Preprint
UCRL-JC-147742

Parallelization of an Adaptive Multigrid Algorithm for Fast Solution of Finite Element Structural Problems

N. K. Crane, I. D. Parsons, K. D. Hjelmstad

This article was submitted to
International Conference on Computational Engineering and Sciences, Reno NV., July 31- August 2, 2002

U.S. Department of Energy
Lawrence Livermore National Laboratory

March 21, 2002

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Parallelization of an Adaptive Multigrid Algorithm for Fast Solution of Finite Element Structural Problems

N. K. Crane[1], I. D. Parsons[2] and K. D. Hjelmstad[3]

Summary

Adaptive mesh refinement selectively subdivides the elements of a coarse user supplied mesh to produce a fine mesh with reduced discretization error. Effective use of adaptive mesh refinement coupled with an a posteriori error estimator can produce a mesh that solves a problem to a given discretization error using far fewer elements than uniform refinement. A geometric multigrid solver uses increasingly finer discretizations of the same geometry to produce a very fast and numerically scalable solution to a set of linear equations. Adaptive mesh refinement is a natural method for creating the different meshes required by the multigrid solver. This paper describes the implementation of a scalable adaptive multigrid method on a distributed memory parallel computer. Results are presented that demonstrate the parallel performance of the methodology by solving a linear elastic rocket fuel deformation problem on an SGI Origin 2000.

Two challenges must be met when implementing adaptive multigrid algorithms on massively parallel computing platforms. First, although the fine mesh for which the solution is desired may be large and scaled to the number of processors, the multigrid algorithm must also operate on much smaller fixed-size data sets on the coarse levels. Second, the mesh must be repartitioned as it is adapted to maintain good load balancing. In an adaptive multigrid algorithm, separate mesh levels may require separate partitioning, further complicating the load balance problem. This paper shows that, when the proper optimizations are made, parallel adaptive multigrid algorithms perform well on machines with several hundreds of processors.

Adaptive Mesh Refinement

Adaptive mesh refinement is a well-known technique for using a posteriori error estimation to automatically produce a non-uniform mesh from a relatively coarse user defined mesh (e.g., [1], [2]). In this paper, h-refinement is employed, in which a low order element, such as an eight node hexahedral element, is subdivided into a number of similar elements. Both isotropic and anisotropic refinement is employed, whereby the error estimator dictates if the original hexahedral element is to be split into eight (for isotropic refinement) or less (for anisotropic refinement) new elements. Linear constraints are introduced on each refined mesh to maintain the required degree of interelement continuity.

To demonstrate the approach, consider Figure 1. The mesh shown on the left of the figure is a section of a coarse star-grain rocket fuel mesh. The entire rocket fuel grain is a long cylinder; a star shaped hole is cut into the cylinder to increase the initial exposed internal surface area. The structural loading on the fuel consists of a large internal pressure and an acceleration loading. The rocket fuel is modeled with first-order hexahedral finite elements and linear elastic material properties. It is desired to find the static displacements and stresses in the fuel to some user specified discretization error

[1] University of Illinois, Urbana, IL 61801; n-crane@uiuc.edu
[2] Lawrence Livermore National Laboratory, Livermore, CA 94551; parsons14@llnl.gov
[3] University of Illinois, Urbana, IL 61801; kdh@uiuc.edu

tolerance. The coarse mesh is solved and the error estimated using an a posteriori error estimator proposed by Zienkiewicz and Zhu [3]. Elements with high error are refined to produce the new mesh shown in the middle of Figure 1. The problem is solved on the new mesh and the error estimated. The new mesh is refined to produce a yet finer mesh shown on the right of Figure 1. This process continues until the user specified discretization error tolerance is reached. For the rocket fuel problem, an adaptively refined mesh can yield a discretization error tolerance of 1% using about 4,000,000 elements. Uniform refinement requires eight times as many elements to yield the same quality of solution.

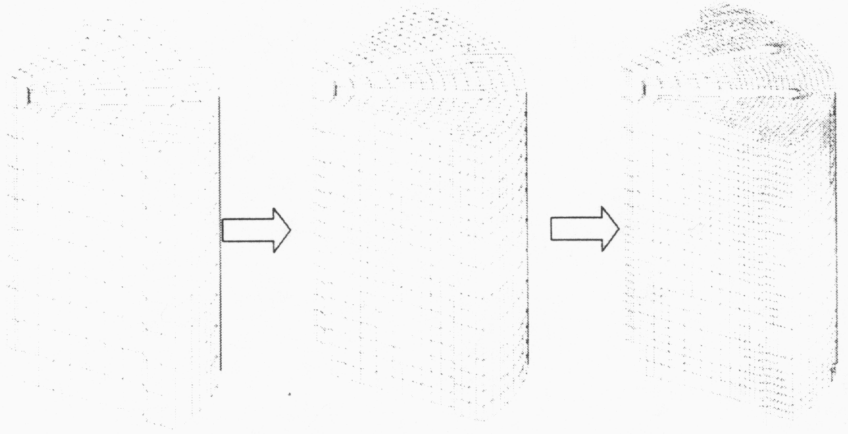

Figure 1: User supplied coarse and adaptively refined meshes.

