A Performance Study of Diffusive vs. Remapped Load-Balancing Schemes *

K. Schloegel G. Karypis V.Kumar
Dept. of Comp. Sci. & Eng.
Army HPC Research Center
University of Minnesota
Minneapolis, MN 55455

R. Biswas L. Oliker
NASA Ames Research Center
Mail Stop T27A-1
Moffett Field, CA 94035

Abstract

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. Eventually, as the graph evolves, it becomes necessary to correct the partition in accordance with the structural changes in the computation. Two different types of schemes to accomplish this task have been developed recently. In one scheme, the graph is partitioned from scratch and then the resulting partition is remapped intelligently to the original partition. The second type of scheme uses a multilevel diffusion repartitioner. In this paper, we conduct a comparison study on repartitioning via intelligent remapping versus repartitioning by diffusion. We show that multilevel diffusion algorithms generally produce significantly lower data migration overhead for adaptive graphs in which low magnitude localized or nonlocalized imbalances occur and for graphs in which high magnitude imbalances occurs globally throughout the domains than partitioning from scratch and remapping the resulting partition. We show that for the class of problems in which high magnitude imbalances occur in localized areas of the graph, partitioning from scratch and remapping the resulting partition will result in higher quality partitions than diffusion schemes, while obtaining data migration overheads which are comparable to those obtained by diffusive schemes. Finally, we show that the run times of the various schemes are all similar.

*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

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 can be represented by the weight of its associated vertex. Similarly, the amount of runtime interaction required between two grid points can be 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 objective that the total number of edges cut by the partitions (hereafter referred to as *edge-cut*) is minimized, and subject to the constraint that each partition has an equal amount of total vertex weight. 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, while requiring that each partition has the same amount of vertex weight ensures load balance. This problem has been well defined and discussed in previous work [3, 5].

For a large class of irregular grid applications, the structure of the problem changes in an incremental fashion from one phase of the computation to another. For example, in adaptive meshes [2], areas of the original graph are selectively coarsened or refined in order to accurately model the dynamic nature of the problem. The adapted meshes can be represented by appropriately modifying the weights of the vertices and edges of the original graph. 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 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 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.

The repartitioning algorithm also needs to satisfy the following objectives.

1. **It minimizes edge-cut.** The redistributed graph should have a small edge-cut to minimize communication overhead in the application.

2. **It minimizes vertex migration time**. Once the mesh is repartitioned, and before the 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 therefore dominate the overall run time, especially if the mesh is adapted frequently.

Partitioning the graph from scratch and then intelligently remapping the resulting partition is one way to meet the above criteria. Various algorithms which do so are described in [1]. Here, the imbalanced graph is partitioned from scratch using one of the multilevel graph partitioning algorithms described in [4, 5]. The resulting partition is then intelligently mapped to the processors in order to reduce the amount of vertex migration required. In [6], a simple greedy remapping algorithm is described and shown to obtain near-optimal results on application graphs.

Another way to meet the above criteria is through diffusive repartitioning. Multilevel diffusion schemes have been developed that incrementally construct a new partition of the graph [7, 8]. These schemes generally have three phases: a graph coarsening phase, a diffusion phase, and a multilevel refinement phase.

In the graph coarsening phase, these algorithms attempt to coarsen the input graph recursively by collapsing matched vertices into a single vertex on the next coarser graph. Vertices are matched together by computing a maximal set of independent edges. In the diffusion phase, the coarsest (and hence smallest) graph is balanced by incrementally modifying the input partition. By beginning this process on the coarsest graph, these algorithms are able to move large chunks of well-connected vertices in a single step. Thus, the bulk of the work required to balance the graph is done quickly. Eventually, due to the coarseness of the graph, an incremental diffusion process may not be able to improve the load balance. At this point, either refinement is begun on the current graph or the partition is projected to the next finer graph and another round of diffusion begins. In these algorithms, diffusion may be directed globally or locally. Once the partition is balanced, multilevel refinement is performed in order to improve the quality of the partition (which may have been upset during diffusion.) In multilevel refinement, border vertices from the coarsest graph are selected in some order. Each vertex is examined to determine if migrating it to an adjacent domain[1] will accomplish the primary objective (usually to reduce the edge-cut) while maintaining the balance constraint. If this is so, the switch is made. Once a local minima in this search space is reached for the current graph, the partition is projected to the next finer level graph and this refinement process begins again.

