Submitted to Supercomputing 1997

Parallel Multilevel Diffusion Algorithms for Repartitioning of Adaptive Meshes *

Kirk Schloegel, George Karypis, and Vipin Kumar

University of Minnesota, Department of Computer Science and Army HPC Center

(kirk,karypis,kumar)@cs.umn.edu

Technical Report #97-014

Abstract

Graph partitioning has been shown to be an effective way to divide a large computation over an arbitrary number of processors. A good partitioning can ensure load balance and minimize the communication overhead of the computation by partitioning an irregular mesh into p equal parts while minimizing the number of edges cut by the partition. For a large class of irregular mesh applications, the structure of the graph changes from one phase of the computation to the next. Eventually, as the graph evolves, the adapted mesh has to be repartitioned to ensure good load balance. Failure to do so will lead to higher parallel run time. This repartitioning needs to maintain a low edge-cut in order to minimize communication overhead in the follow-on computation. It also needs to minimize the time for physically migrating data from one processor to another since this time can dominate overall run time. Finally, it must be fast and scalable since it may be necessary to repartition frequently. Partitioning the adapted mesh again from scratch with an existing graph partitioner can be done quickly and will result in a low edge-cut. However, it will lead to an excessive migration of data among processors. In this paper, we present new parallel algorithms for robustly computing repartitionings of adaptively refined meshes. These algorithms perform diffusion of vertices in a multilevel framework and minimize data movement without compromising the edge-cut. Furthermore, our parallel repartitioners include parameterized heuristics to specifically optimize edge-cut, total data migration, or the maximum amount of data migrated into and out of any one processor. Our results on a variety of synthetic meshes show that our parallel multilevel diffusion algorithms are highly robust schemes for repartitioning adaptive meshes. The resulting edge-cuts are close to those resulting from partitioning from scratch with a state-of-the-art graph partitioner, while data migration is substantially reduced. Furthermore, repartitioning can be done very fast. Our experiments show that meshes with around eight million vertices can be repartitioned on a 256-processor Cray T3D in only a couple of seconds.

1 Introduction

Mesh partitioning is an important problem which has applications in many areas, including scientific computing. In irregular mesh applications, the amount of computation associated with a grid point is represented by the weight of its associated vertex. The amount of interaction required between two grid points is represented by the weight of the edge between the associated vertices. Efficient parallel execution of these irregular grid applications requires the partitioning of the associated graph into p parts with the following two constraints: (i) Each partition has an equal amount of total vertex weight; (ii) The total number of edges cut by the partitions (thereafter referred to as *edge-cut*) is minimized. Since the weight of any given edge represents the amount of communication required between nodes, minimizing the number of edges cut by the partition tends to minimize the overall amount of communication required by the computation. This problem has been well defined and discussed in previous work [2, 7].

*This work was supported by NSF CCR-9423082, by Army Research Office contract DA/DAAH04-95-1-0538, by Army High Performance Computing Research Center cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, by the IBM Partnership Award, and by the IBM SUR equipment grant. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: **http://www.cs.umn.edu/˜karypis**

For a large class of irregular grid applications, the computational structure of the problem changes in an incremental fashion from one phase of the computation to another. For example, in adaptive meshes [1], areas of the original graph are selectively coarsened or refined in order to accurately model the dynamic computation. This causes the weights of the vertices and the edges to change. Eventually, as the graph evolves, it becomes necessary to correct the partition in accordance with the structural changes in the computation and to migrate a certain amount of computation between processors. Thus, we need a partitioning or repartitioning algorithm to redistribute the adapted graph. This algorithm should satisfy the following constraints.

1. **It robustly balances the graph.** Failure to balance the graph will lead to load imbalance, which will result in higher parallel run time. In order to make the repartitioning algorithm general it must be able to balance graphs from a wide variety of application domains.

2. **It minimizes edge-cut.** The redistributed graph should have a small edge-cut to minimize communication overhead in the follow-on computation.

3. **It minimizes vertex migration time**. Once the mesh is repartitioned, and before the grid computation can begin, data associated with the migrated vertices also needs to be moved. In many adaptive computations, the data associated with each vertex is very large. The time for movement of the data can dominate overall run time, especially if the mesh is adapted frequently.

4. **It is fast.** The computational cost of repartitioning should be inexpensive since it is done frequently. Also, since the problem studied in this paper is relevant only in the parallel context, the repartitioning algorithm should be parallelizable.

