Repartitioning Unstructured Adaptive Meshes

José G. Castaños
Department of Computer Science
Brown University
jgc@cs.brown.edu

John E. Savage
Department of Computer Science
Brown University
jes@cs.brown.edu

Abstract

We present a new parallel repartitioning algorithm for adaptive finite-element meshes that significantly reduces the amount of data that needs to move between processors in order to rebalance a workload after mesh adaptation (refinement or coarsening). These results derive their importance from the fact that the time to migrate data can be a large fraction of the total time for the parallel adaptive solution of partial differential equations.

1. Introduction

Adaptive computation offers the potential to provide large storage and computational savings on problems with dissimilar scales by focusing the available computational resources on the regions where the solution changes rapidly. In transient problems, the regions of interest can appear or vanish, and modify their size, shape or location, as occurs in the study of turbulence in fluid flows. In some static as well as transient problems, efficiency requires adaptation of the mesh so that regions of high gradients are not underresolved while maintaining a coarser mesh everywhere else.

The local adaptation of a mesh produces imbalances in the work assigned to processors. Because of the irregular load requirements of parallel adaptive computation, a mesh must be dynamically repartitioned and migrated between processors at runtime. Thus, adaptive finite element methods provide an excellent context for the study of dynamic load balancing schemes on distributed-memory parallel computers.

In this paper we introduce *Parallel Nested Repartitioning*, a new partitioning algorithm sketched in [1], that has its roots in multilevel partitioning algorithms [2, 3]. Our method quickly produces low cost, high quality partitions that minimize the amount of data that needs to be moved to rebalance a workload while keeping the cut size small. It has a very natural parallel implementation that allows us to repartition adapted meshes of arbitrary size.

PNR is implemented in PARED [4], a system for the parallel adaptive solution of PDEs described in Section 2. Starting with a coarse initial mesh, PARED locally adapts the mesh until an error criterion is met. Attached to each coarse element of the initial mesh is a refinement history tree whose leaves are the most refined elements into which the coarse element has been refined.

PARED partitions the mesh by elements. A partition is obtained by PNR from the weighted dual graph G of the coarse mesh. Although G has much less information than is available in the adapted mesh M, we demonstrate through experiment and analysis that partitioning M using G gives balanced partitions with cut sizes comparable to those provided by standard partitioning algorithms [2, 3]. Unfortunately, as we show, small changes in M can produce very different partitions of M and the migration of a large number of elements using either our initial version of PNR [1] or standard partitioning algorithms. This *mesh migration problem* has also been addressed by others [5, 6, 7]. Biswas and Oliker [5] permutes the subsets produced by a standard algorithm to minimize data movement. We show that this heuristic can still require that half the elements be migrated. Walshaw *et al* [6] and Schloegel, Karypis and Kumar [7] determine the number of elements that must move between processors to rebalance them using a technique of Hu and Blake [8] and then try to keep the cut size small by migrating elements on the boundaries between processors. Their heuristics require several iterations in which the same regions of the mesh are repeatedly migrated.

We show the power of PNR through an experiment with PARED in which we track a disturbance through space over time. We show that PNR migrates a very small number of elements while maintaining a partition quality comparable to that produced by RSB [2] and multilevel methods [18, 3].

2. An Overview of PARED

PARED supports the local adaptation of unstructured two- and three-dimensional meshes, and the dynamic repartitioning and load balancing of the work. Our design sup-

ports a dynamically changing environment where elements, vertices, and associated equations and unknowns migrate between processors. References to remote elements and vertices are updated as new elements or vertices are created, deleted or moved to a new processor. PARED runs on distributed memory computers such as the IBM SP and networks of workstations (NOW) in which processing nodes communicate by exchanging messages using MPI [9].

To support the dynamic adaptation of meshes we designed a hierarchical data structure of nested meshes. We assume that the user supplies an initial coarse mesh $M^0(D^0, V^0)$ called the 0-level mesh, where $D^0 = \{\Omega_1, \ldots, \Omega_n\}$ is the set of simple shapes or *elements* that approximate a domain Ω and $V^0 = \{V_1, \ldots, V_m\}$ is the set of vertices associated with these elements. This mesh is loaded into a distinguished processor called the *coordinator* P_C. This processor computes an initial partition of the mesh using the algorithms explained in Section 5. P_C then distributes the mesh between the processors where the numerical simulation starts in parallel.

The adaptation procedure constructs a family of nested meshes $M^0, M^1, \ldots, M^t$. Let $M^t(D^t, V^t)$ be the mesh at time step t where $D^t = \{\Omega_1, \ldots, \Omega_n\}$ is a set of elements that approximate the domain Ω of interest and $V^t = \{V_1, \ldots, V_m\}$ is the set of vertices in the mesh. Every element in D^t at time step t is an unrefined element.

In PARED, when an element is refined, it does not get destroyed. Instead, the refined element inserts itself into a tree. The refined mesh forms a forest of refinement trees, one per initial mesh element. Thus, for every element Ω_a in the initial mesh PARED maintains a refinement history tree τ_a where every element except a *leaf* element is the *parent* of two or more elements. The leaf elements of these trees form the most refined mesh M^t on which the numerical simulation is based at time t. These trees are used in many of our algorithms. For example, in PARED a mesh is coarsened in PARED by replacing all the children of a refined element by their parent. Thus, M^0 is the coarsest mesh that our system can manipulate. Our repartitioning algorithm also takes advantage of these trees; when an element is migrated to another processor all its descendants are migrated as well.

