MULTILEVEL ALGORITHMS FOR GENERATING COARSE GRIDS FOR MULTIGRID METHODS*

Irene Moulitsas and George Karypis

Department of Computer Science & Engineering
University of Minnesota 4-192 CS/EE
200 Union St SE, Minneapolis, MN 55455
karypis, moulitsa@cs.umn.edu

May 3, 2001

*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. Access to computing facilities was provided by the Minnesota Supercomputing Institute.

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 grids is the problem of generating an appropriate set of coarse grids. Even though a number of different agglomerative approaches have been developed for coarse grid construction, there is still a great need for improvement because of the following two reasons. First, existing methods use locally greedy heuristics that often lead to coarse grids whose elements have poor quality (e.g., bad aspect ratios). Second, these algorithms are serial in nature, and they cannot be efficiently parallelized.

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.

2 Previous Works

The most widely used discretization technique on unstructured grids, is the finite volume method on the dual grid (see [12]). Although several variants of the agglomeration technique on such grids exist (see, for example, [4]), the basic approach is the following :

1. Select a starting vertex and list the neighbors;
2. Fuse these neighbors to a new coarse grid control volume;
3. Update the list of neighboring volumes
4. Go to step 2 and repeat until all volumes have been agglomerated

The above algorithm can be applied recursively such that an arbitrary number of coarse grids can be created. This leads to arbitrary control volumes with not any control over the quality of these volumes.

A commonly used improvement to the above algorithm is to build the neighborhood, in step 2, using a greedy algorithm, so that the aspect ratio of the resulting control volume is maximized [17].

3 Serial Multilevel Coarse Grid Construction

The aim of the multilevel coarse grid construction algorithm that we developed is to construct a sequence of coarse grids starting from the original grid. The key difference between our algorithms and the ones that were previously proposed is that each successively coarser grid is generated from the previous one by using the multilevel paradigm to optimize the quality of the fused elements.

In the rest of this section we discuss the various objective functions that we used to measure the overall quality of the generated grids and provide the details of the algorithms. 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 Objectives

In order to ensure fast convergence rates of multigrid algorithms the sequence of coarse grids must contain well shaped elements. A measure of the quality of a cell is its aspect ratio A. In the three-dimensional space the aspect ratio will be defined as

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

where, in this case, S is the surface area, and V the volume, of the control volume respectively.[1]

The aim should be to obtain control volumes that will be as compact as possible. Hence we would like to get control volumes that will be as "spherical" as possible. Therefore we would like to attain aspect ratios that will be as low as possible, i.e. $6\sqrt{\pi}$, which is the aspect ratio for the sphere.

Consequently, the following objective function F_1 can be formulated for the coarse grid cells

$$F_1 = \sum_{i=1}^{NCoarse} \frac{S_i^{3/2}}{V_i} \tag{1}$$

where $NCoarse$ is the number of control volumes on the coarse grid, S_i and V_i are the surface area and volume for control volume i.

One way of formulating the problem of coarse grid construction is to develop an algorithm that optimizes equation (1) subject to the constraints that each fused element will contain at least $Lmin$ and at most $Lmax$ elements of the previous mesh. Alternatively, we can enhance our objective function by allowing it to take into account the weight, that is the number of elements that are fused together, of each control volume i, thus allowing us to focus more on improving the aspect ratios of larger elements. In this case the new objective function will be

$$F_2 = \sum_{i=1}^{NCoarse} w_i \frac{S_i^{3/2}}{V_i}. \tag{2}$$

Potentially, though, objective function F_2 can yield a coarse grid whose overall quality is good, but nonetheless, there still may be a limited number of cells that will have poor quality. This can be avoided by adding another objective in our optimization; that of minimizing the aspect ratio of the worst cell. In particular this is given by

$$F_3 = \max_{i=1\ldots NCoarse} \frac{S_i^{3/2}}{V_i} \tag{3}$$

Experiments (see section 5) have shown that the dual objective of functions F_2 and F_3 tend to yield the best results.

