A Unified Algorithm for Load-balancing Adaptive Scientific Simulations *

Kirk Schloegel and George Karypis and Vipin Kumar
(kirk, karypis, kumar) @ cs.umn.edu
Army HPC Research Center
Department of Computer Science and Engineering
University of Minnesota,
Minneapolis, MN 55455

Technical Report: TR 00-033

May 29, 2000

Abstract

Adaptive scientific simulations require that periodic repartitioning occur dynamically throughout the course of the simulation. The computed repartitionings should minimize both the inter-processor communications incurred during the iterative mesh-based computation and the data redistribution costs required to balance the load. Recently developed schemes for computing repartitionings provide the user with only a limited control of the tradeoffs among these objectives. This paper describes a new Unified Repartitioning Algorithm that can gracefully tradeoff one objective for the other dependent upon a user-defined parameter describing the relative costs of these objectives. We show that the Unified Repartitioning Algorithm is able to minimize the precise overheads associated with repartitioning as well as or better than other repartitioning schemes for a variety of problems, regardless of the relative costs of performing inter-processor communication and data redistribution. Our experimental results show that the Unified Repartitioning Algorithm is extremely fast and scalable to very large problems.

Keywords: Unified Repartitioning Algorithm, Dynamic Graph Partitioning, Multilevel Diffusion, Scratch-remap, Adaptive Mesh Computations

1 Introduction

For large-scale scientific simulations, the computational requirements of techniques relying on globally refined meshes become very high, especially as the complexity and size of the problems increase. By locally refining and de-refining the mesh either to capture flow-field phenomena of interest [1] or to account for variations in errors [14], adaptive methods make standard computational methods more cost effective. The efficient execution of these adaptive scientific simulations on parallel computers requires a periodic repartitioning of the underlying computational mesh. These repartitionings should minimize both the inter-processor communications incurred in the iterative mesh-based computation and the data redistribution costs required

*This work was supported by DOE contract number LLNL B347881, by NSF grant CCR-9972519, by Army Research Office contracts DA/DAAG55-98-1-0441, by Army High Performance Computing Research Center cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Additional support was provided 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-users.cs.umn.edu/~karypis

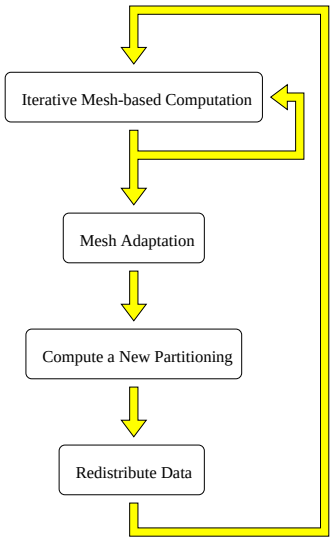

Figure 1: A diagram illustrating the execution of adaptive scientific simulations on high performance parallel computers.

to balance the load. Recently developed schemes for computing repartitionings provide the user with only a limited control of the tradeoffs among these objectives. This paper describes a new Unified Repartitioning Algorithm that can gracefully tradeoff one objective for the other dependent upon a user-defined parameter describing the relative costs of these objectives.

Figure 1 illustrates the steps involved in the execution of adaptive mesh-based simulations on parallel computers. Initially, the mesh is distributed on different processors. A number of iterations of the simulation are performed in parallel, after which mesh adaptation occurs. Here, each processor refines and de-refines its local regions of the mesh resulting in some amount of load imbalance. A new partitioning based on the adapted mesh is computed to re-balance the load, and then the mesh is redistributed among the processors, respectively. The simulation can then continue for another number of iterations until either more mesh adaptation is required or the simulation terminates.

If we consider each round of executing a number of iterations of the simulation, mesh adaptation, and load-balancing to be an *epoch*, then the run time of an epoch can be described by

$$(t_{comp} + f(|E_{cut}|))n + t_{repart} + g(|V_{move}|) \tag{1}$$

where n is the number of iterations executed, t_{comp} is the time to perform the computation for a single iteration of the simulation, $f(|E_{cut}|)$ is the time to perform the communications required for a single iteration of the simulation, and t_{repart} and $g(|V_{move}|)$ represent the times required to compute the new partitioning and to redistribute the data. Here, the inter-processor communication time is described as a function of the edge-cut of the partitioning and the data redistribution time is described as a function of the total amount of data that is required to be moved in order to realize the new partitioning.

