Dynamic Repartitioning of Adaptively Refined Meshes

Kirk Schloegel
Dept. of Computer Science and Engineering
University of Minnesota
Army HPC Research Center
Minneapolis, Minnesota
kirk@cs.umn.edu
http://www.cs.umn.edu/~kirk

George Karypis
Dept. of Computer Science and Engineering
University of Minnesota
Army HPC Research Center
Minneapolis, Minnesota
karypis@cs.umn.edu
http://www.cs.umn.edu/~karypis

Vipin Kumar
Dept. of Computer Science and Engineering
University of Minnesota
Army HPC Research Center
Minneapolis, Minnesota
kumar@cs.umn.edu
http://www.cs.umn.edu/~kumar

Abstract:

One ingredient which is viewed as vital to the successful conduct of many large-scale numerical simulations is the ability to dynamically repartition the underlying adaptive finite element mesh among the processors so that the computations are balanced and interprocessor communication is minimized. This requires that a sequence of partitions of the computational mesh be computed during the course of the computation in which the amount of data migration necessary to realize subsequent partitions is minimized, while all of the domains of a given partition contain a roughly equal amount of computational weight. Recently, parallel multilevel graph repartitioning techniques have been developed that can quickly compute high-quality repartitions for adaptive and dynamic meshes while minimizing the amount of data which needs to be migrated between processors. These algorithms can be categorized as either schemes which compute a new partition from scratch and then intelligently remap this partition to the original partition (hereafter referred to as scratch-remap schemes), or multilevel diffusion schemes. Scratch-remap schemes work quite well for graphs which are highly imbalanced in localized areas. On slightly to moderately imbalanced graphs and those in which imbalance occurs globally throughout the graph, however, they result in excessive vertex migration compared to multilevel diffusion algorithms. On the other hand, diffusion-based schemes work well for slightly imbalanced graphs and for those in which imbalance occurs globally throughout the graph. However, these schemes perform poorly on graphs that are highly imbalanced in localized areas, as the propagation of diffusion over long distances results in excessive edge-cut and vertex migration results. In this paper, we present two new schemes for adaptive repartitioning: *Locally-Matched Multilevel Scratch-Remap* (or *LMSR*) and *Wavefront Diffusion*. The LMSR scheme performs purely local coarsening and partition remapping in a multilevel context. In Wavefront Diffusion, the flow of vertices move in a wavefront from overbalanced to underbalanced domains. We present experimental evaluations of our LMSR and Wavefront Diffusion algorithms on synthetically generated adaptive meshes as well as on some application meshes. We show that our LMSR algorithm decreases the amount of vertex migration required to balance the graph and produces repartitionings of similar quality compared to state-of-the-art scratch-remap schemes. Furthermore, we show that our LMSR algorithm is more scalable in terms of execution time compared to state-of-the-art scratch-remap schemes. We show that our Wavefront Diffusion algorithm obtains significantly lower vertex migration requirements, while maintaining similar edge-cut results compared to state-of-the-art multilevel diffusion algorithms, especially for highly imbalanced graphs. Furthermore, we compare Wavefront Diffusion with LMSR and show that the former will result in lower vertex migration requirements and the later will result in higher quality edge-cut results. These results hold true regardless of the distance which diffusion is required to propagate in order to balance the graph. Finally, we discuss the run times of our schemes which are both capable of repartitioning an eight million node graph in under three seconds on a 128-processor Cray T3E.

Keywords:
Wavefront Diffusion, Multilevel Graph Repartitioning, Scratch-Remap, Multilevel Diffusion

1 Introduction

Graph partitioning is an important problem which has applications in many areas, including scientific computing. In irregular mesh applications, the computation associated with the mesh can be represented by a graph that has weights associated with its vertices and edges. The weight on the vertices of the graph represents the amount of computation associated with a mesh node, and the weight of the edges represents the amount of interaction between the computations associated with the incident vertices. Efficient parallel execution of these irregular mesh applications requires the partitioning of the associated graph into a number of parts equal to the number of processors with the following two criteria. (i) Each partition has a roughly equal amount of total vertex weight. (ii) The total weight of the edges cut by the partitions is minimized. This problem has been well defined and discussed in previous works [2,4,6].

