Multilevel Algorithms for Generating Coarse Grids for Multigrid Methods*

Irene Moulitsas and George Karypis

University of Minnesota, Department of Computer Science / Army HPC Research Center
Minneapolis, MN 55455

{moulitsa, karypis}@cs.umn.edu

Abstract

Geometric Multigrid methods have gained widespread acceptance for solving large systems of linear equations, especially for structured grids. One of the challenges in successfully extending these methods to unstructured grids is the problem of generating an appropriate set of coarse grids. The focus of this paper is the development of robust algorithms, both serial and parallel, for generating a sequence of coarse grids from the original unstructured grid. Our algorithms treat the problem of coarse grid construction as an optimization problem that tries to optimize the overall quality of the resulting fused elements. We solve this problem using the multilevel paradigm that has been very successful in solving the related grid/graph partitioning problem. The parallel formulation of our algorithm incurs a very small communication overhead, achieves high degree of concurrency, and maintains the high quality of the coarse grids obtained by the serial algorithm.

*This work was supported by NSF CCR-9972519, EIA-9986042, ACI-9982274, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SC2001 November 2001, Denver (c) 2001 ACM 1-58113-293-X/01/0011 $5.00

1 Introduction

Geometric multigrid methods have gained widespread acceptance for solving large systems of linear equations, especially for structured grids. One of the challenges in successfully extending these methods to unstructured meshes is that there exists no widely accepted method for generating an appropriate set of coarse grids.

Existing coarse grid construction algorithms operate on the dual graph of the unstructured mesh and obtain the control volumes of the coarse grid by using a variety of agglomerating techniques [9, 2, 10, 11, 12]. The basic idea behind these approaches is to start from a particular vertex of the dual graph and fuse together some of its adjacent vertices into a new coarse grid control volume. The selection of the vertices to be fused together can be based either on the connectivity of the dual graph [2], or it can be done so that the quality of the coarse control volume, as measured by its aspect ratio, is locally optimized [13]. Unfortunately, such locally greedy agglomerating techniques quite often lead to coarse grids of poor quality. Thus, despite the number of different agglomerative approaches already developed, there is still a great need for improvement.

The focus of this paper is to develop robust algorithms, both serial and parallel, for generating a sequence of coarse grids from the original unstructured grid. To this end, we formulate the problem as an optimization problem whose goal is to generate a coarse grid that optimizes a particular objective function which captures the overall quality of the grid subject to lower and upper bound constraints on the size of grid's control volumes.

We investigate two general classes of coarse-grid quality measures that are based on the aspect ratios of the resulting control volumes. The first class tries to optimize an additive function of the individual control volumes' aspect ratios, whereas the second class focuses on bounding the aspect ratio of the worst elements. We develop highly accurate and computationally efficient algorithms for solving these optimization problems that are based on the multilevel paradigm, that has been recently found to be quite effective in solving the related problem of graph and mesh partitioning [1, 3, 5, 4]. We also present a parallel formulation of our serial algorithm that combines our serial multilevel coarse grid construction algorithm with existing scalable parallel formulations of adaptive graph partitioning [15] to effectively parallelize the generation of coarse grids.

We experimentally evaluate the performance of our algorithms in the simulation of unsteady flow on three different unstructured grids. Our experiments show that the coarse grids obtained by our serial algorithms allow the multigrid solver to converge in 5% to 20% fewer iterations when compared to existing agglomerating techniques. Moreover, our parallel evaluation shows that the improved convergence rates carry over to the parallel formulation as well and that our parallel formulation can efficiently scale up to 512 processors even for moderate size grids containing around one million elements.

The rest of the paper is organized as follows. Section 2 motivates and describes the various objective functions that were used to measure the quality of the coarse grids. Section 3 describes the details of our serial multilevel algorithm for building coarse grids, and Section 4 describes its parallel formulation. The experimental evaluation of both the serial and parallel algorithms is shown in Section 5. Finally Section 6 provides some concluding remarks.

2 Objective Functions

The aim of our coarse grid construction algorithms is to generate a sequence of coarse grids such that each grid optimizes a certain function that captures the overall quality of the fused elements, *i.e.*, the control volumes of the coarser grids. In order to ensure fast convergence rates of multigrid algorithms the sequence of coarse grids must contain well shaped control volumes. A widely accepted measure for the quality of a control volume (*i.e.*, how well-shaped it is) is its aspect ratio A. For two–dimensional grids the aspect ratio is defined as

$$A = \frac{l^2}{S},$$

where l is the circumferential length and S is the area of the control volume, respectively. Analogously, for three-dimensional grids the aspect ratio is defined as

$$A = \frac{S^{3/2}}{V},$$

where, S is the surface area and V is the volume of the control volume, respectively. A control volume is well-shaped if it is as compact as possible, *i.e.*, its aspect ratio is minimized. Hence, for two- and three-dimensional grids we want to get control volumes that are as close to being circular and spherical, respectively.

