to divide the vertices into two sets of precisely equal weight.

Recasting the partitioning problem this way does not make it any easier to solve. However, it does identify a possible approximation that will lead to a much simpler problem. Rather than insisting that all x's be exactly ± 1, we allow them to take on any value and consequently replace constraint (b) with a norm condition on the vector x of values $x(i)$. Once we solve the resulting continuous problem, we can find the ± 1 vector which is nearest to the continuous optimum, and use this to partition the graph. Although this strategy does not guarantee that the optimal solution will be found, it works well in practice.

More formally, we approximate (1) by the following.

$$
\begin{aligned}
&\text{Minimize } f(x) = \frac{1}{4}\sum_{e_{ij}\in E} w_e(e_{ij})(x(i)-x(j))^2 \\
&\text{Subject to} \\
&\text{(a)}\quad \sum_i w_v(i)x(i) = 0 \\
&\text{(b)}\quad \sum_i x(i)^2 = n.
\end{aligned}
\tag{2}
$$

We have replaced the previous constraint (b) with a normalization which is appropriate for the ± 1 problem. We have also changed constraint (a) to a strict equality, since this can be achieved in the continuous problem.

The next step is an algebraic transformation of the objective function. It is not hard to show that

$$\sum_{e_{ij}\in E} w_e(e_{ij})(x(i)-x(j))^2 = x^T L x$$

where L is the *Laplacian matrix* of the graph defined by

$$L(i,j) = \begin{cases} \sum_{e_{ik} \in E} w_e(e_{ik}) & \text{if } i = j \\ -w_e(e_{ij}) & \text{if } e_{ij} \in E \\ 0 & \text{Otherwise.} \end{cases}$$

The Laplacian matrix has a number of nice properties. It is symmetric, so it has a complete set of orthonormal eigenvectors, and it is positive semidefinite. Because the sum of all the values in a row is zero, the constant vector is an eigenvector with eigenvalue zero. If the graph is connected, all other eigenvalues are positive. Using this definition of L, Eq. (2) can now be rewritten in matrix terms:

(3)
$$\begin{aligned} &\text{Minimize } f(x) = \frac{1}{4} x^T L x \\ &\text{Subject to} \\ &\text{(a)} \quad w_v^T x = 0 \\ &\text{(b)} \quad x^T x = n. \end{aligned}$$

With a change of variables we can reduce (3) to a form in which we will recognize that its solution is equivalent to the solution of a standard eigenproblem. First define $s(i) = \sqrt{w_v(i)}$ and $t(i) = 1/s(i)$. Let $y = \text{Diag}(s)x$, and let $A = \text{Diag}(t)^T L \text{Diag}(t)$. Since the x values are relaxations of ± 1, the appropriate normalization for the y vector is $y^T y = \sum_i w_v(i)$, which we denote by ω_v. With this notation, we can recast (3) as follows.

(4)
$$\begin{aligned} &\text{Minimize } f(y) = \frac{1}{4} y^T A y \\ &\text{Subject to} \\ &\text{(a)} \quad s^T y = 0 \\ &\text{(b)} \quad y^T y = \omega_v. \end{aligned}$$

It is straightforward to verify that s is an eigenvector of A with eigenvalue zero. A is symmetric and positive semi–definite. Furthermore, if the graph is connected, s is the only eigenvector with a zero eigenvalue. (See [14] for proofs of these properties.)

Now denote the eigenvalues and corresponding normalized eigenvectors of A by λ_i and u_i respectively, where the eigenvalues are indexed in increasing value. We can now write the solution to (4) as $\sum_{i=1,n} \alpha_i u_i$ for some constants α_i, where constraint (b) means that $\sum_i \alpha_i^2 = n$. Constraint (a) disallows any contribution from the first eigenvector, so the summations can start at 2. The value of $y^T A y$ can now be expressed as $\sum_{i=2,n} \alpha_i^2 \lambda_i$, which, of $\lambda_2 < \lambda_3$ is clearly minimized when $\alpha_2 = \sqrt{n}$ and all other α's are zero.