Adaptive repartitioning affects all of terms in Equation 1. How well the new partitioning is balanced influences t_{comp}. The inter-processor communications time is dependent on the edge-cut of the new partitioning. The data redistribution time is dependent on the total amount of data that is required to be moved in order to realize the new partitioning. Recently developed adaptive repartitioning schemes [4, 5, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25] tend to be very fast, especially compared to the time required to perform even a single iteration of a typical scientific simulation. They also tend to balance the new partitioning to within a few percent of optimal. Hence, we can ignore both t_{comp}[1] and t_{repart}. However, depending on the nature of the application, both $f(|E_{cut}|)$ and $g(|V_{move}|)$ can seriously affect parallel run

[1] This is because, in the absence of load imbalance, t_{comp} will be primarily determined by the domain-specific computation and cannot be minimized further.

time and drive down the parallel efficiency. Therefore, it is critical for adaptive partitioning schemes to minimize both the edge-cut and the data redistribution when computing the new partitioning. Viewed in this way, adaptive graph partitioning is a multi-objective optimization problem.

Two approaches have primarily been taken when designing adaptive partitioners. The first approach is to attempt to focus on minimizing the edge-cut and to minimize the data redistribution only as a secondary objective [11, 12, 16, 17, 21, 22, 23, 24, 25]. A good example of such schemes are scratch-remap repartitioners [11, 17, 21]. These use some type of state-of-the-art graph partitioner to compute a new partitioning from scratch and then attempt to intelligently remap the subdomain labels to those of the original partitioning in order to minimize the data redistribution costs. Since a state-of-the-art graph partitioner is used to compute the partitioning, the resulting edge-cut tends to be extremely good. However, since there is no guarantee as to how similar the new partitioning will be to the original partitioning, data redistribution costs can be high, even after remapping [2, 17, 19]. The second approach is to focus on minimizing the data redistribution cost and to minimize the edge-cut as a secondary objective [4, 13, 14, 15, 20]. A good example of this approach are diffusion-based repartitioners [12, 16, 17, 23, 24, 25]. These schemes attempt to perturb the original partitioning just enough so as to balance it. This strategy usually leads to low data redistribution costs, especially when the partitioning is only slightly imbalanced. However, it can result in higher edge-cuts than scratch-remap methods because perturbing a partitioning in order to balance it also tends to adversely affect its quality.

These approaches to adaptive partitioning have two drawbacks. The first is that the two types of repartitioners allow the user to compute partitionings that focus on minimizing either the edge-cut or the data redistribution costs, but give the user only a limited ability to control the tradeoffs among these objectives. This control of the tradeoffs is sufficient if the number of iterations that a simulation performs between load-balancing phases (i. e. the value of n in Equation 1) is either very high or very low. However, when n is neither very high nor very low, neither type of scheme precisely minimizes the combined costs of $f(|E_{cut}|)n$ and $g(|V_{move}|)$. The second disadvantage exists for applications in which n is difficult to predict or those in which n can change dynamically throughout the course of the computation. As an example, one of the key issues concerning the elastic-plastic soil-structure interaction computations required for earthquake simulation is that the number of iterations between load-balancing phases is both unpredictable and dynamic. Here, zones in the 3D solid may become plastic and then unload (during increments of loading) so that the extent of the plastic zone is changing. The change can be both slow and rapid. Slow change usually occurs during initial loading phases, while the later deformation tends to localize in narrow zones rapidly and the rest of the solid unloads rapidly (becomes elastic again) [6].

Recently, Castanos and Savage [2] presented an adaptive repartitioning algorithm that directly minimizes the communication overheads of adaptive multigrid-based finite-element computations. During each load-balancing phase, their algorithm computes a repartitioning of the coarsest mesh of the hierarchy so as to optimize an objective that is similar to Equation 2 (given below). The coarse-mesh repartitioning is then used to partition the entire hierarchy of nested meshes. While this approach addresses the multi-objective nature of the adaptive repartitioning problem, Castanos and Savage's repartitioning algorithm is serial. Therefore, the scheme is suited only for problems in which: (i) the entire mesh refinement history is retained (eg., multigrid solvers), and (ii) a nested partitioning of the successively finer meshes is desired (i. e., for each coarse element, the entire corresponding hierarchy of finer elements belongs to the same processor).