PARED uses a parallel and local h-refinement algorithm based on Rivara's longest edge bisection of triangular [10] and tetrahedral [11] unstructured meshes to adapt a mesh. This is a recursive procedure that in two dimensions splits each triangle Ω_a from a selected set of triangles R by adding an edge between the midpoint of its longest side and the opposite vertex [10]. The refinement propagates to adjacent triangles to maintain the conformality of the mesh. In three dimensions, a tetrahedron is bisected by inserting a triangle between the midpoint of its longest edge and the two vertices not included in that edge [11]. The details of our parallel adaptation algorithm are discussed in [4, 12]. We show that our refinement algorithm propagates refinement across processor boundaries and generates the same refined meshes as would the purely serial bisection algorithm [12].

After the adaptation phase, PARED determines if a user-supplied workload imbalance exists due to increases and decreases in the number of mesh elements on individual processors. If so, it invokes the procedure described in Section 3 to decide how to repartition mesh elements between processors. These elements and the corresponding vertices are migrated between the processors according to the procedure explained in [1, 13]. PARED is then ready to resume another round of equation solving, error estimation, mesh adaptation, mesh repartitioning, and work migration.

3. Partitioning Finite Element Meshes

In PARED each element is assigned to a unique processor and mesh vertices are shared if they are adjacent to elements assigned to different processors. This results in the partition $\Pi = \{\pi_1, \ldots, \pi_p\}$ of the mesh *by elements*, where π_i is the set of elements assigned to processor P_i. Communication is then performed across the edges (in two-dimensional problems) or faces (in three-dimensional problems) that separate two elements of the mesh.

A partition Π of a mesh $M(D, V)$ by elements is obtained from the dual graph G of the mesh. On a mesh partitioned by elements, the communication cost is a function of the cut size, $C_{cut}(\Pi(G))$ and, on machines with a high latency network, on the number of adjacent subdomains. Because there is a one-to-one relation between a partition $\Pi(M)$ of the elements of a mesh M and a partition of the vertices $\Pi(G)$ of its dual graph G, we do not make a distinction between $\Pi(M)$ and $\Pi(G)$.

3.1. Review of Graph Partitioning Methods

The problem of partitioning a graph into p subgraphs of approximately equal size while minimizing the number of edges joining vertices in different subgraphs is known as the *p-way graph partitioning problem*. This problem is NP-hard even in the simple case of bisecting a graph between two processors [14]. As a result, many heuristics have been proposed for it.

One of the most successful heuristics for partitioning unstructured FEM meshes is Recursive Spectral Bisection (RSB) [15]. Local heuristics, such as the Kernighan-Lin algorithm (KL) [16], complement spectral methods by further improving the quality of a partition.

RSB produces high quality partitions but, because of its high cost, it is usually restricted to relatively small graphs. For large graphs, multilevel methods such as Multilevel-KL [17] provide a better tradeoff between partition quality and

speed. Multilevel methods consist of three phases: *graph contraction*, *coarse graph partitioning*, and *projection and improvement*. In the contraction phase a series of graphs of decreasing size, G_0, G_1, ..., G_k, is constructed, typically by contracting edges between disjoint vertices. As each new graph, G_{j+1}, is constructed from its predecessor G_j, its edges and vertices inherit the weights of edges and vertices of G_j. In the second phase, G_k is partitioned among p processors. In the third phase for $j = k$ to 2, G_{j+1} is expanded to G_j and a local search heuristic, such as Kernighan-Lin, is used to improve the allocation of vertices to processors.

Unfortunately, many of the best serial graph partitioning heuristics do not easily accommodate parallel implementation. While Barnard and Simon [18] present a parallel implementation of their spectral algorithm, this approach shows poor scalability. The Kernighan-Lin heuristic used in the projection phase of many multilevel schemes is P-complete [19] and does not parallelize well. Some approaches [19, 20] overcome this problem by moving or swapping clusters of vertices rather than individual vertices. Nevertheless, this parallel process is communication intensive, it is difficult to obtain good performance and it is not efficient on relatively small graphs.

Geometric graph partitioning methods [21] rely on coordinate information, which in the case of finite element meshes, it is usually readily available. Geometric heuristics are scalable but it is shown in [22] that they produce worse partitions than spectral methods.

4. The Repartitioning Problem

Traditional parallel FEM systems partition the mesh in a preprocessing step. The mesh is then mapped to processors and the simulation starts. This static approach to mesh partitioning is not sufficient for methods that dynamically modify the mesh as is the case in adaptive schemes.

The mesh repartitioning problem [5, 6, 7, 23, 24] is not as widely studied as the standard graph partitioning problem. In addition to the traditional goals of balanced partitions and minimum edge cut, the repartitioning of a graph must satisfy a new set of requirements that arise from its dynamic nature, namely, it must frequently rebalance the load between processors. Therefore, the graph repartitioning must have a low cost relative to the solution time and it must be performed in parallel; it is inefficient to move the complete mesh to one processor to repartition it. Finally, the algorithm should consider the current assignment of the mesh so it does not result in unnecessary migration of data between processors.