A large class of scientific and engineering simulations and applications, such as fluid dynamics simulations, weather simulations, and mesh generation, utilizes dynamically changing meshes and graphs in order to model adaptive computations. In such applications, the structure of the graph can change from one phase of the computation to the next. Eventually, as the graph evolves, it needs to be repartitioned and data associated with the nodes has to be moved between processors before the computation can start again in order to ensure good load balance. The newly computed partition should again satisfy the two criteria enumerated above. Additionally, two other requirements are necessary. (iii) The amount of inter-processor data migration required to realize the new partition is minimized. (iv) The new partition is computed quickly. While it is easy to find repartitioning schemes that optimize a subset of these criteria (such as minimizing edge-cuts at the cost of very large interprocessor data migration), the real challenge is to minimize all four of these simultaneously. A repartitioning scheme which attempts to meet all four criteria is more general because any one of: (i) application computation time, (ii) application communication time, (iii) data migration time, or (iv) repartitioning execution time can dominate the overall run time of an application in the general case.

2 Background

Two strategies for computing a repartitioning of a graph are to either partition the graph again from scratch and then attempt to intelligently remap the newly computed partition to the original partition, or to use a diffusive process to migrate vertices from overweighted to underweighted domains. Recently, repartitioning schemes were presented based on both of these strategies [1,9,11,10,12,13,14]. These schemes, which we classify as either scratch-remap schemes [1,9,12] or diffusion-based schemes [11,10,13,14], are all based on the multilevel graph partitioning paradigm. The multilevel graph partitioning paradigm is able to compute very high quality partitions quickly by first constructing a sequence of coarsened graphs, partitioning the coarsest of these, and then refining the computed partition at each finer level graph, starting with the coarsest and working up. The benefit of this paradigm is that the initial partition can be computed on a very small graph and so is done quickly. Then, the quality of this partition is greatly improved by simple local heuristics. Since these are applied at each subsequent finer graph, the result is that the partition refinement algorithm sees multiple views (from global to very local) of the graph. This magnifies the power of simple heuristics and allows high-quality partitions to be computed quickly. Recently developed scratch-remap repartitioners have used this paradigm by partitioning the imbalanced graph from scratch with a multilevel graph partitioner, while recently developed diffusion-based repartitioners are modifications of the multilevel graph partitioning algorithm.

Results in [1,10,12] have shown that current diffusion-based schemes are appropriate when diffusion is not required to propagate far in order to balance the graph. This situation occurs on slightly imbalanced graphs and those in which imbalance occurs globally throughout the graph. Here diffusion can significantly reduce the amount of vertex migration required to balance the graph, while maintaining a high quality edge-cut.

Graphs that are highly imbalanced in localized areas require diffusion to propagate over longer distances. For these class of problems, current diffusion-based repartitioners produce excessive amounts of vertex migration. This is because diffusion repartitioning ties the domain labels of the balanced partition to those of the original partition. While this strategy is highly effective when diffusion is not required to propagate far, it is counter-productive for the opposite case. For extremely imbalanced graphs, vertex migration results can exceed those obtained by scratch-remap algorithms, which do not tie the domain labels of the balanced partition to those of the original partition. Also, as the amount of vertex migration increases so too does the resulting edge-cut (as well as the number of iterations of diffusion required to balance the graph, and so the run time of the algorithm), since vertex migration tends to perturb the quality of the original partition.

Current scratch-remap schemes, on the other hand, are less effective on graphs in which diffusion is not required to propagate far. This is because such schemes result in excessive amounts of vertex migration in comparison with diffusion schemes for these class of problems. At the same time, the quality of the edge-cut produced (and the run time required) is similar to diffusion-based schemes assuming that the original partition is of high quality. This is because diffusion-based schemes will only minimally perturb the edge-cut here.

Scratch-remap schemes are appropriate for graphs in which diffusion is required to propagate over longer distances. This is because scratch-remap schemes can consistently produce very high quality partitions quickly, regardless of the weight characteristics of the graph. Furthermore, since current diffusion-based schemes obtain excessively high vertex migration requirements for these class of problems, scratch-remap schemes can match or improve upon them here.

Improving diffusion-based schemes to the point in which they obtain lower vertex migration requirements than scratch-remap schemes for all problems would result in a clear tradeoff between the two schemes. That is, scratch-remap schemes would produce the highest quality repartitions for all problems, while diffusion schemes would produce the lowest amount of vertex migration required for all problems. Thus, the choice of repartitioners would be clear depending on whether the inter-processor communications or the cost to redistribute data amoung processors dominanted the run time.

