Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations

James D. Teresco[1], Karen D. Devine[2], and Joseph E. Flaherty[3]

[1] Department of Computer Science, Williams College, Williamstown, MA 01267 USA terescoj@cs.williams.edu
[2] Discrete Algorithms and Mathematics Department, Sandia National Laboratories, Albuquerque, NM 87185 USA kddevin@sandia.gov
[3] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 USA flaherje@cs.rpi.edu

Summary. In parallel simulations, partitioning and load-balancing algorithms compute the distribution of application data and work to processors. The effectiveness of this distribution greatly influences the performance of a parallel simulation. Decompositions that balance processor loads while keeping the application's communication costs low are preferred. Although a wide variety of partitioning and load-balancing algorithms have been developed, their effectiveness depends on the characteristics of the application using them. In this chapter, we review several partitioning algorithms, along with their strengths and weaknesses for various PDE applications. We also discuss current efforts toward improving partitioning algorithms for future applications and architectures.

The distribution of data among cooperating processes is a key factor in the efficiency of parallel solution procedures for partial differential equations (PDEs). This distribution requires a data-partitioning procedure and distributed data structures to realize and use the decomposition. In applications with constant workloads, a static partition (or static load balance), computed in a serial or parallel pre-processing step, can be used throughout the computation. Other applications, such as adaptive finite element methods, have workloads that are unpredictable or change during the computation, requiring dynamic load balancers that adjust the decomposition as the computation proceeds. Partitioning approaches attempt to distribute computational work equally, while minimizing interprocessor communication costs. Communication costs are governed by the amount of data to be shared by cooperating processes (communication volume) and the number of partitions sharing the data (number of messages). Dynamic load-balancing procedures should also operate in parallel on distributed data, execute quickly, and minimize data

movement by making the new data distribution as similar as possible to the existing one. The partitioning problem is defined in more detail in Section 1.

Numerous partitioning strategies have been developed. The various strategies are distinguished by trade-offs between partition quality, amount of data movement, and partitioning speed. Characteristics of an application (e.g., computation-to-communication ratio, cost of data movement, and frequency of repartitioning) determine which strategies are most appropriate for it. For example, geometric algorithms like recursive bisection and space-filling curve partitioning provide high-speed, medium-quality decompositions that depend only on geometric information (e.g., particles' spatial coordinates, element centroids). Graph-based algorithms provide higher quality decompositions based on connectivity between application data, but at a higher cost. Several strategies, with their relative trade-offs, are described in detail in Section 2 and Section 3.

Many partitioning procedures have been implemented directly in applications, using application-specific data structures. While this approach can provide high execution efficiency, it usually limits the application to a single procedure and burdens the application programmer with partitioning concerns. A number of software libraries are available that provide high-quality implementations of partitioning procedures, provide flexibility to switch among available methods, and free the application programmer from those details. Some of these software packages are described in Section 4.

While existing methods have been very successful, research challenges remain. New models, such as hypergraphs, can more accurately model communication. Multi-criteria partitioning can improve efficiency when different phases of a computation have different costs. Resource-aware computation, achieved by adjusting the partitioning or other parts of the computation according to processing, memory and communication resources, is needed for efficient execution on modern hierarchical and heterogeneous computer architectures. Current research issues are explored further in Section 5.

1 The Partitioning and Dynamic Load Balancing Problems

The most common approach to parallelizing PDE solution procedures assigns portions of the computational domain to cooperating processes in a parallel computation. Typically, one process is assigned to each processor. Data are distributed among the processes, and each process computes the solution on its local data (its *subdomain*). Inter-process communication provides data that are needed by a process but "owned" by a different process. This model introduces complications including (*i*) assigning data to subdomains (i.e., *partitioning*, or when the data is already distributed, *dynamic load balancing*), (*ii*) constructing and maintaining distributed data structures that allow for efficient data migration and access to data assigned to other processes, and

(*iii*) communicating the data as needed during the solution process. The focus of this chapter is on the first issue: data partitioning.

1.1 The Partitioning Problem

The computational work of PDE simulation is often associated with certain "objects" in the computation. For particle simulations, computation is associated with the individual particles; adjusting the distribution of particles among processors changes the processor load balance. For mesh-based applications, work is associated with the entities of the mesh — elements, surfaces, nodes — and decompositions can be computed with respect to any of these entities or to a combination of entities (e.g., nodes and elements). The partitioning problem, then, is the division of objects into groups or subdomains that are assigned to cooperating processes in a parallel computation.

At its simplest, a partitioning algorithm attempts to assign equal numbers of objects to partitions while minimizing communication costs between partitions. A partition's subdomain, then, consists of the data uniquely assigned to the partition; the union of subdomains is equal to the entire problem domain. For example, Figure 1 shows a two-dimensional mesh whose elements are divided into four subdomains. Often communication between partitions consists of exchanges of solution data for adjacent objects that are assigned to different partitions. For example, in finite element simulations, "ghost elements" representing element data needed by but not assigned to a subdomain are updated via communication with neighboring subdomains. While this data distribution is the most commonly used one for parallelization of PDE applications (and, indeed, will be assumed without loss of generality in the rest of this chapter), other data layouts are possible. In Mitchell's full-domain partition (FuDoP) [77], for example, each process is assigned a disjoint subdomain of a refined mesh. Then within each process, a much coarser mesh is generated for the rest of the problem domain, giving each process a view of the entire domain. This layout reduces the amount of communication needed to update subdomain boundary values during adaptive multigrid, at the cost of extra degrees of freedom and computation. A similar idea has been applied to parallel solution procedures by Bank and Holst to reduce communication costs for elliptic problems [3].

Objects may have weights proportional to the computational costs of the objects. These nonuniform costs may result from, e.g., variances in computation time due to different physics being solved on different objects, more degrees of freedom per element in adaptive p-refinement [1, 105], or more small time steps taken on smaller elements to enforce timestep contraints in local mesh-refinement methods [42]. Similarly, nonuniform communication costs may be modeled by assigning weights to connections between objects. Partitioning then has the goal of assigning equal total object weight to each subdomain while minimizing the weighted communication cost.

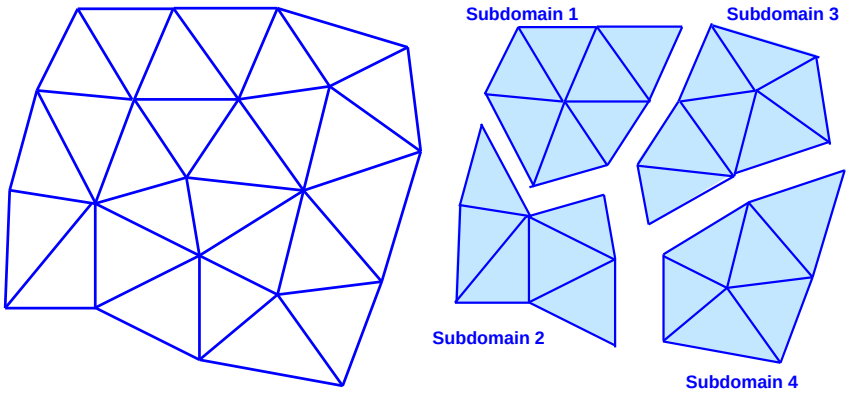

Fig. 1. An example of a two-dimensional mesh (left) and a decomposition of the mesh into four subdomains (right).

1.2 Dynamic Repartitioning and Load Balancing Problem

Workloads in dynamic computations evolve in time, so a partitioning approach that works well for a static problem or for a slowly-changing problem may not be efficient in a highly dynamic computation. For example, in finite element methods with adaptive mesh refinement, process workloads can vary dramatically as elements are added and/or removed from the mesh. Dynamic repartitioning of mesh data, often called dynamic load balancing, becomes necessary.

Dynamic repartitioning is also needed to maintain geometric locality in applications like crash simulations and particle methods. In crash simulations, for example, high parallel efficiency is obtained when subdomains are constructed of geometrically close elements [96]. Similarly, in particle methods, particles are influenced by physically near particles more than by distant ones; assigning particles to processes based on their geometric proximity to other particles reduces the amount of communication needed to compute particle interactions.

Dynamic load balancing has the same goals as partitioning, but with the additional constraints that procedures (*i*) must operate in parallel on already distributed data, (*ii*) must execute quickly, as dynamic load balancing may be performed frequently, and (*iii*) should be incremental (i.e., small changes in workloads produce only small changes in the decomposition) as the cost of redistribution of mesh data is often the most significant part of a dynamic load-balancing step. While a more expensive procedure may produce a higher-quality result, it is sometimes better to use a faster procedure to obtain a lower-quality decomposition, if the workloads are likely to change again after a short time.

1.3 Partition Quality Assessment

The goal of partitioning is to minimize time to solution for the corresponding PDE solver. A number of statistics may be computed about a decomposition that can indicate its suitability for use in an application.

The most obvious measure of partition quality is computational load balance. Assigning the same amount of work to each processor is necessary to avoid idle time on some processors. The most accurate way to measure imbalance is by instrumenting software to determine processor idle times. However, imbalance is often reported with respect to the number of objects assigned to each subdomain (or the sum of object weights, in the case of non-uniform object computation costs).

Computational load balance alone does not ensure efficient parallel computation. Communication costs must also be considered. This task often corresponds to minimizing the number of objects on sharing data across subdomain boundaries, since the number of adjacencies on the bounding surface of each subdomain approximates the amount of local data that must be communicated to perform a computation. For example, in element decompositions of mesh-based applications, this communication cost is often approximated by the number of element faces on boundaries between two or more subdomains. (In graph partitioning, this metric is referred to as "edge cuts"; see Section 2.2.) A similar metric is a subdomain's *surface index*, the percentage of all element faces within a subdomain that lie on the subdomain boundary. Two variations on the surface index can be used to estimate the cost of interprocess communication. The *maximum local surface index* is the largest surface index over all subdomains, and the *global surface index* measures the percentage of all element faces that are on subdomain boundaries [14]. In three dimensions, the surface indices can be thought of as surface-to-volume ratios if the concepts of surface and volume are expanded beyond conventional notions; i.e., the "volume" is the whole of a subdomain, and the elements on subdomain boundaries are considered the "surface." The global surface index approximates the total communication volume, while the maximum local surface index approximates the maximum communication needed by any one subdomain.

A number of people [14, 50, 111] have pointed out flaws in minimizing only the edge cut or global surface index statistics. First, the number of faces shared by subdomains is not necessarily equal to the communication volume between the subdomains [50]; an element could easily share two or more faces with elements in a neighboring subdomain, but the element's data would be communicated only once to the neighbor (Figure 2). Second, interconnection network latency is often a significant component of communication cost; therefore, interprocess connectivity (the number of processes with which each process must exchange information during the solution phase) can be as significant a factor in performance [14] as the total volume of communication. Third, communication should be balanced, not necessarily minimized [95]. A

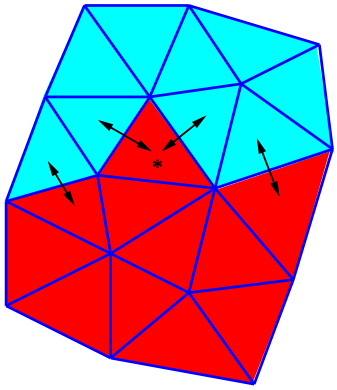

Fig. 2. Example where the number of elements on the subdomain boundary is not an accurate measure of communication costs. The shading indicates subdomain assignments. The element indicated by "*" needs to send its value to two neighbors in the other subdomain, but the value need only be communicated once.

balanced communication load often corresponds to a small maximum local surface index.