Thus the solution to (4) is a multiple of the second eigenvector of A. This vector can be easily transformed to find the solution to (2), which can be used to find a nearby discrete point which partitions the graph. The whole procedure is sketched in Fig. 7. The second eigenvector of a Laplacian matrix is often known as a Fiedler vector in recognition of the pioneering work of Miroslav Fiedler [8, 9].

Form L, the Laplacian matrix of the graph
Generate $A = \mathrm{Diag}(t) L \mathrm{Diag}(t)$
Compute y = second lowest eigenvector of A
Generate $x = \mathrm{Diag}(t) y$
Find median value γ among entries in x
Partition 1 = vertices with x value less than or equal to γ
Partition 2 = vertices with x value greater than γ.

Fig. 7. The weighted spectral bisection algorithm

The dominant cost of this spectral bisection algorithm is the calculation of an eigenvector of L. The traditional approach to this problem is the Lanczos algorithm [10], an iterative method in which each iteration is dominated by a matrix–vector multiplication. Barnard and Simon have described a multilevel eigensolver that can significantly speed up the standard spectral bisection algorithm [1].

5.2. Spectral Bisection with Terminal Propagation. In [23, 24] Van Driessche and Roose show how to modify the standard spectral bisection algorithm to include certain kinds of constraints. The original motivation for their work was reduction of data movement in a dynamic repartitioning, but their basic idea can also be applied to the constraints associated with terminal propagation. As with the standard spectral technique, the basic idea is to construct a discrete optimization problem and then to relax the discreteness constraint.

In the standard derivation (1) we began with an algebraic formulation of the exact partitioning problem. We now need to enhance the objective function to include terminal propagation considerations. Since we are working in the terminal propagation context we now need to distinguish between the subgraph we are currently partitioning and the remainder of the graph which may impose constraints on this partitioning. If $d(i)$ is the preference for a vertex to be in the set denoted by $+1$ which will define to be, say, V_1, then the new problem we want to solve is

$$\text{(5)} \qquad \begin{aligned} &\text{Minimize } f(x) = \frac{1}{4} \sum_{e_{ij} \in \underline{E}} w_e(e_{ij})(x(i) - x(j))^2 - \frac{1}{2} \sum_{i \in \underline{V}} d(i) x(i) \\ &\text{Subject to} \\ &\text{(a)} \quad \sum_{i \in \underline{V}} w_v(i) x(i) \approx 0 \\ &\text{(b)} \quad x(i) = \pm 1. \end{aligned}$$

We now make the same approximation as in the standard spectral bisection problem, replacing constraint (b) by a normalization condition to obtain

$$\text{(6)} \qquad \begin{aligned} &\text{Minimize } f(x) = \frac{1}{4} x^T L x - \frac{1}{2} d^T x \\ &\text{Subject to} \\ &\text{(a)} \quad w_v^T x = 0 \\ &\text{(b)} \quad x^T x = n. \end{aligned}$$

We now make the same variable transformation used earlier to take us from (3) to (4). If we also let $h = \mathrm{Diag}(t)d$ and multiply the objective function by 4, we have

$$\begin{aligned} &\text{Minimize } f(y) = y^T A y - 2h^T y \\ &\text{Subject to} \\ &\text{(a)} \quad s^T y = 0 \\ &\text{(b)} \quad y^T y = \omega_v. \end{aligned} \tag{7}$$

Unfortunately, the solution to (7) is not as simple as the solution to (4), its standard counterpart. We introduce Lagrange multipliers η and μ, and look for stationary points of the function

$$F(y, \eta, \mu) = y^T A y - 2h^T y + \eta(s^T y) + \mu(\omega_v - y^T y). \tag{8}$$

Setting the partial derivative of F with respect to η or μ yields the two constraint equations. Taking the derivatives with respect to the components of y, we obtain

$$2Ay - 2h + \eta s - 2\mu y = 0. \tag{9}$$

We can calculate η by left multiplying (9) by s^T. Since s is orthogonal to y and s is a zero eigenvector of A, we discover that $\eta = 2s^T h/\omega_v$. We now define

$$g = h - \frac{s^T h}{\omega_v} s, \tag{10}$$

which allows us to rewrite (9) as

$$Ay = \mu y + g. \tag{11}$$

This *extended eigenproblem* must be solved subject to the constraints in (7). Although this problem generally has multiple solutions, Van Driessche and Roose have shown that the solution which minimizes the objection function is always the y vector associated with the smallest possible value for μ [23]. As with the standard spectral bisection approach, once a solution to (11) is computed, it is transformed back to a solution of (6), from which a nearby discrete solution can be found.