If the adapted graph is partitioned from scratch using a state-of-the-art multilevel graph partitioner such as the one implemented in METIS [6], then it will reasonably optimize criterion 1 and 2. Since a highly parallel formulation of this algorithm exists [6], criterion 4 can also be met to a large extent. Partitioning from scratch will, however, result in high vertex migration, as the partitioning does not take the initial location of the vertices into account. A partitioning method that incrementally constructs a new partition as simply a modification of the input partition (e.g. by diffusion [11]) can potentially move a much smaller number of vertices. Such a method can also be potentially faster than partitioning the graph from scratch.

Repartitioning schemes that incrementally modify an existing partition have been quite successful on graphs that are small perturbations of the original graphs [9, 11]. For these graphs, such a scheme inherits a good (*i.e.*, low edge-cut), but imbalanced starting point in the initial partition. It then attempts to fix the imbalance of this partition while maintaining its good edge-cut. It does this by minimizing the amount of disturbance (*i.e.*, vertex migration) to the initial partition in the balancing phase. For only slightly imbalanced graphs, the initial partition does not need to be disturbed very much, and so these algorithms are able to maintain an edge-cut comparable with the initial partition. However, if the initial partition is highly imbalanced, then many vertices need to move in order to balance the graph. Thus, even if the disturbance to the initial partition is minimized, the final partition will necessarily end up quite a bit removed from it. Hence, the balancing phase of such a method will increase the edge-cut considerably. Local refinement [9, 11] can only provide a limited improvement in the edge-cut of the resulting partition.

One promising solution to the problem of edge-cut degradation as the degree of imbalance increases in size and complexity is the use of a multilevel scheme [2, 4] that takes the initial location of the vertices into consideration. The multilevel paradigm allows the local refinement to be performed at multiple coarsened versions of the graph, which has been shown to be quite effective in reducing the edge-cut. In addition to the refinement, the movement of graph vertices (to achieve load balance) can also be done at multiple coarsened versions. This multilevel diffusion scheme can move large chunks of vertices at coarser levels, and then achieve better load balance at finer levels. In a multilevel context, a global picture of the graph [3] can be used to guide graph balancing while utilizing a multilevel view to guide refinement.

We have developed multilevel diffusion schemes that incrementally constructs a new partition of the graph [10]. These schemes coarsen the graph, begin multilevel diffusion on the coarsest graphs, and then begin multilevel refinement as soon as the partition is sufficiently balanced. By applying diffusion on the coarsest graphs, we speed up the costly diffusion process, since this is done on relatively small graphs. We are also able to effectively control the trade-off between edge-cut and vertex migration since diffusion and refinement are linked in a multilevel context. In this paper, we present a brief overview of these schemes and provide experimental results of an MPI-based implementation of these schemes on a 256-processor Cray T3D for a variety of adaptive graphs. Our experimental results show that we can repartition adaptive meshes in less time than it takes to partition the meshes from scratch, maintain a comparable edge-cut, and move far less vertices in the process.

2 Notations, Definitions, and Issues

Our discussion will include the concepts of both vertex weight and vertex size. Vertex weight is the computational cost of the work represented by the vertex while size is its migration cost. Thus, the repartitioner should attempt to balance the graph with respect to vertex weight while minimizing vertex migration with respect to vertex size. Depending on the representation and storage policy of the data, size and weight may not necessarily be equal. One example of such a situation arises in [8].

Let $G = (V, E)$ be an undirected graph of V vertices and E edges and P be a set of p processors. Let s_i represent the cost of movement of vertex v_i. We will refer to s_i as the *size* of vertex i. Let w_i represent the weight (*i.e.*, computational work) of vertex v_i and $w_e(v_1, v_2)$ equal the amount of communication required between v_1 and v_2. A vertex's *density* is equal to its weight divided by its size. We denote $B(q)$ as the set of vertices with partition q. The weight of any partition q can then be defined as:

$$W(q) = \sum_{v_i \in B(q)} w_i$$

A vertex is *clean* if its current partition is its initial partition on the input graph. Otherwise it is *dirty*. A vertex is a *border* vertex if one of its adjacent vertices is in another partition. If so, then all such partitions are the vertex's *neighbor* partitions. If a partition contains at least one vertex which has another partition as a neighbor partition, then those two partitions are neighbor partitions to each other.

TotalV is defined as the sum of the sizes of vertices which change partitions as the result of partitioning or repartitioning. Thus, it is the sum of the sizes of the dirty vertices. MaxV is defined as the maximum of the sums of the sizes of those vertices which migrate into and out of any one partition as a result of partitioning or repartitioning.

2.1 Multilevel Diffusion Algorithms: A Brief Overview

Our serial multilevel diffusion algorithms have three phases: a coarsening phase, a multilevel diffusion phase, and a multilevel refinement phase. In the coarsening phase, only pairs of nodes that belong to the same partition are considered for merging. Hence, the initial partition of the coarsest level graph is identical to the input partition of the graph that is being repartitioned, and thus does not need to be computed.