Another measure of partition quality is the internal connectivity of the subdomains. Having multiple disjoint connected components within a subdomain (also known as *subdomain splitting* [57]) can be undesirable. Domain decomposition methods for the solution of the linear systems will converge slowly for partitions with this property [25, 38]. Additionally, if a relatively small disjoint part of one subdomain can be merged into a neighboring subdomain, the boundary size will decrease, thereby improving the surface indices.

Subdomain aspect ratio has also been reported as an important factor in partition quality [32, 38], particularly when iterative methods such as Conjugate Gradient (CG) or Multigrid are used to solve the linear systems. Diekmann, et al. [32] provide several definitions of subdomain aspect ratio, the most useful being the ratio of the square of the radius of smallest circle that contains the entire subdomain to the subdomain's area. They show that the number of iterations needed for a preconditioned CG procedure grows with the subdomain aspect ratio. Furthermore, large aspect ratios are likely to lead to larger boundary sizes.

Geometric locality of elements is an important indicator of partition effectiveness for some applications. While mesh connectivity provides a reasonable approximation to geometric locality in some simulations, it does not represent geometric locality in all simulations. (In a simulation of an automobile crash, for example, the windshield and bumper are far apart in the mesh, but can be quite close together geometrically.) Geometric locality is also important in particle methods, where a natural representation of connectivity is not often available. Quality metrics based on connectivity are not appropriate for these types of simulations.

2 Partitioning and Dynamic Load Balancing Taxonomy

A variety of partitioning and dynamic load balancing procedures have been developed. Since no single procedure is ideal in all situations, many of these alternatives are commonly used. This section describes many of the approaches, grouping them into geometric methods, global graph-based methods, and local graph-based methods. Geometric methods examine only coordinates of the objects to be partitioned. Graph-based methods use the topological connections among the objects. Most geometric or graph-based methods operate as global partitioners or repartitioners. Local graph-based methods, however, operate among neighborhoods of processes in an existing decomposition to improve load balance. This section describes the methods; their relative merits are discussed in Section 3.

2.1 Geometric Methods

Geometric methods are partitioners that use only objects' spatial coordinates and objects' computational weights in computing a decomposition. For example, in mesh partitioning, any mesh entities' coordinates (e.g., nodal coordinates, element centroids, surface element centroids) can be used. Geometric methods assign objects that are physically close to each other to the same partition in a way that balances the total weight of objects assigned to each partition. This goal is particularly effective for applications in which objects interact only if they are geometrically close to each other, as in particle methods and crash simulations.

Recursive Bisection

Recursive bisection methods divide the simulation's objects into two equally weighted sets; the bisection algorithm is then applied to each set until the number of sets is equal to the number of desired partitions. (This description implies that the number of partitions must be a power of two; however, only minor changes in the algorithm are needed to allow an arbitrary number of partitions.)

Perhaps the most well-known geometric bisection method is Recursive Coordinate Bisection (RCB), developed by Berger and Bokhari [9]. In RCB, two sets are computed by cutting the problem geometry with a plane orthogonal to a coordinate axis (see Figure 3, left). The plane's direction is selected to be orthogonal to the longest direction of the geometry; its position is computed so that half of the object weight is on each side of the plane. In a twist on RCB, Jones' and Plassmann's Unbalanced Recursive Bisection (URB) algorithm [61] halves the problem geometry (instead of the set of objects) and then assigns processes to each half proportionally to the total object weight within the half.

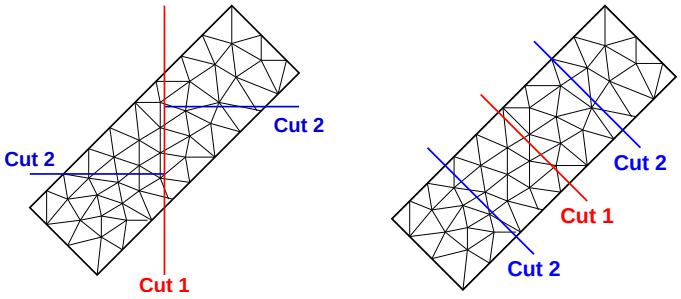

Fig. 3. Example of RCB cuts along coordinate axes (left) and RIB cuts along the principal axis of inertia (right).

Like RCB, Recursive Inertial Bisection (RIB) [107, 113] uses cutting planes to bisect the geometry. In RIB, however, the direction of the plane is computed to be orthogonal to long directions in the actual geometry, rather than to a coordinate axis (see Figure 3, right). Treating objects as point masses, the direction of principle inertia in the geometry is found by computing eigenvectors of a 3×3 matrix assembled from the point masses.

Space-Filling Curves

A second class of geometric partitioners utilizes a one-dimensional "traversal" or linearization to order objects or groups of objects. After determining a one-dimensional ordering, subdomains are formed from contiguous segments of the linearization. This technique produces well-formed subdomains if the ordering preserves *locality*, i.e., if objects that are close in the linearization are also close in the original coordinate space.

The linearization is often achieved using space-filling curves (SFCs). SFCs provide continuous mappings from one-dimensional to d-dimensional space [99]. They have been used to linearize spatially-distributed data for partitioning [2, 17, 33, 88, 89, 94], storage and memory management [22, 79], and computational geometry [7].

SFCs are typically constructed recursively from a single stencil. Each level of refinement replaces segments of the SFC with a new copy of the curve's stencil, subject to spatial rotations and reflections. The SFC can come arbitrarily close to any point in space. Most importantly for partitioning, some SFCs preserve locality, which Edwards [33] defines formally. Several orderings with different degrees of complexity and locality are possible; only the commonly-used Morton and Hilbert orderings are included here.

The Morton (Z-code or Peano) ordering [80, 84] is a simple SFC that traverses a quadrant's children in a "Z"-like pattern (in the order I, II, III, IV in Figure 4). The pattern at each refinement is identical to that used by its ancestors; no rotations or reflections are performed. However, there are

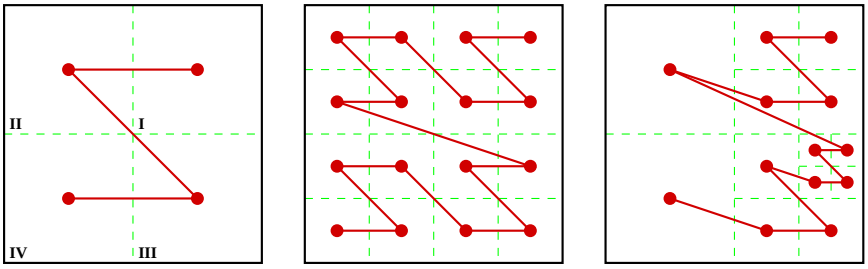

Fig. 4. Template curve for the Morton ordering (left), its first level of refinement (center), and an adaptive refinement (right).

large "jumps" in its linearization, particularly as the curve transitions from quadrant II to quadrant III, so Morton ordering does not always preserve locality. The jumps are even more apparent in three dimensions. Nevertheless, because of its simplicity, Morton ordering is viable in some circumstances, and provides a base ordering for all SFCs [17, 60].

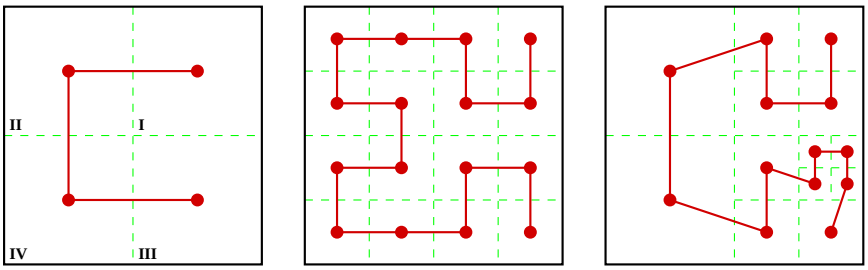

Fig. 5. Template curve for the Hilbert ordering (left), its first level of refinement (center), and an adaptive refinement (right).

The Hilbert ordering uses the Peano-Hilbert SFC [11, 90, 91] to order quadrants. It uses a bracket-like template with rotations and inversions to keep quadrants closer to their neighbors. (Figure 5). Hilbert ordering is locality preserving, and tends to be the most useful for partitioning.

SFC orderings can be applied directly to objects given only the objects' spatial coordinates [2]. Each object is assigned a unique "key" representing the object's position along the SFC. This key is a number in the range $[0, 1]$ that specifies the point on the SFC that passes closest to the object. The object are then ordered by their keys; this ordering can be done via global sorting, binning [8, 27, 29], or traversing an octree representing the SFC [17, 42, 44, 71, 75]. The one-dimensional ordering is then partitioned into appropriately sized pieces; all objects within a piece are assigned to one subdomain.

SFC partitioning was first used by Warren and Salmon [127] in particle-based gravitational simulations. They used a Morton ordering, but acknowledged that Hilbert ordering would improve locality. Patra and Oden [81, 89], Parashar and Browne [88], and Edwards [33] used Hilbert SFC ordering for finite element meshes. Patra and Oden choose cuts along the SFC to balance computational work in their *hp*-adaptive computation. Pilkington and Baden [94] apply SFCs for dynamic load balancing with a uniform mesh where computational workloads vary. Steensland, et al. [111] looked at SFCs for partitioning structured grids which undergo adaptive refinement. Octree partitioning [42, 71, 75] implements SFC partitioning using octree data structures commonly used in mesh generation. Mitchell's Refinement Tree partitioning [76, 78] uses nodal connectivity in adaptively refined meshes (instead of coordinate values) to generate a SFC through mesh elements; while this approach is not strictly a geometric method, the resulting decompositions are qualitatively identical to SFC-produced decompositions.

2.2 Global Graph-Based Partitioning

A popular and powerful class of partitioning procedures make use of connectivity information rather than spatial coordinates. These methods use the fact that the partitioning problem in Section 1.1 can be viewed as the partitioning of an induced graph $G = (V, E)$, where objects serve as the graph vertices (V) and connections between objects are the graph edges (E). For example, Figure 6 shows an induced graph for the mesh in Figure 1; here, elements are the objects to be partitioned and, thus, serve as vertices in the graph, while shared element faces define graph edges.

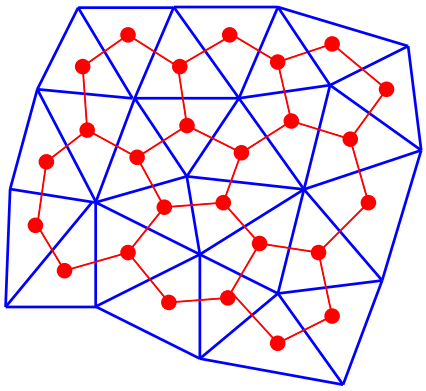

Fig. 6. Example two-dimensional mesh from Figure 1 (left) with its induced graph.