The standard graph partitioning problem minimizes the cut size while maintaining balance whereas the repartitioning problem maintains balance while keeping both the cut size and the number of elements migrated small. The latter problem is formulated as minimizing $C_{\text{repartition}}(\widehat{\Pi}, \Pi, \alpha) = C_{\text{cut}}(\widehat{\Pi}) + \alpha C_{\text{migrate}}(\Pi, \widehat{\Pi})$ where Π is the current unbalanced partition, $\widehat{\Pi}$ is the desired balanced partition, and parameter α is used to penalize partitions that would only provide a marginal improvement in $C_{cut}(\widehat{\Pi})$ but require significant movement of data between processors.

Since the graph partitioning problem is a special case of the graph repartitioning problem in which $\alpha = 0$, graph repartitioning is also NP-hard and heuristics are needed for it. A natural one to use is the Kernighan-Lin algorithm with a gain function that reflects changes in the cost $C_{\text{repartition}}(\widehat{\Pi}, \Pi, \alpha)$ defined above. This idea is the basis for the heuristic introduced in Section 9.

5. Parallel Nested Repartitioning (PNR)

PARED uses a procedure for repartitioning adapted meshes that was originally outlined in [1, 13]. An initial coarse mesh $M^0(D, V)$ is refined where needed to obtain a refined mesh M^t at time t. As M^t is constructed, PARED builds a refinement history tree τ_a for each coarse element Ω_a in M^0 whose leaves are corresponds to elements of M^t. PARED constructs a dual graph G of M^0 that has one vertex w_a for each element Ω_a in M^0 and an edge between two vertices if the corresponding elements have a common edge in 2D or a common face in 3D. The weight of a vertex w_a in G is the number of leaves in τ_a. The weight of an edge (w_a, w_b) in G is the number of leaves of τ_a and τ_b that are adjacent. Rather than computing directly a partition of M^t, PARED invokes PNR that first computes a partition of M^0 using G.

After the current mesh M^t is refined to produce M^{t+1}, each processor notifies the coordinating processor P_C of the changes in vertex and edge weights of G, which is then partitioned. PARED then moves elements in M^{t+1} to potentially new processors by moving refinement history trees. This algorithm is sketched in Figure 2. It has an initial phase, P0, and three active phases, P1, P2, and P3. In P0 a mesh is refined; in P1 processors compute new edge and vertex weights for G; in P2 these weights are transmitted to the coordinating processor, P_C; in P3 G is repartitioned and processors are notified of the new assignments of refinement history trees.

Many of the heuristics designed for graph partitioning can also be used to repartition the updated graph G. Unfortunately, when these heuristics are applied to slightly different problems they can generate very different results. For example, standard graph partitioning algorithms such as Multilevel-KL or RSB usually compute a new distribution of the adapted mesh that is very different from the current one and require a large movement of elements and vertices

between processors. In Section 7 we discuss the sensitivity of heuristics to the repartition of G.

6. Quality of the Partitions Obtained from PNR

Because there are many more ways to partition the adapted mesh M^t than to partition M^0, a good partition of G does not necessarily imply a good partition of M^t. In this section to test the effectiveness of PNR we compare the partitions provided by PNR with those produced by Chaco's Multilevel-KL [17] on adaptively refined two- and three- dimensional meshes and show that the resulting partitions are of similar quality.

Our test problem involved computing a solution to Laplace's equation $\Delta u = 0$ defined in $\Omega^2 = (-1, 1)^2$ with the Dirichlet boundary conditions

$$g(x, y) = \cos(2\pi(x - y)) \frac{\sinh(2\pi(x + y + 2))}{\sinh(8\pi)}.$$

The analytical solution of this problem is known to be $u(x, y) = g(x, y)$. This solution is smooth but changes rapidly close to the corner (1,1). A similar problem has been defined in three dimensions.

The initial two- and three-dimensional meshes for this problems had 12,498 triangles and 9,540 tetrahedra, each of about the same size in each case. The mesh was adapted using the L_∞ norm creating a large number of elements in the region of rapid change in the solution, as shown in Figure 1. Eight levels of refinement were needed in the 2D case and five in the 3D case to reduce the error to 10^{-3}. The number of elements increased from 12,498 to 135,371 in the 2D case and from 9,540 to 70,185 in the 3D case. After each refinement, a new partition of the adapted mesh was computed using both Multilevel-KL and PNR with $\alpha = 0.1$. Shown in Figure 3 are the results of these comparisons for some of the levels of refinements. Clearly, PNR provides very high quality partitions.

6.1. Competitive Analysis of PNR

We have shown analytically that PNR can provide 2D mesh partitions that are competitive, that is, whose cut sizes are within a small constant factor of the best partitioning algorithm at the expense of a small additive change in the balance of mesh elements between processors. The 3D case is unresolved. The result is stated below.