In the multilevel diffusion phase, balance is sought by means of either ***undirected*** or ***directed*** diffusion. In the case of undirected diffusion, the border vertices are visited in a random order. If a vertex belongs to an overweight partition (*i.e.*, a partition whose weight is higher than the average weight), then it is moved to an adjacent partition with lower weight. If there are more than one adjacent partition satisfying this condition, the one that leads to the smaller edge-cut is selected. If a vertex belongs to a partition with average weight, then it is moved to a partition that leads to a reduction in the edge-cut as long as it does not make the destination partition overweight. Again, if there are more than one partition satisfying this condition, the one that leads to the smaller edge-cut is selected. If a vertex belongs to an under-weight partition it is not moved. Note that a vertex belonging to an overweight partition can move even if this migration leads to an increase in the edge-cut. This process is repeated for a small number of steps or until either balance is obtained or no progress is made in balancing. Note that in this scheme, diffusion is performed using only local information.

In the case of directed diffusion, a global picture is used to guide the vertex migration. This global picture is computed by the 2-norm minimization solution described in [3]. The result of this computation is a transfer matrix that indicates how much weight needs to be transfered between neighboring partitions. Using this transfer matrix, the directed diffusion scheme works as follows. Again, the boarder vertices are visited in a random order. If a vertex is neighbors with a partition which according to the transfer matrix needs to transfer work to, then that vertex can be migrated to the neighbor partition. If a vertex is neighbors with more than one such partition, it is migrated to that partition which will produce the highest gain in edge-cut. The vertex is migrated even if the gain in edge-cut is negative. After a vertex is migrated, the transfer matrix is updated to reflect the vertex migration (*i.e.*, the weight of the vertex that was moved is subtracted from the appropriate entry of the transfer matrix). After each border vertex is visited exactly once, the process repeats until either balance is obtained or no progress is made in balancing.

In both of these schemes used during the multilevel diffusion phase, it may not be possible to balance the graph at the coarsest graph level. That is, there may not be sufficiently fine vertices on the coarsest graph to allow for total balancing.

If this is the case, the graph needs to be uncoarsened one level in order to increase the number of finer vertices. The process described above is then begun on the next coarsest graph.

After the graph is balanced, multilevel diffusion ends and multilevel refinement begins on the current graph. Here, the emphasis is on improving the edge-cut. The border vertices are again visited randomly and are checked to see if they can be migrated to another partition so that

1. maintain the edge-cut, maintain the balance, and the selected partition is the vertex's initial partition from the input graph, or

2. decrease the edge-cut while maintaining the graph balance, or

3. maintain the edge-cut and improve graph balance.

If so, vertices are migrated. These three conditions make up the *refinement phase vertex migration criteria*. Criterion 1 allows vertices to migrate to their initial partitions (as long as the migration does not increase the edge-cut and worsen the load balance), and therefore, to lower TotalV and possibly MaxV [10].

Dynamic suppression of vertices as described in [10] is also utilized during multilevel diffusion to reduce MaxV. Essentially with this variation, the migration of relatively non-dense vertices is suppressed during the balancing phase. The concept of vertex cleanness as described in [10] is incorporated in multilevel refinement in order to reduce the TotalV. That is, only those vertices which have either been displaced in the process of balancing or those which will result in a sufficiently high edge-cut-gain-to-size-ratio are eligible for migration during refinement.

A complete description of the scheme and extensive experimental results conducted on a single processor for a variety of adapted meshes are presented in [10].

3 Parallel Multilevel Diffusion Algorithms

We have developed parallel versions of the multilevel diffusion algorithms as follows. Vertices are initially assumed to be distributed across p processors. This division of vertices corresponds to the original partition of a static partitioner and is assumed to be of good quality (*i.e.*, low edge-cut). However, the sums of the vertex weights of the vertices resident on each processor are assumed to be variant. Thus, the original partition is not balanced and so there is a need for repartitioning.

Our parallel algorithms begin with a coarsening phase in which a sequence $G_i = (V_i, E_i)$ for $i = 0, 1, ..., m$, of successively coarser graphs is constructed. Graph G_{i+1} is constructed from G_i by first computing a matching of vertices of G_i and then collapsing together the matched vertices. The matchings computed are restricted to vertices residing on the same processors. By adhering to this restriction, coarsening is inherently very parallel. Otherwise, this phase is identical as described in [7].