A k-way partition of the graph G is obtained by dividing the vertices into subsets $V_1, ..., V_k$, where $V = V_1 \cup ... \cup V_k$, and $V_i \cap V_j = \oslash$ for $i \neq$

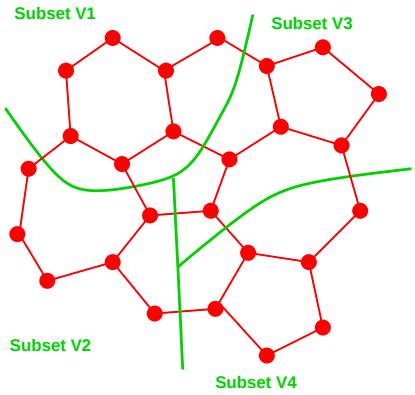

Fig. 7. Four-way partitioning of the graph from Figure 6.

j. Figure 7 shows one possible decomposition of the graph induced by the mesh in Figure 6. Vertices and edges may have weights associated with them representing computation and communication costs, respectively. The goal of graph partitioning, then, is to create subsets V_k with equal vertex weights while minimizing the weight of edges "cut" by subset boundaries. An edge e_{ij} between vertices v_i and v_j is cut when v_i belongs to one subset and v_j belongs to a different one. In Figure 7, eight edges are cut. Algorithms to provide an optimal partitioning are NP-complete [46, 47], so heuristic algorithms are generally used. The graph partitioning is related back to the mesh partitioning problem by creating subdomains of the mesh corresponding to each subset V_i. Figure 1 (right) shows the partitioning of the mesh based on the graph partitioning of Figure 7.

A number of algorithms have been developed to partition graphs. Many of these were developed as static partitioners, intended for use as a preprocessing step rather than as a dynamic load balancing procedure. Some of the multilevel procedures do operate in parallel and can be used for dynamic load balancing.

Greedy Partitioning

Farhat [36] applied graph partitioning to the mesh partitioning problem. The graph is partitioned by a *greedy algorithm* (GR) that builds each subdomain by starting with a vertex and adding adjacent vertices until the subdomain's target size has been reached. The procedure then chooses another unassigned vertex and builds the next subdomain. Farhat [38] reports success using these procedures. Such greedy procedures can also be components of the more commonly used multilevel partitioners described below.

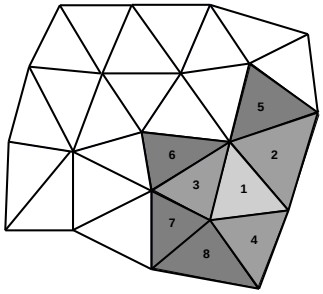

Fig. 8. Example greedy partitioning of a small mesh. Numbers indicate the order in which elements are added to the subdomain being constructed.

Spectral Partitioning

A very well known static graph partitioning method is Recursive Spectral Bisection (RSB) [97, 107]. In RSB, the Laplacian matrix L of a graph is constructed. Each diagonal entry l_{ii} is the degree of vertex i; non-diagonal entries l_{ij} are -1 if edge e_{ij} exists in the graph, and 0 otherwise. The eigenvector x associated with the smallest non-zero eigenvalue of L is then used to divide the vertices into two sets. The median value of x is found. Then, for each x_i, if x_i is less than the median, vertex i is assigned to the first set; otherwise, it is assigned to the second set. This bisection procedure is repeated on the subgraphs until the number of sets is equal to the number of desired partitions.

RSB generally produces high quality partitions. The eigenvector calculation, however, is very expensive and, thus, RSB is used primarily for static partitioning. Strategies using additional eigenvectors to compute four or eight partitions in each stage have proven to be effective while reducing the cost to partition [54].

Multilevel Partitioning

By far, the most successful global graph-based algorithms for static partitioning are multilevel graph partitioners [15, 55, 66], as evidenced by the number of static graph partitioning packages available [53, 65, 92, 98]. Multilevel methods' operation is much like the V-cycle used in multigrid solvers, in that an initial solution is computed on a coarse representation of the graph and used to obtain better solutions on finer representations.

Multilevel graph partitioning involves three major phases: (*i*) *coarsening*, the construction of a sequence of smaller graphs that approximate the original, (*ii*) *partitioning* of the coarsest graph, and (*iii*) *uncoarsening*, the projection of the partitioning of the coarsest graph onto the finer graphs, with a local optimization applied to improve the partitioning at each step. A simple example of this procedure for a small graph with two levels of coarsening is shown in Figure 9.

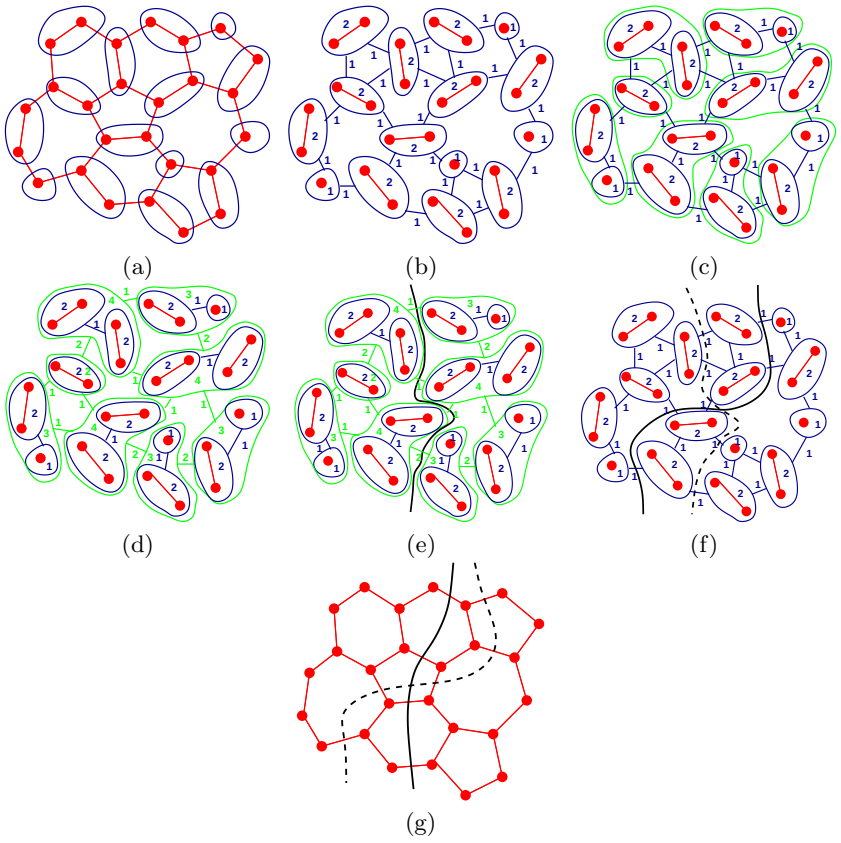

Fig. 9. Multilevel partitioning of the induced graph of Figure 6. Vertex matching in (a) leads to the coarse graph in (b). A second round of vertex matching in (c) produces the coarse graph in (d). This coarsest graph is partitioned in (e). The graph is uncoarsened one level and the partitioning is optimized in (f). The second level of uncoarsening, and another round of local optimization on this partitioning produces the final two-way partitioning shown in (g).

Coarsening procedures typically use a vertex matching algorithm that identifies vertices that can be combined to create coarse vertices. The set of edges from the coarse vertex is taken as the union of the edges for the combined vertices. The sum of the combined vertices' weights is used as the coarse vertex's weight. In this way, the structure and workloads of the input graph are preserved in the coarse representations. Matching at each level can be done by randomly selecting unmatched vertices [15, 55, 66, 124] or using heuristics [6, 23, 45, 48, 49, 66]. For example, heavy-edge matching combines the two vertices sharing the edge with the heaviest edge weight [66], sug-

gesting that vertices with the strongest affinity toward each other should be combined.

The coarsest graph is then partitioned. Since this graph is small, a spectral method [55, 66] can be used efficiently. Faster greedy methods [66] can also be used; while they produce lower quality coarse partitions, local optimizations during the uncoarsening phase improve partition quality. A local optimization may also be used at this point to attempt to encourage incrementality [103]. A geometric procedure such as an SFC may also be used for this coarse partitioning [68].

The coarse decomposition is projected to the finer graphs, with refinements of the partitions made at each graph level. Typically a local optimization technique reduces a communication metric while maintaining and improving load balance. Most of the local optimization approaches are based on the Kernighan-Lin (KL) graph bisection algorithm [69] or its linear-time implementation by Fiduccia and Maytheses (FM) [39]. These techniques make a series of vertex moves from one partition to another, measuring the "gain" or improvement in the metric for each move; moves with high gain are accepted. Karypis and Kumar [66] perform only a few iterations of their KL-like procedure, noting that most of the gain is usually achieved by the first iteration. Hendrickson and Leland [55] continue their KL-like procedure to allow for the discovery of sequences of moves that, while individually making the decomposition worse, may lead to a net improvement. This allows the procedure to escape from local minima. Walshaw, et al. [124] define a relative gain value for each vertex, intended to avoid collisions (i.e., vertices on opposite sides of a boundary each being selected to move).

Parallel implementation of multilevel graph partitioners has allowed them to be used for dynamic load balancing [67, 124]. These methods produce very high quality partitionings, but at a higher cost than geometric methods. Graph partitioners are not inherently incremental, but modifications such as the local methods described below can make them more effective for dynamic repartitioning.

2.3 Local Graph-based Methods

In an adaptive computation, dynamic load balancing may be required frequently. Applying global partitioning strategies after each adaptive step can be costly relative to solution time. Thus, a number of dynamic load balancing techniques that are intended to be fast and incrementally migrate data from heavily to lightly loaded processes have been developed. These are often referred to as local methods.

Unlike global partitioning methods, local methods work with only a limited view of the application workloads. They consider workloads within small, overlapping sets of processors to improve balance within each set. Heavily loaded processors within a set transfer objects to less heavily loaded processors in the same set. Sets can be defined by the parallel architecture's processor

connectivity [70] or by the connectivity of the application data [58, 130]. Sets overlap, allowing objects to move between sets through several iterations of the local method. Thus, when only small changes in application workloads occur through, say, adaptive refinement, a few iterations of a local method can correct imbalances while keeping the amount of data migrated low. For dramatic changes in application workloads, however, many iterations of a local method are needed to correct load imbalances; in such cases, invocation of a global partitioning method may result in a better, more cost-effective decomposition.

Local methods typically consist of two steps: (*i*) computing a map of how much work (nodal weight) must be shifted from heavily loaded to lightly loaded processors, and (*ii*) selecting objects (nodes) that should be moved to satisfy that map. Many different strategies can be used for each step.

Most strategies for computing a map of the amount of data to be shifted among processes are based on the diffusive algorithm of Cybenko [24]. Using processor connectivity or application communication patterns to describe a computational "mesh," an equation representing the workflow is solved using a first-order finite-difference scheme. Since the stencil of the scheme is compact (using information only from neighboring processes), the method is local.