Our Contributions In this paper, we present a parallel adaptive repartitioning scheme (called the *Unified Repartitioning Algorithm*) for the dynamic load-balancing of scientific simulations that attempts to solve the precise multi-objective optimization problem. By directly minimizing the combined costs of $f(|E_{cut}|)n$ and $g(|V_{move}|)$, our scheme is able to gracefully tradeoff one objective for the other as required by the specific application. Our experimental results show that when inter-processor communication costs are much greater in scale than data redistribution costs, our scheme obtains results that are similar to those obtained by an optimized scratch-remap repartitioner and better than those obtained by an optimized diffusion-based repartitioner. When these two costs are of similar scale, our scheme obtains results that are similar to the diffusive repartitioner and better than the scratch-remap repartitioner. When the cost to perform data

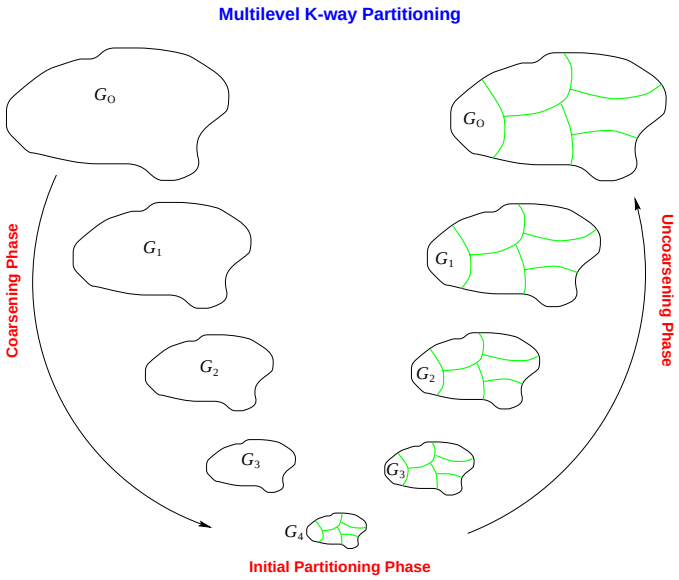

Figure 2: The three phases of multilevel k-way graph partitioning. During the coarsening phase, the size of the graph is successively decreased. During the initial partitioning phase, a k-way partitioning is computed, During the multilevel refinement (or uncoarsening) phase, the partitioning is successively refined as it is projected to the larger graphs. G_0 is the input graph, which is the finest graph. G_{i+1} is the next level coarser graph of G_i. G_4 is the coarsest graph.

redistribution is much greater than the cost to perform inter-processor communication, our scheme obtains better results than the diffusive scheme and much better results than the scratch-remap scheme. Finally, our experimental results show that our Unified Repartitioning Algorithm is fast and scalable to very large problems.

2 Unified Repartitioning Algorithm

We have developed a new parallel Unified Repartitioning Algorithm (URA) for dynamic load-balancing of scientific simulations that improves upon the best characteristics of scratch-remap and diffusion-based repartitioning schemes. A key parameter used in URA is the *Relative Cost Factor* (RCF). This parameter describes the relative times required for performing the inter-processor communications incurred during parallel processing and to perform the data redistribution associated with balancing the load. Using this parameter, it is possible to unify the two minimization objectives of the adaptive graph partitioning problem into the precise cost function

$$|E_{cut}| + \alpha |V_{move}| \tag{2}$$

where α is the Relative Cost Factor, $|E_{cut}|$ is the edge-cut of the partitioning, and $|V_{move}|$ is the total amount of data redistribution. The Unified Repartitioning Algorithm attempts to compute a repartitioning while directly minimizing this cost function.

The Unified Repartitioning Algorithm is based upon the multilevel paradigm that is illustrated in Figure 2. We next describe its three phases: graph coarsening, initial partitioning, and uncoarsening/refinement. In the graph coarsening phase, coarsening is performed using a purely local variant of heavy-edge matching [7, 16, 17, 25]. That is, vertices may be matched together only if they are in the same subdomain on the original partitioning. This matching scheme has been shown to be very effective at helping to minimize both the edge-cut and data redistribution costs and is also inheritly more scalable than global matching schemes [16, 17, 25].

Selecting an initial partitioning scheme for an adaptive partitioner is complicated for a number of reasons.

Graph	Num of Verts	Num of Edges	Description
auto	448,695	3,314,611	3D mesh of GM Saturn
*mdual*2	988,605	1,947,069	dual of a 3D mesh
*mrng*3	4,039,160	8,016,848	dual of a 3D mesh

Table 1: Characteristics of the graphs used in some of the experiments.