The effectiveness of repartitioning algorithms quite often is determined by how successful they are in load balancing the computations while minimizing the edge-cut as well as the cost associated in redistributing the load in order to realize the new partitioning. Two metrics that are widely used [6, 7] for measuring this redistribution cost are TotalV which measures the total volume of data moved among all processors, and MaxV which measures maximum flow of data to or from any single processor. Specifically, TotalV is defined as the sum of the sizes of the vertices which change domains as the result of repartitioning, while MaxV is defined as the maximum of the sums of the sizes of those vertices which migrate into and out of any one domain as a result of repartitioning.

2 Experimental Results

We tested our parallel repartitioning algorithms on a Cray T3E-1200 with 128 processors. Each processor on the T3E-1200 is a 600 MHz Dec Alpha (EV5). The processors are interconnected via a three dimensional torus network that has a peak unidirectional bandwidth of 450 bytes per second, and a small latency. We

[1] In this paper, we refer to a k-way partition as being composed of k disjointed *domains*.

used Cray's MPI library for communication. Cray's MPI achieves a peak bandwidth of 200 MBytes and an effective startup time of 30 microseconds.

We evaluated the performance of the parallel repartitioning algorithms described above on synthetically generated adaptive meshes, derived from three medium to large size 3D finite element meshes, on 32, 64, and 128 processors. The characteristics of the corresponding dual graphs of these meshes are described in Table 1. Our synthetically generated problems simulate the process of mesh adaptation by changing the weight of the vertices and the edges of the graph. The weight of some of the vertices was changed from one (prior to adaptation) to a number greater than one (indicating the degree of adaptation). The weight of edges between *adapted* vertices was also changed to reflect the higher degree of connectivity in the adapted graph. For every edge with incident vertices v_i and v_j, its weight was set to $(\min(w_i, w_j))^{2/3}$ (where w_i is the weight of vertex v_i). Note that this edge-weighting model tries to capture the face connectivity of 3D finite element meshes. For each of these synthetically generated graphs, parallel repartitioning algorithms are used to compute a new partitioning such that the sum of the weight of the vertices assigned to each processor is the same.

Graph Name	Num. of Vertices	Num. of Edges
MRNGB	1017253	2015714
MRNGC	4039160	8016848
MRNGD	7833224	15291280

Table 1: The various graphs used in the experiments.

For each of the three meshes and for each processor configuration, we generated two types of adaptive meshes, TypeA and TypeB. Meshes of TypeA attempt to simulate the situation in which the mesh is adapted at various regions throughout the mesh, whereas meshes of TypeB attempt to simulate the situation in which the adaptation occurs in a small region of the mesh. For both TypeA and TypeB meshes on p processors, we first computed a p-way partitioning of the graph (using the parallel k-way graph partitioner implemented in ParMETIS), and then redistributed the graph according to this partitioning. This became the initial partitioning that we used to adjust the weight of the vertices to emulate the effect of adaptation.

For meshes of TypeA, the weights of the vertices were changed as follows. Each processor generated a random number r between zero and 100, then it randomly selected r% of its vertices, and set their weight to α. The weight of the remaining vertices was set to one. For meshes of TypeB, the weights of the vertices were changed as follows. Each processor generated a random number r between zero and $p-1$, and then for the processors in which r was less than $0.05p$, the weight of all the vertices in these processors was set to α. The weight of the remaining vertices was set to one. Note that the expected number of processors that will increase the weight of its vertices is 1.6, 3.2, and 6.4 for 32, 64, and 128 processors, respectively. For both TypeA and TypeB problems, we let α take the values of 2, 5, 10, 20, 30, and 40. These synthetically generated graphs are then used as the input to the parallel repartitioning algorithms described in [6, 7] and implemented in ParMETIS. In all of our experiments, a graph that has less than 5% load imbalance is assumed to be well balanced. Also, in all of our experiments, the cost to migrate a vertex between processors is assumed to be proportional to its weight.

The parallel implementations which we used for the experiments are all from ParMETIS. That is, ParMETIS implements the undirected diffusion algorithm described in [7] in the subroutine PARUAMETIS, the directed diffusion algorithm in PARDAMETIS, and the intelligent greedy remapping algorithm in PARPAMETIS.