5.3. Solving the Extended Eigenproblem. Van Driessche and Roose [23] describe a Lanczos–like algorithm for solving this extended eigenproblem, and similar techniques can be found in [11]. Unfortunately, Barnard and Simon's multilevel eigensolver [1] does not seem amenable to solving the extended eigenproblem. Here we briefly review the technique developed in [23], show how to monitor its convergence in a more traditional manner than in [23] and comment on its extension to incorporate selective reorthogonalization.

Our problem is to find the smallest μ and the corresponding y such that

$$\begin{aligned} &Ay = \mu y + g \\ &\text{Subject to} \\ &\text{(a)} \quad s^T y = 0 \\ &\text{(b)} \quad y^T y = \omega_v. \end{aligned} \tag{12}$$

We defer treatment of the constraint equations and consider first the extended eigenproblem. Multiplying through by $Q_j^T \in \mathbb{R}^{j\times n}$, the transpose of the Lanczos basis at step j, and looking for a solution of form $y = Q_j v$ we obtain

$$Q_j^T A Q_j v = \mu Q_j^T Q_j v + Q_j g. \tag{13}$$

From the standard Lanczos process we have after j steps that $Q_j^T A Q_j = T_j + Q_j^T r_j e_j^T$ where T_j is a tridiagonal matrix, Q_j is orthogonal, r_j is the residual vector and $e_j = (1,1,1,\ldots)^T$. Hence (13) becomes $(T_j + Q_j^T r_j e_j^T)v = \mu v + Q_j^T g$. Since in exact arithmetic $||Q_j^T r_j e_j^T||$ is zero, we have $(T_j - \mu I)v = Q_j^T g$ where we seek the pair (μ, v) corresponding to the left-most value of μ such that the applicable constraints are satisfied. We can simplify this a bit further by exploiting a degree of freedom in the Lanczos process, the choice of the initial basis vector. If we set $q_i = g/||g||$ we have, given the orthogonality of Q, that $Q_j^T g = Q_j^T(||g||q_1) = ||g||e_1$ where $e_1 = (1,0,0,0\ldots)^T$. Substituting we obtain

$$(T_j - \mu I)v = ||g||e_1, \tag{14}$$

and have effectively transformed the extended eigenproblem into Lanczos space.

We now address the constraint equations. Using $y = Q_j v$ the norm constraint on v becomes $\omega_v = y^T y = v^T Q_j^T Q_j v = v^T v = \omega_v$. The orthogonality constraint can be enforced during the Lanczos process. We compute the jth approximation to y as $y_j = \sum_{i=1}^{j} v(i) q_i$, so to force $s^T y$ to be zero we insist that $s^T Y_j = \sum_{i=1}^{j} v(i) s^T q_i = 0$. Hence the constraint is satisfied if $s^T q_i = 0$ for $i \in [2,\ldots,j]$. Although in exact arithmetic $s^T q_i = 0$, with finite precision the q vectors may not be orthogonal to s. We can enforce this by orthogonalizing the residual (and hence Lanczos vectors) against s. Then we just require $s^T q_1 = ||g||s^T g = 0$ which is true by the construction of g given in (10).

Solve $(T_j - \mu I)v = \|\|g\|\|e_1$
    s.t. $v^T v = \omega_v$ (for the left-most pair (μ, v))
where T_j is computed by applying the Lanczos iteration to
    $Ay = \mu y + g$
with $q_1 = \frac{g}{\|\|g\|\|}$ and $s^T q_i = 0 \quad 2 \le \forall i \le j$
enforced by explicit orthogonalization.
Compute
    $y = \sum_{i=1}^{j} v(i) q_i$.

Fig. 8. An algorithm for the extended eigenproblem.