Theorem 6.1 *Let the partition Π^t of the refined mesh $M^t(D, V)$ have cut size C and assign at most $(|G|/p)(1 + \epsilon)$ elements to any processor. Under the assumption that each coarse mesh element is refined uniformly to depth d, it is possible to produce from Π^t a partition Π^0 of $M^t(D, V)$ with cut size at most $9C$ that respects the boundaries of elements in $M^0(D, V)$ and for which each processor has at most $(|G|/p)(1 + \epsilon) + (p - 1)d^2$ mesh elements.*

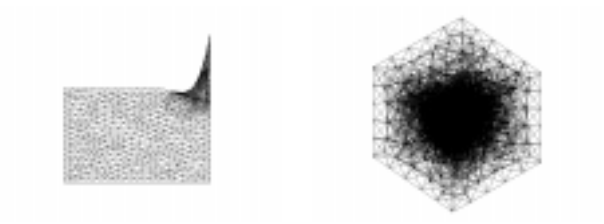


Figure 1. Irregular two- and three-dimensional regular meshes adaptively refined to solve Laplace's equation of a problem that exhibits high physical activity in one of its corners.

This result is established by exhibiting an algorithm that moves the boundaries of a partition of M^t so that it respects the boundaries of M^0. When a boundary between two processors passes through a coarse mesh element of M^0, we move it to the shorter periphery of the element (this causes a fixed expansion in the cut size) unless this will cause this number of elements to exceed $2d^2$, the number of elements into a coarse element is refined by a d-level refinement. In this case, the longer periphery is taken. We bound the number of elements that are displaced from one processor to another before the number exceeds $2d^2$ and use this to bound the expansion in the size of the cut when the longer periphery is taken.

7. The High Migration Cost of Standard Heuristics

RSB and Multilevel-KL are very effective methods for partitioning unstructured meshes. Nevertheless, when applied to the repartitioning of adapted meshes, the resulting partitions require a large movement of data between processors, an amount that is usually proportional to the size of the mesh, as determined through experiments. Figure 4 shows the results of repartitioning a series of 2D meshes using the RSB algorithm. (Similar results are obtained for 3D meshes and Multilevel-KL.) The meshes are those generated by the Laplace problem described in Section 6.

After a new mesh M^t was generated from a previous mesh mesh M^{t-1}, it was partitioned between 4 to 64 processors using RSB. The cut size before and after the partition, denoted $C_{cut}(\Pi^{t-1})$ and $C_{cut}(\widehat{\Pi}^t)$, are shown in Table 4. Also shown is $C_{migrate}(\Pi^t, \widehat{\Pi}^t)$, the amount of work that needs to be migrated, and $C_{migrate}(\Pi^t, \widetilde{\Pi}^t)$, where $\widehat{\Pi}^t$ is

P0. $M^0(D^0, V^0)$ is the initial mesh and $M^t(D, V)$ is the mesh after t adaptations.
$\widetilde{R} \subseteq D$ and $\widetilde{C} \subseteq D$ are the regions refined and coarsened at time t.
P1. In parallel, compute *ElemWeight*(Ω_a) and *EdgeWeight*(Ω_a, Ω_b) for $\Omega_a \in \widetilde{R} \cup \widetilde{C}$ and $\Omega_b \in D$.
P2. Each processor sends its new weights to P_C.
P3. P_C updates the graph G and computes a partition $\widehat{\Pi}^t = \{\widehat{\pi}_1^t, \ldots, \widehat{\pi}_p^t\}$.
for each processor P_i **do**
  for each vertex $w_a \in \pi_i^{t-1}$ and $w_a \in \widehat{\pi}_j^t$ but $i \neq j$ **do**
    P_C directs P_i to move element Ω_a and its refinement tree τ_a to P_j.
    P_i executes the move.
  end for
end for

Figure 2. Outline of the Parallel Nested Repartitioning Algorithm.

the partition obtained from RSB and $\widetilde{\Pi}^t$ is a permutation of $\widehat{\Pi}^t$ that minimizes the movement of elements [5]. As can be seen, in the best case, almost half the elements in the refined mesh need to be migrated.

8. Bounding the Migration Cost

In the previous section we have shown that standard graph partitioning algorithms, when applied to the repartitioning problem, often require significant movement of data between processors which creates serious contention for an interprocessor network. In this section we derive a lower estimate on the migration cost under certain reasonable assumptions. In the next section we present a new repartitioning heuristic. The migration cost for this heuristic is close to the lower bound derived in this section.

Assume that Π^{t-1} is a balanced distribution of a mesh between p processors and that m new elements are created in the refinement phase in only one processor, say P_o, resulting in an unbalanced partition Π^t. Also assume that balance can be obtained by restricting the migration of elements to be between adjacent processors. Let H^t be the processor connectivity graph at time t using the current distribution Π^t. H^t has one vertex for every processor and an undirected edge between two processors that have adjacent elements. To obtain a balanced distribution $\widehat{\Pi}^t$ processor P_o must send m/p elements to every other processor P_j, but only along the edges of H^t. This procedure effectively shifts the boundaries of the mesh and reduces the probability of creating disconnected subsets in each processor.