The parallel formulation of the multilevel diffusion phase depends on whether or not we are using directed or undirected diffusion. In the case of directed diffusion, we chose to perform the directed diffusion for the coarsest graph serially. Since the coarse graph is very small (its size is proportional to the number of processors), this serial computation does not significantly affect the overall performance and scalability of our parallel multilevel directed diffusion algorithm. Furthermore, we use the additional processors to obtain a better directed diffusion as follows. The graph is broadcast to all processors. Each processor then simultaneously balances the coarsest graph using the directed diffusion algorithm described in Section 2.1. Since the diffusion scheme is inherently random, each processor computes a potentially unique partition. The best partition is selected. The selection criteria are either; (i) lowest edge-cut, (ii) lowest TotalV, (iii) lowest MaxV, or (iv) greatest balance. In our experiments, we select the partition that has the lowest edge-cut. In some cases, the coarsest graph may be too coarse to allow for complete balancing. For this reason we use the parallel undirected diffusion algorithm (described in the next paragraph) to balance any minor imbalances.

The parallel formulation of the undirected diffusion algorithm is modeled after our coarse-grained parallel multilevel refinement algorithm [7]. Each iteration of the parallel multilevel refinement algorithm consists of two sub-phases. During the first sub-phase, vertices are migrated only from lower- to higher-numbered partitions. During the second sub-phase, vertices are migrated from higher- to lower-numbered partitions. In this way, unexpected edge-cut increases caused by the simultaneous migration of neighboring vertices is avoided. Furthermore, our schemes avoids any bias towards the lower- or higher-numbered partitions, by using at each step a random partition ordering. In each sub-phase, vertices are visited and selected for migration according to the criteria for undirected diffusion described in Section 2.1.

The parallel formulation of the multilevel refinement algorithm is similar to that for undirected diffusion with the exception that vertices are moved according to the *refinement phase vertex migration criteria* described in Section 2.1.

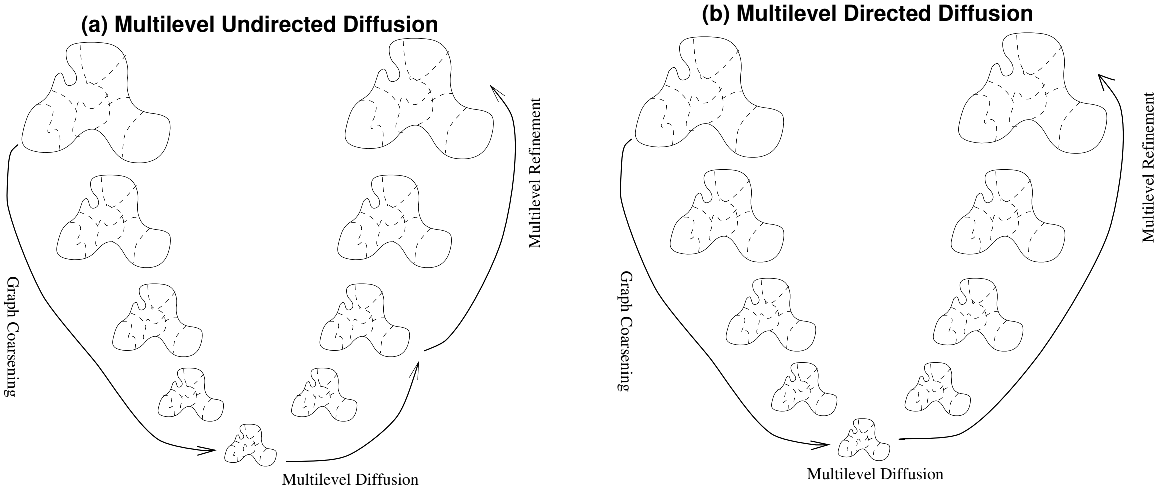

Figure 1: Multilevel Diffusion Algorithms: (a) undirected diffusion and (b) directed diffusion.

Furthermore, the concept of vertex cleanness as described in [10] is also employed in our scheme in order to reduce TotalV. Otherwise, our multilevel refinement algorithm is identical to the coarse-grained parallel multilevel refinement algorithm described in [7].

In summary, as illustrated in Figure 1, our parallel multilevel diffusion algorithms are made up of three phases, graph coarsening, multilevel diffusion, and coarse-grained multilevel refinement. In the case of undirected diffusion all three phases are done in parallel, whereas in the case of directed diffusion two of the three phases, coarsening and refinement, are done in parallel, and the diffusion is done serially on all processors.