Using the aspect ratio of a control volume as a measure of its quality we can derive a number of different measures to capture the overall quality of a coarse grid. In particular, if $NCoarse$ is the number of control volumes in the coarse grid, a simple way of measuring its quality is to add the aspect ratios of all of its control volumes. That is,

$$F_1 = \sum_{i=1}^{NCoarse} A_i, \tag{1}$$

where A_i is the aspect ratio of the ith control volume. Since our goal is to obtain grids that contain well-shaped control volumes, a coarse grid that minimizes F_1 is preferred. One of the limitations of the F_1 measure is that it does not take into account the size of the different control volumes which can potentially be somewhat different. This measure can be easily modified to give higher weight to larger control volumes; thus, penalizing (or rewarding) it when the aspect ratio of large control volumes are poor (or good). In this case the new grid quality measure is

$$F_2 = \sum_{i=1}^{NCoarse} w_i \, A_i, \tag{2}$$

where w_i is the number of elements that were fused together to obtain the ith control volume. Again, coarse grids that minimize the F_2 measure are also preferred. Potentially, though, either the F_1 or the F_2 measures can fail to account for coarse grids whose overall quality is good, but nonetheless, they still contain a limited number of control volumes that are of poor quality. To correct this problem, we can measure the quality of a coarse grid by looking at the aspect ratio of its worst (*i.e.*, higher as-

pect ratio) control volume. In this case, the quality of the coarse grid is given by

$$F_3 = \max_{i=1...NCoarse} A_i. \quad (3)$$

Coarse grids that minimize the F_3 measure will be preferred.

These three different functions for measuring the quality of a coarse grid can be used as the objectives to be optimized while generating the next level coarse grid. In particular, the coarse grid construction problem can be formulated as the following optimization problem.

> Given an initial grid, generate a coarse grid such that each of its control volumes contains at least $Lmin$ and at most $Lmax$ elements of the initial grid, and the coarse grid minimizes one of the different grid quality measures, that is either F_1, F_2, or F_3.

Note that the various grid-quality measures can be combined to perform multi-objective optimizations. In particular, F_3 can be easily combined with either F_1 or F_2. In this case, the goal of the optimization algorithm is to generate a coarse grid that first minimizes the F_3 measure, and among the solutions that have the same F_3 value, it focuses on constructing a grid that minimizes either F_1 or F_2.

In the rest of this paper we will refer to the various grid-quality measures as the corresponding objective functions.

3 Serial Multilevel Coarse Grid Construction

Our algorithm for generating coarse grids solves the optimization problem described in Section 2, using the multilevel paradigm. The basic idea of the multilevel paradigm is the following. If M is the original problem, a sequence of successive approximations of M is obtained $\{M_1, M_2, \ldots, M_n\}$ such that each successive approximation M_{i+1} is a smaller problem than its previous approximation M_i. Once that sequence has been obtained, a solution of the optimization problem is computed at the coarsest approximation M_n. This solution is used to derive a solution for the next-level finer approximation M_{n-1}, which is further optimized locally on M_{n-1}. Now, the solution to M_{n-1} is again used to derive a solution for M_{n-2} which is also further optimized. This process continues, until a solution propagates all the way up to the original problem M, which becomes the final solution. The three distinct phases of the multilevel paradigm are called *coarsening*, *initial solution*, and *uncoarsening* phases, respectively.

The multilevel paradigm has been used successfully in many applications such as traveling salesman and graph partitioning [1, 3, 16, 5, 4]. In fact, the widely used graph partitioning algorithms available in Chaco, METIS, and JOSTLE are based on this paradigm. Also note that the overall idea of the multilevel optimization paradigm is very similar in nature to the multigrid algorithm itself.

The method used to obtain approximate representations of the original problem plays an important role in the success of the multilevel paradigm. In particular, the approximate representations must be such that the quality of a solution obtained in them is similar (if not identical) to the quality of the same solution when viewed in the original problem. This ensures that the optimization being performed at the coarsest levels of the approximations, does optimize the solution of the original problem. This is true in the context of graph partitioning, and as we will show in the rest of this section, is true for our problem as well.

In the rest of this section we discuss how the various grids are represented, and describe our algorithms for the coarsening, initial solution, and uncoarsening phases. Note that our discussion will only focus on generating the next level coarse grid and the overall sequence of grids can be obtained by applying the same algorithms repeatedly.

3.1 Grid and Solution Modeling

Our algorithms model the grid using its dual graph. In the dual graph $G = (V, E)$, each vertex corresponds to an element of the grid (*i.e.*, triangle, tetrahedron, brick), and two vertices are connected via an edge, if the corresponding grid elements share a segment or face depending on whether or not the grid is two- or three-dimensional. An example of a two-dimensional triangular mesh and its corresponding dual graph is shown in Figure 1.