3 Our Contributions

In this paper, we present two new schemes for adaptive repartitioning: *Locally-Matched Multilevel Scratch-Remap* (or *LMSR*) and *Wavefront Diffusion*. The LMSR scheme performs purely local coarsening and partition remapping in a multilevel context. In Wavefront Diffusion, the flow of vertices move in a wavefront from overbalanced to underbalanced domains. We present experimental evaluations of our LMSR and Wavefront Diffusion algorithms on synthetically generated adaptive meshes as well as on some application meshes. We show that LMSR decreases the amount of vertex migration required to balance the graph, is more scalable in terms of execution time, and produces repartitionings of similar quality compared to current state-of-the-art scratch-remap schemes. We show that Wavefront Diffusion obtains significantly lower vertex migration requirements, while maintaining similar edge-cut results compared to current state-of-the-art multilevel diffusion algorithms, especially for highly imbalanced graphs. Furthermore, we compare Wavefront Diffusion with LMSR and show that in general the former results in lower vertex migration requirements and the later results in higher quality edge-cut results. These results hold true regardless of the distance diffusion is required to propagate in order to balance the graph. Finally, we discuss the run times of our new schemes which are both capable of repartitioning an eight million node graph in under three seconds on a 128-processor Cray T3E.

4 LMSR

Scratch-remap schemes [9] partition the imbalanced graph from scratch utilizing a state-of-the-art graph partitioner and then remap the newly computed partition to the imbalanced partition such that the amount of data migration required to realize the new partition is minimized. Note that fast and parallel multilevel graph partitioning algorithms exist which are able to consistently compute high-quality partitions for finite-element graphs [6,15]. Hence, the edge-cut and run time results obtained by scratch-remap schemes are extremely difficult to improve upon. Furthermore, these schemes produce domains which are balanced to within a small constant. However, even an intelligent remapping of the newly computed partition results in vertex migration requirements which are significantly higher than those of diffusion-based repartitioners for all but the most extremely imbalanced problems [12].

The vertex migration required by scratch-remap schemes can be reduced if we can increase the degree of overlap between many of the domains of the initial partition with domains of the newly computed partition. One way to maximize this overlap is to partition the graph from scratch using a purely local coarsening during the coarsening phase of multilevel graph partitioning. This ensures that each coarse vertex contains vertices from exactly one domain of the old partition. Hence, for any partitioning of this coarse graph, significant portions of the domains on the newly computed partition will overlap with domains from the old partition. Also, local coarsening is more scalable than global coarsening because it requires significantly less inter-processor communication.

Another technique is to perform partition remapping immediately after computing the initial partition and before multilevel refinement begins. This makes it possible to minimize the edge-cut and maximize the overlap between the old and new partitions at each level during the multilevel refinement phase using local refinement heuristics.

We refer to our new scheme that implements both of these enhancements as *LMSR*.

5 Wavefront Diffusion

Results in [10,12,13] show that multilevel diffusion repartitioning schemes perform well when the degree of graph imbalance in the original graph is low to moderate. However, they tend to break down for graphs that are highly imbalanced in localized areas [10,1]. The reason is that for such graphs, diffusion is required to propagate over long distances. As a result, many domains are simultaneously both recipients and donors of vertices during diffusion [3]. For these domains, standard diffusion algorithms interleave the outgoing flow of vertices with the incoming flow of vertices from neighboring domains. Such a domain is often forced to move out vertices before it has received all of the vertices it is supposed to receive from its neighbors. Hence, it will have only a limited choice for selecting good outgoing vertices with respect to minimizing the edge-cut and the required vertex migration.

To address this problem, we have developed a new diffusion algorithm. The spirit of the algorithm is to begin the diffusion of vertices from those domains which have no required flow of vertices into them. Then after these domains reach balance, the diffusion solution is recomputed and the next iteration is begun on the set of domains whose required flow of vertices into them was satisfied during the previous iteration, and so on, until all of the domains are balanced. Furthermore, vertices which were migrated in previous iterations and so are no longer in the same domains as they are assigned on the original partition (referred to as *dirty* vertices) are eligible to migrate at any time, since doing so in preference to unmigrated (or *clean*) vertices will help to minimize the data migration cost. This is because no matter how many times a vertex is migrated during the computation of the repartitioning, the actual data migration cost is paid only once at the end. Our experimental results (not included in this paper) have shown that as diffusion is required to propagate over greater distances, the percentage of previously migrated vertices which are again selected to migrate increases as well. This reuse of dirty vertices results in very low vertex migration requirements, even when diffusion is required to propagate over extreme distances. We refer to this scheme as *Wavefront Diffusion*, since the flow of vertices is in a wavefront from overbalanced domains to underbalanced domains.

