AIAA JOURNAL
Vol. 40, No. 3, March 2002

Improvements in Parallel Chimera Grid Assembly

Nathan C. Prewitt*
Mississippi State University, Mississippi State, Mississippi 39759
Davy M. Belk†
U.S. Air Force Research Laboratory, Eglin Air Force Base, Florida 32542
and
Wei Shyy‡
University of Florida, Gainesville, Florida 32611

Parallel implementation and performance assessment of the grid assembly function in a chimera grid approach, realized by the Beggar code, are presented. A mixed programming model is used combining message passing with shared-memory programming constructs using standard POSIX calls. The POSIX calls are used to store large, common data structures in shared memory. The associated reduction in the total memory requirements allows more processors to be used. The effective utilization of this increased processor count is based on improvements in load balancing because of a finer decomposition of the work associated with the grid assembly function. Parallel efficiency is demonstrated using a three-store, ripple release, trajectory calculation.

Introduction

THE chimera grid scheme[1] (i.e., the use of overlapping, structured grids for domain decomposition) allows the calculation of moving-body, fluid dynamics problems using time-accurate computational fluid dynamics (CFD). Thus, chimera has seen application in the analysis of store separation,[2] tilt-rotor aircraft aerodynamics,[3] and many other areas. However, an expensive grid assembly process, involving hole cutting and interpolation, is required to establish communication between the overlapping grids. And the expense increases further with the time-accurate solution of moving-body problems because the grid assembly has to be updated after each time step.

Numerous attempts have been made to create robust and efficient algorithms for handling the grid assembly problem. The number of algorithms that have been applied to dynamic, moving-body problems is somewhat smaller. The number of parallel implementations for dynamic problems is fewer still.

Barszcz et al.[4] and Wissink and Meakin[5] have performed the parallel solution of many dynamic problems using chimera methods. However, their decomposition of the work related to the grid assembly function was based on the same decomposition used for balancing the work of the flow solver. Modifications in the work distribution of the grid assembly lessened the parallel performance of the flow solver and, thus, the parallel performance of the overall simulation.[5]

Prewitt et al.[6] performed the first parallel, chimera CFD solution of a dynamic problem that used a data decomposition of the grid assembly separate from that of the flow solver. However, large load imbalances were seen in the grid assembly function, and the relatively small number of grids used in the test problem did not offer much opportunity for scalability. This work presents the improvements in load balancing offered by using a fine-grain decomposition of the work associated with grid assembly.

Received 16 April 2001; revision received 12 August 2001; accepted for publication 14 September 2001.

*PET Onsite CFD Lead, U.S. Department of Defense Engineer Research and Development Center, Major Shared Resource Center. Senior Member AIAA.

†Technical Advisor, Computational Mechanics Branch, Assessments and Demonstrations Division. Associate Fellow AIAA.

‡Professor and Chairman, Department of Aerospace Engineering, Mechanics and Engineering Science. Fellow AIAA.

Grid Assembly

Two major steps occur in the grid assembly function: hole cutting and the stencil search. Holes are cut into overlapping grids to mark cells that intersect solid boundaries (rather than conforming to them) as invalid. This creates hole boundaries that require some sort of applied boundary condition. The points on the fringe of these hole boundaries and the points on the outer boundaries of any embedded grids are collectively referred to as intergrid boundary points (IGBPs). Boundary conditions are supplied at the IGBPs by interpolating the flow solution from any overlapping grids. Finding appropriate interpolation stencils is the goal of the stencil search.

In this work the concern is with the grid assembly algorithms used in the Beggar code.[7] Beggar is capable of solving three-dimensional inviscid and viscous flow problems involving multiple moving objects. It allows blocked, patched, and overlapping structured grids in a framework that includes grid assembly, flow solution, force and moment calculation, and the integration of the rigid body, six degree-of-freedom equations of motion.

For cutting holes Beggar uses an outline and fill algorithm. The facets of the hole-cutting surfaces are used to create an outline of the hole, and then the hole is filled by a fast sweep through the grid. The algorithm first marks the cells surrounding the vertices of a hole-cutting facet as being on the hole side or the world side of the facet. Then the facet is recursively refined in order to ensure a complete outline of the hole. Each of the hole-cutting facets is treated independently. The only restriction on the hole-cutting facets is that they form completely closed surfaces.

Parallel Implementations

Beggar has gone through an evolutionary process of parallel-processing development. Single-program, multiple-data and multiple-program, multiple-data programming models have been used along with the message-passing paradigm. Implementations have used both parallel virtual machine (PVM) and message passing interface (MPI) for the message-passing interface.