Let the minimum distance between processors P_i and P_j in H^t be $d_{i,j}$. The movement of m/p elements from P_o to P_j has a migration cost of $d_{o,j}(m/p)$ if only edges of H^t are used, giving the following total migration cost:

$$C_{migrate}(\Pi^t, \widehat{\Pi}^t) = \sum_{j \neq o} d_{o,j} \left(\frac{m}{p} \right).$$

For example, if processors in the graph H^t form a two-dimensional $\sqrt{p} \times \sqrt{p}$ mesh and P_o is located in one of its corners, the total migration cost required to rebalance the mesh after creating m new elements in P_i is

$$C_{migrate}(\Pi^t, \widehat{\Pi}^t) \leq 2(\sqrt{p} - 1)(p - 1)\frac{m}{p} \leq 2\sqrt{p}\, m.$$

Under these assumptions, the total migration cost $C_{migrate}(\Pi^t, \widehat{\Pi}^t)$ only depends on the number of processors p and the number of new elements m and is independent of the mesh size.

9. Minimizing the Migration Cost

In this section we introduce a new heuristic that we have developed to repartition the weighted graph G located in the coordinator. It greatly reduces the number of mesh elements that need to move in order to repartition a good unbalanced partition.

The critical step in PNR is the selection of the graph partitioning algorithm to partition and repartition the dual weighted graph G in the coordinator. PARED can use a variety of algorithms to partition G, including Chaco's implementation of RSB and Multilevel-KL. Although traditional graph partitioners generate partitions with small cuts, they do not consider migration cost.

In PNR we use a standard multilevel algorithm for the initial partition of G. On subsequent repartitions we use a Multilevel-KL heuristic of the type described in Section 3.1 that is modified in two ways: a) we do not partition the coarsest graph G_k that results from the graph contraction phase, and b) we use a form of KL that is designed to minimize the migration while keeping the cut small and the processors balanced.

The KL heuristic is based on a gain function that is associated with vertices of G. The typical multiprocessor version of the KL heuristic, which is designed to minimize the

2D Mesh												
	Multilevel-KL						PNR					
Level	4	8	16	32	64	128	4	8	16	32	64	128
0	179	333	525	792	1141	1614	157	297	465	739	1043	1523
1	202	335	534	801	1167	1702	197	343	521	773	1164	1633
2	263	445	674	1023	1500	2118	245	437	675	996	1458	2076
3	270	473	775	1194	1748	2456	305	471	745	1120	1609	2316
4	350	571	895	1400	2080	2906	363	571	932	1352	1995	2809
5	388	642	1061	1595	2324	3341	350	624	980	1495	2179	3134
6	448	749	1202	1829	2706	3945	444	733	1175	1775	2620	3699
7	493	830	1357	2111	3112	4503	563	808	1351	2048	2971	4315
8	554	950	1547	2337	3544	5151	539	994	1557	2360	3595	5152

3D Mesh												
	Multilevel-KL						PNR					
Level	4	8	16	32	64	128	4	8	16	32	64	128
0	334	489	674	935	1174	1437	372	536	737	931	1193	1458
1	321	478	729	975	1230	1495	382	517	682	979	1226	1483
2	366	559	785	1046	1350	1667	364	572	819	1088	1406	1695
3	398	681	979	1349	1717	2120	406	698	975	1302	1716	2038
4	631	1020	1453	1893	2441	3024	618	999	1481	1935	2410	2761
5	1243	1742	2561	3380	4374	5446	1377	1895	2551	3374	4306	5225

Figure 3. Comparison of the quality of the partitions produced by Multilevel-KL and PNR. The tables show the number of shared vertices obtained by partitioning a sequence of locally adapted meshes with Multilevel-KL and PNR into 4 to 128 subsets.

cut size while balancing processors, uses a gain function that measures the change in the cut size and moves vertices from sets that have too many of them to sets that have too few. The latter insures that balance is maintained.

Our variant of the KL heuristic addresses all three measures, balance, cut size, and migration cost by attaching a gain function to vertices that reflects changes in the measure $C_{repartition}(\Pi^t, \widehat{\Pi}^t, \alpha, \beta)$ defined below where $\alpha, \beta > 0$ are constants that reflect the cost of migration and balance relative to the cost of cut size.

$$\begin{aligned} C_{repartition}(\Pi^t, \widehat{\Pi}^t, \alpha, \beta) &= C_{cut}(\widehat{\Pi}^t) + \alpha C_{migrate}(\Pi^t, \widehat{\Pi}^t) \\ &+ \beta C_{balance}(\widehat{\Pi}^t). \end{aligned} \quad (1)$$

Here

$$C_{balance}(\widehat{\Pi}^t) = \sum_i^p \left(weight(\widehat{\pi}_i^t) - \frac{weight(\widehat{\Pi}^t)}{p} \right)^2$$

where $\widehat{\pi}_i{}^t$ is the ith subset in the partition $\widehat{\Pi}^t$.

As in Multilevel-KL, we maintain a square table with an entry for each pair of subsets consisting of priority queues based on gains. That is, the possible movements between a pair of subsets is sorted by potential gain. To implement the local heuristic, we select the vertex movement with largest gain from this table, move the vertex between subsets and update the entries in the table corresponding to adjacent vertices. The moved vertex is marked so it is not inserted in the table again. This process iterates through the heads of the P^2 priority queues. The removing of the maximum gain from a priority queue and neighbor update time is bounded by $O(\log(n))$, where n is the number of boundary elements in a subdomain π_i. A vertex move between π_i and π_j modifies the difference $weight(\pi_i) - weight(\pi_j)$. Rebuilding these priority queues requires $O(n)$ steps. Fortunately, n, the number of vertices in G, is small.