For every vertex v in the graph we have three values associated with it, that we call them *vertex-weight* (v^w), *vertex-boundary-surface* (v^s), and *vertex-volume* (v^v). The vertex-weight captures the number of original grid elements that this particular element represents. Initially, all vertex-weights are set to one, but as the original grid is coarsened, their weights are updated to account for the number of elements that the particular control volume is

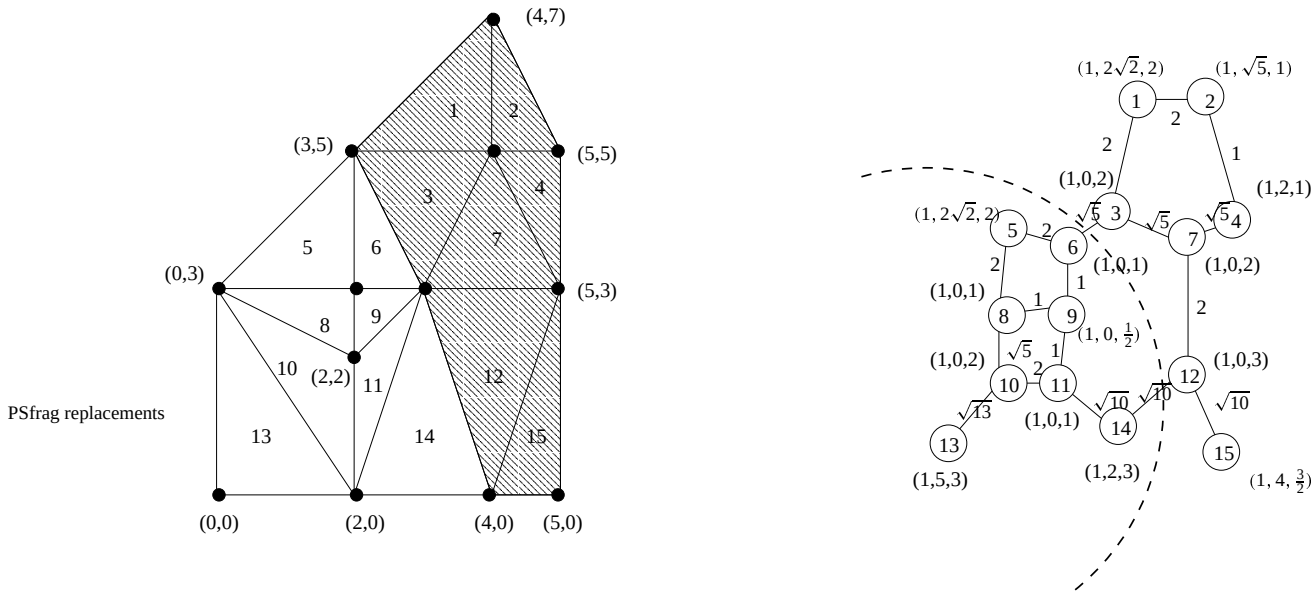

Figure 1: Sample two-dimensional triangular mesh and the dual Graph.

made off. The vertex-boundary-surface is used to capture the length or area of the element's segments or faces that are not shared by other elements, *i.e.*, they are on the boundary of the grid. Note that for all interior elements v^s will be zero. The vertex-volume is used to capture the area (for two-dimensional grids) or the volume (for three-dimensional grids) of that particular element. Finally, for every edge (v, u), our dual graph has a weight associated with it that corresponds to either the length of the shared line segment or the area of the shared face, in two- or three-dimensional grids, respectively. The various vertex and edge weights are illustrated in Figure 1.

The weighted dual graph contains all the information that is necessary to compute the aspect ratios of each grid element as well as any control volume derived by combining any of these elements. In particular, in the case of two-dimensional grids, the area of an element is nothing more than v^v, whereas its circumferential length is the sum of the weights on its edges plus v^s. Similarly, the area of a control volume consisting of a collection of such elements B, is nothing more than the sum of the vertex-volumes of the vertices in B, and the circumferential length is equal to the sum of the weights of the edges connecting a vertex in B with a vertex in $V - B$ plus the sum of the vertex-boundary-surfaces of the vertices in B. For example, consider the control volume that is derived from the shaded elements of the grid shown in Figure 1. The area of that control volume is equal to the sum of the v^v weights of the corresponding vertices, and its circumferential length is equal to the sum of the weights of the edges that are being cut by the corresponding partitioning plus the v^s weights of all the vertices corresponding to the shaded region. A similar method can be used to compute the corresponding quantities in three-dimensional grids.