In order to compare the three schemes we graphically depict the results in the sequence of graphs shown in Figs. 1–6. In particular, Fig. 1 compares the quality in terms of edge-cut and TotalV produced by the two multilevel diffusion schemes (PARUAMETIS and PARDAMETIS) relative to partitioning from scratch and then performing a greedy remapping (PARPAMETIS) for TypeA experiments in which α was set to 2, 5, and 10. For each experiment, we computed the ratio of the edge-cut and TotalV produced by the diffusion algorithms to that of PARPAMETIS and plotted it using a bar chart.

Performance on Slightly to Moderately Adapted Graphs The experiments shown in Fig. 1 simulate a low to moderate degree of adaption taking place globally throughout the adapted graph. Fig. 1 shows that the edge-cuts obtained by the multilevel diffusion algorithms were, in general, 0% to 10% higher than those obtained by PARPAMETIS. Furthermore, the degradation in edge-cut for the diffusion schemes tended to increase with α. Thus, as the level of imbalance increased, PARPAMETIS did an increasingly better job than either PARUAMETIS or PARDAMETIS in minimizing the edge-cut. The TotalV produced by both of the diffusion schemes is considerably less than that obtained by PARPAMETIS. In general, diffusion resulted in TOTALV which is 15% to 30% that produced by the intelligent remapping scheme. As the level of imbalance increased, the difference in TOTALV results tended to decrease. Comparing PARUAMETIS with PARDAMETIS, the edge-cut and TOTALV results are generally very similar (within 5%). However, PARUAMETIS tended to obtain slightly lower TOTALV results than PARDAMETIS.

The experiments shown in Fig. 2 simulate a low degree of adaption taking place in localized areas of the adapted graph. Fig. 2 shows that the edge-cuts obtained by the multilevel diffusion algorithms were, in general, 0% to 20% higher than those obtained by PARPAMETIS. Again, the degradation in edge-cut for the diffusion schemes tended to increase with α. The TOTALV produced by both of the diffusion schemes is again considerably less than that obtained by PARPAMETIS. Here, diffusion resulted in TOTALV which is 10% to 70% of that produced by PARPAMETIS. Furthermore, as the level of imbalance increased, the difference in TOTALV results tended to decrease dramatically. This indicates that as the level of localized imbalance increases, PARPAMETIS's performance increasingly approaches that of the diffusion algorithm with respect to TOTALV. Comparing PARUAMETIS with PARDAMETIS, the edge-cut and TOTALV results are again similar (usually within 4-6%).

In summary, for this class of problems, the results indicate that multilevel diffusion can significantly reduce the amount of total vertex migration required to realize the new partition while maintaining quality in terms of edge-cut which is comparable to that of intelligent remapping algorithms.

Performance on Highly Adapted Graphs The experiments shown in Fig. 3 simulate a high degree of adaption taking place globally throughout the adapted graph. Fig. 3 shows that the edge-cuts obtained by the multilevel diffusion algorithms were, in general, 20% to 30% greater than those obtained by PARPAMETIS. Again, the degradation in edge-cut for the diffusion schemes tended to increase with α. The TOTALV produced by both of the diffusion schemes is considerably less than that obtained by PARPAMETIS. Diffusion resulted in TOTALV which is 30% to 40% that produced by the intelligent remapping scheme. Here however, the relative TOTALV results remain constant (independent of α) between the diffusion algorithms and PARPAMETIS. Comparing PARUAMETIS with PARDAMETIS, we again see only a slight difference among the schemes.

The experiments shown in Fig. 4 simulate a high degree of adaption taking place in localized areas of the adapted graph. Fig. 4 shows that the edge-cuts obtained by the multilevel diffusion algorithms were, in general, 20% to 30% greater than those obtained by PARPAMETIS. Notice that for three of the experiments, the edge-cuts obtained by the diffusion algorithms were 58% to 98% higher than those obtained by PARPAMETIS. This is because in these experiments, the diffusion algorithms failed to balance the graphs to within 5%. Because of this, multilevel refinement never began, and so the edge-cut quality suffered. Fig. 4 also shows that the TOTALV produced by PARPAMETIS is generally similar or lower than that produced by the diffusion algorithms. This is especially true of the highly imbalanced problems. Note, these results indicate that for this class of problems, PARPAMETIS is more effective in reducing both the edge-cut and the TOTALV than multilevel diffusion algorithms.