We performed the tests described in Section 7 using our new PNR algorithm to partition and repartition the coarse mesh. These results are shown in Figure 5 for the same two- and three-dimensional mesh In these tests we used $\alpha = 0.1$ and $\beta = 0.8$, obtaining balanced partitions with $\epsilon < 0.01$. The quality of the partitions measured by the number of shared vertices generated by PNR and RSB (shown in Table 4) is very similar. On the other hand, the total migration cost $C_{migrate}(\Pi^t, \widehat{\Pi}^t)$ is much smaller than those measured previously and does not significantly increase with mesh size.

10. A Transient Problem

In the previous section we have shown that on locally adapted meshes PNR produces partitions with a cut size similar to the ones produces by standard graph partitioning

Proc	M^{t-1} (before ref) Elem	$C_{cut}(\Pi^{t-1})$	M^t (after ref) Elem	$C_{cut}(\widetilde{\Pi}^t)$	$C_{migrate}(\Pi^t, \widehat{\Pi}^t)$	$C_{migrate}(\Pi^t, \widetilde{\Pi}^t)$
4	5094	99	5269	95	2627	2627
8		168		159	3341	831
16		273		274	4458	1551
32		421		421	5046	2270
64		615		629	5129	2354
4	11110	137	11411	152	9192	2010
8		249		250	9696	3383
16		405		410	10444	4747
32		633		647	11061	5684
64		926		960	11230	5284
4	23749	311	23902	291	16477	14519
8		488		480	19182	13117
16		700		670	22620	11104
32		1000		980	23441	11374
64		1463		1425	23530	11711
4	49915	331	50072	410	35601	23152
8		569		680	49190	18507
16		920		977	49264	22147
32		1408		1431	49776	21972
64		2067		2159	50050	23639
4	103585	788	103786	863	38433	38433
8		1121		1193	77099	43272
16		1690		1728	93892	51125
32		2380		2403	99397	50264
64		3297		3310	102277	50278

Figure 4. Migration cost resulting from repartitioning a series of two-dimensional unstructured meshes of increasing size using the RSB algorithm. M^{t-1} is the mesh before refinement and distributed according a balanced partition Π^{t-1} obtained from RSB. M^t is the refined mesh. $\widehat{\Pi}^t$ is a new balanced partition of M^t also produced by the RSB algorithm and $\widetilde{\Pi}^t$ is a permutation of $\widehat{\Pi}^t$ that minimizes data movement.

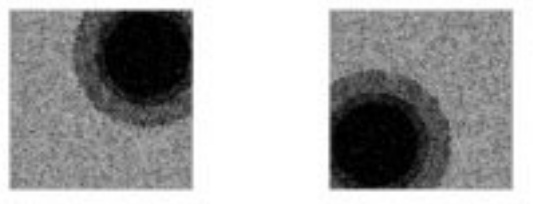
methods but requires a much smaller data movement. In this section we use our method to follow a disturbance across a domain. We show that the repeated use of our heuristic maintains the quality of the partitions while retaining its small migration cost.

To study these issues we solve Poisson's equation $\Delta u = f$ over the domain $\Omega^2 = (-1, 1)^2$ where the solution $u(x, y, t)$ is the known function

$$u(x, y, t) = \frac{1}{1 + 100(x + t)^2 + 100(y + t)^2}$$

and compare the partitions produced by RSB and PNR. We solved this problem for 100 time steps in which t varies from -0.5 to 0.5. This function is smooth with a peak of 1 at the coordinates $x = y = -t$ and zero almost everywhere else. Thus, as t varies from -0.5 to 0.5, the peak moves along a diagonal from $(0.5, 0.5)$ to $(-0.5, -0.5)$.

The initial and final adapted meshes are shown in Figure $6(a)$ and (b). In each of the 100 time steps, after the solution and adaptation of the mesh, we also compute a new partition $\widehat{\Pi}^t$ using RSB and PNR.



(a) (b)

Figure 6. (a) and (b) show the adapted mesh at $t = -0.5$ and $t = 0.5$.

Figure 7 shows the number of shared vertices of each partition of $\widehat{\Pi}^t$ obtained by RSB and PNR for 4, 8, 16 and 32 processors. In PNR we used the parameters $\alpha = 0.1$ and $\beta = 0.8$ in Equation 1. The resulting partitions have less than 0.01 imbalance between subsets. Even though PNR is a local heuristic, in these examples the cut size of the