Belk and Strasburg[8] developed the first parallel implementation. The grid assembly was performed as an initialization step, and only the flow solver was executed in parallel. Thus, this implementation was only applicable to static problems.

The extension to solving dynamic problems in parallel was accomplished in several phases based on the parallel implementation of the grid assembly function and its integration with the parallel implementation of the flow solver. Prewitt et al.[9] presents a complete review of this development process.

Prewitt et al.[10] presented the first implementation, phase I, which used a single front-end (FE) process to perform the grid assembly in a serial fashion with respect to the parallel execution of the flow solver across multiple backend (BE) processes. Proper communication between the flow solver and the grid assembly function was required; however, no consideration was given to load balancing or partitioning of the grid assembly function.

In the phase II implementation Prewitt et al.[10] showed an improvement in the parallel efficiency by hiding the execution time of the grid assembly function behind that of the flow solver. This was possible because of the Newton-relaxation scheme used in the flow solver (complete details on the flow solver can be found in the literature; see Whitfield,[11] for example).

As the number of BE processes increases, the time to compute the flow solution, and therefore the time available for hiding the work of the grid assembly, decreases. To continue to see the optimum speedup, multiple processes must be used to decrease the total execution time of the grid assembly function.

In the phase III implementation Prewitt et al.[6] distributed the work of the grid assembly across multiple FE processes using data decomposition techniques. The basis for decomposition was the grids or groupings of grids called *superblocks*. The grid assembly execution times were monitored, and the superblocks were migrated between FE processes in order to dynamically improve the balance of the grid assembly workload.

This implementation showed significant improvements over the preceding implementations; however, its performance was hampered by a poor load balance and limited scalability in the grid assembly function because of the use of superblocks as the basis for domain decomposition. The load balancing and scalability can be improved by using a fine-grain decomposition of the work associated with the grid assembly.

The work of the grid assembly function is associated with the number of hole-cutting surface facets, the number of cells that are cut, and the number of cells that require interpolation. In this paper, the hole-cutting facets are used as the basis for decomposition of the hole cutting. This is done to demonstrate the improvements in performance and scalability that are possible with a fine-grain decomposition. Of course, hole cutting is only half of the work of the grid assembly. The other half, the stencil search, is more difficult to implement using a fine-grain decomposition (e.g., based on IGBPs); therefore, the stencil search portion of the grid assembly function is still decomposed based on the mapping of superblocks to FE processes.

Parallel Performance

If one considers the entire workload of a complete simulation to be broken down into a part that can be executed in parallel and a part that must be executed serially, the maximum speedup that can be achieved is determined by Amdahl's law[12]:

$$s \le 1/[(f_p/npes) + f_s] \tag{1}$$

where s is the speedup, f_s is the serial fraction of the work, f_p is the parallel fraction of the work, and $npes$ is the number of processors on which the parallel portion of the code is running.

Generalizing Eq. (1) to include the effect of a load imbalance, the speedup becomes

$$s \simeq 1/[(f_p/npes) \times f_i + f_s] \tag{2}$$

where f_i is the load imbalance factor. This is a scale factor that is used to increase the average parallel work per processor to represent the maximum work on any processor.

The phase I implementation fits this model of performance very well. The flow solver represents the parallel work, whereas the grid assembly represents the serial work. Thus, for the phase I implementation the speedup can be approximated from Eq. (2) as

$$s \simeq 1/[(F_p/nbes) \times F_i + G_s] \tag{3}$$

where F_p is the parallel fraction of the work represented by the flow solver, F_i is the load imbalance factor from the distribution of the flow solver, $nbes$ is the number of BE processes, and G_s is the serial fraction of the work represented by the grid assembly.

For the phase II implementation one must take into account the fact that the execution time of the grid assembly can be hidden by the execution time of the flow solver. If the fraction of time spent in the flow solver after the first Newton step, or dt iteration, is

$$F_t = \left(\frac{F_p}{nbes}\right) \times F_i \times \left(\frac{ndt - 1}{ndt}\right) \tag{4}$$

where ndt is the number of dt iterations being run per time step, then the speedup can be approximated by the equation

$$s \simeq 1/[(F_p/nbes) \times F_i + G_t] \tag{5}$$

where

$$G_t = \begin{cases} G_s - F_t & \text{if } F_t < G_s \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

If F_t is greater than G_s, the time to do the grid assembly is completely hidden by the flow solver. If the time to compute the grid assembly is only partially hidden by the flow solver, the speedup is degraded by the portion of the grid assembly process that is not hidden.