Our actual serial implementation of Wavefront Diffusion simplifies the algorithm described above. Here, we utilize an array, *flow*, with one element per domain. $flow_i$ contains the sum of the vertex weight which domain i is required to send out to other domains minus the sum of the vertex weight which domain i is required to receive in from other domains. In each iteration, only the domain, i, with the maximum value for $flow_i$ is allowed to migrate clean vertices. All domains are allowed to migrate dirty vertices. After each iteration, the diffusion solution is recomputed. Thus, at any iteration, the majority of domains will only be able to migrate dirty vertices (if they possess them).

6 Parallel LMSR and Parallel Wavefront Diffusion Algorithms

Many efficient parallel formulations are available for partitioning graphs from scratch. The computation of the remapping phase can be performed serially on a single processor, as its run time is usually much smaller than the time to move the nodes to their destinations. Therefore, parallelizing scratch-remap schemes in this way is straightforward.

Parallel versions of multilevel diffusion algorithms have been described in [11,14]. Here, 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.

Parallel multilevel repartitioning 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 almost embarrassingly parallel.

After graph coarsening, the coarsest graph is assembled and broadcast to all of the processors. Next, depending on its processor number, each processor simultaneously performs some version of the serial Wavefront Diffusion algorithm as described in Section 5. Processor 0 performs (Sorted) Wavefront Diffusion with the following modifications. (i) Vertices are sorted with respect to their amount of edge weight which is cut by the current partition prior to each diffusion iteration. Thus, vertices which are highly connected to vertices in different domains are selected first for migration. This modification tends to decrease the perturbation to edge-cut. (ii) The three domains which have the highest values for $flow_i$ at any given time are allowed to migrate clean vertices. This modification tends to decrease the number of iterations required to balance the graph significantly. The remainder of the processors perform the serial Wavefront Diffusion algorithm as described in Section 5 with modification (ii) and a unique random number seed. Thus, each processor is likely to explore a different solution path. After at least one processor has balanced the graph to within 10%, then all of the computed partitions are compared and the partition which has the lowest value for (edge-cut balance) is selected. Note that multiple runs of the sorted version of Wavefront Diffusion will result in identical solutions. Therefore, it is not beneficial to have more than one processor performing this algorithm.

The parallel formulation of the multilevel refinement phase is described in [11] and modeled after coarse-grained parallel multilevel refinement algorithm [6]. 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 domains. During the second sub-phase, vertices are migrated from higher- to lower-numbered domains. In this way, unexpected edge-cut increases caused by the simultaneous migration of neighboring vertices is avoided. Furthermore, these schemes avoid any bias towards the lower- or higher-numbered domains by using a random partition ordering at each step. In each sub-phase, vertices are visited and selected for migration according to the *refinement phase vertex migration criteria* described in [11].

7 Experimental Results

In this section, we present experimental results for the parallel implementations of our LMSR and

Wavefront Diffusion algorithms. We evaluated the performance of the parallel repartitioning algorithms on synthetically generated adaptive meshes. These meshes were derived from two large size 3D finite element meshes of four and eight million nodes. We first computed a *p*-way partitioning of the graph, 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. Each processor generated a random number, *r*, between zero and *p*-1. Then for the processors in which *r* was less than 0.05*p*, the weight of all of the vertices in these processors was set to *alpha*. The weight of the remaining vertices was set to one. 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 *(pow ((min* (w_i, w_j)*)),* 2/3*)* (where w_i is the weight of vertex v_i). Since only 5% of the domains changed the weights of their vertices, the adapted graph is imbalanced in localized areas. Also, the value of *alpha* determines the level of imbalance in these localized areas.

Each figure contains six sets of four results. Each set contains results in which *alpha* was set to 5, 10, 20, and 30 for a given graph and number of processors. The level of *alpha* increases from left to right. The first set is for the four million node finite element graph on 32 processors. The second set is for the eight million node finite element graph on 32 processors. Together these are labeled *32 Processors*. For all of the experiments, a partition in which no domain contains more than 105% of the average domain weight is considered to be well-balanced. All of the repartitioning schemes were able to compute well-balanced partitions for every experiment.