Several variations of this strategy have been developed to reduce data movement or improve convergence. Hu and Blake [58] take a more global view of load distributions, computing a diffusion solution while minimizing work flow over edges of a graph of the processes. Their method is used in several parallel graph-partitioning libraries [100, 124]. Such diffusion methods have been coupled with multilevel graph partitioners (see Section 2.2) to further improve their effectiveness [56, 100, 103, 123, 124].

Other techniques for accelerating the convergence of diffusion schemes include use of higher-order finite difference schemes and dimensional exchange. Watts, et al. [128, 129] use a second-order implicit finite difference scheme to solve the diffusion equation; this scheme converges to global balance in fewer iterations, but requires more work and communication per iteration. In dimensional exchange [24, 31, 132, 134], a hypercube architecture is assumed. (The algorithm can be used on other architectures by logically mapping the architecture to a hypercube.) Processes exchange work with neighbors along each dimension of the hypercube; after looping over all dimensions, the workloads are balanced.

Demand-driven models are also common [26, 34, 70, 86, 130, 131, 132]. These models operate in either of two ways: (*i*) underloaded processes request work from their overloaded neighboring processes, or (*ii*) overloaded processes send work to their underloaded neighbors. The result is similar to diffusion algorithms, except that nodes are transferred to only a subset of neighbors rather than distributed to all neighbors. Version (*i*) of this model has shown to be more effective than (*ii*), as the majority of load-balancing work is performed by the underloaded process and overloading of the receiving process is avoided [132]. As in the diffusion algorithm, neighbors can be

defined by following the physical processor network [70] or the logical data connections [131]. Ozturan's iterative tree-balancing procedure [26, 86] groups processes into trees based upon their work requests, moving work among processes within trees. This more global view accelerates the convergence of the diffusion, but also increases the average number of neighboring processes per process in the application's communication graph.

The second step of a local method is deciding which objects (graph nodes) to move to satisfy the workload transfers computed in the first step. Typically, variants of the KL [69] or FM [39] local optimization algorithms (used for refinement of multilevel partitions) are used. For each object, the gain toward a specific goal achieved by transferring the object to another process is computed. Many options for the gain function have been used (e.g., [28, 55, 124, 131, 134]). Most commonly, the weight of graph edges cut by subdomain boundaries is minimized. However, other goals might include minimizing the amount of data migrated [28, 100], minimizing the number of process neighbors, optimizing subdomain shape [30, 118], or some combination of these goals. The set of objects producing the highest gain is selected for migration. Selection continues until the actual workload transferred is roughly equal to the desired workload transfers.

3 Algorithm Comparisons

A number of theoretical and empirical comparisons of various partitioning strategies have been performed [14, 27, 37, 38, 41, 51, 57, 111, 112, 115]. Selection of the method that is most effective for an application depends on trade-offs between incrementality, speed and quality that can be tolerated by the application. A PDE solver which uses a single decomposition throughout the computation should consider strategies that produce high-quality partitions, with less concern for execution speed of the partitioner. A solver which uses frequent adaptivity will want to consider strategies that execute quickly and are incremental, with less emphasis on partition quality. A procedure which does not readily provide adjacency information will be restricted to geometric methods. This section summarizes and cites key results.

- RCB, URB
  - Geometric method: only coordinate information needed.
  - Incremental and suitable for dynamic load balancing [51].
  - Executes very quickly [115].
  - Moderate quality decompositions. Cutting planes help keep the number of objects on subdomain boundaries small for well-shaped meshes [18]. Unfortunate cuts through highly refined regions [115] or complex domain geometry [75, 73] can lead to poor decompositions. URB produces more uniform subdomain aspect ratios than RCB when there is a large variation in object density [61].

  - Conceptually simple; straightforward to implement in parallel [29].
  - Maintains geometric locality [51, 122].
  - Simple to determine intersections of objects with subdomains, e.g., for parallel contact detection and smoothed particle hydrodynamics simulations [96]; subdomains are described by simple parallelepipeds.
- RIB
  - Geometric method: only coordinate information needed.
  - Not incremental; may be unsuitable for dynamic load balancing [42].
  - Executes almost as quickly as RCB [115].
  - Slightly higher quality decompositions than RCB; lower quality than spectral and multilevel graph partitioning [38, 115]. Unfortunate cuts through highly refined regions can cause poor decompositions [115].
  - Conceptually simple; straightforward to implement in parallel [29, 106].
  - Maintains geometric locality.
  - Simple to determine intersections of objects with subdomains, e.g., for parallel contact detection and smoothed particle hydrodynamics simulations [96].
- SFC
  - Geometric method: only coordinate information needed.
  - Incremental and suitable for dynamic load balancing [42, 51].
  - Executes very quickly [42, 94, 112].
  - Slightly lower quality decompositions than geometric bisection methods [89].
  - Conceptually simple; straightforward to implement in parallel [94].
  - Choice of SFC used depends on locality requirements; Hilbert is usually best [17].
  - The global ordering induced by sorting SFC keys can be exploited to order data to improve cache performance during computation, and can provide automated translations between global and per-process numbering schemes [33, 87].
  - Possible to determine intersections of objects with subdomains, e.g., for parallel contact detection and smoothed particle hydrodynamics simulations [27].
- Greedy partitioning
  - Graph-based method: connectivity information is required.
  - Not incremental; may be unsuitable for dynamic load balancing.
  - Executes quickly [38, 118, 122, 123].
  - Medium-quality decompositions [122], better than RIB [38], and good with respect to subdomain aspect ratio [38]. Tends to leave non-connected or stripwise subdomains in the last few partitions computed [57].
  - Difficult to implement in parallel.
  - Does not maintain geometric locality [122].
- Spectral graph partitioning

  - Graph-based method: connectivity information is required.
  - Not incremental; may be unsuitable for dynamic load balancing [112]. Van Driessche and Roose [117] developed modifications to include incrementality.
  - Executes very slowly [123].
  - Very high quality decompositions [123].
  - More difficult to parallelize than geometric methods [5, 109].
  - Does not maintain geometric locality [122].
  - Suitable primarily for static partitioning.
- Multilevel graph partitioning
  - Graph-based method: connectivity information is required.
  - Not incremental; may be unsuitable for dynamic load balancing [122]. Metrics may include migration cost to improve incrementality [103].
  - Executes slowly [115, 123].
  - Very high quality decompositions [67, 124].
  - Difficult to implement in parallel [67].
  - Does not maintain geometric locality.
- Local graph-based methods
  - Graph-based method: connectivity information is required.
  - Incremental; suitable for dynamic load balancing [26, 86].
  - Usually execute quickly, but several iterations may be needed for global balance. Also, more sophisticated techniques can be more expensive [112].
  - High quality decompositions, given a good starting decomposition.
  - Straightforward to implement in parallel [26, 86]; can be incorporated into multilevel strategies [55, 67, 124].
  - Useful as a post-processing step for other methods to improve partition quality [42, 75].

4 Software

Many software packages are available to provide static and dynamic load balancing to applications. Using these packages, application developers can access a variety of high-quality implementations of partitioning algorithms. Many packages also include supporting functionality (e.g., data migration tools and unstructured communication tools) commonly needed by applications using load balancing. Use of these packages saves application developers the effort of learning and implementing partitioning algorithms themselves, while allowing them to compare partitioning strategies within their applications. Moreover, many of the packages are available as open-source software; see bibliography entries for the packages cited for distribution details.

Static partitioning software is typically used as a pre-processor to the application. It can be used in two ways: as a stand-alone tool or as a function call from the application. In stand-alone mode, input files describe the problem

domain to be partitioned; the format of these files is determined by the partitioning software. The computed decomposition is also written to files. The application must then read the decomposition files to distribute data appropriately. Function-call interfaces to static partitioners allow them to be called directly by applications during pre-processing phases of the application.

Several graph partitioning packages have been developed for static load balancing; they include Chaco [53], Metis [65], Jostle [126], Party [98] and Scotch [93]. These tools run in serial and have both stand-alone and function interfaces. For the stand-alone mode, users provide input files describing the problem domain in terms of a graph, listing vertices (objects), edges between vertices, vertex and edge weights, and possibly coordinates. The function-call interfaces accept a graph description of the problem through arrays using compressed sparse row (CSR) format. In both modes, applications have to convert their application data into the appropriate graph format.

By necessity, dynamic load-balancing software uses function call interfaces, as file-based interfaces would be unacceptable for balancing during a computation. Similarly, dynamic load-balancing software is executed in parallel, assuming an existing distribution of data; parallel execution is required to maintain scalability of the application. Two types of dynamic partitioning software are available: algorithm-specific libraries and toolkits of partitioning utilities.

ParMETIS [68] and PJostle [126] are two widely used algorithm-specific libraries. Both provide multi-level and diffusive graph partitioning. Like their serial counterparts, they accept input in CSR format, with extensions describing the existing partition assignment of the vertices; the arrays describing the application data as a graph in this compressed format must be built by the application. ParMETIS includes support for multiple weights per vertex [63] and edge [101], enabling multi-constraint and multi-objective partitioning (see Section 5.2). PJostle allows multiple vertex weights for multiphase applications [125] (see Section 5.2) and a network description [121] to allow partitioning for heterogeneous computing systems (see Section 5.3).

Load-balancing toolkits such as Zoltan [29] and DRAMA [72] incorporate suites of load-balancing algorithms with additional functionality commonly needed by dynamic applications. Both Zoltan and DRAMA include geometric partitioners (through implementations in the toolkits) and graph-based partitioners (through interfaces to ParMETIS and PJostle). They enable comparisons of various methods by providing a common interface for all partitioners and allowing applications to select a method via a single parameter. They also provide support for moving data between processors to establish a new decomposition.

The Zoltan toolkit [29] provides parallel dynamic load balancing and data management services to a wide range of applications, including particle simulations, mesh-based simulations, circuit simulations, and linear solvers. It includes geometric bisections methods (RCB, RIB), space-filling curve methods (HSFC, Octree, Refinement Tree), and graph-based partitioning (through

ParMETIS and PJostle). Unlike the graph-partitioning libraries, Zoltan's design is "data-structure neutral"; i.e., Zoltan does not require the application to use or build particular data structures for Zoltan. Instead, a callback-function interface provides a simple, general way for applications to provide data to Zoltan. Applications provide simple functions returning, e.g., lists of objects to be partitioned, coordinates for the objects, and relationships between objects. Zoltan calls these functions to obtain application information needed to build its data structures. Once an application implements these callback functions, switching between load-balancing methods requires changing only one parameter.

Zoltan also includes a number of utilities that simplify development of dynamic applications. Its data migration tools assist in the movement of data among processors as they move from an old decomposition to a new one. Because Zoltan does not have information about application data structures, it cannot update them during migration. But given callback functions that pack and unpack data from communication buffers, its migration tools perform all communication needed for data migration. Zoltan's unstructured communication package provides a simple mechanism for complex communication among processors, freeing application developers from the details of individual message sends and receives. Its distributed data directory provides an efficient, scalable utility for locating data in the memory space of other processes. Key kernels of contact detection simulations—finding the partitions owning points and regions in space—are included for Zoltan's geometric and HSFC methods.