4 Experimental Results

We tested our parallel multilevel repartitioning algorithms on a Cray T3D with 256 processors. Each processor on the T3D is a 150Mhz Dec Alpha (EV4). The processors are interconnected via a three dimensional torus network that has a peak unidirectional bandwidth of 150Bytes per second, and a small latency. We used Cray's MPI library for communication. Cray's MPI achieves a peak bandwidth of 45MBytes and an effective startup time of 57 microseconds.

We evaluated the performance of our parallel adaptive multilevel diffusion algorithms on four medium to large size graphs arising in finite element computations. The characteristics of these graphs are described in Table 1. For each graph we synthetically generated adaptive graphs by randomly changing the weights of the vertices. For example, on 64 processors, we first computed a 64-way partition and moved the graph so that processor P_i stores the vertices that belong to partition i. Next, each processor P_i selects a random number r_i between zero and the number of vertices that it stores, and randomly selects r_i of its vertices and changes their weight from one to three. We found that this scheme leads to graphs that are about 50% load imbalanced, *i.e.*, there is a partition whose weight is 50% higher than the average weight. These synthetically generated graphs are then used as the input to our parallel multilevel diffusion and partitioning algorithms. In all of our experiments, a graph that is 5% load imbalance is assumed to be well balanced, and in all experiments this load imbalance level is easily achieved.

Graph Name	Number of Vertices	Number of Edges	Description
MRNGA	257000	505048	Dual of a 3D Finite element mesh
MRNGB	1017253	2015714	Dual of a 3D Finite element mesh
MRNGC	4039160	8016848	Dual of a 3D Finite element mesh
MRNGD	7833224	15291280	Dual of a 3D Finite element mesh

Table 1: The various graphs used in the experiments.

Table 2 shows the results obtained by our parallel multilevel directed and undirected diffusion algorithms for the test problems on 64, 128, and 256 processors. For each problem, this table shows the edge-cut of the resulting partitioning, the total number of vertices that needs to be moved (TotalV), the maximum number of vertices that needs to be moved in and out of any particular processor (MaxV), as well as the amount of time (in seconds) required to compute the repartitioning. The rows labeled 'Scratch' show similar performance metrics for the scheme in which the adaptive graph is being repartitioned from scratch using the parallel multilevel k-way partitioning algorithm described in [7]. Finally, the rows labeled 'Imbalance' indicate the load imbalance resulted by our synthetic adaptation scheme for each one of the graph-processor combinations. As you can see, the load imbalance ranges between 1.42 (42%) and 1.52 (52%). Note that MRNGD could not run on 64 processors due to the limited amount of memory on each processor.

Graph	Scheme	64 Processors				128 Processors				256 Processors			
		Cut	TotalV	MaxV	Time	Cut	TotalV	MaxV	Time	Cut	TotalV	MaxV	Time
MRNGA	Imbalance	1.4244				1.4391				1.4971			
	Undirected	26457	24819	2427	0.573	32898	24089	1391	0.424	42536	25423	730	0.436
	Directed	25895	25858	2363	0.666	34635	26670	1430	0.547	42502	25620	755	1.066
	Scratch	25099	253856	9815	0.902	32855	252054	4991	1.019	43130	256020	2660	2.247
MRNGB	Imbalance	1.4876				1.4955				1.4854			
	Undirected	69428	92806	10625	1.377	87687	94704	5342	1.017	112522	94328	2821	0.817
	Directed	67032	94397	10447	1.534	86836	90785	5376	1.174	112199	95074	3028	1.161
	Scratch	65395	996825	39444	2.039	85812	1016832	20365	1.799	110459	1016166	10507	3.294
MRNGC	Imbalance	1.4715				1.4920				1.4995			
	Undirected	169336	342078	46182	3.994	217165	358453	26919	2.435	278607	368216	11513	1.689
	Directed	169559	386720	44182	4.385	223031	382812	27300	2.685	279832	351109	11299	2.092
	Scratch	167617	3943621	166742	5.056	222368	3998118	83983	3.558	281195	3984525	42061	4.352
MRNGD	Imbalance					1.4470				1.5152			
	Undirected					345122	640172	40995	4.145	452386	743892	22439	2.747
	Directed					350334	700727	46231	4.511	445263	768296	22902	3.203
	Scratch					343354	7487361	156524	5.511	438704	7507463	83205	4.692

Table 2: The performance of the parallel adaptive repartitioning schemes on the four test graphs for 64, 128, and 256 processors on Cray T3D. All run-times are in seconds.