The Geometric Multigrid Method

A geometric multigrid solver uses multiple refinements of the same coarse mesh to accelerate the solution of a system of equations [4]. Let the meshes shown on the left, center and right of Figure 1 represent the coarse, the intermediate and the fine meshes in a multigrid sequence, respectively. These meshes will be denoted as meshes 1, 2 and 3, respectively. It is required to solve the system of equations produced by the finite element discretization of mesh 3. The coarser meshes are used to accelerate this solution process via a multigrid V-cycle shown in Figure 2.

Depending on the problem, between one and ten multigrid V-cycles are usually required to achieve an accurate solution to the fine mesh equations. The relaxation steps use a few cycles of conjugate gradient (CG) relaxation to improve the solution on a given mesh. The rocket fuel problem uses 6 CG cycles in each relaxation step. The coarse mesh solution step uses CG relaxation as a solver. The number of iteration cycles to solve the coarse mesh rocket fuel problem is about 200. The restriction step restricts the residual (i.e., forces) from a fine mesh to the next coarser mesh. The interpolation step interpolates a solution (i.e., displacements) on a coarse mesh to the fine mesh.

The advantage of a multigrid solver is that the computational work is $O(N)$, where N is the number of fine mesh unknowns. An iterative solver such as the CG method is typically $O(N^{4/3})$, while a direct factorization method is generally around $O(N^{7/3})$ for three dimensional problems. Using an adaptive multigrid solver to compute the static solution of a 4,000,000-element mesh rocket fuel problem on a

single processor of an Origin 2000 takes approximately 7 hours. Solving the same problem using the CG method as a solver requires 192 hours, about 27 times as long.

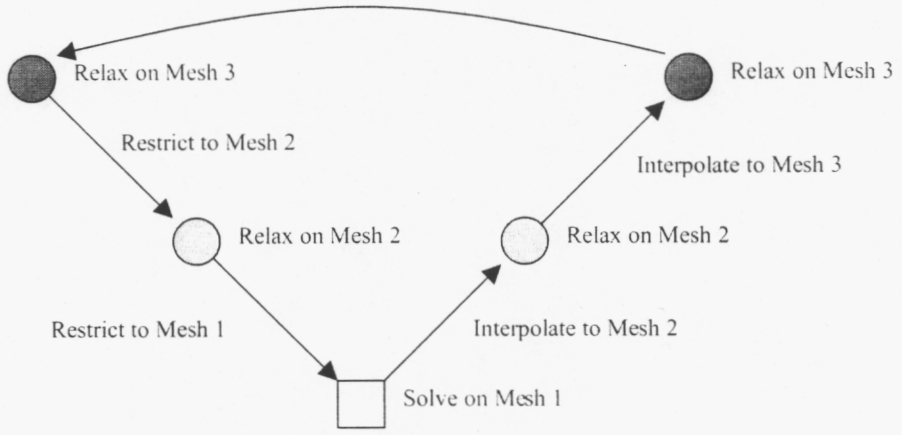

Figure 2: Schematic of a 3 mesh multigrid V-cycle.

Parallelization of an Adaptive Multigrid Algorithm

The majority of the computational work in a multigrid cycle is spent performing the relaxations on the various mesh levels. The serial timings for an example multigrid solution cycle are shown in Table 1. For the serial case, the total cycle time is dominated by the fine mesh relaxation step. The coarse mesh solution phase of the cycle requires only about 2% of the total solution time.

Mesh Number	Number of Elements	Number of Relaxation Cycles	Serial Time (Seconds)
1	1,440	200	28.9
2	10,801	6	6.42
3	70,282	6	40.9
4	388,339	6	228
5	1,747,098	6	1040

Table 1: Serial multigrid cycle timings.

Figure 3 shows CG relaxation speed-up curves for the different mesh sizes considered in Table 1. Two important trends are evident in this figure. First, the larger meshes of the multigrid sequence produce much better speed-ups than the smaller meshes. This phenomenon can be explained by considering a solid cube with cube-shaped mesh partitions. Let a coarse mesh of this region have 64 elements and be partitioned onto 64 processors. The computational workload of each partition is proportional to the number of elements, 1. The communication load for each partition is proportional to the number of boundary nodes, 8. A fine mesh of this solid has a total of 32,758 elements partitioned with 512 elements per processor. The computational workload for each partition is 512; the number of boundary nodes is 386. For this example the coarse mesh has a computation to communication ratio of 0.125; the fine mesh a computation to communication ratio of 1.33. Thus, the coarse meshes must bear more communication load per computational effort than the fine meshes.