For the phase III implementation the grid assembly is executed in parallel; therefore, the grid assembly time is

$$G_p/nfes \times G_i \tag{7}$$

where G_p is the parallel fraction of the work represented by the grid assembly function, G_i is the load imbalance in distributing the grid assembly work, and $nfes$ is the number of FE processors that are executing the grid assembly function. Functional overlapping of the flow solver and the grid assembly function is still being used; therefore, the speedup is still estimated using Eq. (5) with

$$G_t = \begin{cases} G_p/nfes \times G_i - F_t & \text{if } F_t < G_p/nfes \times G_i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

With the change in the basis for the decomposition of the hole-cutting work, the grid assembly time should be broken down into hole-cutting time and stencil-searching time for the purpose of performance modeling so that separate load imbalance factors can be identified. However, Eqs. (5) and (8) will continue to be used. Any improvement in the overall performance is as a result of improvements in the load balance of the hole cutting.

Results

The generic store, ripple release case is being used for collecting timing information. Three stores are ejected from a triple ejector rack configuration in bottom, outboard, inboard order with 0.04 s between releases. A sense of the motion and the overlapping grid system can be obtained from Fig. 1. The complete calculation consists of 600 time steps of 0.0005 s, representing 0.3 s of the trajectory. Two dt iterations and six inner iterations are used per time step of the flow solver. This problem was first presented by Thoms and Jordan.[13] Complete details of the problem definition can be found in Ref. 13.

All solutions were run in double precision (64 bit) on an SGI Origin 2000 machine. This particular machine was configured with 64–195-MHz R10000 processors and 16 GB of distributed, shared memory with two processors and 512 MB of memory per node card.

A single-processor simulation was computed to establish the base solution time of 9384 min (about 6.5 days). Comparing the execution time of the grid assembly function to the total execution time from the sequential run, work fractions of $G_p = 0.05$ and $F_p = 0.95$ were established.

All of the phase III runs used four FE processes for grid assembly, with dynamic load balancing performed after each iteration. Performance data, originally presented by Prewitt et al.,[6] are shown in Fig. 2. Two estimated speedup curves are shown. One is for nominal load imbalance factors of $F_i = 1.05$ and $G_i = 1.08$. This represents the level of performance that was anticipated. However, the grid assembly experienced a much larger load imbalance, and the overall

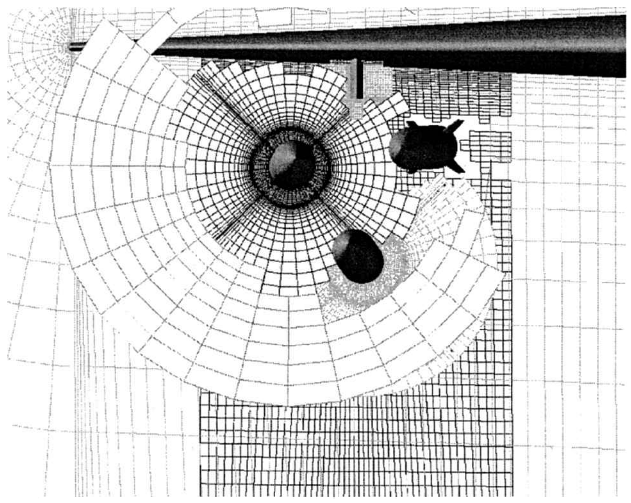
Fig. 1 Three-store, ripple release overlapping grid system with holes.

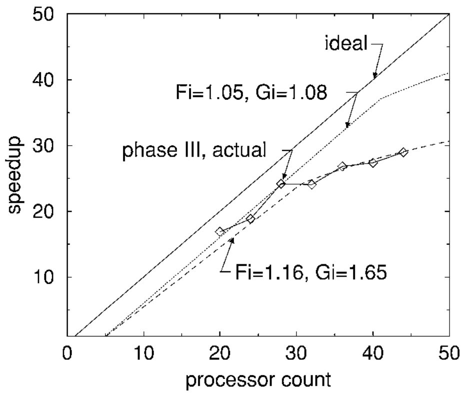

Fig. 2 Measured speedups for the phase III runs.