Proc	M^{t-1} (before ref) Elem	$C_{cut}(\Pi^{t-1})$	M^t (after ref) Elem	$C_{cut}(\widetilde{\Pi}^t)$	$C_{migrate}(\Pi^t, \widehat{\Pi}^t)$	$C_{migrate}(\Pi^t, \widetilde{\Pi}^t)$
4	5094	89	5269	91	132	132
8		154		162	280	280
16		261		290	430	430
32		394		442	483	483
64		591		642	681	681
4	11110	151	11411	151	226	226
8		260		262	489	489
16		400		415	773	773
32		601		659	967	967
64		866		935	1146	1146
4	23749	197	23902	199	115	115
8		347		352	245	245
16		564		578	332	332
32		883		932	415	415
64		1302		1351	512	512
4	49915	291	50072	289	156	156
8		547		549	251	251
16		885		899	373	373
32		1346		1368	531	531
64		1995		2038	581	581
4	103585	426	103786	429	151	151
8		802		789	321	321
16		1314		1319	469	469
32		1970		1971	623	623
64		2982		3042	731	731

Figure 5. Migration cost resulting from repartitioning a series of two-dimensional unstructured meshes of increasing size using the PNR algorithm. M^{t-1} is the mesh before refinement and distributed according a balanced partition Π^{t-1} obtained from PNR. M^t is the refined mesh. $\widehat{\Pi}^t$ is a new balanced partition of M^t also produced by the PNR algorithm and $\widetilde{\Pi}^t$ is a permutation of $\widehat{\Pi}^t$ that minimizes data movement.

partitions that it produces does not deteriorate over time and is similar to the ones produced by a very successful graph partitioning method, RSB.

Figure 8 shows the number of elements migrated by the three methods, a) RSB, b) RSB after computing a permutation $\widetilde{\Pi}^t$ of $\widehat{\Pi}^t$ that reduces migration, as explained in Section 7, and c) PNR. RSB usually migrates between 50% and 100% of the total number of elements between repartitions. Although not reported here, the results for Multilevel-KL are similar. The total movement significantly decreases with the permutation $\widetilde{\Pi}^t$ of the subsets to processors but we still observe peaks of more than 46% of the total elements and an average movement of 21% for 32 processors. This method is characterized by sharp peaks, where some repartitions resulted in small migrations while others require a significant movement of data.

The total migration cost resulting from PNR is small compared with the other methods (on the average it is between 1.2% of the elements for 4 processors and 5.5% for 32 processors) and is smooth. PNR resulted in only two peaks with more than 10% data movement between iterations. The total data movement produced by PNR over all iterations was between 12% and 27% of the total number of elements moved by the permuted RSB heuristic.

11. Conclusion

In this paper we introduce PNR, a new graph partitioning algorithm that provides balanced multi-processor partitions with small cut size of two- and three-dimensional meshes. We show that PNR moves very small numbers of mesh elements when used to repartition meshes even when only a few elements have been refined or coarsened. This feature is important when meshes are frequently adapted to follow disturbances in the solution of computational science problems using the FEM.

We have shown that PNR provides balanced partitions with small cut size through experiments in which the partitions it produces are compared with those produced by some of the best partitioning algorithms. For the 2D problem we

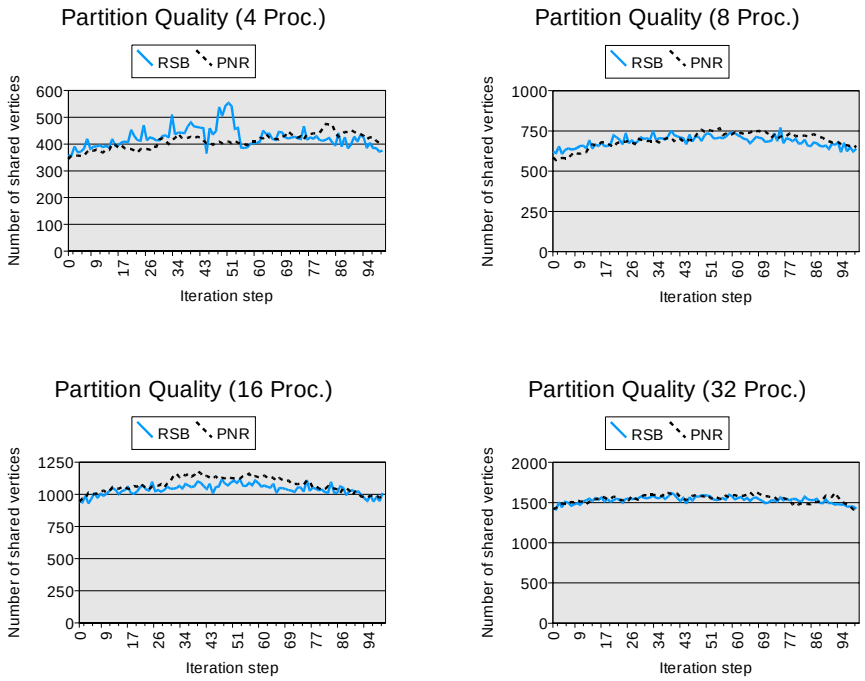


Figure 7. Quality of the partitions measured by the number of shared vertices produced by RSB and PNR for 4, 8, 16 and 32 processors for each of the 100 time steps between $t = -0.5$ to $t = 0.5$.

have also shown through competitive analysis that partitions produced by PNR can have a cut size that is at worst a small multiple of that produced by the best partitioning algorithms on the finest FEM graph.