To better compare the three schemes we graphically depicted the results shown in Table 2 in the sequence of graphs shown in Figures 2– 5. In particular, Figure 2 compares the quality in terms of edge-cut produced by the two multilevel diffusion algorithms relative to partitioning from scratch. For each experiment, we computed the ratio of the edgecut produced by the diffusion algorithms to that of partitioning from scratch and plotted it using a bar chart. For example, the bars for MRNGB-128 corresponds to partitioning the adaptively refined graph MRNGB on 128 processors. Looking at this figure we can see that the edge-cuts of the partitionings produced by the two multilevel diffusion algorithms are in general within 5% of the edge-cut produced by partitioning the adaptive graphs from scratch. Furthermore, for some of these graphs (*e.g.*, MRNGA-256 and MRNGC-128) the multilevel diffusion algorithms produced partitionings whose edge-cut was slightly better than that of partitioning from scratch. These results show that our adaptive multilevel diffusion algorithms are able to produce partitionings whose quality is comparable to the quality obtained by the parallel multilevel partitioning algorithm. Comparing the two multilevel diffusion algorithms together, we see that in general they lead to partitionings that have comparable edge-cuts. Out of the eleven experiments, the undirected diffusion scheme leads to smaller edge-cuts for six of them, whereas the directed diffusion does better in five experiments.

The performance in terms on the number of vertices that need to be moved (TotalV and MaxV) is shown in Figures 3 and 4, respectively. Looking at these two figures we can clearly see the advantage of the multilevel diffusion algorithms over partitioning from scratch in significantly reducing the number of vertices that needs to be moved in order to achieve load balance. The multilevel diffusion schemes move in general less than 10% percent of the total number of vertices that are required to be moved by partitioning from scratch (Figure 3), and the maximum number of vertices that is sent and received by any processor is less than 30% of that required by partitioning from scratch (Figure 4). This dramatic reduction in data movement coupled with the comparable edge-cut quality makes the multilevel diffusion algorithms a clear choice for repartitioning adaptively refined graphs. As it was the case with the edge-cut, the relative performance of directed and undirected diffusion is quite similar, with both schemes performing within a couple percentage points of each other.

Finally, Figure 5 shows the amount of time required by the two multilevel diffusion algorithms relative to partitioning from scratch. From this figure we can see that the multilevel diffusion algorithms are considerably faster than partitioning the graph from scratch. In particular, for MRNGD on 256 processors, undirected diffusion is about 40% faster than partitioning from scratch, requiring only 2.75 seconds (partitioning from scratch requires 4.69 seconds). This run-time difference is mostly due to the fact that the multilevel diffusion algorithms do not have to compute an initial k-way partitioning of the coarsest graph, as they inherit the original partitioning. Comparing the multilevel undirected

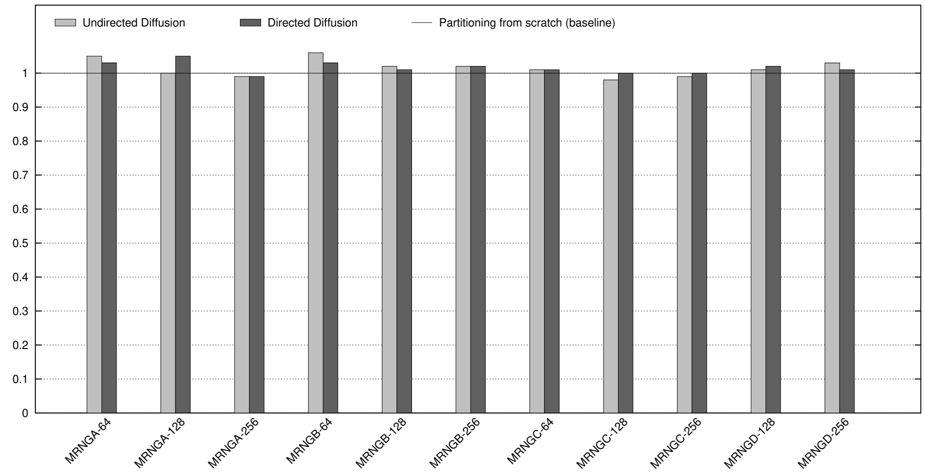

Figure 2: Quality in terms of edge-cuts of the partitionings produced by the adaptive multilevel directed and undirected diffusion algorithms relative to partitioning from scratch. For each graph, the ratio of the edge-cut of the adaptive multilevel diffusion algorithms to that of the multilevel k-way partitioning is plotted for 64-, 128-, and 256-way partitionings on 64, 128, and 256 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than partitioning from scratch.

diffusion with the directed diffusion algorithm we see that the former is faster, and the performance gap increases with the number of processors. This is because, as discussed in Section 3, the directed diffusion algorithm needs to diffuse the coarsest graph serially, whereas the undirected diffusion algorithm does not have this serial component. Looking at the run-times shown in Table 2 we can see that both multilevel diffusion algorithms scale very well with the number of processors, and they are able to partition graphs with around eight million vertices in around three seconds.