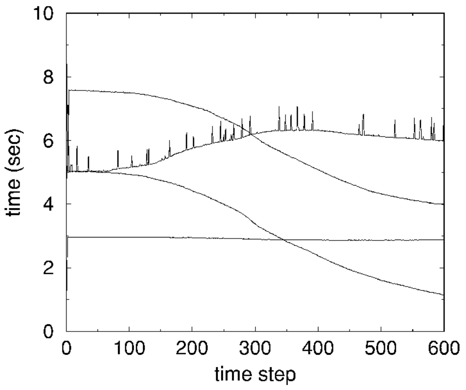

Fig. 3 Detailed execution times for hole cutting for the 40 + 4 phase III run.

performance suffered. The second estimated speedup curve shows that the actual imbalance in the grid assembly is about 65%.

The iterative algorithm used by the grid assembly function requires several synchronization points to ensure that each process has access to the proper cell state information (i.e., iblanking information). The large imbalance is caused by this need for synchronization and the limited number of superblocks with which to decompose the workload.

Figure 3 shows the evolution of the execution times for hole cutting from the 40 + 4 (40 BE and 4 FE processes) phase III run. Each curve represents the execution time vs the time-step number for the work done on an FE process. If the workload was well balanced, the FE processes would be executing for nearly equal times. However, the separation in these curves shows that considerable differences exist in the execution time of each process at any given time step.

There are about 60,000 hole-cutting facets and only 10 superblocks in this particular test case. Therefore, the use of the hole-cutting facets as a basis for decomposition adds flexibility for distributing the workload. The use of shared memory to store the cell state information also reduces communication and the need for synchronization.

Figure 4 shows plots of the hole-cutting execution times vs the time-step number when the hole-cutting facets were used as the basis for decomposition of the hole-cutting workload. The plots represent runs on four and eight FE processes, respectively. These figures contain separate curves for each of the FE processes. The tight grouping of the curves shows the excellent load balance experienced by the hole-cutting function. A direct comparison of the two plots in Fig. 4 shows that the execution time decreased by a factor of two. This, again, is ideal behavior because the number of FE processes was doubled.

During the initial time steps and at several points during the simulation, noise appears to be in the timing data. These are variations in the execution times of different FE processes indicating some load

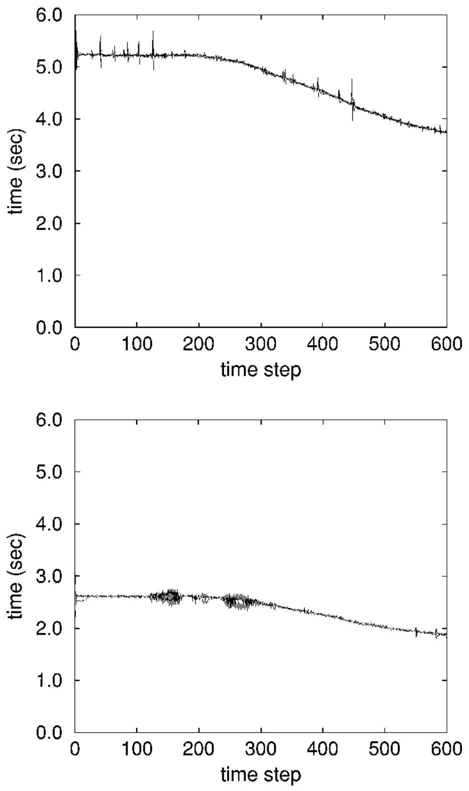

Fig. 4 Measured execution times for hole cutting on four (top) and eight (bottom) FE processes with a decomposition based on hole-cutting facets.

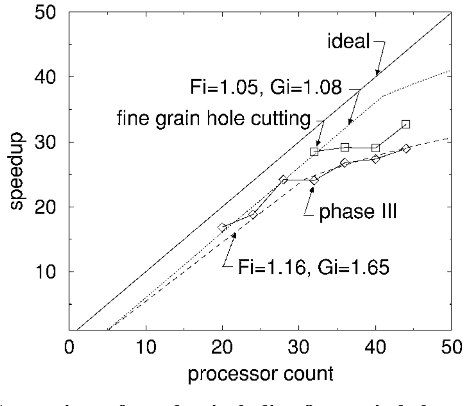

Fig. 5 Comparison of speedup including fine-grain hole cutting with dynamic load balancing.

imbalance. However, the dynamic load balancing routine quickly drives the execution times back toward the ideal load balance.