3.2 Overview of the approach

We build a graph from our initial grid as follows: every cell of the grid is represented by a vertex in the graph. An edge between two vertices in the graph signifies that the corresponding grid cells share a common face. For every vertex in the graph we have three values associated with it. These are the vertex weight, the vertex surface and the vertex volume. The vertex weight shows how many cells the current cell is made up of. It is initialized with the value of one. The vertex surface is zero for the cells whose all faces are internal. For these cells which have boundary faces, the surface area of the boundary faces is represented by vertex surface. The vertex volume is the volume of the cell. Every edge in the cell is associated with the edge weight. The edge weight represents the surface area of this cells in the grid.

[1] The aspect ratio can be analogously defined for two–dimensional meshes as well.

In the spirit of graph partitioning methods, our approach consists of two phases; the coarsening phase and the refinement phase. Our coarsening and refining phases are similar in nature to those used by multilevel graph partitioning algorithms (see, for example, [6], [7]). Successive coarser graphs are constructed by computing a matching of the vertices. During refinement a greedy refinement algorithm is used that randomly traverses the vertices and attempts to move them to adjacent cells if such moves improve the objective function subject to the *Lmin* and *Lmax* constraints.

3.3 Globular Agglomeration

The key step of our algorithm is the method used to compute the matching during coarsening. Motivated by earlier research on graph partitioning we use a method called "globular matching" that was inspired by the heavy edge heuristic. In this approach the vertices are visited in decreasing order of their degree. If a vertex has not been matched yet, we match it with one of its adjacent unmatched vertices that will lead to the smallest aspect ratio. Note that this algorithm does not guarantee that the matching obtained has the best properties (over all possible matchings), but our experiments have shown that it works very well in practice. The complexity of computing a globular agglomeration is $O(|E|)$, which is asymptotically similar to that for computing the random agglomeration.

4 Parallel Implementation

In recent years, a number of scalable parallel formulations of multilevel graph partitioning algorithms have been developed (see [9], [8], [16]). 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 lead 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.

4.1 Overview of the approach

The overall structure of our parallel algorithm is shown in Fig. 1. 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 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.

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

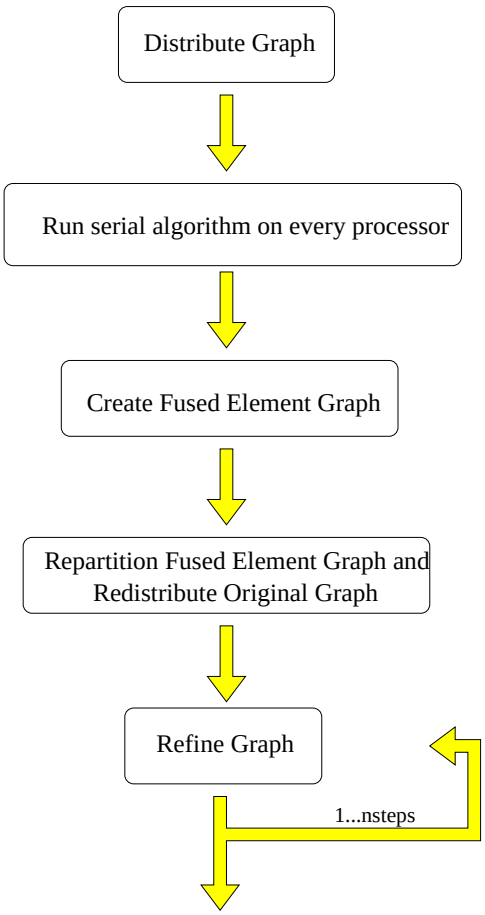

Figure 1: The various phases of the the parallel procedure.