MAXV **Results** Fig. 5 gives the relative MAXV performance of the three schemes for TYPEA experiments. Results show that, in general, the diffusion schemes produced much lower MAXV results than PARPAMETIS. Specifically, the MAXV results obtained by the multilevel diffusion algorithms were about 20% to 70% of those obtained by PARPAMETIS.

Fig. 6 gives the relative MAXV performance of the three schemes for TYPEB experiments and shows that the diffusion schemes again produced lower MAXV results, in general, than PARPAMETIS. However, here there was a convergence point as α increased in which all three algorithms produced generally similar MAXV results. This is because the MAXV of these problems is dominated by the amount of vertex weight that needs to move out of the few overbalanced domains. This migration is required in order to balance the graph, and so becomes the lower bound for MAXV.

Run Time Results Table 2 gives some sample data points with respect to the execution times of the various algorithms. All of these results are from the largest graph, MRNGD. Table 2 shows that the run times of all of the schemes are quite fast. Also, the run times are very similar among the schemes. Notice, however, that the diffusion algorithms (labeled *UA* for PARUAMETIS and *DA* for PARDAMETIS) tend to take longer for the TypeB problems than they do for TypeA problems, while the PARPAMETIS (labeled *PA*) times are more consistent. This is because localized imbalances cause diffusion to propagate further on average than global imbalances. Thus, diffusion algorithms will require more iterations of diffusion in order to become balanced for TypeB problems than for TypeA problems. Hence, the run times are higher. Computing a new partition from scratch avoids this problems. Thus, the PARPAMETIS times are not affected by the localized imbalances.

Problem	Partition	α	UA	DA	PA
TypeA	32-way	5	5.469	5.529	5.134
TypeA	32-way	30	5.555	5.551	5.198
TypeA	64-way	5	2.834	2.859	2.725
TypeA	64-way	30	2.816	2.870	2.799
TypeA	128-way	5	1.546	1.602	1.791
TypeA	128-way	30	1.581	1.661	1.846
TypeB	32-way	5	5.568	5.488	5.079
TypeB	32-way	30	5.470	5.488	5.047
TypeB	64-way	5	2.860	2.880	2.743
TypeB	64-way	30	2.930	3.029	2.758
TypeB	128-way	5	1.563	1.641	1.764
TypeB	128-way	30	1.895	2.059	1.817

Table 2: The run times of selected experiments.

3 Conclusion and Related Work

In this paper, we have shown that multilevel diffusion algorithms are generally able to produce significantly lower TotalV and MaxV results for adaptive graphs in which either localized or non-localized imbalances are low in magnitude as well as for graphs in which high magnitude imbalances occur globally throughout the grid. This is because the optimal solution for these problems is relatively near to the initial partition in the search space. Hence, diffusion, which attempts to minimize the difference between the original partition and the output partition, is a good strategy here. For these classes of problems, diffusive repartitioning should be applied a number of times, in order to keep data migration overhead low, followed by a single iteration of partitioning from scratch and remapping the resulting partition. This will allow edge-cut results to remain low, while minimizing the data migration overhead.

For the class of problems in which a large amount of imbalance occurs in localized areas of the graph, partitioning from scratch and remapping the resulting partition will result in very low edge-cuts and TotalV and MaxV results which are similar to those obtained by multilevel diffusive schemes. This is because the optimal solution for these problems is substantially removed from the initial partition in the search space. Hence, partitioning from scratch and intelligent remapping should be followed in favor of multilevel diffusion for such cases.

References

[1] R. Biswas and L. Oliker. Experiments with repartitioning and load balancing adaptive meshes. Technical Report NAS-97-021, NASA Ames Research Center, Moffett Field, CA, October 1997.

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

[3] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. *Proceedings Supercomputing '95*, 1995.

[4] 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.

[5] G. Karypis and V. Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the 8th SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[6] L. Oliker and R. Biswas. Plum: Parallel load balancing for adaptive unstructured meshes. Technical Report NAS-97-020, NASA Ames Research Center, Moffett Field, CA, 1997.

[7] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997.

[8] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47(2):102–108, 1997.