Given such a graph representation of the original grid, the desired coarse grid can be viewed as a k-way partitioning of the vertices of the dual graph such that each partition contains between $Lmin$ and $Lmax$ vertices, and optimizes a particular grid-based objective function. Essentially, the vertices in each partition represent the grid elements that will be combined to form the control volumes of the coarse grid. This graph-partitioning-based view of the solution will be used through out our description of the algorithm. In particular, our algorithms will operate on the dual graph and will try to find such a k-way partitioning. The various vertex and edges weights of the dual graph contain all the information necessary to accurately evaluate the various objective functions discussed in Section 2.

3.2 Coarsening Phase

Our algorithms, starting from the original grid represented by its dual graph $G = (V, E)$, construct a sequence of approximate representations by obtaining a sequence of successively coarser graphs $\{G_1, G_2, \ldots, G_n\}$. Each graph G_i is obtained from the previous graph G_{i-1} by first finding a maximal indepen-

dent set I_{i-1} of edges of G_{i-1}, and then collapsing together the pairs of vertices that are connected by edges in I_{i-1}. As a result of this operation, G_i contains $|I_{i-1}|$ fewer vertices than G_{i-1}. An *independent* set of edges of a graph is a set of edges no two of which are incident to the same vertex. An independent set is *maximal* if it is not possible to add any other edge to it without making two edges become incident on the same vertex.

To preserve the geometric properties of the original grid, the weights of the vertices and edges in G_i are updated to account for the different vertices and edges that are being collapsed together. In particular, if (v_1, v_2) is an edge in I_{i-1}, and u_1 is the vertex of G_i that is obtained by collapsing together vertices v_1 and v_2, our algorithm sets the vertex-weight u^w of u to be equal to $v_1^w + v_2^w$, its vertex-boundary-surface u^s to be equal to $v_1^s + v_2^s$, and its vertex-volume u^v to be equal to $v_1^v + v_2^v$. To preserve the connectivity information in the coarser graph, the edges of u are the union of the edges of the vertices v_1 and v_2. If both vertices v_1 and v_2 contain edges to the same vertex v_3, then the weight of the edge (u, v_3) is set equal to the sum if the weights of these edges. By updating the weights in this fashion, we can accurately compute the aspect ratio of the control volumes that each vertex represents.

The key step of our algorithm is the method used to find the maximal independent sets of edges. Motivated by earlier research on graph partitioning we use a method called ***globular matching*** that was inspired by the heavy edge heuristic [5]. The idea behind this approach is to select a maximal independent set of edges such that the control volumes that are created as a result of collapsing the pairs of vertices for each edge, have the smallest aspect ratio. In globular matching, we try to achieve that using the following greedy algorithm. The vertices are visited in decreasing order of their degree. If a vertex has not been matched yet (*i.e.*, is not part of an edge already in the independent set), we match it with its adjacent unmatched vertex that leads to the smallest aspect ratio, and the resulting vertex does not violate the constraint of the maximum size of the control volume (*i.e.*, the $Lmax$ constraint). The edge corresponding to this pair of vertices is then added to the independent set. Note that this algorithm does not guarantee that the obtained independent set leads to the smallest aspect ratios, but our experiments have shown that it works very well in practice. The complexity of computing a globular agglomeration is $O(|E|)$.

The coarsening phase ends when the graph cannot be further coarsened without violating the $Lmax$ constraint on the maximum size of each control volume.

3.3 Initial Solution Phase

The goal of the initial solution phase is to compute a solution of the original problem using its coarsest approximation. In our algorithm, the coarsest approximation corresponds to a graph G_n whose vertices were obtained by collapsing adjacent elements of the original grid. Also, due to the termination condition of the coarsening phase, each vertex is guaranteed to contain no more than $Lmax$ elements of the original grid. Given these characteristics of G_n, we obtain a solution to the original problem by essentially treating each vertex of G_n as a single coarse element of the original grid. Note that this solution may not necessarily satisfy the $Lmin$ constraint, since some of the vertices in G_n may actually contain fewer than $Lmin$ elements of the original grid. This is something that will be corrected during the uncoarsening phase.

3.4 Uncoarsening Phase

The goal of the uncoarsening phase is to take the solution computed in the coarsest graph and propagate it all the way up to the original graph, by going through graphs $G_{n-1}, \ldots, G_1, G$, and further refine it during this propagation.

The initial solution at the coarsest graph is nothing more than a $|V_n|$-way partitioning of this graph, such that the ith part contains only the ith vertex of this graph. When this solution is propagated to the next level finer graphs, most of these $|V_n|$ partitions will contain more than just a single vertex. The overall quality of the solution can be potentially improved by moving some of the vertices to different partitions, as long as the size-constraints are not violated. Such vertex movements take a subset of elements from one control volume and assign them to a different control volume. Such improvements are possible because of three reasons. First, the maximal independent set were computed using a greedy algorithm and does lead to an optimal independent set. Second, the objective function that we try to optimize may be somewhat different than the heuristic used to guide the construction of the maximal independent set. Third, finer graphs have more degrees of freedom that can be used to further improve the quality of a solution derived from a coarser graph.