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. 2, 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 ([11]), 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 algorithm on two grids. The first case, M6, is an unstructured mesh with 94493 elements that corresponds to an M6 wing. The second case, F22, is an unstructured mesh, with

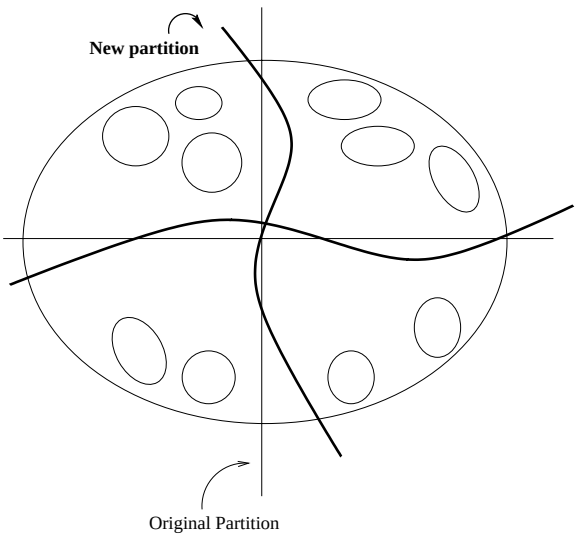

Figure 2: Partition.

428748 elements, corresponding to the wing of an F22 airplane. Both cases are 3D tetrahedral meshes. We have evaluated the performance of our parallel algorithm for constructing coarse grids on a 1024–processor CRAY T3E parallel computer where each processor has 512MB of memory.

5.1 Serial Algorithm Results

We have tested the quality of the grids obtained from our serial algorithm in the simulation of an unsteady flow of moving grids, arising in aero–elasticity problems, using an edge–based multigrid solver.

In Fig. 3 and Fig. 4 we show the convergence of the multigrid method using different techniques for the construction of the coarse grids, for M6 and F22 respectively. These figures show the results for six different algorithms for constructing coarse grids. The results labeled "Trad 1" correspond to the single neighborhood based agglomerative scheme described in section 2. The results labeled "Trad 2" correspond to the neighborhood based agglomerative scheme that takes into account the aspect ratio of the new cells, see [14], [17]). 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 (1), F_2 (2), F_3 (3), and $F_3 + F_2$ objective functions respectively from section 3.1.

From these results we can see that, in general, the multilevel algorithm produces grids that lead to fewer multigrid iterations. The only notable exception is "ML_F3_F2" for F22, in which the multigrid solver fails to converge.