The development thus far is summarized in Fig. 8. Solution of the shifted tridiagonal system there is not simply a matter of a linear solve because the norm constraint on v must also be satisfied. We therefore use an iterative approach. That is we guess a value of μ, solve for v and check whether $v^T v = \omega_v$ and adjust our guess at μ accordingly. The simple bisection algorithm shown in Fig. 9 can be used for this. There are a few rare cases in which this must be modified which are detailed in [23]. The tridiagonal linear solve called for in fig. (9) can now be solved by a standard algorithm; see for example [10].

Compute the first eigenpair (θ_1, z_1) of T
$\mu_l := \theta_1 - \frac{\|\|g\|\|}{\sqrt{\omega_v}}$
$\mu_h := \theta_1 - \frac{\|\|g\|\|}{\sqrt{\omega_v}} z(1)$
Assign tol for μ accuracy
$\delta := 2 \times tol$
While $(\delta > tol)$
    $\mu = (\mu_l + \mu_h)/2$
    Solve $(T - \mu I) = \|\|g\|\| e_1$
    If $v^T v \leq \omega_v$ then $\mu_l = \mu$
    If $v^T v \geq \omega_v$ then $\mu_h = \mu$
    $\delta := \mu_h - \mu_l$

Fig. 9. A bisection algorithm for the norm constrained linear system.

A remaining issue is convergence monitoring. We need an inexpensive way of estimating the norm of the eigenresidual $r(y) = Ay - \mu y - g$. Recall $y = Q_j v$, $q_1 = g/||g||$ and, from the Lanczos process, $T_j v = \mu v + Q_j^T g$ and $AQ_j = Q_j T_j + r_j e_j^T$. We can therefore write

$$
\begin{aligned}
r(y) &= Ay - \mu y - g \\
&= AQ_j v - \mu Q_j v - g \\
&= (Q_j T_j + r_j e_j^T) v - \mu Q_j v - g \\
&= Q_j(\mu v + Q_j^T g) + r_j e_j^T v - \mu Q_j v - g \\
&= Q_j Q_j^T g + r_j e_j^T v - g \\
&= Q_j Q_j^T ||g|| q_1 - ||g|| q_1 + r_i e_i^T v \\
&= r_j e_j^T v
\end{aligned} \tag{15}
$$

Hence

$$
||r(y)|| = ||r_j e_j^T v|| = ||r_j v_j|| = \beta_j |v_j| \tag{16}
$$

where, in standard Lanczos notation, β_j is the last off-diagonal in T_j. This means that our residual estimate is essentially free since we must already compute v in order to check the norm constraint.

This residual estimate will be accurate only if orthogonality is maintained between the Lanczos basis vectors, the columns of Q_j. Maintaining this orthogonality reliably is in practice a difficult task to which much research has been devoted; see for example [16, 18, 19, 21]. The method we have found most effective in the graph partitioning context is a modified version of the algorithm known as selective reorthogonalization due to Parlett and Scott [19]. This method relies on a key result due to Paige [16, 17] characterizing the loss of orthogonality among the Lanczos basis vectors. Essentially this result says that the loss of orthogonality is in the direction of the well–converged approximate eigenvectors $y_i = Q_j z_i$, where z_i is the ith eigenvector of T_j. Hence if the approximate eigenvector which converges first happens to be the (sole) eigenvector we seek, this loss of orthogonality is not detrimental and is in fact to be welcomed as an indication of convergence. The Lanczos process converges in most cases to the extremal eigenvectors first, and in the spectral bisection algorithm we are seeking the left–most eigenvector (assuming we project out the s eigenvector), so very often no reorthogonalization is needed. We have found, however, that the addition of selective reorthogonalization leads to a more efficient and reliable algorithm.

Paige's result also shows that the magnitude of orthogonality loss is essentially inversely proportional to $\beta_j z_i(j)$:

$$y_i^T q_{j+1} = \frac{g_{ii}}{\beta_j z_i(j)} \tag{17}$$

where g_{ii} is an element of a round-off error matrix. This cheaply computable estimate is the key to development of an efficient reorthogonalization algorithm. Since we are computing T_j and Q_j in exactly the same manner for the extended eigenproblem as we would in the standard eigenproblem, the reorthogonalization algorithm is unchanged. There is, however, one possible point of confusion. Notice that $\beta_j z_i(j)$ term is very similar to the expression given for the residual magnitude in (16). In fact when solving the standard eigenproblem rather than the extended eigenproblem these terms are identical – both orthogonality loss and convergence are monitored in terms of $z_i(j)$. But in the extended eigenproblem it is $v(j)$, not $z(j)$ which should be used in monitoring convergence.