Figure 1 shows the edge-cut, vertex migration cost, and run time results of the state-of-the-art scratch-remap scheme described in [9] and implemented in ParMeTiS [12] as PARPAMETIS. In this figure, the results from our LMSR algorithm are normalized against those obtained from PARPAMETIS. Hence, a bar below the 1.0 index line indicates that our LMSR algorithm obtained results lower than PARPAMETIS. Here, we can see that the edge-cut results obtained from the two schemes are generally similar. However, the LMSR algorithm resulted in run time results which decrease in relation to PARPAMETIS as the number of processors increases. This indicates that our LMSR algorithm scales better than PARPAMETIS. The reason is that PARPAMETIS does not utilize true local matching. It utilizes locally-preferred matching. That is, only local vertices are eligible for matching during the first iteration through the vertices. In the second iteration, all unmatched vertices are eligible. Since global matching is possible, additional interprocessor communication is required here which is not needed for purely local matching. This makes our LMSR algorithm more scalable than PARPAMETIS. Finally, the LMSR algorithm obtains vertex migration costs which are generally lower than those obtained by PARPAMETIS by up to 40%.

Figure 1

Figures 2 and 3 compare the vertex migration and edge-cut results obtained by the multilevel diffusion algorithm, PARUAMETIS, implemented in ParMeTiS [8], with our LMSR algorithm and our Wavefront Diffusion algorithm. In these figures, the edge-cut (and vertex migration) results from PARUAMETIS and Wavefront Diffusion are normalized against those obtained from our LMSR algorithm. Hence, a bar above the 1.0 index line indicates that the LMSR algorithm obtained edge-cut (or vertex migration) results lower than the indicated algorithm.

In Figure 2, the total amount of vertex migration required is compared. Here PARUAMETIS obtained generally better results than our LMSR scheme. However, these results tended to converge for higher levels of imbalance. Wavefront Diffusion, on the other hand, obtained consistently better results compared to the LMSR algorithm, and increasingly better results as the level of imbalance increased compared to PARUAMETIS. Such low migration requirements for our Wavefront Diffusion algorithm could reduce the run times of a class of adaptive applications by a factor of two to three compared with the same application repartitioned by the other schemes.

Figure 2

Figure 3 shows that our LMSR algorithm obtained edge-cut results which are better than both of the other schemes across the board. In most cases, the difference is within 20%. Both of the diffusion-based schemes obtained results which are generally within 10% of each other. This shows that our LMSR algorithm is able to compute very high-quality repartitions regardless of the level of imbalance of the graph, and that our Wavefront Diffusion algorithm obtains repartitionings of comparable quality with state-of-the-art multilevel diffusion repartitioning schemes.

Figure 3

Finally, it is important to note that the run times of all of the parallel implementations compared are *extremely* fast. None of the run times for any of the schemes were over two seconds for the four million node graph or over three seconds for the eight million node graph on 128 processors. All of the schemes obtained generally similar run times for a given experiment to within 30%.

8 Helicopter Blade Results

Experimental results given in Section 7 were for synthetically generated adaptive meshes. In this section, we present results from our schemes on an application domain. These experiments were performed with the serial versions of our algorithms. However, the parallel results for these graphs will be included in the full-length version of this paper [11]. Figure 4 shows the repartitioning results from a series of application meshes with a high degree of adaptation at each stage. These graphs are 3-

dimensional mesh models of a rotating helicopter blade. As the blade spins, the mesh must be adapted by refining it in the area where the rotor has entered and coarsening it in the area of the mesh where the rotor has passed through. These meshes were provided by the authors of [9].

Here, the first of a series of six graphs, G_1, G_1, ... G_6, was originally partitioned into 16 domains with the multilevel graph partitioner implemented in *METIS* [5,7]. The partition of graph G_1 acted as the input partition for graph G_2. Repartitioning the imbalanced graph, G_2, resulted in the experiment named *First* and the input partition for graph G_3. Similarly, the repartition of graph G_3 resulted in experiment *Second*, and so on, through experiment *Fifth*. The last set of results is marked *Sum*. This is the sum of the raw scores of all five experiments.

Figure 4 gives a comparison of the edge-cut (and vertex migration) results of the five experiments, (followed by the sum of these) for Wavefront Diffusion, LMSR, and the state-of-the-art scratch-remap algorithm described in [9] and referred to here as SR. The edge-cut (and vertex migration) results obtained by Wavefront Diffusion, and LMSR are normalized by those obtained by the SR algorithm. Hence, a bar below the index line indicates that the corresponding algorithm obtained results lower than those obtained by the SR algorithm.