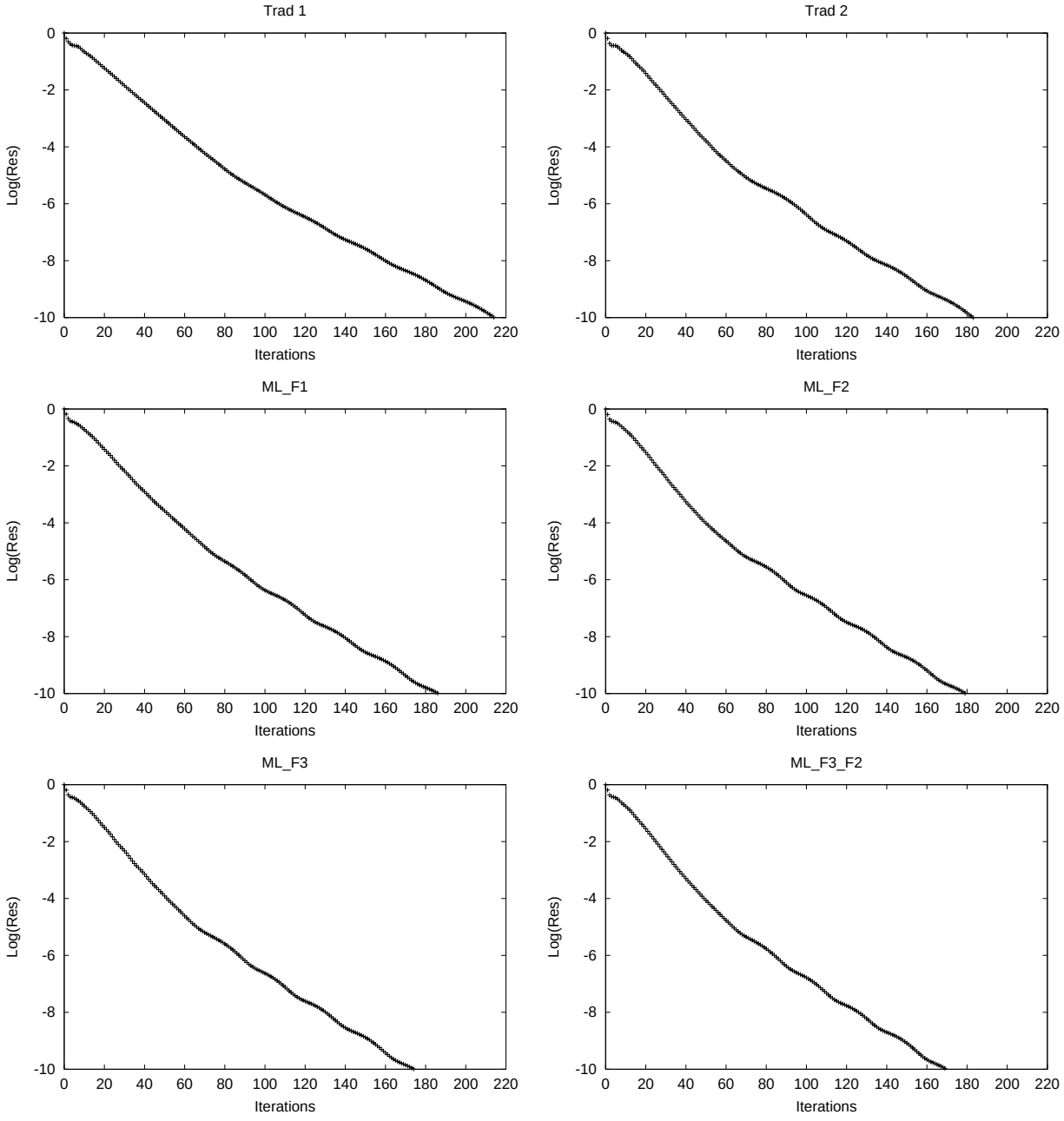

Figure 3: Convergence of multigrid algorithm on M6.