The DRAMA (Dynamic Re-Allocation of Meshes for parallel finite element Applications) toolkit [72] provides parallel dynamic load balancing and support services to mesh-based applications. DRAMA assumes a basic data structure of a mesh and enables partitioning of the mesh nodes, elements or both. The mesh is input to DRAMA through array-based arguments. Like Zoltan, DRAMA provides a number of partitioning strategies, including recursive bisection methods and graph partitioning through interfaces to ParMETIS and PJostle.

DRAMA includes a robust cost-model for use in partitioning. This model accounts for both computation and communication costs in determining effective decompositions. Because it assumes a mesh data structure, DRAMA includes more sophisticated support for data migration than Zoltan. It migrates its input mesh to its new location; this migrated mesh can then serve as a starting point for the application data migration. DRAMA provides support for heterogeneous computing architectures through PJostle's network description [120] (see Section 5.3). It also includes extensive support for contact detection and crash simulations.

Load-balancing tools that are tied more closely to specific applications also exist. For example, the PLUM system [82] provides dynamic load balancing for applications using adaptively refined meshes. Its goal is to minimize load-balancing overhead during adaptive computations. To do so, it balances with respect to a coarse mesh in the adaptive simulation using element weights pro-

portional to the number of elements into which each coarse element has been refined. It uses an external partitioning library (e.g., ParMETIS) to compute a decomposition, and then uses a similarity matrix to remap partitions in a way that minimizes data movement between the old and new decompositions. Another example, the VAMPIRE library [110], produces decompositions for structured adaptive mesh refinement applications. Assuming the refined mesh is represented as a tree of uniform grids, it uses a SFC algorithm to distribute the grids to processors to evenly distribute work while attempting to minimize communication between the grids. Load-balancing systems are also included as parts of larger parallel run-time systems; see, e.g., CHARM++ [62] and PREMA [4].

5 Current Challenges

As parallel simulations and environments become more sophisticated, partitioning algorithms must address new issues and application requirements. Software design that allows algorithms to be compared and reused is an important first step; carefully designed libraries that support many applications benefit application developers while serving as test-beds for algorithmic research. Existing partitioners need additional functionality to support new applications. Partitioning models must more accurately represent a broader range of applications, including those with non-symmetric, non-square, and/or highly-connected relationships. And partitioning algorithms need to be sensitive to state-of-the-art, heterogeneous computer architectures, adjusting work assignments relative to processing, memory and communication resources.

5.1 Hypergraph Partitioning

Development of robust partitioning models is important in load-balancing research. While graph models (see Section 2.2) are often considered the most effective models for mesh-based PDE simulations, they have limitations for larger classes of problems (e.g., electrical systems, computational biology, linear programming). These new problems are often more highly connected, more heterogeneous, and less symmetric than mesh-based PDE problems.

As an alternative to graphs, hypergraphs can be used to model application data [19, 20]. A hypergraph $HG = (V, HE)$ consists of a set of vertices V representing the data objects to be partitioned and a set of hyperedges HE connecting two *or more* vertices of V. By allowing larger sets of vertices to be associated through edges, the hypergraph model overcomes many of the limitations of the graph model.

A key limitation of the graph model is that its edge-cut metric only approximates communication volume induced by a decomposition (see Section 1.3). While this approximation is adequate for traditional finite-element, finite-volume, and finite-difference simulations, it is not sufficient for more highly

connected and unstructured data. In the hypergraph model, however, the number of hyperedge cuts is equal to the communication volume, providing a more effective partitioning metric.

Catalyurek and Aykanat [20] also describe the greater expressiveness of hypergraph models over graph models. Because edges in the graph model are non-directional, they imply symmetry in all relationships, making them appropriate only for problems represented by square, structurally symmetric matrices. Systems A with non-symmetric structure must be represented by a symmetrized model $A + A^T$, adding new edges to the graph and further skewing the communication metric. While a directed graph model could be adopted, it would not improve the accuracy of the communication metric. Likewise, graph models can not represent rectangular matrices, such as those arising in linear programming. Kolda and Hendrickson [52] propose using bipartite graphs. For an $m \times n$ matrix A, vertices $m_i, i = 1, \ldots, m$ represent rows, and vertices $n_j, j = 1, \ldots, n$ represent columns. Edges e_{ij} connecting m_i and n_j exist for non-zero matrix entries a_{ij}. But as in other graph models, the number of edge cuts only approximates communication volume.

Hypergraph models, on the other hand, do not imply symmetry in relationships, allowing both structurally non-symmetric and rectangular matrices to be represented. For example, the rows of a rectangular matrix could be represented by the vertices of a hypergraph. Each matrix column would be represented by a hyperedge connecting all non-zero rows in the column [20].

The improved communication metric and expressiveness of hypergraph models lead to impressive results. Using hypergraph partitioning, Catalyurek and Aykanat [20] report reductions in communication volume of 12-15% compared to graph partitioning for matrices from traditional finite difference applications. But for a broader range of matrices, including examples from linear programming, circuit simulations and stochastic programming, hypergraph partitioning produced reductions of 30-38% on average. Time to compute the hypergraph decomposition was 34-130% greater than that required to compute a graph decomposition.

Hypergraph partitioning's effectiveness has been demonstrated in many areas, including VLSI layout [16], sparse matrix decompositions [20, 119], and database storage and data mining [21, 85]. Serial hypergraph partitioners are available (e.g., hMETIS [64], PaToH [20, 19], Mondriaan [119]). Research into parallel hypergraph partitioning includes a disk-based implementation used for partitioning Markov matrices [116] and a distributed memory implementation in Zoltan [13]. Parallel implementation is needed for hypergraph partitioning to be viable for very large simulations. Additionally, incremental hypergraph algorithms (analogous to diffusive graph algorithms [24]) will be needed for dynamic applications.

5.2 Multi-criteria Partitioning

Most load-balancing research has focused on cases having a single load to be balanced. Multi-phase simulations, however, might have different work loads in each phase of a simulation. For example, a multiphysics simulation might include both fluid flow and solid mechanics phases. Crash simulations typically have a finite-element solve phase and a contact-detection phase. Even within a finite element simulation, the matrix assembly and matrix solve phases may have significantly different load characteristics depending on the physics of the problem.

One approach to balancing multi-phase simulations is to use separate decompositions for each phase, mapping data between decompositions when needed. This approach has been used with great success in crash simulations, where static graph-based decompositions were used for the finite element phase and dynamic geometric decompositions were used for contact detection [96]; data were transferred between the decompositions as needed between phases.

Still, the idea of having a single decomposition that is balanced with respect to multiple loads is attractive. With such a decomposition, no mapping of data is needed between phases, reducing application communication costs. Each object to be balanced would have a vector v of weights associated with it; the j^{th} component of v would represent the object's workload in phase j. A single decomposition would then be generated that balances each vector component.

Walshaw, et al. [125] developed a multiphase graph partitioner in Jostle [126]. Assuming components of weight vector v represent a vertex's participation in a phase, they say the "type" of the vertex is the first phase j in which the vertex participates, i.e., for which $v[j] > 0$. They then balance each type of vertex separately, maintaining partition information from lower types as "stationary" vertices in the partitioning of higher types. That is, in computing a partition for vertices of type k, $k > j$, all vertices of type j within a partition are represented by a single "supervertex" whose partition assignment is fixed to a particular partition; edges between these stationary vertices and vertices of type k are maintained to represent data dependencies between the phases. A standard graph partitioner is used to partition each type of vertices; in attempting to minimize cut edges, the graph partitioner is likely to assign type k vertices to the same partition as type j vertices to which they are connected, keeping inter-phase communication costs low.

The multi-constraint graph-partitioning model of Karypis, et al. [63, 104] in METIS [65] and ParMETIS [67] uses vertex weight vectors to create multiple load-balancing constraints. Using this model, they can compute both multiphase decompositions and decompositions with respect to multiple criteria, e.g., workloads and memory usage. Their approach is built on the multi-level framework commonly used in graph partitioning (see Section 2.2), with modifications made in the coarsening, coarse-partitioning, and refinement steps

to accommodate multiple vertex weights. During coarsening, the same heavy-edge metric used in single-constraint partitioning is used to select vertices to be combined; this metric combines a vertex with the neighboring vertex sharing the heaviest edge weight. In multi-constraint partitioning, ties between combinations with the same metric value are broken by a "balanced-edge" metric that attempts to make all weights of the combined vertex as close to the same value as possible, as more uniform weights are easier to balance in the coarse-partitioning and refinement steps. A greedy recursive graph bisection algorithm is used to compute the coarse partition; at each level of recursion, two subdomains A and B are created by removing vertices from A (which initially contains the entire domain) and adding them to B (which initially is empty). In the multi-constraint case, vertices are selected based on their ability to reduce the heaviest weight of A the most. In refinement, KL [69] or FM [39] procedures are used. For multi-constraint partitioning, queues of vertices that can be moved are maintained for each weight and neighboring partition; vertices are again selected based on their ability to reduce the maximum imbalance over all weights while reducing the number of edges cut. To enforce the balance constraints in multi-constraint partitioning, an additional shifting of vertices among processors without regard to increases in the edge cut weight is sometimes needed before refinement.

Because geometric partitioners are preferred for many applications, Boman, et al. pursued multi-criteria partitioning for geometric partitioners, specifically RCB [12]. Their implementation is included in Zoltan [29]. RCB consists of a series of one-dimensional partitioning problems; objects i are ordered linearly by their coordinate values corresponding to the direction of the cut. Like other approaches, objects i have vector weights v_i representing the load-balance criteria. Instead of imposing multiple constraints, however, Boman, et al. formulate each one-dimensional problem as an optimization problem where the objective is to find a cut s such that

$$\min_s \max(g(\sum_{i \leq s} v_i), g(\sum_{i > s} v_i)),$$

where g is a monotonically non-decreasing function in each component of the input vector (typically $g(x) = \sum_j x_j^p$ with $p = 1$ or $p = 2$, or $g(x) = \|x\|$ for some norm). This objective function is unimodal with respect to s. In other words, starting with $s = 1$ and increasing s, the objective decreases, until at some point the objective starts increasing; that point defines the optimal bisection value s. (Note that the objective may be locally flat (constant), so there is not always a unique minimizer.) An optimal cut is computed in each coordinate direction; the cut producing the best balance is accepted.

In general, computing multi-criteria decompositions becomes more difficult as the number of criteria and/or number of partitions increases. As a result, partition quality can degrade. Likewise, multi-criteria partitions are more expensive to compute than single-criterion partitions; the extra cost,

however, may be justified by the improved load balance and reduction of data transfer.

5.3 Resource-Aware Balancing

Cluster and grid computing have made hierarchical and heterogeneous computing systems increasingly common as target environments for large-scale scientific computation. Heterogeneity may exist in processor computing power, network speed, and memory capacity. Clusters may consist of networks of multiprocessors with varying computing and memory capabilities. Grid computations may involve communication across slow interfaces between vastly different architectures. Modern supercomputers are often large clusters with hierarchical network structures. Moreover, the characteristics of an environment can change during a computation due to increased multitasking and network traffic. For maximum efficiency, software must adapt dynamically to the computing environment and, in particular, data must be distributed in a manner that accounts for non-homogeneous, changing computing and networking resources. Several projects have begun to address resource-aware load balancing in such heterogeneous, hierarchical, and dynamic computing environments.