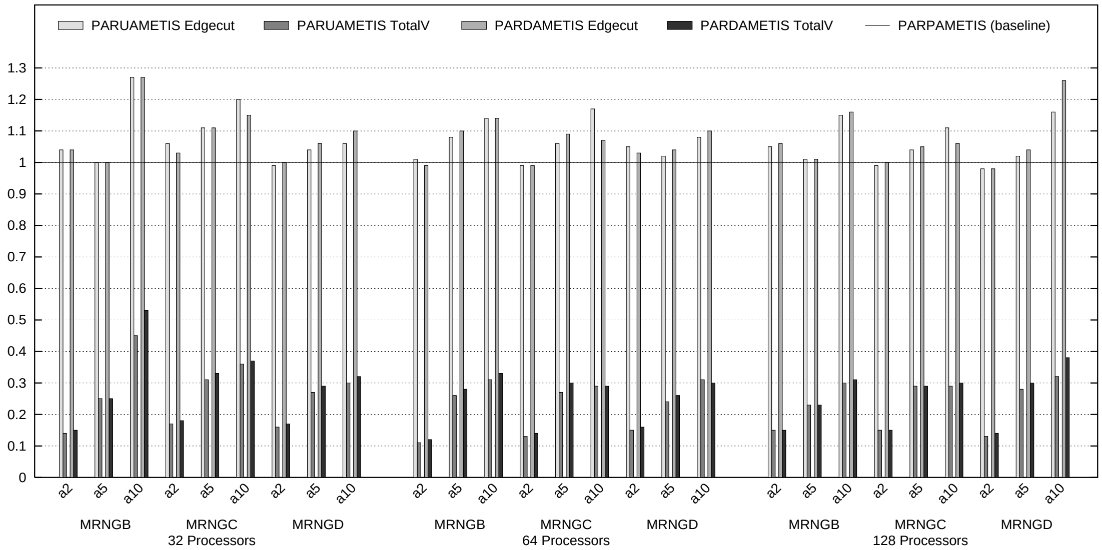

Figure 1: Quality in terms of edge-cut and TOTALV of the partitions produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for low imbalance TYPEA problems. For each graph, the ratio of the edge-cut and TOTALV of the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.

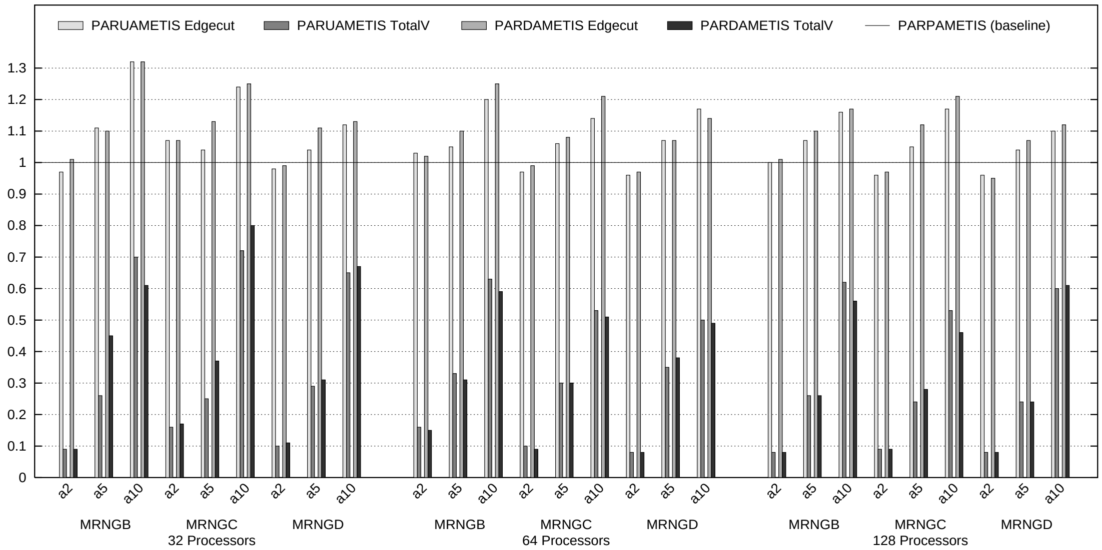

Figure 2: Quality in terms of edge-cut and TOTALV of the partitionings produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for low imbalance TYPEB problems. For each graph, the ratio of the edge-cut and TOTALV of the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.