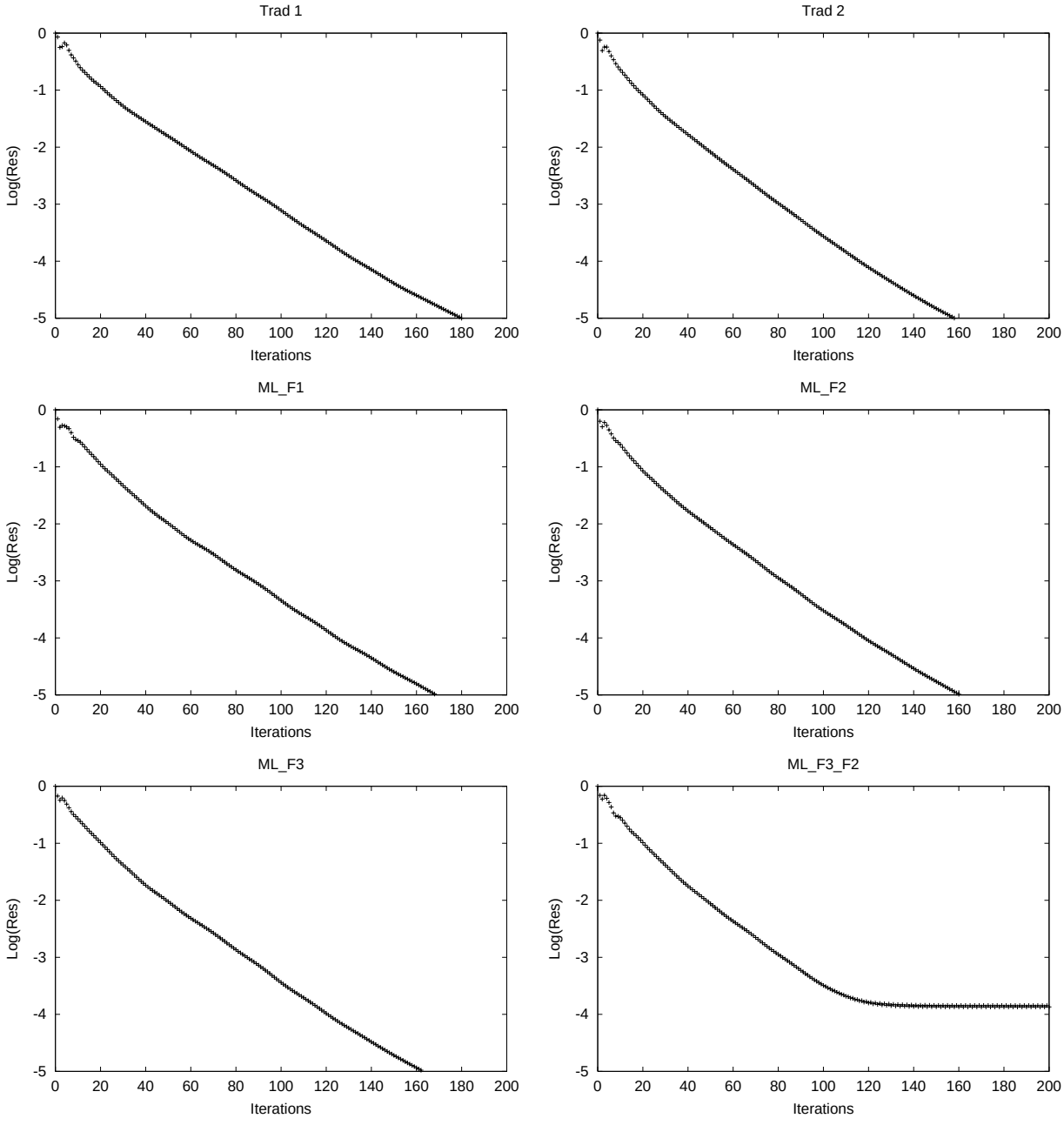

Figure 4: Convergence of multigrid algorithm on F22.

5.2 Parallel Algorithm Results

We evaluated the performance of our parallel formulation of the multilevel coarse grid construction algorithm on the same two unstructured meshes used by the serial algorithm. Since our multigrid solver could only run on an SGI platform we were not able to directly evaluate the quality of the coarse grids in terms of multigrid convergence. For this reason our evaluation was focused on how effective the parallel algorithm is in optimizing the objective function, and on its scalability.

Table 1 shows the quality of the coarse grids produced by our parallel algorithm using 1, 2, 4, 8, 16, 32, 64 and 128 processors for both the M6 and the F22 data sets. These results were obtained using the dual objective function of F_3 and F_2. Some of the results could not be obtained due to the fact that there was not enough memory on small number of processors. From these results we can see that with respect to the F2 objective function, 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 F2 objective 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. In particular $8.207008e+06$ on 128 processors for F22 was $8.288461e+06$ on 1 processor. With respect to the F3 objective function 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 F3 objective 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 1: Quality measures on 1, 2, 4, 8, 16, 32, 64 and 128 processors.

	M6		F22	
processors	Maximum Aspect Ratio	Weighted Sum	Maximum Aspect Ratio	Weighted Sum
1	$2.436233e+01$	$1.834413e+06$	NOT ENOUGH MEMORY	NOT ENOUGH MEMORY
2	$2.331621e+01$	$1.825575e+06$	NOT ENOUGH MEMORY	NOT ENOUGH MEMORY
4	$3.191382e+01$	$1.824597e+06$	NOT ENOUGH MEMORY	NOT ENOUGH MEMORY
8	$2.550975e+01$	$1.821121e+06$	$1.022124e+03$	$8.288461e+06$
16	$2.265289e+01$	$1.815118e+06$	$2.315982e+02$	$8.259121e+06$
32	$2.263738e+01$	$1.809697e+06$	$1.022124e+03$	$8.238999e+06$
64	$2.263738e+01$	$1.803522e+06$	$1.022124e+03$	$8.220569e+06$
128			$1.022124e+03$	$8.207008e+06$
256			$1.781039e+02$	$8.189893e+06$

Finally, table 2 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 Table 1. The single processor run times were obtained by using the serial algorithm. A number of observations can be made from this table. First, looking at the results of M6 (for which we were able to run on single processor), 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 tends to decrease linearly (both for the M6 and the F22). In fact on 64 processors M6 can be coarsened in 1.56 seconds and on 256 processors F22 can be coarsened in 3.58 seconds.