We use a randomized refinement algorithm that is

similar in nature to that used in the context of multilevel k-way partitioning [4]. It consists of a number of iterations. In each iteration the vertices are visited in a random order. For each vertex v we compute the reduction in the value of the objective function that will be achieved if v was to move to another neighboring partition (*i.e.*, control volume). If there exist some moves that lead to actual improvements without violating the size-constraints, then v is moved to the partition that leads to the highest improvement among them. If no such move exists, then v is not moved. This process stops when no vertex can be moved during an entire iteration. Our experiments show that in each graph, our refinement algorithm converges within two to five iterations.

This randomized refinement algorithm can be used with all the different objectives functions described in Section 2. Moreover, all of these objective functions can be efficiently evaluated as they depend on the aspect ratios of the partitions that are adjacent to the vertex that we try to move.

One of the side-effects of our refinement algorithm is that it does not guarantee that the discovered partitions will be contiguous, even if the initial partitioning contained contiguous partitions. This is because during the refinement, vertices corresponding to graph articulation points can be moved when such moves improve the particular objective function. To handle these cases, at the end of each refinement phase, we identify which partitions contain non-contiguous control-volumes, and create different partitions for each one of them. As a result, the final solution may contain more than $|V_n|$ partitions. Due to this contiguity enforcement as well as due to the coarsening, the partitions may actually contain fewer than $Lmin$ elements. To correct this problem after the partition contiguity has been enforced, we try to merge small partitions with some of their adjacent partitions, as long as the resulting partition does not violate the $Lmax$ constraint. The selection of which partitions to merge is guided by the particular objective function that we try to optimize. Once the merging operation is finished, the number of control volumes that have fewer than $Lmin$ elements is now greatly minimized. Only those control volumes that are adjacent to prohibitively "large" control volumes will remain unmerged. As a last step, the neighbors of the small partitions, that can afford to lose some of their vertices, have to contribute these vertices to these partitions. In this way we can ensure that the $Lmin$ and $Lmax$ constraints are satisfied.

4 Parallel Implementation

In recent years, a number of scalable parallel formulations of multilevel graph partitioning algorithms have been developed [8, 14, 6, 15]. However, even though our serial coarse grid construction algorithm shares many characteristics with these multilevel partitioning algorithms, their parallel formulations cannot be used to efficiently parallelize the grid construction algorithm. This is because, unlike graph partitioning, in which the number of partitions is very small relatively to the size of the graph, in coarse grid construction, the number of fused elements (which correspond to the number of partitions) is very large and of the same order as the number of vertices. This difference makes existing parallel formulations of multilevel graph partitioning unscalable, as their communication overhead is lower bounded by the number of partitions. This led us to develop an entirely new approach of parallelizing the coarse grid construction algorithm that does not rely on existing parallel formulations of multilevel graph partitioning.

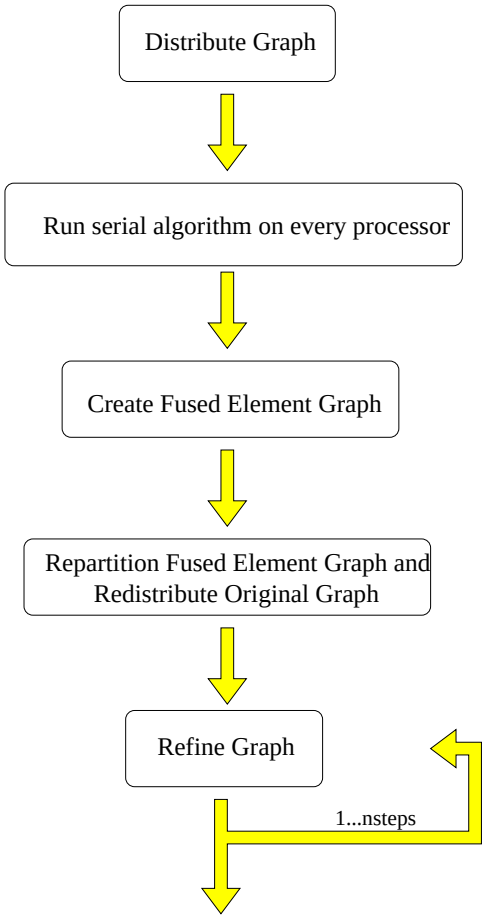

Figure 2: The various phases of the the parallel procedure.

The overall structure of our parallel algorithm is shown in Fig. 2. Initially, the mesh is distributed among the processors using a parallel multilevel graph partitioning algorithm. This results in a distribution in which each processor contains a well-shaped subdomain, in the sense that the number of elements between processors is minimized. Now, each processor, operates on its locally stored portion of the overall mesh, and it cuts all connections to other processors. It then uses the serial multilevel coarse grid construction algorithm to generate a coarse grid for its local subdomain. Since the processors operate only on their own subdomains, this approach leads to a coarse grid whose interior contains elements with good aspect ratios, but because it is not allowed to create fused elements across processor subdomain boundaries, the quality of the elements along the processor subdomain boundaries may be poor.