We have studied the amount of data that is moved by PNR and shown through experiment that it is comparable to the amount predicted by analysis. We have also conducted an experiment using PNR in the system PARED to track a disturbance that moves in space over a time. We show that the number of mesh elements that are moved by PNR during this computation is very small yet we obtain cut sizes comparable to those produced by Recursive Spectral Bisection and Multilevel-KL, two standards for graph partitioning.

References

[1] José G. Castaños and John E. Savage. The dynamic adaptation of parallel mesh-based computation. In *SIAM 7th Symposium on Parallel and Scientific Computation*, 1997.

[2] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *Proceedings of the 6th SIAM conference on Parallel Processing for Scientific Computing*, pages 711–718, 1993.

[3] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[4] José G. Castaños and John E. Savage. PARED: a framework for the adaptive solution of PDEs. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.

[5] Rupak Biswas and Leonid Oliker. Load balancing unstructured adaptive grids for CFD. In *SIAM 7th Symposium on Parallel and Scientific Computation*, 1997.

[6] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel Processing and Distributed Computing*, 47:102–108, 1997.

[7] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusing schemes for repartitioning of adaptive meshes. *Journal of Parallel Processing and Distributed Computing*, 47:109–124, 1997.

[8] Y.F. Hu and R.J. Blake. An optimal dynamic load balancing algorithm. Technical Report Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA4 4AD, UK, 1995.

[9] Message Passing Interface Forum:. MPI: A message passing interface standard, 1994.

[10] Maria Cecilia Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *Inter-*

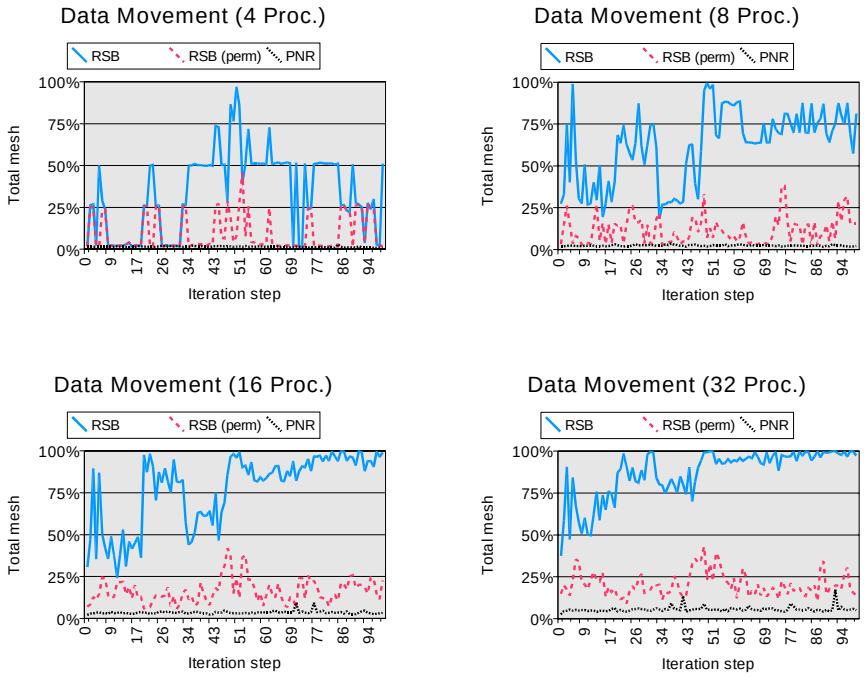


Figure 8. Elements moved between time steps of partitions produced by RSB, permuted RSB and PNR for 4, 8, 16 and 32 processors.

national Journal for Numerical Methods in Engineering, 28:2889–2906, 1989.

[11] Maria Cecilia Rivara. A 3-D refinement algorithm suitable for adaptive and multi-grid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[12] José G. Castaños and John E. Savage. Parallel refinement of unstructured meshes. In *IASTED International Conference on Parallel and Distributed Computing and Systems*, 1999.

[13] José G. Castaños and John E. Savage. The dynamic adaptation of parallel mesh-based computation. Technical Report CS-96-31, Department of Computer Science, Brown University, October 1996.

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

[15] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11(3):430–452, 1990.

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

[17] B. Hendrickson and R. Leland. The Chaco user's guide, version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, 1995.

[18] S. T. Barnard and H. Simon. A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. In *Proceedings of the seventh SIAM conference on Parallel Processing for Scientific Computing*, 1995.

[19] J. E. Savage and M. Wloka. Parallelism in graph partitioning. *Journal of Parallel and Distributed Computing*, 13:257–272, 1991.

[20] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. Technical Report CORR 95-036, University of Minnesota, Dept. of Computer Science, 1995.

[21] G. L. Miller, S.H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Sparse Matrix Computations:Graph Theory Issues and Algorithms*, pages 57–84. New York, 1993.

[22] H. D. Simon. Partitioning of unstructured meshes for parallel processing. *Computing Systems Eng.*, 1991.

[23] R. D. Williams. DIME: Distributed irregular mesh environment. Technical Report C3P 861, California Institute of Technology, 1990.

[24] P. Diniz, S. Plimpton, B. Hendrickson, and R. Leland. Parallel algorithms for dynamically partitioning unstructured grids. In D. Bailey et al., editor, *Parallel Processing for Scientific Computing*, pages 615–620. SIAM, 1995.