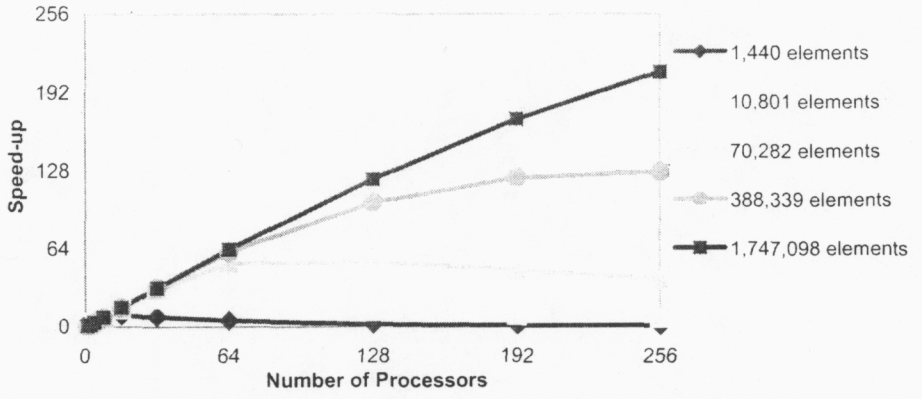

Figure 3: CG relaxation cycle speed-up curves on Origin 2000 by mesh size.

Second, on the coarse meshes the algorithm reaches a point where adding more processors not only produces little additional speed-up, but actually has a detrimental effect. This effect is caused by increasing communication latency when using larger sets of processors. As more processors are used, the processors needed by the computation are physically farther apart. Thus, as more and more processors are used, the latency of communicating between processors increases. The postulate that relaxation slow-down is caused by increasing message latency is reinforced by the observation that when using the SGI Origin 2000 a significant performance reduction occurs when going from 64 to 65 or more processors. The Origin is constructed from boxes containing 64 processors each. Once more than 65 processors are used, inter-box communication is required, which has a much higher latency compared to intra-box communication.

Using a reduced set of processors for the coarse mesh relaxations brings the communication to computation ratio back into balance and reduces the detrimental effect of communication latency. Using a reduced set of processors deliberately allows some of the available processors go idle during the coarse mesh computations. Let E denote the number of elements on a mesh. Based on Origin 2000 timing runs, the optimal number of processors P_0 to use is approximately

$$P_0 = 0.5\sqrt{E}. \tag{1}$$

When using the optimal number of processors, the maximum obtainable speed-up is

$$SU_0 = 0.25\sqrt{E}. \tag{2}$$

Table 2 shows how the multigrid mesh sequence described in Table 1 performs in parallel on 256 processors. The relative contribution of the coarse mesh solve time is much greater than in the serial case. The coarse mesh solution requires about30% of the total solution time in parallel versus only 2% in the serial case. Addition of more processors will continue to improve speed-up on the fine mesh, but will have a reduced effect on the performance of the coarse meshes. Thus, the overall parallel performance of a multigrid code is often strongly dependent on the performance on coarse mesh operations.

Mesh Number	Number of Elements	Maximum Obtainable Speed-up	Parallel Time (Seconds)
1	1,440	9	3.21
2	10,801	26	0.251
3	70,282	66	0.620
4	388,339	128	1.78
5	1,747,098	208	5.14

Table 2: Parallel multigrid cycle timings for 256 processors.

The second obstacle to obtaining good speed-up in a parallel adaptive multigrid code is achieving good load balance on each of the multigrid meshes. A mesh with good load balance is partitioned such that the computational work across all processors is balanced.Fo r the case of CG relaxations, good load balance occurs when all processors own approximately the same number of elements. A sample multigrid mesh hierarchy partitioned on two processors is shown in Figure 4. The shaded elements are given to processor 1, the white elements to processor 2. The mesh partitioning shown has good load balance on mesh 3. However, using the same spatial partitions on mesh 1 and mesh 2 yields poor load balance on the mesh 1 and mesh 2 operations.