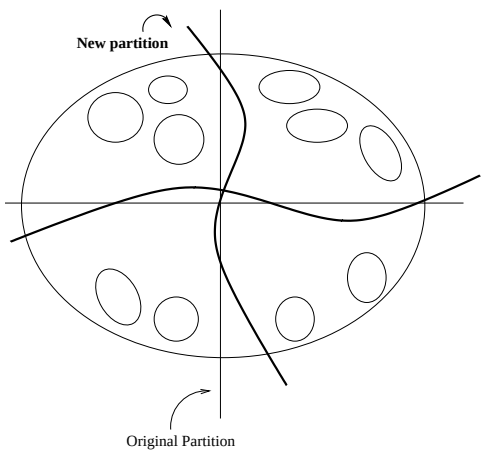

Figure 3: Partition.

One way of correcting this problem is to allow fused elements on the boundaries to participate in refinement iterations with the fused elements stored in neighboring (with respect to the mesh) processors. However, this approach leads to fine-grain communication and synchronization that can potentially limit the overall parallel efficiency. For this reason we developed an alternate approach for correcting the quality of these interface elements. The key idea of our approach is to use an adaptive graph partitioning algorithm to perturb the existing mesh partition, so that the elements along the processor interface boundary move closer to the interior, and away from the boundary. The motivation behind this approach is that if the fused elements along the interface move towards the interior of the subdomain, then their adjacent fused elements will move to the same subdomain as well, and their quality can be improved by simply performing local refinement. This is illustrated in Fig. 3, in which the dark black lines correspond to the new partitioning of the original mesh. Note that the repartitioning of the mesh can be done in such a way so that the elements that have already been fused together are assigned to the same processor, thus preserving the quality of the existing fused elements.

Our parallel algorithm uses the adaptive graph partitioning algorithm available in PARMETIS([7]), and the overall process of adaptive repartitioning followed by local refinement is performed until the overall quality the coarse grid does not improve any further. Our experiments have shown that the overall process converges within a small number of iterations (less than ten).

5 Experimental Results

We evaluated the performance of our serial and parallel algorithms using three different 3D tetrahedral meshes whose characteristics are shown in Table 1. The size of these meshes ranged from 94.5K to 1.1M elements. We tested the quality of the grids obtained from our algorithms in the simulation of an unsteady flow of moving grids, arising in aero–elasticity problems, using an edge–based multigrid solver, for each one of the three meshes. The parallel performance was evaluated on two different architectures. The first one was a 1024–processor CRAY T3E-1200 parallel computer with EV6 Alpha processors running at 600MHz and 512MB of memory at each processor. The second was a 16-processor Linux-based cluster of workstations connected via a 100MBit Ethernet switch. Each processor was an Intel Pentium III at 650Mhz with 1GB of main memory. We will refer to this cluster as the "BEO" machine.

Name	# Elements	Description
M6	94, 493	M6 wing
F22	428, 748	F22 wing
F16	1, 124, 648	F16 wing

Table 1: Characteristics of the test data sets.

5.1 Evaluation of the Serial Algorithm

Our first set of experiments was focused on evaluating the quality of the coarser grids produced by our serial multilevel coarse grid construction algorithm using the unsteady flow simulation.

Table 2 shows the convergence characteristics of the

multigrid solver for different methods for constructing the coarse grids. In particular, the table shows the results for six different algorithms for coarse grid construction. The row labeled "Trad1" corresponds to the results obtained by the simple neighborhood based agglomerative scheme [2]. The results labeled "Trad2" correspond to the neighborhood based agglomerative scheme that takes into account the aspect ratio of the new cells [11, 13]). The remaining results labeled "ML_F1", "ML_F2", "ML_F3", "ML_F3_F2", correspond to our multilevel coarse grid construction algorithms using the F_1, F_2, F_3, and $F_3 + F_2$ objective functions respectively from section 2.

	M6	F22	F16
Technique	# Iterations	# Iterations	# Iterations
Trad1	215	181	399
Trad2	160	153	358
ML_F1	146	155	349
ML_F2	149	159	345
ML_F3	156	157	349
ML_F3_F2	148	160	339

Table 2: Convergence of serial multigrid algorithm.

Looking at the results in this table we can see that the different objectives of the proposed multilevel algorithms for constructing the coarse grids, tend to perform very similar to each other. In most cases, the number of iterations required by the multigrid solver on the coarse grids generated by the four objectives are within 3% from each other. The only exception was ML_F3 that required 7% more iterations on M6 than the other schemes. Comparing Trad1 against Trad2 we can see that the former performs significantly worse. Comparing the multilevel approaches against Trad2 we can see that they lead to coarse grids that are better (at least in terms of the number of iterations) for M6 and F16 and they achieve comparable quality for F22. In particular, the number of iterations required by the multigrid solver is lower by 10% on M6 and 6% on F16 when the multilevel schemes are used.