5 Conclusion and Related Work

Walshaw, Cross, and Everett also implemented a parallel partitioner and directed diffusion repartitioner based on an optimization of the Hu and Blake diffusion solver [3, 12, 11]. Their algorithms has two distinct phases, called balancing and refinement phase. The first is a balancing phase in which the diffusion solution guides vertex migration in order to balance the graph. Unlike our algorithms, this diffusion is performed on the original graph (*i.e.*, it does not utilize a multilevel representation), and requires a significant amount of time especially for graphs that are grossly load imbalanced. The second is a local refinement phase in which a local view of the graph guides vertex migration in order to decrease the edge-cut upset by the balancing phase. For this refinement phase they use either a single-level refinement scheme (performed on the full graph), or they use a multilevel refinement scheme similar to that described in [5]. They report results on a set of small- to medium-size two-dimensional meshes with very moderate imbalance problems (around 10%). In the case of their single-level refinement scheme, they report results in which the TotalV is low and the edge-cut increases only slightly over partitioning from scratch. However, when they use the multilevel refinement scheme, they are able to decrease the edge-cut, at the cost of increasing the TotalV considerably.

In this paper we presented scalable and highly parallel formulations of multilevel diffusion algorithms. Our algorithms are able to quickly repartition graphs corresponding to adaptively refined meshes and obtain partitionings whose quality is comparable to those obtained by the high-quality parallel multilevel k-way partitioning algorithm of METIS [7], while dramatically reducing the amount of data movement required to realize this new partition. Unlike the algorithms of Walshaw *et al.*, our parallel multilevel diffusion algorithms combine both diffusion and refinement in a unified multilevel framework making it possible to quickly balance graphs that are highly imbalanced while producing edgecuts comparable to partitioning from scratch and substantially reducing the total vertex movement. Furthermore, our algorithms utilize the concepts of vertex weight, vertex size, and vertex cleanness to specifically minimize TotalV and MaxV in addition to the edge-cut. Our experiments has shown that graphs with around eight million vertices can be repartitioned in under three seconds on a 256-processor Cray T3D making it possible to perform large scale simulations that require frequent mesh refinements.

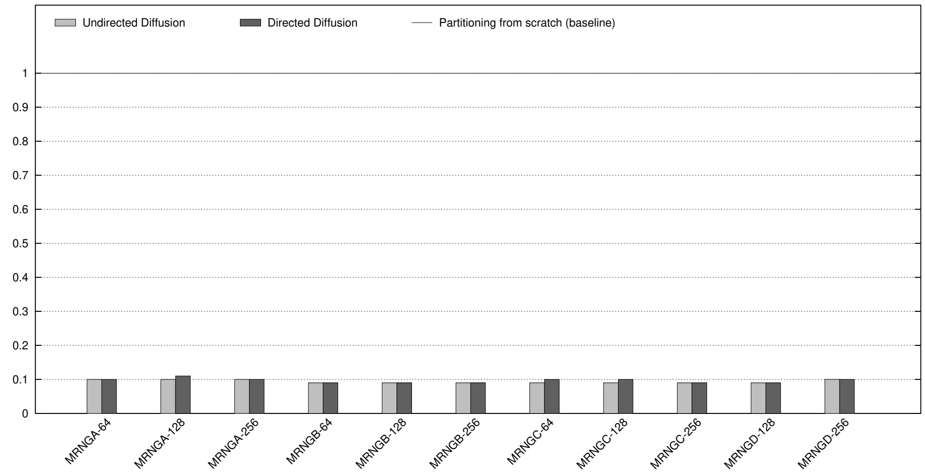

Figure 3: Quality in terms of the total number of vertices that need to be moved (TotalV) of the partitionings produced by the adaptive multilevel directed and undirected diffusion algorithms relative to partitioning from scratch. For each graph, the ratio of the TotalV of the adaptive multilevel diffusion algorithms to that of the multilevel k-way partitioning is plotted for 64-, 128-, and 256-way partitionings on 64, 128, and 256 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than partitioning from scratch.

References

[1] R. Biswas and R. C. Strawn. A new procedure for dynamic adaption of three-dimensional unstructured grids. *Applied Numerical Mathematics*, 13:437–452, 1994.

[2] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

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

[4] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

[5] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 95-064, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/˜karypis.

[6] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 96-036, Department of Computer Science, University of Minnesota, 1996. Also available on WWW at URL http://www.cs.umn.edu/˜karypis. A short version appears in Supercomputing 96.