6. Results. The algorithms described in the previous sections have been implemented in **Chaco 2.0** [13], and we report some experimental results here. All the runs were performed a Sun Sparcstation 20 with a 50MHz clock and 64 Mbytes of RAM. We will describe results from four different algorithms:

MLKL: the multilevel–KL method from §4.1,

MLKLTP: the multilevel–KL algorithm with terminal propagation described in §4.2,

Spec+KL: spectral bisection from §5.1 combined with a pass of standard KL/FM from §3.1, and

SpecTP+KL spectral bisection with terminal propagation as presented in §5.2 combined with the terminal propagation version of KL/FM discussed in §3.2.

For the spectral algorithms we solved to residual tolerances of 10^{-3}, and we used a variant of Barnard and Simon's multilevel eigensolver for standard spectral bisection. For multilevel–KL and the multilevel eigensolver, the smallest graph had at most 200 vertices.

We monitored four metrics of the quality of the partitioner. First was the number of edges cut, which corresponds closely to the total communication volume. Second was *hops* in which we multiply each cut edge by the architectural distance between the two processors owning the endpoints. Third was *messages* which is the total number of messages required to execute a step of an iterative solver using the decomposition. The final metric was the time required to generate the decomposition.

Our first example graph is *barth5*, a 2D finite element grid with triangular elements containing 15606 vertices, and 45878 edges[2] The results of partitioning and mapping this graph to a 6–dimensional hypercube are presented in Table 1.

Method	MLKL	MLKLTP	Spec+KL	SpecTP+KL
cuts	2844	3187	2959	3530
hops	4832	3594	5052	3892
messages	288	322	282	360
time (secs)	10.1	12.2	30.8	32.6

Table 1. Results of using different partitioning algorithms to decompose the barth5 mesh for a 6–dimensional hypercube.

As expected, terminal propagation significantly improves the data locality as evidenced by the significant reduction in hops. The average distance a datum has to travel is reduced from 1.7 to 1.1 in both algorithms. This comes at the cost of a modest increase in communication volume as reflected by the increase in the cuts metric, as well as an increase in number of messages. The time required to perform the partitioning is slightly increased by the use of terminal propagation.

For our next experiment we partitioned the *ocean* mesh among the processors of a 10×20 mesh. This a 3D finite difference grid of the world's oceans comprised of about 143K vertices and 410K edges. The results are presented in Table 2. Note that for this problem, we need to be able to bisect into two sets of unequal size. This is straightforward to do with the multilevel–KL method, and a generalization to this case of spectral bisection with terminal propagation is described in [23].

Method	MLKL	MLKLTP	Spec+KL	SpecTP+KL
cuts	37847	45715	38365	52292
hops	103672	60210	103870	63163
messages	1652	1174	1614	1104
time (secs)	157.6	186.7	690.3	477.5

Table 2. Results of using different partitioning algorithms to decompose the ocean mesh for a 10×20 grid.

Again we observe that terminal propagation significantly improves locality, reducing the average number of wires traversed by a message from 2.7 to 1.3 in the multilevel–KL algorithm, and from 2.7 to 1.2 in the spectral method. As before this locality is paid for by an increase in communication volume. However, unlike the previous

[2] This, and and other meshes, can be obtained via anonymous ftp to riacs.edu in the directory /pub/grids.

problem the number of messages is significantly reduced by terminal propagation. Since communication is local and meshes have many fewer processors in their neighborhood than hypercubes, this result isn't surprising. For this problem, the spectral terminal propagation algorithm was significantly faster than its standard counterpart. We don't know why this happened since we used RQI/Symmlq as the standard eigensolver.

From these and similar experiments we make several observations.