Minyard and Kallinderis [74] use octree structures to conduct partitioning in dynamic execution environments. To account for the dynamic nature of the execution environment, they collect run-time measurements based on the "wait" times of the processors involved in the computation. These "wait" times measure how long each CPU remains idle while all other processors finish the same task. The objects are assigned load factors that are proportional to the "wait" times of their respective owning processes. Each octant load is subsequently computed as the sum of load factors of the objects contained within the octant. The octree algorithm then balances the load factors based on the weight factors of the octants, rather than the number of objects contained within each octant.

Walshaw and Cross [121] conduct multilevel mesh partitioning for heterogeneous communication networks. They modify a multilevel algorithm in PJostle [126] seeking to minimize a cost function based on a model of the heterogeneous communication network. The model gives a static quantification of the network heterogeneity as supplied by the user in a Network Cost Matrix (NCM). The NCM implements a complete graph representing processor interconnections. Each graph edge is weighted as a function of the length of the path between its corresponding processors.

Sinha and Parashar [108] present a framework for adaptive system-sensitive partitioning and load balancing on heterogeneous and dynamic clusters. They use the Network Weather Service (NWS) [133] to gather information about the state and capabilities of available resources; then they compute the load capacity of each node as a weighted sum of processing, memory, and communi-

cations capabilities. Reported experimental results show that system-sensitive partitioning resulted in significant decrease of application execution time.

Faik, et al. [35] present the Dynamic Resource Utilization Model (DRUM) for aggregating information about the network and computing resources of an execution environment. Through minimally instrusive monitoring, DRUM collects dynamic information about computing and networking capabilities and usage; this information determines computing and communication "powers" that can be used as the percentage of total work to be assigned to processes. DRUM uses a tree structure to represent the underlying interconection of hierarchical network topologies (e.g., clusters of clusters, or clusters of multiprocessors). Using DRUM's dynamic monitoring and power computations, they achieved 90% of optimal load distribution for heterogeneous clusters [35].

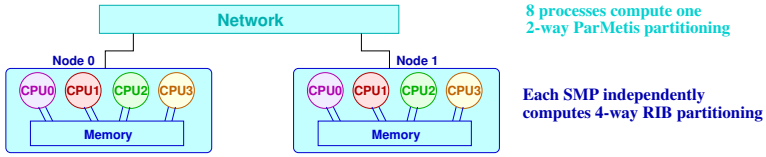

Fig. 10. Hierarchical balancing algorithm selection for two 4-way SMP nodes connected by a network.

Teresco [114] has implemented hierarchical partitioning procedures within Zoltan. These procedures can be used alone, or can be guided by DRUM [35]. Hierarchical partitioning allows any combination of Zoltan's load-balancing procedures to be used on different levels and subtrees of hierarchical machine models. Tradeoffs in execution time, imbalance, and partition quality (*e.g.*, surface indices, interprocess connectivity) can hold greater importance in heterogeneous environments [115], making different methods more appropriate in certain types of environments. For example, consider the cluster of SMPs connected by Ethernet shown in Figure 10. A more costly graph partitioning can be done to partition into two subdomains assigned to the SMPs, to minimize communication across the slow network interface, possibly at the expense of some computational imbalance. Then, a fast geometric algorithm can be used to partition independently within each SMP. Teresco [114] reports that while multilevel graph partitioning alone often achieves the fastest computation times, there is some benefit to using this hierarchical load balancing, particularly in maintaining strict load balance within the SMPs.

5.4 Migration Minimization

The costs of dynamic load balancing include (*i*) preparation of the input to the partitioner, (*ii*) execution of the partitioning algorithm, and (*iii*) migration of application data to achieve the new decomposition. The migration step is

often the most expensive, leading to efforts to reduce this cost. As described in Section 3, selection of appropriate load-balancing procedures contributes to reduced migration costs. Incremental procedures (e.g., RCB, SFC, Octree, diffusive graph partitioning) are preferred when data migration costs must be controlled. The unified partitioning strategy in ParMETIS computes both a multilevel graph decomposition ("scratch-remap") and a diffusive decomposition [102, 103]; it then selects the better decomposition in terms of load balance and migration costs.

Clever techniques can be used within an application to reduce data migration costs. For example, the most straightforward way to use partitioning and dynamic load balancing in a parallel adaptive computation is shown on the left in Figure 11. Here, an initial mesh is partitioned, and the computation proceeds, checking periodically to determine whether the solution resolution is sufficient. If not, the mesh is enriched adaptively, the load is rebalanced, and the computation continues. Alternatively, the rebalancing can be done *before* the mesh is actually enriched, if the error indicators used to predict refinement can also predict appropriate weights for the mesh before enrichment [43, 83] (Figure 11, right). This "predictive balancing" approach can improve computational balance during the refinement phase, and leads to less data migration, as redistribution occurs on the smaller mesh. Moreover, without predictive balancing, individual processors may have nearly all of their elements scheduled for refinement, leading to a memory overflow on those processors, when in fact the total amount of memory available across all processors is sufficient for the computation to proceed following refinement [40]. If the error indicators predict the resulting refinement with sufficient accuracy, the predictive balancing step also achieves a balanced partitioning of the refined mesh. In some cases, a corrective load balancing step, e.g., with one of the local methods outlined in Section 2.3, may be beneficial.

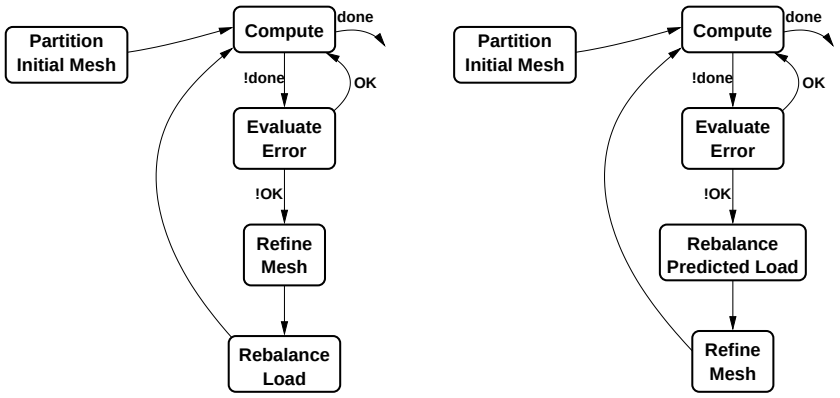

Fig. 11. Non-predictive (left) and predictive (right) program flows for a typical parallel adaptive computation.

Techniques within load-balancing procedures can also reduce migration costs. The similarity matrix in PLUM [82] represents a maximal matching between an old decomposition and a new one. Old and new partitions are represented by the nodes of a bipartite graph, with edges between old and new partitions representing the amount of data they share. A maximal matching, then, numbers the new partitions to provide the greatest overlap between old and new decompositions and, thus, the least data movement. Similar strategies have been adopted by ParMETIS [68] and Zoltan [29].

Load-balancing objectives can also be adjusted to reduce data migration. Heuristics used in local refinement (see Section 2.3) can select objects for movement that have the lowest data movement costs. They can also select a few heavily weighted objects to satisfy balance criteria rather than many lightly weighted objects. Hu and Blake compute diffusive decompositions to achieve load balance subject to a minimization of data movement [59]. Berzins extends their idea by allowing greater load imbalance when data movement costs are high [10]; he minimizes a metric combining load imbalance and data migration to reduce actual time-to-solution (rather than load imbalance) on homogeneous and heterogeneous networks.

Acknowledgments

The authors thank the following people for their collaborations and discussions: Andrew Bauer, Diane Bennett, Rob Bisseling, Erik Boman, Paul Campbell, Laura Effinger-Dean, Jamal Faik, Luis Gervasio, Robert Heaphy, Bruce Hendrickson, Steve Plimpton, Robert Preis, Arjun Sharma, Lida Ungar, and Courtenay Vaughan.

References

1. Adjerid, S., Flaherty, J. E., Moore, P., and Wang, Y.: High-order adaptive methods for parabolic systems. *Physica-D*, 60:94–111, (1992)
2. Aluru, S. and Sevilgen, F.: Parallel domain decomposition and load balancing using space-filling curves. In *Proc. International Conference on High-Performance Computing*, pages 230–235, (1997)
3. Bank, R. E. and Holst, M. J.: A new paradigm for parallel adaptive meshing algorithms. *SIAM J. Scien. Comput.*, 22:1411–1443, (2000)
4. Barker, K. J. and Chrisochoides, N. P.: An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications. In *Proc. Supercomputing 2003*, Phoenix, (2003)
5. Barnard, S. T.: PMRSB: parallel multilevel recursive spectral bisection. In Baker, F. and Wehmer, J., editors, *Proc. Supercomputing '95*, San Diego, (1995)
6. Barnard, S. T. and Simon, H. D.: Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, (1994)

7. Bartholdi, J. J. and Platzman, L. K.: An $O(n \log n)$ travelling salesman heuristic based on spacefilling curves. *Operation Research Letters*, 1(4):121–125, (1982)
8. Bauer, A. C.: *Efficient Solution Procedures for Adaptive Finite Element Methods – Applications to Elliptic Problems.* PhD thesis, State University of New York at Buffalo, (2002)
9. Berger, M. J. and Bokhari, S. H.: A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36:570–580, (1987)
10. Berzins, M.: A new metric for dynamic load balancing. *Appl. Math. Modelling*, 25:141–151, (2000)
11. Bially, T.: Space-filling curves: their generation and their application to band reduction. *IEEE Trans. Inform. Theory*, IT-15:658–664, (1969)
12. Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Heroux, M., and Preis, R.: LDRD report: Parallel repartitioning for optimal solver performance. Technical Report SAND2004–0365, Sandia National Laboratories, Albuquerque, NM, (2004)
13. Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Mitchell, W. F., John, M. S., and Vaughan, C.: Zoltan: Data-management services for parallel applications. URL: `http://www.cs.sandia.gov/Zoltan`
14. Bottasso, C. L., Flaherty, J. E., Özturan, C., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H.: The quality of partitions produced by an iterative load balancer. In Szymanski, B. K. and Sinharoy, B., editors, *Proc. Third Workshop on Languages, Compilers, and Runtime Systems*, pages 265–277, Troy, (1996)
15. Bui, T. and Jones, C.: A heuristic for reducing fill in sparse matrix factorization". In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, (1993)
16. Caldwell, A., Kahng, A., and Markov, J.: Design and implementation of move-based heuristics for VLSI partitioning. *ACM J. Experimental Algs.*, 5, (2000)
17. Campbell, P. M., Devine, K. D., Flaherty, J. E., Gervasio, L. G., and Teresco, J. D.: Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, (2003)
18. Cao, F., Gilbert, J. R., and Teng, S.-H.: Partitioning meshes with lines and planes. Technical Report CSL–96–01, Xerox PARC, (1996). ftp://parcftp.xerox.com/pub/gilbert/index.html
19. Catalyurek, U. and Aykanat, C.: Decomposing irregularly sparse matrices for parallel matrix-vector multiplications. *Lecture Notes in Computer Science*, 1117:75–86, (1996)
20. Catalyurek, U. and Aykanat, C.: Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Dist. Systems*, 10(7):673–693, (1999)
21. Chang, C., Kurc, T., Sussman, A., Catalyurek, U., and Saltz, J.: A hypergraph-based workload partitioning strategy for parallel data aggregation. In *Proc. of 11th SIAM Conf. Parallel Processing for Scientific Computing.* SIAM, (2001)
22. Chatterjee, S., Lebeck, A. R., Patnala, P. K., and Thottethodi, M.: Recursive array layouts and fast parallel matrix multiplication. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, (1999)
23. Cheng, C.-K. and Wei, Y.-C. A.: An improved two-way partitioning algorithm with stable performance. *IEEE Trans. Computer Aided Design*, 10(12):1502–1511, (1991)

24. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7:279–301, (1989)
25. Dagum, L.: Automatic partitioning of unstructured grids into connected components. In *Proc. Supercomputing Conference 1993*, pages 94–101, Los Alamitos, (1993). IEEE, Computer Society Press
26. de Cougny, H. L., Devine, K. D., Flaherty, J. E., Loy, R. M., Özturan, C., and Shephard, M. S.: Load balancing for the parallel adaptive solution of partial differential equations. *Appl. Numer. Math.*, 16:157–182, (1994)
27. Devine, K. D., Boman, E. G., Heaphy, R. T., Hendrickson, B. A., Teresco, J. D., Faik, J., Flaherty, J. E., and Gervasio, L. G.: New challenges in dynamic load balancing. Technical Report Technical Report CS-04-02, Williams College Department of Computer Science, (2004). To appear, *Appl. Numer. Math.*
28. Devine, K. D. and Flaherty, J. E.: Parallel adaptive *hp*-refinement techniques for conservation laws. *Appl. Numer. Math.*, 20:367–386, (1996)
29. Devine, K. D., Hendrickson, B. A., Boman, E., St. John, M., and Vaughan, C.: *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, (1999). Tech. Report SAND99-1377. Open-source software distributed at `http://www.cs.sandia.gov/Zoltan`.
30. Diekmann, R., Meyer, D., and Monien, B.: Parallel decomposition of unstructured fem-meshes. In *Proc. Parallel Algorithms for Irregularly Structured Problems*, pages 199–216. Springer LNCS 980, (1995)
31. Diekmann, R., Monien, B., and Preis, R.: Load balancing strategies for distributed memory machines. In Topping, B., editor, *Parallel and Distributed Processing for Computational Mechanics: Systems and Tools*, pages 124–157, Edinburgh, (1999). Saxe-Coburg
32. Diekmann, R., Preis, R., Schlimbach, F., and Walshaw, C.: Shape-optimized mesh partitioning and load balancing for parallel adaptive fem. *Parallel Comput.*, 26(12):1555–1581, (2000)
33. Edwards, H. C.: *A Parallel Infrastructure for Scalable Adaptive Finite Element Methods and its Application to Least Squares C^∞ Collocation*. PhD thesis, The University of Texas at Austin, (1997)
34. Enbody, R., Purdy, R., and Severance, C.: Dynamic load balancing. In *Proc. 7th SIAM Conference on Parallel Processing for Scientific Computing*, pages 645–646. SIAM, (1995)
35. Faik, J., Gervasio, L. G., Flaherty, J. E., Chang, J., Teresco, J. D., Boman, E. G., and Devine, K. D.: A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-04-03, Williams College Department of Computer Science, (2004). Presented at Cluster '04
36. Farhat, C.: A simple and efficient automatic FEM domain decomposer. *Computers and Structures*, 28(5):579–602, (1988)
37. Farhat, C., Lanteri, S., and Simon, H. D.: TOP/DOMDEC: a software tool for mesh partitioning and parallel processing. *Comp. Sys. Engng.*, 6(1):13–26, (1995)
38. Farhat, C. and Lesoinne, M.: Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. Numer. Meth. Engng.*, 36:745–764, (1993)
39. Fiduccia, C. M. and Mattheyses, R. M.: A linear time heuristic for improving network partitions. In *Proc. 19th IEEE Design Automation Conference*, pages 175–181. IEEE, (1982)

40. Flaherty, J. E., Dindar, M., Loy, R. M., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H.: An adaptive and parallel framework for partial differential equations. In Griffiths, D. F., Higham, D. J., and Watson, G. A., editors, *Numerical Analysis 1997 (Proc. 17th Dundee Biennial Conf.)*, number 380 in Pitman Research Notes in Mathematics Series, pages 74–90. Addison Wesley Longman, (1998)
41. Flaherty, J. E., Loy, R. M., Özturan, C., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H.: Parallel structures and dynamic load balancing for adaptive finite element computation. *Appl. Numer. Math.*, 26:241–263, (1998)
42. Flaherty, J. E., Loy, R. M., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H.: Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws. *J. Parallel Distrib. Comput.*, 47:139–152, (1997)
43. Flaherty, J. E., Loy, R. M., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H.: Predictive load balancing for parallel adaptive finite element computation. In Arabnia, H. R., editor, *Proc. PDPTA '97*, volume I, pages 460–469, (1997)
44. Flaherty, J. E., Loy, R. M., Shephard, M. S., and Teresco, J. D.: Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In Cockburn, B., Karniadakis, G., and Shu, S.-W., editors, *Discontinous Galerkin Methods Theory, Computation and Applications*, volume 11 of *Lecture Notes in Compuational Science and Engineering*, pages 113–124, Berlin, (2000). Springer
45. Garbers, J., Promel, H. J., and Steger, A.: Finding clusters in VLSI circuits. In *Proc. IEEE Intl. Conf. on Computer Aided Design*, pages 520–523, (1990)
46. Garey, M., Johnson, D., and Stockmeyer, L.: Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, (1976)
47. Garey, M. R. and Johnson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, (1979)
48. Hagen, L. and Kahng, A.: Fast spectral methofs for ratio cut partitioning and clustering. In *Proc. IEEE Intl. Conf. on Computer Aided Design*, pages 10–13, (1991)
49. Hagen, L. and Kahng, A.: A new approach to effective circuit clustering. In *Proc. IEEE Intl. Conf. on Computer Aided Design*, pages 422–427, (1992)
50. Hendrickson, B.: Graph partitioning and parallel solvers: Has the emperor no clothes? In *Proc. Irregular'98*, volume 1457 of *Lecture Notes in Computer Science*, pages 218–225. Springer-Verlag, (1998)
51. Hendrickson, B. and Devine, K.: Dynamic load balancing in computational mechanics. *Comput. Methods Appl. Mech. Engrg.*, 184(2–4):485–500, (2000)
52. Hendrickson, B. and Kolda, T. G.: Graph partitioning models for parallel computing. *Parallel Comput.*, 26:1519–1534, (2000)
53. Hendrickson, B. and Leland, R.: The Chaco user's guide, version 2.0. Technical Report SAND94–2692, Sandia National Laboratories, Albuquerque, (1994). Open-source software distributed at `http://www.cs.sandia.gov/~bahendr/chaco.html`.
54. Hendrickson, B. and Leland, R.: An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Scien. Comput.*, 16(2):452–469, (1995)

55. Hendrickson, B. and Leland, R.: A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, (1995)
56. Horton, G.: A multi-level diffusion method for dynamic load balancing. *Parallel Comput.*, 19:209–218, (1993)
57. Hsieh, S.-H., Paulino, G. H., and Abel, J. F.: Evaluation of automatic domain partitioning algorithms for parallel finite element analysis. Structural Engineering Report 94-2, School of Civil and Environmental Engineering, Cornell University, Ithaca, (1994)
58. Hu, Y. F. and Blake, R. J.: An optimal dynamic load balancing algorithm. Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA4 4AD, UK, (1995)
59. Hu, Y. F., Blake, R. J., and Emerson, D. R.: An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice and Experience*, 10:467 – 483, (1998)
60. Jagadish, H. V.: Linear clustering of objects with multiple attributes. In *Proc. ACM SIGMOD*, pages 332–342, (1990)
61. Jones, M. T. and Plassmann, P. E.: Computational results for parallel unstructured mesh computations. *Comp. Sys. Engng.*, 5(4–6):297–309, (1994)
62. Kale, L. V. and Krishnan, S.: CHARM++: A portable concurrent object oriented system based on C++. *ACM SIGPLAN notices*, 28(10):91–128, (1993)
63. Karypis, G. and Kumar, V.: Multilevel algorithms for multiconstraint graph paritioning. Technical Report 98-019, Department of Computer Science, University of Minnesota, (1998)
64. Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S.: Multilevel hypergraph partitioning: application in VLSI domain. In *Proc. 34th conf. Design automation*, pages 526 – 529. ACM, (1997)
65. Karypis, G. and Kumar, V.: Metis: Unstructured graph partitioning and sparse matrix ordering system. Tech. Report, University of Minnesota, Department of Computer Science, Minneapolis, MN, (1995). Open-source software distributed at `http://www-users.cs.umn.edu/~karypis/metis`.
66. Karypis, G. and Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scien. Comput.*, 20(1), (1999)
67. Karypis, G. and Kumar, V.: Parallel multivelel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, (1999)
68. Karypis, G., Schloegel, K., and Kumar, V.: *ParMetis Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.1.* University of Minnesota Department of Computer Science and Engineering, and Army HPC Research Center, Minneapolis, (2003). Open-source software distributed at `http://www-users.cs.umn.edu/~karypis/metis`.
69. Kernighan, B. and Lin, S.: An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 29:291–307, (1970)
70. Leiss, E. and Reddy, H.: Distributed load balancing: design and performance analysis. *W. M. Kuck Research Computation Laboratory*, 5:205–270, (1989)
71. Loy, R. M.: *Adaptive Local Refinement with Octree Load-Balancing for the Parallel Solution of Three-Dimensional Conservation Laws.* PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, (1998)
72. Maerten, B., Roose, D., Basermann, A., Fingberg, J., and Lonsdale, G.: DRAMA: A library for parallel dynamic load balancing of finite element applications. In *Proc. Ninth SIAM Conference on Parallel Processing for Scientific*