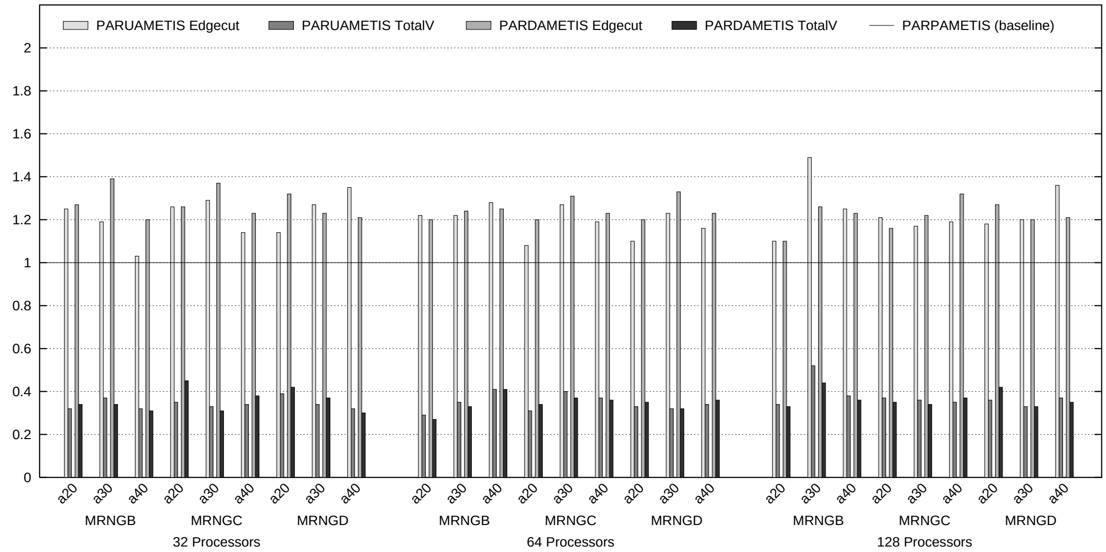

Figure 3: Quality in terms of edge-cut and TotalV of the partitionings produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for high imbalance TypeA problems. For each graph, the ratio of the edge-cut and TotalV of the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.

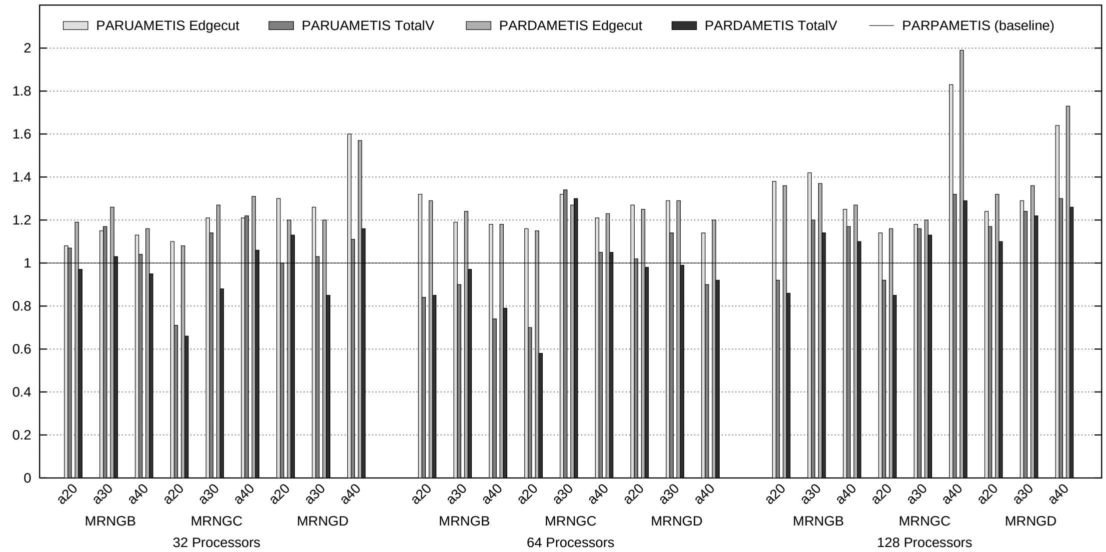

Figure 4: Quality in terms of edge-cut and TotalV of the partitionings produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for high imbalance TypeB problems. For each graph, the ratio of the edge-cut and TotalV of the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.