Experimental results [19] have shown that for some types of repartitioning problem instances, scratch-remap repartitioners tend to obtain better results compared to diffusive repartitioners, while for other types of problem instances, diffusive repartitioners tend to do better than scratch-remap repartitioners. Furthermore, the effectiveness of each type of repartitioning scheme is highly dependent on the value of the Relative Cost Factor. For these reasons, in the initial partitioning phase of URA, repartitioning is performed on the coarsest graph twice by alternative methods. Optimized variants of scratch-remap and global diffusion [17] are both used to compute new partitionings. The cost functions are then computed for each of these and the one with the lowest cost is selected. This technique tends to give a very good starting point from which to start multilevel refinement, regardless of the type of repartitioning problem or the value of the Relative Cost Factor. Note that the fact that URA computes two initial partitionings does not impact the scalability of the algorithm as long as the size of the coarsest graph is suitably small [8].

Most current adaptive graph partitioning algorithms perform partition refinement in order to minimize the edge-cut of the new partitioning. Some of these schemes [16, 17] also attempt to minimize the data redistribution cost as a tie breaking scheme. (That is, this objective is considered when the gain to the edge-cut that will result from moving a vertex is the same for two or more subdomains.) However, even with such a tie-breaking scheme, these do not directly minimize the precise cost function described in Equation 2. A refinement algorithm that does so, especially one that is applied in the multilevel context, can potentially minimize the cost function much better than current refinement schemes. In the uncoarsening phase of URA, the cost function from Equation 2 is directly minimized during multilevel refinement. Except for this important modification, the refinement algorithm used in URA is similar to the parallel adaptive refinement algorithm described in [17]. Note that since each time a vertex is moved, the new cost function can be computed in constant time using information local to the processor, these two algorithms have the same asymptotic run times.

3 Experimental Results

In this section, we present experimental results comparing the cost function and run time results of the Unified Repartitioning Algorithm with optimized versions of scratch-remap (LMSR) [17] and multilevel diffusion repartitioners (Wavefront Diffusion) [17].

Experimental Setup The experiments presented in this section were conducted on graphs derived from finite-element computations. These graphs are described in Table 1. For each graph, we modified the vertex and edge weights in order to simulate various types of repartitioning problems. Specifically, we constructed four repartitioning problems for each graph that simulate adaptive computations in which the work imbalance is distributed globally throughout the mesh. An example of an application in which this might occur is a particle-in-mesh computation. Here, particles may be located anywhere within the mesh and are free move to any other regions of the mesh. The result is that both the densely and sparsely populated regions are likely to be distributed globally throughout the mesh. Typically, this type of repartitioning problem is easier for diffusion-based schemes compared to scratch-remap schemes [17, 19]. We also constructed four problems that simulate adaptive mesh computations in which adaptation occurs in localized regions of the mesh. An example is the simulation of a helicopter blade. Here, the finite-element mesh must be extremely fine around

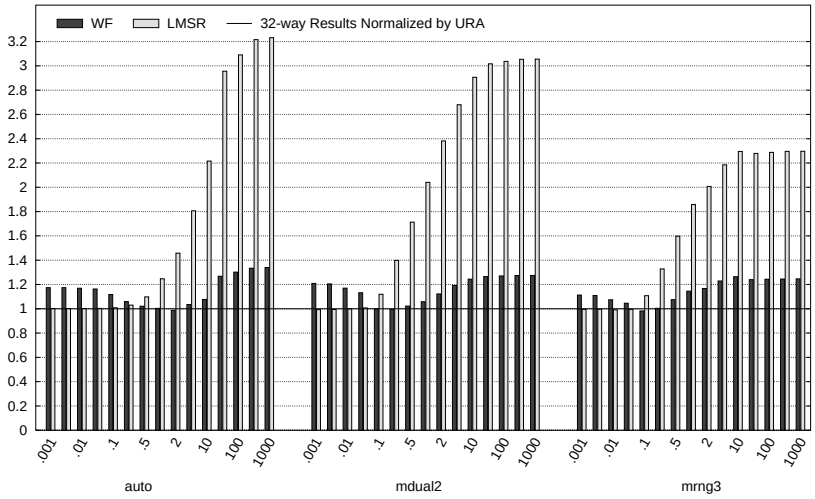

Figure 3: The cost function results obtained from the Unified Repartitioning Algorithm compared to the results obtained from optimized scratch-remap (LMSR) and multilevel diffusion (WF) algorithms on 32 processors of a Cray T3E.