Computing, San Antonio, (1999). Library distributed under license agreement from `http://www.ccrl-nece.de/~drama/drama.html`.
73. Minyard, T. and Kallinderis, Y.: Octree partitioning of hybrid grids for parallel adaptive viscous flow simulations. *Int. J. Numer. Meth. Fluids*, 26:57–78, (1998)
74. Minyard, T. and Kallinderis, Y.: Parallel load balancing for dynamic execution environments. *Comput. Methods Appl. Mech. Engrg.*, 189(4):1295–1309, (2000)
75. Minyard, T., Kallinderis, Y., and Schulz, K.: Parallel load balancing for dynamic execution environments. In *Proc. 34th Aerospace Sciences Meeting and Exhibit*, number 96-0295, Reno, (1996)
76. Mitchell, W. F.: Refinement tree based partitioning for adaptive grids. In *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing*, pages 587–592. SIAM, (1995)
77. Mitchell, W. F.: The full domain partition approach to distributing adaptive grids. *Appl. Numer. Math.*, 26:265–275, (1998)
78. Mitchell, W. F.: The refinement-tree partition for parallel solution of partial differential equations. *NIST Journal of Research*, 103(4):405–414, (1998)
79. Moon, B., Jagadish, H. V., Faloutsos, C., and Saltz, J. H.: Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Trans. Knowledge and Data Engng.*, 13(1):124–141, (2001)
80. Morton, G. M.: A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., (1966)
81. Oden, J. T., Patra, A., and Feng, Y.: Domain decomposition for adaptive *hp* finite element methods. In *Proc. Seventh Intl. Conf. Domain Decomposition Methods*, State College, Pennsylvania, (1993)
82. Oliker, L. and Biswas, R.: PLUM: Parallel load balancing for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, 51(2):150–177, (1998)
83. Oliker, L., Biswas, R., and Strawn, R. C.: Parallel implementaion of an adaptive scheme for 3D unstructured grids on the SP2. In *Proc. 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems*, Santa Barbara, (1996)
84. Orenstein, J. A.: Spatial query processing in an object-oriented database system. In *Proc. ACM SIGMOD*, pages 326–336, (1986)
85. Ozdal, M. and Aykanat, C.: Hypergraph models and algorithms for data-pattern based clustering. *Data Mining and Knowledge Discovery*, (2004). Accepted for publication
86. Özturan, C.: *Distributed Environment and Load Balancing for Adaptive Unstructured Meshes*. PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, (1995)
87. Parashar, M., Browne, J. C., Edwards, C., and Klimkowski, K.: A common data management infrastructure for adaptive algorithms for PDE solutions. In *Proc. SC97*, San Jose, CA, (1997)
88. Parashar, M. and Browne, J. C.: On partitioning dynamic adaptive grid hierarchies. In *Proc. 29th Annual Hawaii International Conference on System Sciences*, volume 1, pages 604–613, (1996)
89. Patra, A. and Oden, J. T.: Problem decomposition for adaptive *hp* finite element methods. *Comp. Sys. Engng.*, 6(2):97–109, (1995)
90. Patrick, E. A., Anderson, D. R., and Brechtel, F. K.: Mapping multidimensional space to one dimension for computer output display. *IEEE Trans. Computers*, C-17(10):949–953, (1968)

91. Peano, G.: Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, (1890)
92. Pellegrini, F.: SCOTCH 3.1 User's guide. Technical Report 1137-96, LaBRI, Université Bordeaux I, (1996). Library available at `http://www.labri.fr/Perso/~pelegrin/scotch/`.
93. Pellegrini, F. and Roman, J.: Experimental analysis of the dual recursive bipartitioning algorithm for static mapping. Technical Report 1038-96, Universite Bordeaux I, (1996)
94. Pilkington, J. R. and Baden, S. B.: Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Trans. on Parallel and Distributed Systems*, 7(3):288–300, (1996)
95. Pınar, A. and Hendrickson, B.: Graph partitioning for complex objectives. In *Proc. 15th Int'l Parallel and Distributed Processing Symp. (I PDPS)*, San Francisco, CA, (2001)
96. Plimpton, S., Attaway, S., Hendrickson, B., Swegle, J., Vaughan, C., and Gardner, D.: Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics. *J. Parallel Distrib. Comput.*, 50:104–122, (1998)
97. Pothen, A., Simon, H., and Liou, K.-P.: Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, (1990)
98. Preis, R. and Diekmann, R.: *Advances in Computational Mechanics with Parallel and Distributed Processing*, chapter PARTY – A Software Library for Graph Partitioning, pages 63–71. CIVIL-COMP PRESS, (1997). Library distributed under free research and academic license at `http://wwwcs.upb.de/fachbereich/AG/monien/RESEARCH/PART/party.html`.
99. Sagan, H.: *Space-Filling Curves*. Springer-Verlag, (1994)
100. Schloegel, K., Karypis, G., and Kumar, V.: Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. Parallel Distrib. Comput.*, 47(2):109–124, (1997)
101. Schloegel, K., Karypis, G., and Kumar, V.: A new algorithm for multi-objective graph partitioning. Tech. Report 99-003, University of Minnesota, Department of Computer Science and Army HPC Center, Minneapolis, (1999)
102. Schloegel, K., Karypis, G., and Kumar, V.: A unified algorithm for load-balancing adaptive scientific simulations. In *Proc. Supercomputing*, Dallas, (2000)
103. Schloegel, K., Karypis, G., and Kumar, V.: Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes. *IEEE Trans. Parallel Distrib. Syst.*, 12(5):451–466, (2001)
104. Schloegel, K., Karypis, G., and Kumar, V.: Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation – Practice and Experience*, 14(3):219–240, (2002)
105. Shephard, M. S., Dey, S., and Flaherty, J. E.: A straightforward structure to construct shape functions for variable p-order meshes. *Comp. Meth. in Appl. Mech. and Engng.*, 147:209–233, (1997)
106. Shephard, M. S., Flaherty, J. E., de Cougny, H. L., Özturan, C., Bottasso, C. L., and Beall, M. W.: Parallel automated adaptive procedures for unstructured meshes. In *Parallel Comput. in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, (1995)
107. Simon, H. D.: Partitioning of unstructured problems for parallel processing. *Comp. Sys. Engng.*, 2:135–148, (1991)

108. Sinha, S. and Parashar, M.: Adaptive system partitioning of AMR applications on heterogeneous clusters. *Cluster Computing*, 5(4):343–352, (2002)
109. Sohn, A. and Simon, H.: S-HARP: A scalable parallel dynamic partitioner for adaptive mesh-based computations. In *Proc. Supercomputing '98*, Orlando, (1998)
110. Steensland, J.: Vampire homepage. `http://user.it.uu.se/~johans/research/vampire/vampire1.html`, (2000). Open-source software distributed at `http://user.it.uu.se/~johans/research/vampire/download.html`.
111. Steensland, J., Chandra, S., and Parashar, M.: An application-centric characterization of domain-based SFC partitioners for parallel SAMR. *IEEE Trans. Parallel and Distrib. Syst.*, 13(12):1275–1289, (2002)
112. Steensland, J., Söderberg, S., and Thuné, M.: A comparison of partitioning schemes for blockwise parallel SAMR algorithms. In *Proc. 5th International Workshop on Applied Parallel Computing, New Paradigms for HPC in Industry and Academia*, volume 1947 of *Lecture Notes in Computer Science*, pages 160–169, London, (2000). Springer-Verlag
113. Taylor, V. E. and Nour-Omid, B.: A study of the factorization fill-in for a parallel implementation of the finite element method. *Int. J. Numer. Meth. Engng.*, 37:3809–3823, (1994)
114. Teresco, J. D., Faik, J., and Flaherty, J. E.: Hierarchical partitioning and dynamic load balancing for scientific computation. Technical Report CS-04-04, Williams College Department of Computer Science, (2004). Submitted to Proc. PARA '04.
115. Teresco, J. D. and Ungar, L. P.: A comparison of Zoltan dynamic load balancers for adaptive computation. Technical Report CS-03-02, Williams College Department of Computer Science, (2003). Presented at COMPLAS '03
116. Trifunovic, A. and Knottenbelt, W. J.: Towards a parallel disk-based algorithm for multilevel k-way hypergraph partitioning. In *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 236b, Santa Fe, (2004)
117. Van Driessche, R. and Roose, D.: An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Comput.*, 21:29–48, (1995)
118. Vanderstraeten, D., Farhat, C., Chen, P., Keunings, R., and Ozone, O.: A retrofit based methodology for the fast generation and optimization of large-scale mesh partitions: beyond the minimum interface size criterion. *Comput. Methods Appl. Mech. Engrg.*, 133:25–45, (1996)
119. Vastenhouw, B. and Bisseling, R. H.: A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. Preprint 1238, Dept. of Mathematics, Utrecht University, (2002)
120. Walshaw, C. and Cross, M.: Multilevel Mesh Partitioning for Heterogeneous Communication Networks. Tech. Rep. 00/IM/57, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, (2000)
121. Walshaw, C. and Cross, M.: Multilevel Mesh Partitioning for Heterogeneous Communication Networks. *Future Generation Comput. Syst.*, 17(5):601–623, (2001). (originally published as Univ. Greenwich Tech. Rep. 00/IM/57)
122. Walshaw, C. and Cross, M.: Dynamic mesh partitioning and load-balancing for parallel computational mechanics codes. In Topping, B. H. V., editor, *Computational Mechanics Using High Performance Computing*, pages 79–94.

Saxe-Coburg Publications, Stirling, (2002). (Invited Chapter, Proc. Parallel & Distributed Computing for Computational Mechanics, Weimar, Germany, 1999)
123. Walshaw, C., Cross, M., and Everett, M.: A localized algorithm for optimizing unstructured mesh partitions. *Intl. J. of Supercomputer Applications*, 9(4):280–295, (1995)
124. Walshaw, C., Cross, M., and Everett, M.: Parallel dynamic graph-partitioning for unstructured meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, (1997)
125. Walshaw, C., Cross, M., and McManus, K.: Multiphase mesh partitioning. *Appl. Math. Modelling*, 25(2):123–140, (2000). (originally published as Univ. Greenwich Tech. Rep. 99/IM/51)
126. Walshaw, C.: *The Parallel JOSTLE Library User's Guide, Version 3.0*. University of Greenwich, London, UK, (2002). Library distributed under free research and academic license at `http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/`.
127. Warren, M. S. and Salmon, J. K.: A parallel hashed oct-tree n-body algorithm. In *Proc. Supercomputing '93*, pages 12–21. IEEE Computer Society, (1993)
128. Watts, J.: A practical approach to dynamic load balancing. Master's Thesis, (1995)
129. Watts, J., Rieffel, M., and Taylor, S.: A load balancing technique for multiphase computations. In *Proc. High Performance Computing '97*, pages 15–20. Society for Computer Simulation, (1997)
130. Wheat, S.: *A Fine Grained Data Migration Approach to Application Load Balancing on MP MIMD Machines*. PhD thesis, University of New Mexico, Department of Computer Science, Albuquerque, (1992)
131. Wheat, S., Devine, K., and MacCabe, A.: Experience with automatic, dynamic load balancing and adaptive finite element computation. In El-Rewini, H. and Shriver, B., editors, *Proc. 27th Hawaii International Conference on System Sciences*, pages 463–472, Kihei, (1994)
132. Willebeek-LeMair, M. and Reeves, A.: Strategies for dynamic load balancing on highly parallel computers. *IEEE Parallel and Distrib. Sys.*, 4(9):979–993, (1993)
133. Wolski, R., Spring, N. T., and Hayes, J.: The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Comput. Syst.*, 15(5-6):757–768, (1999)
134. Xu, C., Lau, F., and Diekmann, R.: Decentralized remapping of data parallel applications in distributed memory multiprocessors. Tech. Rep. tr-rsfb-96-021, Dept. of Computer Science, University of Paderborn, Paderborn, Germany, (1996)