[7] George Karypis and Vipin Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the eighth SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[8] Leonid Oliker and Rupak Biswas. Efficient load balancing and data remapping for adaptive grid calculations. Technical report, NASA Ames Research Center, Moffett Field, CA, 1997.

[9] Chao-Wei Ou and Sanjay Ranka. Parallel incremental graph partitioning using linear programming. Technical report, Syracuse University, Syracuse, NY, 1992.

[10] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. Technical Report TR 97-013, University of Minnesota, Department of Computer Science, 1997. http://www.cs.umn.edu/ karypis.

[11] C. Walshaw, M. Cross, and M. G. Everett. Dynamic load-balancing for parallel adaptive unstructured meshes. *Parallel Processing for Scientific Computing*, 1997.

[12] C. Walshaw, M. Cross, and M. G. Everett. Dynamic mesh partitioning: A unified optimisation and load-balancing algorithm. Technical Report 95/IM/06, Centre for Numerical Modelling and Process Analysis, University of Greenwich, London, UK, December 1995.

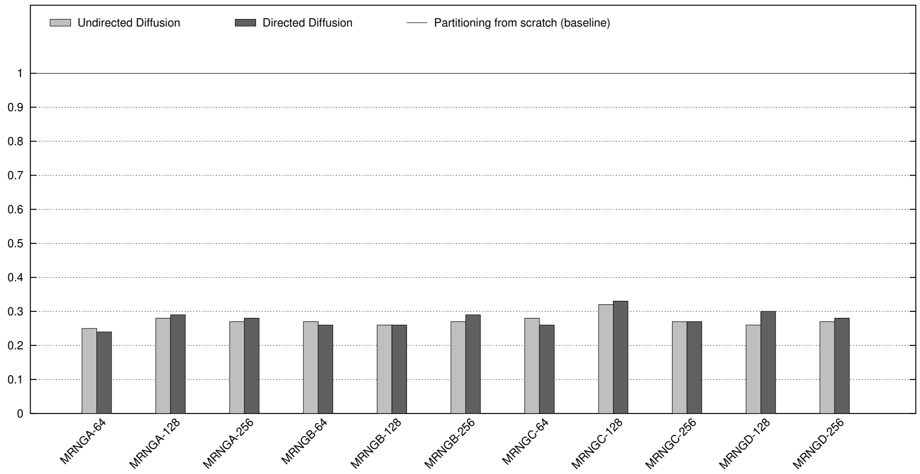

Figure 4: Quality in terms of the maximum number of vertices that need to be moved in and out of a partition (MaxV) of the partitionings produced by the adaptive multilevel directed and undirected diffusion algorithms relative to partitioning from scratch. For each graph, the ratio of the MaxV of the adaptive multilevel diffusion algorithms to that of the multilevel k-way partitioning is plotted for 64-, 128-, and 256-way partitionings on 64, 128, and 256 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than partitioning from scratch.

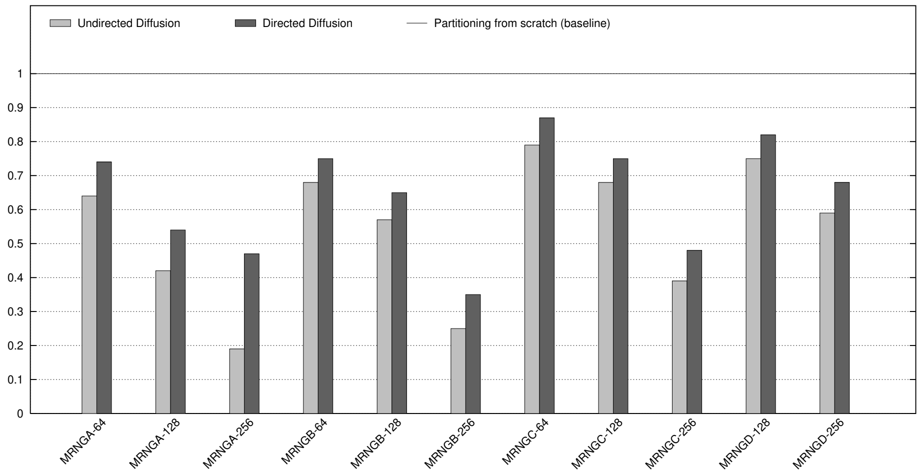

Figure 5: Run time of adaptive multilevel directed and undirected diffusion algorithms relative to partitioning from scratch. For each graph, the ratio of the time required by adaptive multilevel diffusion algorithms to that of the multilevel k-way partitioning is plotted for 64-, 128-, and 256-way partitionings on 64, 128, and 256 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms run faster than partitioning from scratch.