Figure 4 shows that the two scratch-remap schemes obtained similar edge-cuts. However, our LMSR scheme obtained significantly lower vertex migration results compared to SR. Figure 4 also shows that Wavefront Diffusion results in significantly lower vertex migration requirements at the cost of somewhat lower quality edge-cut results compared with the scratch-remap schemes. Note that these results concur with those given in Section 7.

Figure 4

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

T. Bui and C. Jones.
A heuristic for reducing fill in sparse matrix factorization.
In *6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445-452, 1993.

3

R. Diekmann, A. Frommer, and B. Monien.
Nearest neighbor load balancing on graphs.
Technical report, University of Paderborn, Department of Mathematics and Computer Science, 1998.

4

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

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.

6

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.

7

G. Karypis and V. Kumar.
METIS 3.0: Unstructured graph partitioning and sparse matrix ordering system.
Technical Report 97-061, Department of Computer Science, University of Minnesota, 1997.

8

G. Karypis, K. Schloegel, and V. Kumar.
PARMETIS: Parallel graph partitioning and sparse matrix ordering library.
Technical report, University of Minnesota, Department of Computer Science and Engineering, 1997.

9

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.

10

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.

11
K. Schloegel, G. Karypis, and V. Kumar.
Dynamic Repartitioning of Adaptively Refined Meshes.
University of Minnesota, Department of Computer Science and Engineering, 1998.

12
K. Schloegel, G. Karypis, V. Kumar, R. Biswas, and L. Oliker.
A performance study of diffusive vs. remapped load-balancing schemes.
ISCA 11th Int'l Conference on Parallel and Distributed Computing Systems, September 1998.

13
C. Walshaw, M. Cross, and M. G. Everett.
Dynamic load-balancing for parallel adaptive unstructured meshes.
Parallel Processing for Scientific Computing, 1997.

14
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.

15
C. Walshaw, M. Cross, S. Johnson, and M. G. Everett.
Jostle: Partitioning of unstructured meshes for massively parallel machines.
Proc. Parallel CFD'94, Kyoto, 1994.

Author Biography

Kirk Schloegel Kirk Schloegel is currently working on his Ph.D. in Computer Science at the University of Minnesota. He received his M.Sc. at the University of Edinburgh, Edinburgh, Scotland. His research interests include parallel algorithm design and parallel computing.

George Karypis George Karypis received his Ph.D. in Computer Science at the University of Minnesota, and he is currently an assistant professor at the department of Computer Science & Engineering at the University of Minnesota. His research interests spans the areas of parallel algorithm design, applications of parallel processing in scientific computing and optimization, sparse matrix computations, parallel programming languages and libraries, and data mining. His recent work has been in the areas of parallel sparse direct solvers, serial and parallel graph partitioning algorithms, parallel matrix ordering algorithms, and scalable parallel preconditioners. His research has resulted in the development of software libraries for unstructured mesh partitioning (METIS and ParMETIS), and for parallel Cholesky factorization (PSPASES). He has author of over 20 research articles, and a coauthor of a widely used text book ``Introduction to Parallel Computing''.

Vipin Kumar Vipin Kumar received his Ph.D. in Computer Science at the University of Maryland, and he is currently a professor at the department of Computer Science & Engineering at the University of Minnesota. His current research interests include parallel computing, parallel algorithms for scientific computing problems, and data mining. His research has resulted in the development of highly efficient parallel algorithms and software for sparse matrix factorization (PSPASES), graph partitioning (METIS and ParMETIS) and dense hierarchical solvers. Kumar's research in performance analysis resulted in the development of the isoefficiency metric for analyzing the scalability of parallel algorithms. He is author of over 100 research articles, and a coauthor of a widely used text book ``Introduction to Parallel Computing''. Kumar has given over 50 invited talks at various conferences, workshops, national labs, and has served as chair/co-chair for many conferences/workshops in the area of parallel computing. Kumar serves on the editorial boards of IEEE Parallel and Distributed Technology, Parallel Computing, the Journal of Parallel and Distributed Computing, and served on the editorial board of IEEE Transactions of Data and Knowledge Engineering during 93-97. He is a senior member of IEEE, a member of SIAM, and ACM.