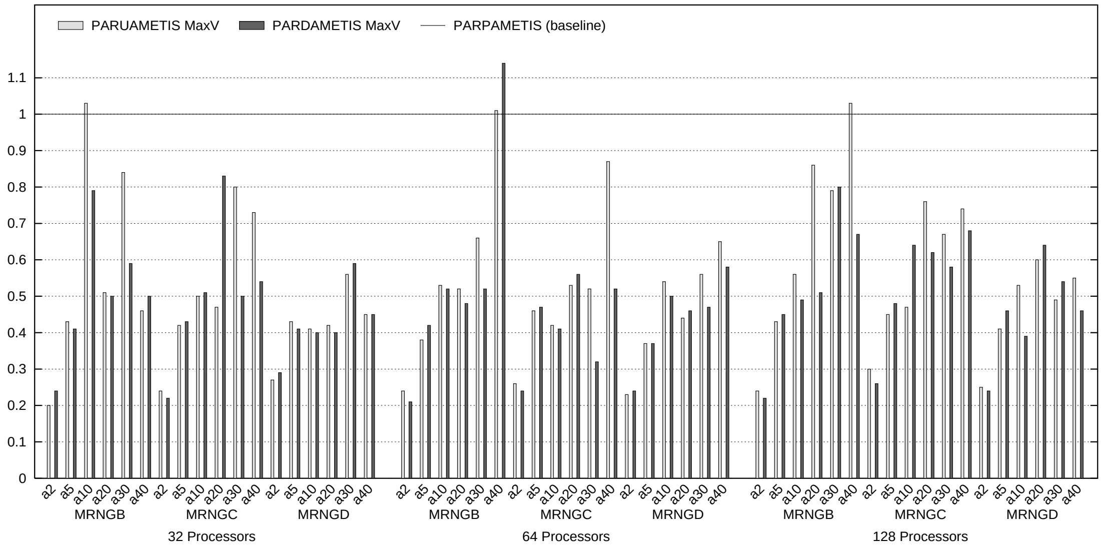

Figure 5: Quality in terms of MaxV of the partitionings produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for TypeA problems. For each graph, the ratio of the MaxV produced by the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.

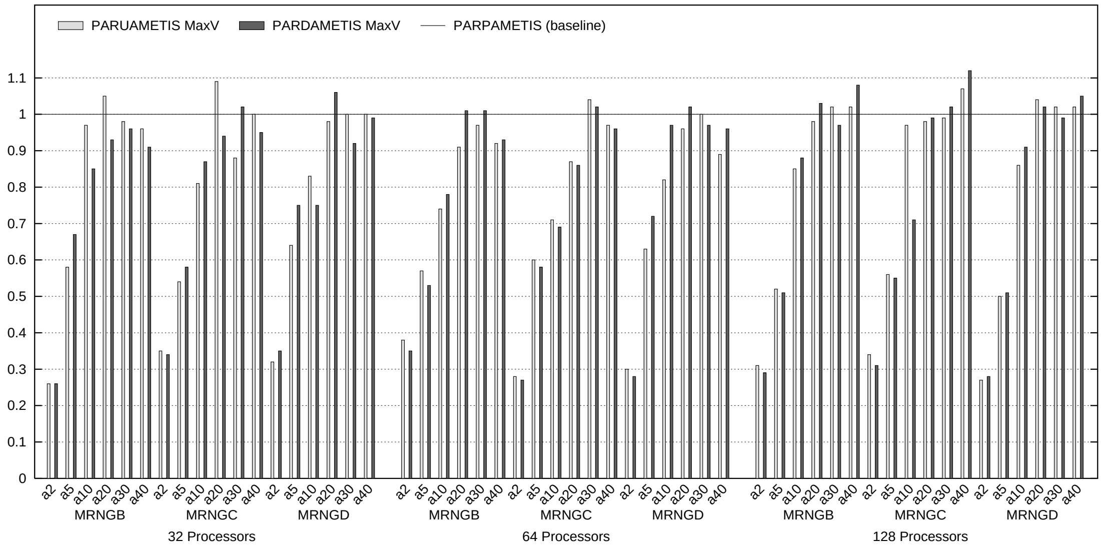

Figure 6: Quality in terms of MaxV of the partitionings produced by the multilevel directed and undirected diffusion algorithms relative to partitioning from scratch followed by intelligent remapping for TypeB problems. For each graph, the ratio of the MaxV produced by the multilevel diffusion algorithms to that of multilevel k-way partitioning and subsequent greedy remapping is plotted for 32-, 64-, and 128-way partitions on 32, 64, and 128 processors, respectively. Bars under the baseline indicate that the multilevel diffusion algorithms perform better than the remapping algorithm.