- Terminal propagation is an effective approach for coupling recursive applications of partitioning with the desire to restrict communication to nearby processors. This is evidenced by the fact that the cuts and hops values are very similar in all the tables where terminal propagation was used, while the cuts and startups are only modestly larger than those obtained by traditional algorithms which ignore interprocessor distances.
- For the fairly nice graphs associated with scientific computing, the multilevel–KL algorithm produces partitions at least as good as spectral+KL, both with and without terminal propagation, while requiring significantly less time.
- For meshes, and for large hypercubes, terminal propagation usually results in fewer messages needing to be sent.
- We have also observed that Lanczos with terminal propagation is typically somewhat faster than standard Lanczos, perhaps because we initialize the iteration with an intelligent rather than a random vector.

7. Conclusion. We have described a general method for coupling the partitioning and mapping problems in such a way that contention for communication links is significantly reduced. In applications where many messages are simultaneously competing for limited bandwidth, this approach may significantly improve performance. The general idea can undoubtedly be applied to a wide variety of recursive partitioning methods. Here we have focused on two techniques which are currently popular in the parallel computing community. The approach presented is sufficiently flexible to allow for the user to weight the relative importance of cuts and hops and hence trade off communication volume and message congestion. More generally, we believe there are likely to be other important ideas which can be adapted from from the circuit placement community to assist with parallel computing.

The techniques described in this paper can be extended in several ways. The KL/FM terminal propagation algorithm can be generalized to work on more than two sets at once. This leads to a similar generalization of the multilevel terminal propagation scheme. (Both algorithms are implemented in **Chaco 2.0**.) The spectral terminal propagation method can also be extended to work on four sets at once [25], and in principle it can be extended to work on eight sets simultaneously as well.

Acknowledgements. Hendrickson and Leland were supported by the Applied Mathematical and Computer Sciences program, U.S. Department of Energy, Office of Energy Research, and work at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AC04-76DP00789. Van Driessche was supported by the Belgian Incentive Program "Information Technology"—Computer Science of the Future

(IT/IF/5), and of the Belgian Programme on Interuniversity Poles of Attraction (IUAP 17), initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture. The scientific responsibility for this paper rests with its authors.

REFERENCES

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

[2] T. Bui and C. Jones, *A heuristic for reducing fill in sparse matrix factorization*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 445–452.

[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] A. E. Dunlop and B. W. Kernighan, *A procedure for placement of standard–cell VLSI circuits*, IEEE Trans. CAD, CAD-4 (1985), pp. 92–98.

[6] C. Farhat and H. Simon, *TOP/DOMDEC - a software tool for mesh partitioning and parallel processing*, Tech. Rep. RNR-93-011, NASA Ames Research Center, Moffett Field, CA 94035, June 1993.

[7] 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.

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

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

[10] G. Golub and C. Van Loan, *Matrix Computations, Second Edition*, Johns Hopkins University Press, Baltimore, MD, 1989.

[11] G. Golub and U. von Matt, *Quadratically constrained least squares and quadratic problems*, Numer. Math., 59 (1991), pp. 561–580.

[12] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, Tech. Rep. SAND93–1301, Sandia National Laboratories, Albuquerque, NM, October 1993.

[13] ——, *The Chaco user's guide, version 2.0*, Tech. Rep. SAND94–2692, Sandia National Laboratories, Albuquerque, NM, October 1994.

[14] ——, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM J. Sci. Comput., 16 (1995).

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

[16] C. C. Paige, *Computational variants of the Lanczos method for the eigenproblem*, J. Inst. Math. Applics., 10 (1972), pp. 373–381.

[17] ——, *Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix*, J. Inst. Math. Applics., 18 (1976), pp. 341–349.

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

[19] B. Parlett and D. Scott, *The Lanczos algorithm with selective orthogonalization*, Math. Comp., 33 (1979), pp. 217–238.

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

[21] H. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.

[22] 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.

[23] R. Van Driessche and D. Roose, *A spectral algorithm for constrained graph partitioning I: The bisection case*, TW Report 216, Department of Computer Science, Katholieke Universiteit

Leuven, Belgium, October 1994.

[24] ———, *Dynamic load balancing with a spectral bisection algorithm for the constrained graph partitioning problem*, in High-Performance Computing and Networking, no. 919 in Lecture Notes in Computer Science, Springer, 1995, pp. 392–397. Proceedings of the International Conference and Exhibition, Milan, Italy, May 1995.

[25] ———, *A spectral algorithm for constrained graph partitioning II: The bisection case*, Tech. Rep., Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1995. In preparation.