Table 2: Run Times (in seconds)

	M6	F22
processors	Time	Time
1	48.56	NOT ENOUGH MEMORY
2	61.06	NOT ENOUGH MEMORY
4	28.88	NOT ENOUGH MEMORY
8	13.46	79.39
16	6.51	35.92
32	3.25	17.97
64	1.85	8.73
128		4.80
256		3.58

6 Conclusions

In this paper we presented serial and parallel algorithms for building coarse grids, in the context of multigrid solvers, that use the multilevel paradigm. Our results show that this approach leads to coarse grids that have well–shaped elements and the corresponding parallel formulation can scale linearly to large number of processors.

References

[1] M. Adams, *Heuristics for the Automatic Construction of Coarse Grids in Multigrid Solvers for Finite Element Problems in Solid Mechanics.*, Technical Report UCB//CSD-98-994, University of California, Berkeley, (1998).

[2] T.F. Chan and B.F. Smith, *Domain Decomposition and Multigrid Algorithms for Elliptic problems on unstructured meshes.*, Electronic Transactions on Numerical Analysis, Volume 2, (1994), pp. 171-182.

[3] C.M. Fiduccia and R.M. Mattheyses, *A linear time heuristic for improving network partitions.*, In Proc. 19th IEEE Design Automation Conference (1982), pp. 175-181.

[4] H. Guillard, *Node–nested multigrid with Delaunay coarsening.*, Technical Report INRIA–Sophia Antipolis No 1898, France (1993).

[5] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs.*, Technical Report SAND93-1301, Sandia National Laboratories (1993).

[6] G. Karypis and V. Kumar, *A fast and highly quality multilevel scheme for partitioning irregular graphs.*, SIAM Journal on Scientific Computing.

[7] G. Karypis and V. Kumar, *Multilevel k-way partitioning scheme for irregular graphs.*, Journal of Parallel and Distributed Computing

[8] G. Karypis and V. Kumar, *Parallel multilevel k-way partitioning scheme for irregular graphs.*, Supercomputing (1996).

[9] G. Karypis and V. Kumar, *A coarse-grain parallel formulation of a multilevel k-way graph partitioning algorithm.*, 8th SIAM Conference on Parallel Processing for Scientific Computing.

[10] B.W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal 49(2) (1970), pp. 291-307.

[11] G. Karypis, K. Schloegel, and V. Kumar, *ParMetis Parallel graph partitioning and sparse matrix ordering library.*, Department of Computer Science and Engineering and Army HPC Research Center (1998).

[12] M. Lallemand, H. Steve, and A. Dervieux, *Unstructured Multigridding by Volume Agglomeration: Current Status.*, Computers and Fluids, Volume 21, No. 3, (1992), pp. 397-433.

[13] D.J. Mavriplis, *Directional Coarsening and Smoothing for Anisotropic Navier–Stokes Problems.*, Electronic Transactions on Numerical Analysis, Volume 6, (1997), pp. 182-197.

[14] D.J. Mavriplis, *Three–Dimensional High–Lift Analysis Using a Parallel Unstructured Multigrid Solver.*, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, Technical Report 98-20 (1998).

[15] D.J. Mavriplis and S. Pirzadeh, *Large Scale Parallel Unstructured Mesh Computations for 3D High–Lift Analysis*, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, Technical Report 99-9 (1999).

[16] K. Schloegel, G. Karypis and V. Kumar, *Parallel multilevel algorithms for multi–constraint graph partitioning.*, Department of Computer Science and Engineering, Technical Report 99-031 (1999).

[17] V. Venkatakrishnan D.J. Mavriplis, *Agglomeration Multigrid for the three–dimensional Euler equations.*, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, Technical Report 94-5 (1994).