5.2 Evaluation of the Parallel Algorithm

We evaluated the performance of our parallel formulation of the multilevel coarse grid construction algorithm on the same three unstructured meshes used by the serial algorithm. Our evaluation was done at three levels. First we compared the aspect-ratio characteristics of the generated coarse grids as the number of processors increased. Second, we compared the performance of the multigrid solver on the parallel generated grids. Finally, we evaluated the scalability of the parallel algorithm itself.

Table 3 shows the quality of the coarse grids produced by our parallel algorithm on the CRAY using $1, 2, 4, 8, 16, 32, 64, 128, 256,$ and 512 processors for the M6, F22 and the F16 data sets. These results were obtained using the scheme ML_F3_F2. Note that some of the results could not be obtained due to the fact that there was not enough memory on small number of processors on the CRAY T3E. Table 4 shows the results of the same tests ran on the BEO cluster for $1, 2, 4, 8,$ and 16 processors. Results from tests that were not obtained on the CRAY due to memory limitations can be seen in this table. From these results we can see that with respect to the F_2 measure, the quality of the coarse grids produced by the parallel algorithm remains the same as we increase the number of processors. In fact the overall F_2 measure seems to improve as we increase the number of processors. This is primarily due to the fact that the larger processor configurations produced grids with slightly more elements. With respect to the F_3 measure the overall quality still remains the same, even though there are certain instances in which the results are somewhat different. This is primarily due to the fact that the F_3 measure is entirely determined by the quality of a single fused element, and it is much more sensitive to the underlying randomization of the algorithm.

Table 5 shows the convergence of the multigrid solver using the various grids generated by the different objective functions of the multilevel algorithm. Since the current version of our multigrid solver runs only serially, these results were obtained by first generating the coarse grids in parallel, and then using them as input to the serial multigrid algorithm. Due to time limitations the results reported in this table show only the convergence performance for up to 16 processors. Nevertheless, even from these limited results, we can see that, in general, our parallel multilevel algorithms produce grids that lead to similar convergence rates for the multigrid algorithm, as our serial multilevel algorithm. In fact, the number of iterations required by the various algorithms remained pretty much constant as we increased the number of processors. The only exception was with problem F16, for which the number of iterations did increase slightly for ML_F2, ML_F3, and ML_F3_F2. One of the reasons for that may be due to the fact the size of the coarse grids

	M6		F22		F16	
# PES	F_3	F_2	F_3	F_2	F_3	F_2
1	$2.43e+01$	$1.82e+06$	—	—	—	—
2	$2.26e+01$	$1.82e+06$	—	—	—	—
4	$2.25e+01$	$1.82e+06$	$2.71e+01$	$8.29e+06$	—	—
8	$2.27e+01$	$1.82e+06$	$2.93e+01$	$8.28e+06$	—	—
16	$2.26e+01$	$1.81e+06$	$2.31e+01$	$8.25e+06$	$2.24e+01$	$2.02e+07$
32	$2.26e+01$	$1.80e+06$	$2.36e+01$	$8.23e+06$	$2.64e+01$	$2.02e+07$
64	$2.26e+01$	$1.80e+06$	$2.40e+01$	$8.21e+06$	$7.11e+01$	$2.01e+07$
128	$2.26e+01$	$1.80e+06$	$2.31e+01$	$8.20e+06$	$2.28e+01$	$2.01e+07$
256	$2.30e+01$	$1.79e+06$	$1.68e+02$	$8.19e+06$	$2.84e+01$	$2.01e+07$
512	$2.43e+01$	$1.78e+06$	$4.57e+01$	$8.18e+06$	$3.50e+01$	$2.01e+07$

Table 3: Quality measures on CRAY T3E on 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512 processors.

	M6		F22		F16	
# PES	F_3	F_2	F_3	F_2	F_3	F_2
1	$2.70e+01$	$1.82e+06$	$2.55e+01$	$8.33e+06$	$2.28e+01$	$2.04e+07$
2	$2.29e+01$	$1.82e+06$	$2.32e+01$	$8.30e+06$	$2.84e+01$	$2.03e+07$
4	$3.19e+01$	$1.82e+06$	$2.55e+01$	$8.29e+06$	$2.84e+01$	$2.03e+07$
8	$2.27e+01$	$1.81e+06$	$2.41e+01$	$8.28e+06$	$2.28e+01$	$2.03e+07$
16	$2.26e+01$	$1.81e+06$	$2.31e+01$	$8.25e+06$	$2.84e+01$	$2.02e+07$

Table 4: Quality measures on BEO on 1, 2, 4, 8, and 16 processors.

generated as the number of processors increases is somewhat larger. We are currently investigating these results.