both the helicopter blade and in the vicinity of the sound vortex that is created by the blade in order to accurately capture flow-field phenomena of interest. It should be coarser in other regions of the mesh for maximum efficiency. As the simulation progresses, neither the blade nor the sound vortex remain stationary. Therefore, the new regions of the mesh that these enter need to be refined, while those regions that are no longer of key interest should be de-refined. In this case, mesh refinement and de-refinement is often performed in very localized regions of the mesh. This type of repartitioning problem tends to be easier for scratch-remap schemes than diffusion-based schemes [17, 19].

Further experiments were performed on a real problem set from the simulation of a diesel internal combustion engine[2]. This is a particles-in-cells computation. The mesh consists of 175-thousand mesh elements. At first, no fuel particles are present in the combustion chamber. As the computation progresses, fuel particles are injected into the chamber at a single point and begin to spread out. Thus, they may enter regions of the mesh belonging to different processors. Load imbalance occurs as processors are required to track different numbers of particles.

Results on Synthetic Data Sets Figures 3 through 5 show the cost functions obtained by the Unified Repartitioning Algorithm compared to those obtained by the optimized scratch-remap and multilevel diffusion algorithms, LMSR and Wavefront Diffusion [17], as implemented in ParMeTiS [9] on up to 128 processors of a Cray T3E. Specifically, these figures show three sets of results, one from each of the graphs described in Table 1. Each set is composed of fifteen pairs of bars. These pairs represent the averaged results from the eight experiments (simulating global and localized imbalances) that are described above. For each pair of bars, the Relative Cost Factor was set to a different value. These values are .001, .002, .01, .02, .1, .25, .5, 1, 2, 4, 10, 50, 100, 500, and 1000. Therefore, for each set of results, minimizing the edge-cut is the dominate objective for the results on the left, while minimizing the data redistribution cost is the dominate objective for the results on the right. The results in the middle represent varying tradeoffs between the two objectives.

The bars in Figures 3 through 5 give the averaged relative performances of the optimized scratch-remap and multilevel diffusion repartitioners normalized by those of the Unified Repartitioning Algorithm. These are obtained by evaluating Equation 2 for each resulting partitioning. Therefore, a result above the 1.0 index line

[2] These test sets were provide to us by Boris Kaludercic, HPC Product Coordinator, Computational Dynamics Ltd, London, England.

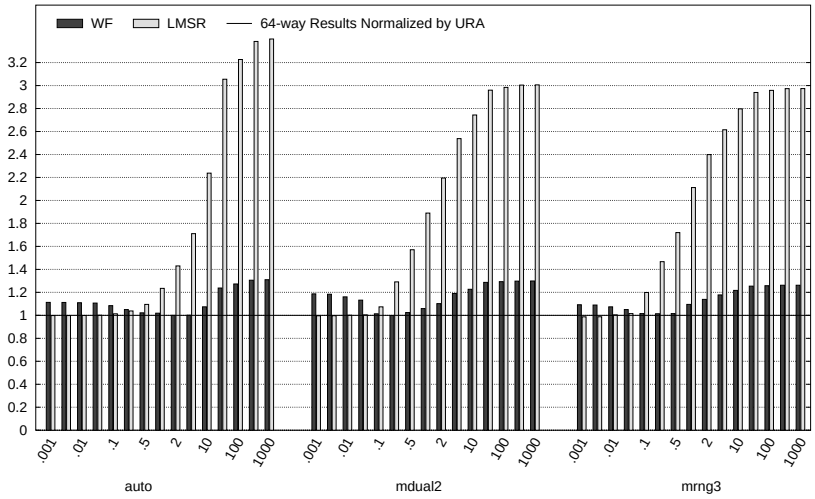

Figure 4: The cost function results obtained from the Unified Repartitioning Algorithm compared to the results obtained from optimized scratch-remap (LMSR) and multilevel diffusion (WF) algorithms on 64 processors of a Cray T3E.

indicates that the Unified Repartitioning Algorithm minimized Equation 2 better than the corresponding scheme.