Figure 5 shows the speedup for the entire simulation when a fine-grain decomposition of the hole-cutting workload is used along with dynamic load balancing. The additional data are for runs on 4 FE processes and 28, 32, 36, and 40 BE processes. Only slight improvements in the overall speedup are seen because only the load imbalance of the hole cutting is being improved. A fine-grain decomposition of the stencil search must be included in order to reap the full benefits.

Conclusions

Execution timing data for the ripple release test problem were presented to show the performance improvements seen by using a fine-grain decomposition of the work associated with hole cutting, as opposed to a coarse-grain decomposition based on the superblocks. This implementation used shared memory to store the cell state information, thereby reducing the communication cost and the requirements for synchronization. A dynamic load balancing algorithm was also used to keep the load balance of the hole-cutting work near ideal.

Several equations were derived from Amdahl's law for use in modeling the performance of the different parallel implementations of the grid assembly function. These equations proved effective in approximating the speedup to be gained and are of great utility in deciding the optimum number of processors on which to run a job.

Acknowledgments

This work was performed under the funding and oversight of the U.S. Air Force Seek Eagle Office. Support was provided in part by a grant of computer time from the U.S. Department of Defense High Performance Computing Modernization Program at the U.S. Army Space and Strategic Defense Command Distributed Center, Aeronautical Systems Center Major Shared Resource Center, and Engineer Research and Development Center Major Shared Resource Center. Additional support was provided by the Air Force Office of Scientific Research, Air Force Materiel Command, U.S. Air Force, under Grant F499620-98-1-0092. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1]Benek, J. A., Steger, J. L., and Dougherty, F. C., "A Flexible Grid Embedding Technique with Applications to the Euler Equations," AIAA Paper 83-1944, June 1983.

[2]Lijewski, L. E., and Suhs, N., "Time-Accurate Computational Fluid Dynamics Approach to Transonic Store Separation Trajectory Prediction," *Journal of Aircraft*, Vol. 31, No. 4, 1994, pp. 886–891.

[3]Meakin, R. L., "Moving Body Overset Grid Methods for Complete Aircraft Tiltrotor Simulations," *AIAA Computational Fluid Dynamics Conference*, AIAA, Washington, DC, 1993, pp. 576–588.

[4]Barszcz, E., Weeratunga, S. K., and Meakin, R. L., "Dynamic Overset Grid Communication on Distributed Memory Parallel Processors," AIAA Paper 93-3311, July 1993.

[5]Wissink, A. M., and Meakin, R. L., "On Parallel Implementations of Dynamic Overset Grid Methods," *SC97: High Performance Networking and Computing* [CD ROM], San Jose, CA, Nov. 1997, URL: http://www.supercomp.org/sc97/proceedings/TECH/WISSINK/INDEX.HTM.

[6]Prewitt, N. C., Belk, D. M., and Shyy, W., "Distribution of Work and Data for Parallel Grid Assembly," AIAA Paper 99-0913, Jan. 1999.

[7]Maple, R. C., and Belk, D. M., "Automated Set Up of Blocked, Patched, and Embedded Grids in the Beggar Flow Solver," *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, edited by N. P. Weatherill, J. Hauser, J. Thompson, and P. Eiseman, Pine Ridge Press, Swansea, England, U.K., 1994, pp. 305–314.

[8]Belk, D. M., and Strasburg, D. W., "Parallel Flow Solution on the T3D with Blocked and Overset Grids," *3rd Symposium on Overset Composite Grid and Solution Technology*, 1996.

[9]Prewitt, N. C., Belk, D. M., and Shyy, Wei, "Parallel Computing of Overset Grids for Aerodynamic Problems with Moving Objects," *Progress in Aerospace Sciences*, Vol. 36, No. 2, 2000, pp. 117–172.

[10]Prewitt, N. C., Belk, D. M., and Shyy, Wei, "Implementations of Parallel Grid Assembly for Moving Body Problems," AIAA Paper 98-4344, Aug. 1998.

[11]Whitfield, D. L., "Newton-Relaxation Schemes for Nonlinear Hyperbolic Systems," Engineering and Industrial Research Station, Mississippi State Univ., Rept. MSSU-EIRS-ASE-90-3, Mississippi State, MS, Oct. 1990.

[12]Amdahl, G. M., "Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities," *AFIPS Conference Proceedings*, Vol. 30, AFIPS Press, Reston, VA, 1967, pp. 483–485.

[13]Thoms, R. D., and Jordan, J. K., "Investigations of Multiple Body Trajectory Prediction Using Time Accurate Computational Fluid Dynamics," AIAA Paper 95-1870, June 1995.

J. Kallinderis
Associate Editor