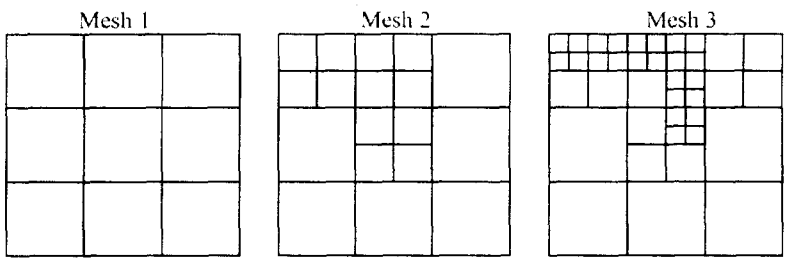

Figure 4: Example partitioning using the same spatial partitions on each mesh.

A good solution to the multi-mesh load balance problem is multi-partitioning. A multi-partition algorithm creates a separate partition for each mesh. Each mesh's partition balances the load and minimizes the communication for that mesh only. Multi-partitioning requires additional communication during the interpolation and restriction phases of the multigrid cycle. During these phases data must be transferred from the old partitioning layout to the new partitioning layout. Communication (represented by gray lines) is shown for the multigrid solution of a multi-partitioned mesh in Figure 5. Partition boundary communication is required in the mesh relaxation steps and partition change communication is required in the interpolation and restriction steps. ParMETIS is used for mesh partitioning to produce partitions which simultaneously have good load balance, minimalbou ndary communication, and are as similar to one another as possible (to reduce data transfer volume in the interpolation and restriction steps.)

Results

When using reduced processor sets on the coarse meshes and multi-partitioning, good speed-ups can be obtained for a parallel adaptive multigrid solver. The total code speed-up for a 4,000,000 element static deflection rocket fuel problem is shown in Figure 6. The speed-ups are for the entire code consisting of error estimation, adaptive refinement and multigrid solution steps. On 256 processors, the 4,000,000 element problem is solved in 150 seconds, which is a speed-up of 190 over the serial time.

Although the speed-up of the multigrid solver (190) is not as good as a CG solver would likely be (255), the numerical efficiency of the multigrid solver (27 times faster than CG for this mesh) makes the overall solution time for the multigrid solver much less. The overall code speed-up gains from adaptive refinement, multigrid solution and parallel execution are multiplicative. Adaptive refinement allows solving a problem with a smaller mesh, multigrid allows solving that mesh very efficiently, and parallelization substantially accelerates the multigrid solver. Thus, parallel adaptive multigrid can be used to produce very fast solutions to structural problems.

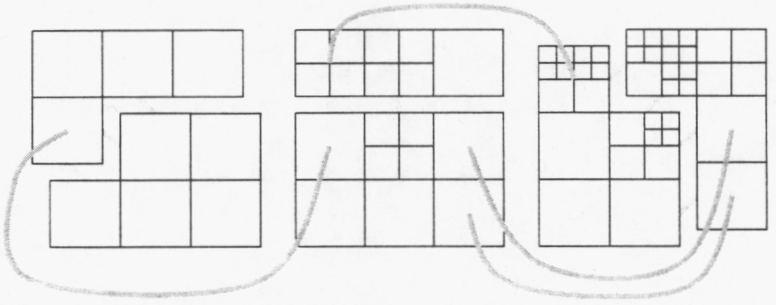

Figure 5: communication in a multi-partitioned mesh multigrid cycle.

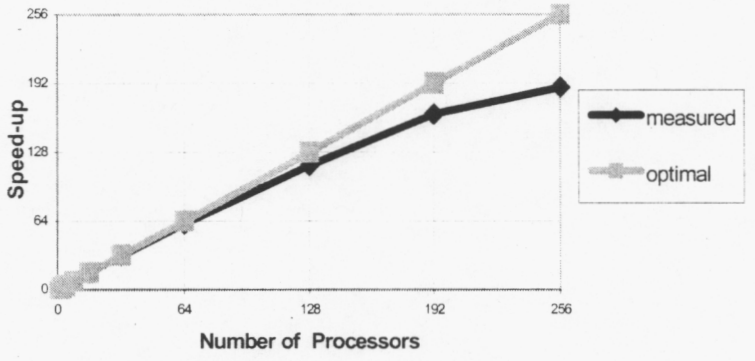

Figure 6: Parallel adaptive multigrid speed-up for 4,000,000 element rocket fuel mesh

References

1 Johnson, C., 1997. *Numerical Solution of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, Cambridge-Lund.

2 Babuska, I. and Rheinboldt, W. C., 1980. Reliable error estimation and mesh adaptation for the finite element method. *Computational Methods in Nonlinear Mechanics, Proceedings of the TICOM Second International Conference*, pp 67-108, Amsterdam, Netherlands 1980.

3 Zienkiewicz, O. C. and Zhu, J. Z., 1987. A simple error estimator and adaptive procedure for practical engineering analysis. *International Journal for Numerical Methods in Engineering*, 24, pp. 337-357.

4 Trottenberg, U., 2000. *Multigrid*, Academic Press.