Figures 3 through 5 show that the Unified Repartitioning Algorithm is able to minimize the cost function as well as or better than either of the other two schemes across the board. Specifically, when the Relative Cost Factor is set low (i. e., minimizing the edge-cut is the key objective), the Unified Repartitioning Algorithm minimizes the cost function as well as the scratch-remap scheme and better than the multilevel diffusion scheme. Note that the Unified Repartitioning Algorithm does quite well here, because when the RCF is set very low, it means that the edge-cuts of the partitionings are primarily being compared. Therefore, in order to obtain costs that are similar to the scratch-remap scheme, URA must be computing partitionings of similar edge-cut to a multilevel graph partitioner.

For the experiments in which the Relative Cost Factor is set high (i. e., minimizing the data redistribution cost is the key objective), the Unified Repartitioning Algorithm minimizes the cost function better than the multilevel diffusion scheme and much better than the scratch-remap scheme. URA beat the diffusion scheme here because it attempts to minimize the true cost function during multilevel refinement. The multilevel diffusion scheme, on the other hand, minimizes the edge-cut as the primary objective and the data redistribution cost as the secondary objective during refinement.

For the experiments in which the Relative Cost Factor was set near one, URA also tended to do as well as or better than either of the other two schemes in minimizing the cost function. It is interesting to note that the multilevel diffusion algorithm performs well in this region. This is because in the initial partitioning phase of this algorithm, the partitioning is balanced while aggressively minimizing the data redistribution cost. During the uncoarsening phase, the multilevel refinement algorithm focuses on aggressively minimizing the edge-cut. The result is that this scheme presents an almost even tradeoff between the edge-cut and the data redistribution cost.

The results presented in Figures 3 through 5 indicate that the Unified Repartitioning Algorithm is able to meet or beat the results obtained from either the scratch-remap or the multilevel diffusion repartitioner for a variety of experiments regardless of the value for the Relative Cost Factor. The other two schemes perform well only for limited ranges of values of the Relative Cost Factor.

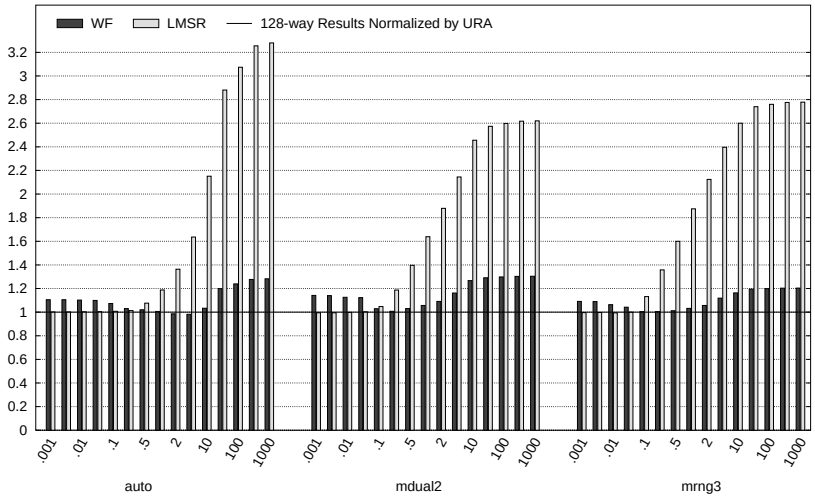

Figure 5: The cost function results obtained from the Unified Repartitioning Algorithm compared to the results obtained from optimized scratch-remap (LMSR) and multilevel diffusion (WF) algorithms on 128 processors of a Cray T3E.

Results on the Mesh from a Simulation of a Diesel Combustion Engine Table 2 gives the cost function and average run time results for URA, the optimized scratch-remap algorithm (LMSR), and the optimized multilevel diffusion algorithm (WF) on the diesel combustion engine test set for 8, 16, and 32 processors of a Cray T3E. The numbers at the top of each column indicate the Relative Cost Factor of the experiments in that column. Table 2 shows that URA minimizes the cost functions better than either of the other two schemes across the board. In this case, URA obtains somewhat better results than the scratch-remap scheme even when the Relative Cost Factor is set low. These results confirm that the URA scheme is able to minimize the cost function as well as or better than current repartitioning schemes.

Table 2 shows that all three repartitioning schemes obtained similar average run time results on 8 and 16 processors. However, URA obtains significantly worse average run time results on 32 processors than either of the other schemes. This is due to thrashing that occurred during multilevel refinement. This phenomenon is this is discussed in more detail below.