Finally, Table 6 shows the run times (in seconds) required by the different processors to construct the coarse grids. The times appearing here correspond to the runs made for creating Tables 3 and 4. The single processor run times were obtained using the serial algorithm. A number of observations can be made from this table. First, looking at the results of M6 (which we were able to run on single processor on the CRAY), and the results of all grids on the BEO, we can see that the two-processor time is actually higher than the serial time. This is due to the fact that the parallel algorithm performs more computations, as it needs to refine the solutions multiple times (once after each repartitioning). However as the number of processors increases, the amount of time required does decrease. In the case of the CRAY T3E, the runtime actually reduces linearly all the way up to 256 processors. The run-time reduction is lower when going from 256 to 512 processors as all three problems are quite small (even for the larger mesh, F16, there are only about 2150 elements at each processor). In the case of the BEO cluster, a significant reduction in runtime is obtained all the way up to 8 processors. However, when going from 8 to 16 processors, the amount of time required for M6 actually increases. The performance degradation is due to the poor interprocessor interconnection network of the cluster. Despite that, as the problem size increases, better speedup can still be obtained as illustrated by the results obtained for F22, for which the runtime reduced by 32% when going from 8 to 16 processors.

	CRAY T3E			BEO	
	M6	F22	F16	M6	F22
# PES	Time	Time	Time	Time	Time
1	80.66	—	—	32.74	173.61
2	103.17	—	—	46.14	246.66
4	50.43	256.13	—	28.21	153.21
8	22.30	125.21	—	14.18	74.71
16	9.95	61.06	163.14	18.64	50.55
32	4.38	29.71	90.08	—	—
64	2.24	14.86	40.47	—	—
128	1.68	7.11	19.15	—	—
256	1.06	4.52	10.72	—	—
512	1.06	3.55	7.11	—	—

Table 6: Run Times (in seconds) on CRAY T3E and BEO cluster.

6 Conclusions

In this paper we presented serial and parallel algorithms for building coarse grids for geometric multigrid solvers that use the multilevel paradigm. Our results show that this approach leads to coarse grids that have well–shaped

	M6				F22				F16			
	# Iterations				# Iterations				# Iterations			
Technique	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
ML_F1	146	147	149	150	158	159	160	157	353	352	350	344
ML_F2	147	146	148	149	158	157	158	156	341	342	344	355
ML_F3	148	148	150	152	157	156	156	156	339	348	345	354
ML_F3_F2	146	146	147	152	159	158	161	158	338	341	349	357

Table 5: Convergence of multigrid algorithm on 1, 2, 4, 8, and 16, processors.

elements and the corresponding parallel formulation can scale to large number of processors.

Acknowledgements

The authors express their appreciation to Edwin van der Weide, of the DaimlerChrysler Aerospace Military Aircraft in Germany, for making available to us several 3D meshes as well as the source code to check the results of the serial algorithm.

References

[1] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[2] H. Guillard. Node–nested multigrid with delaunay coarsening. Technical Report No 1898, INRIA–Sophia Antipolis, France, 1993.

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

[4] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[5] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1), 1999. Also available on WWW at URL http://www.cs.umn.edu/~karypis. A short version appears in Intl. Conf. on Parallel Processing 1995.

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

[7] G. Karypis, Kirk Schloegel, and V. Kumar. PARMETIS 2.0: Parallel graph partitioning and sparse matrix ordering library. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

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

[9] H. Steve M. Lallemand and A. Dervieux. Unstructured multigridding by volume agglomeration: Current status. *Computers and Fluids, Volume 21*, (3):397–433, 1992.

[10] D.J. Mavriplis. Directional coarsening and smoothing for anisotropic navier–stokes problems. *Electronic Transactions on Numerical Analysis*, (6):182–197, 1997.

[11] D.J. Mavriplis. Three–dimensional high–lift analysis using a parallel unstructured multigrid solver. Technical Report 98-20, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, 1998.

[12] D.J. Mavriplis and S. Pirzadeh. Large scale parallel unstructured mesh computations for 3d high–lift analysis. Technical Report 99-9, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, 1999.

[13] V. Venkatakrishnan D.J. Mavriplis. Agglomeration multigrid for the three–dimensional euler equations. Technical Report 94-5, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, 1994.

[14] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel diffusion algorithms for repartitioning of adaptive meshes. *Journal of Parallel and Distributed Computing*, 47(2):109–124, 1997. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[15] Kirk Schloegel, George Karypis, and Vipin Kumar. Wavefront diffusion and lmsr: Algorithms for dynamic repartitioning of adaptoive meshes. *IEEE Transactions on Parallel and Distributed Systems*, 12(5):451–466, 2001.

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