Parallel Run Time Results Tables 3 and 4 give the run time results of the optimized multilevel diffusion algorithm (WF), the optimized scratch-remap algorithm (LMSR), and URA for selected experiments from Figures 3 through 5 on a Cray T3E and for similar experiments run on up to 8 processors of a cluster of Pentium Pro workstations connected by a Myrinet switch. Table 3 shows that the repartitioning algorithms studied in this paper are very fast. For example, they are all able to compute a 128-way repartitioning of a four million node graph in only a couple of seconds on 128 processors of a Cray T3E. The Unified Repartitioning Algorithm shows somewhat worse scalability than the other two algorithms for high numbers of processors on the Cray T3E. This is due to *thrashing* that occurs during multilevel refinement. Here, vertices are repeatedly moved out of their original subdomains in order to balance the partitioning and then move right back into these subdomains in an effort to minimize the data redistribution term of the cost function. (Therefore, this thrashing does not occur when the Relative Cost Factor is set low.) The thrashing of vertices does not significantly affect the ability of the algorithm to minimize the cost function, but it does increase the run time of the uncoarsening phase.

Note that all of the reported run times were obtained on non-dedicated machines. Therefore, these results may contain a certain amount of noise. This reason, along with cache effects, explains the few super-linear speedups observed.

Scheme	0.001	0.01	0.1	1	10	100	1000	Avg. Time
8-processors								
WF	12,149	12,615	17,273	63,856	529,687	5,187,997	51,771,097	0.45
LMSR	7,934	8,349	12,499	54,000	469,008	4,619,088	46,119,887	0.43
URA	6,700	7,119	11,285	50,696	346,872	3,049,575	30,285,375	0.44
16-processors								
WF	18,914	19,450	24,802	78,323	613,535	5,965,655	59,486,855	0.33
LMSR	12,944	13,735	21,642	100,722	891,516	8,799,456	87,878,853	0.39
URA	11,381	12,157	20,852	62,239	398,284	4,107,916	40,775,713	0.36
32-processors								
WF	35,018	35,533	40,684	92,191	607,261	5,757,961	57,264,960	0.29
LMSR	18,815	19,632	27,800	109,479	926,274	9,094,224	90,773,725	0.27
URA	18,027	18,856	27,190	82,021	559,527	5,028,104	49,949,802	0.42

Table 2: Objective function costs and run time results of the adaptive graph partitioners for various RCF values on problems derived from a particles-in-cells simulation on a Cray T3E. The numbers at the top of each column indicate the RCF of the experiments in that column.

Graph	Scheme	8-processors	16-processors	32-processors	64-processors	128-processors
auto	WF	2.34	1.49	0.86	0.61	0.55
auto	LMSR	2.30	1.46	0.75	0.56	0.51
auto	URA	2.37	1.53	1.09	0.88	1.03
*mdual*2	WF	3.25	1.84	1.01	0.77	0.68
*mdual*2	LMSR	3.21	1.82	0.96	0.63	0.58
*mdual*2	URA	3.26	1.89	1.25	1.08	1.71
*mrng*3	WF	10.75	6.15	3.20	1.88	1.45
*mrng*3	LMSR	10.75	6.13	3.18	1.76	1.21
*mrng*3	URA	10.90	6.31	3.48	2.55	2.43

Table 3: Parallel run times of selected experiments for the adaptive graph partitioners WF, LMSR, and URA on a Cray T3E.

Scheme	2-processors	4-processors	8-processors
WF	12.13	6.36	3.07
LMSR	12.18	6.12	3.03
URA	12.41	6.46	3.17

Table 4: Parallel run times of experiments performed on the graph *auto* for the adaptive graph partitioners WF, LMSR, and URA on a cluster of Pentium Pro workstations connected by a Myrinet switch.

4 Conclusions

We have presented a Unified Repartitioning Algorithm for dynamic load-balancing of scientific simulations and shown that this scheme is fast and effective. Our Unified Repartitioning Algorithm is significant because it is able to gracefully tradeoff the objectives of minimizing the edge-cut and the amount of data redistribution required to balance the load regardless of the relative costs of these. In fact, the scheme is so general in this sense that the algorithm even can be used as a static graph partitioner. That is, the scheme acts as a multilevel graph partitioner when the Relative Cost Factor is set to zero (i. e., the data redistribution cost is not considered when computing the partitioning). The Unified Repartitioning Algorithm is also significant because it represents a key component in developing tools for automatically performing dynamic load-balancing. For example, dynamic load-balancing tools such as DRAMA [10] and Zoltan [3] can measure the times required to perform inter-processor communications and data redistribution for an application and use this information to automatically compute an accurate Relative Cost Factor to be used as an input for the repartitioning algorithm.

5 Extensions to be Included in the Full Paper

In the full paper, we plan to include a more complete description of the scheme and additional results from experiments derived from real application domains. We also plan to modify the refinement algorithm of URA so as to eliminate the thrashing of vertices discussed in Section 3. We are confident that this can be accomplished by the same technique used in the parallel formulation of our multi-constraint graph partitioner [18]. Incorporating this technique into our Unified Repartitioning Algorithm should further improve the already good parallel run time results reported in this paper, while maintaining the effectiveness of the algorithm.

Acknowledgements

We would like to thank Boris Kaludercic, HPC Product Coordinator, Computational Dynamics Ltd, London, England for providing us with the data for the diesel combustion engine test sets.

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] J. Castanos and J. Savage. Repartitioning unstructured adaptive meshes. In *Proc. Intl. Parallel and Distributed Processing Symposium*, 2000.

[3] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Design of dynamic load-balancing tools for parallel applications. In *Proc. of the Intl. Conference on Supercomputing*, 2000.

[4] P. Diniz, S. Plimpton, B. Hendrickson, and R Leland. Parallel algorithms for dynamically partitioning unstructured grids. *Proc. 7th SIAM Conf. Parallel Proc.*, 1995.

[5] J. Flaherty, R. Loy, C. Ozturan, M. Shephard B. Szymanski, J. Teresco, and L. Ziantz. Parallel structures and dynamic load balancing for adaptive finite element computation. *Appl. Numer. Maths*, 26:241–263, 1998.

[6] B. Jeremic and C. Xenophontos. Application of the p-version of the finite element method to elasto-plasticity with localization of deformation. *Communications in Numerical Methods in Engineering*, 15(12):867–876, 1999.

[7] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[8] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *Siam Review*, 41(2):278–300, 1999.

[9] G. Karypis, K. Schloegel, and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical report, Univ. of MN, Dept. of Computer Sci. and Engr., 1997.

[10] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale. DRAMA: A library for parallel dynamic load balancing of finite element applications. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[11] L. Oliker and R. Biswas. PLUM: Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 52(2):150–177, 1998.

[12] C. Ou and S. Ranka. Parallel incremental graph partitioning using linear programming. *Proceedings Supercomputing '94*, pages 458–467, 1994.

[13] C. Ou, S. Ranka, and G. Fox. Fast and parallel mapping algorithms for irregular and adaptive problems. *Journal of Supercomputing*, 10:119–140, 1996.

[14] A. Patra and D. Kim. Efficient mesh partitioning for adaptive *hp* finite element meshes. Technical report, Dept. of Mech. Engr., SUNY at Buffalo, 1999.

[15] J. Pilkington and S. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. Technical report, Dept. of Computer Science and Engineering, Univ. of California, 1995.

[16] 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.

[17] K. Schloegel, G. Karypis, and V. Kumar. Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes. Technical Report TR 98-034, Univ. of Minnesota, Dept. of Computer Sci. and Engr., 1998.

[18] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Proc. EuroPar-2000*, 2000. Accepted as a Distinguished Paper.

[19] K. Schloegel, G. Karypis, V. Kumar, R. Biswas, and L. Oliker. A performance study of diffusive vs. remapped load-balancing schemes. *ISCA 11th Intl. Conf. on Parallel and Distributed Computing Systems*, pages 59–66, 1998.

[20] A. Sohn. S-HARP: A parallel dynamic spectral partitioner. Technical report, Dept. of Computer and Information Science, NJIT, 1997.

[21] A. Sohn and H. Simon. JOVE: A dynamic load balancing framework for adaptive computations on an SP-2 distributed-memory multiprocessor. Technical Report 94-60, Dept. of Computer and Information Science, NJIT, 1994.

[22] R. VanDriessche and D. Roose. Dynamic load balancing of iteratively refined grids by an enhanced spectral bisection algorithm. Technical report, Dept. of Computer Science, K. U. Leuven, 1995.

[23] A. Vidwans, Y. Kallinderis, and V. Venkatakrishnan. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA Journal*, 32:497–505, 1994.

[24] 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, 1995.

[25] 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.