NASA Contractor Report 189742
ICASE Report No. 92-68

ICASE

IMPLEMENTATION OF A PARALLEL UNSTRUCTURED EULER SOLVER ON SHARED AND DISTRIBUTED MEMORY ARCHITECTURES

D. J. Mavriplis
Raja Das
Joel Saltz
R. E. Vermeland

NASA Contract Nos. NAS1-18605 and NAS1-19480
December 1992

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681-0001

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

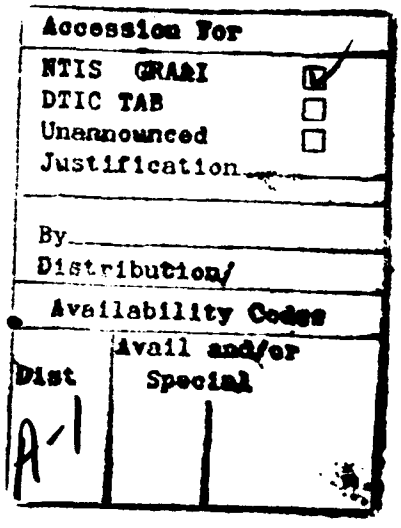
IMPLEMENTATION OF A PARALLEL UNSTRUCTURED EULER SOLVER ON SHARED AND DISTRIBUTED MEMORY ARCHITECTURES

D. J. Mavriplis, Raja Das, and Joel Saltz[1]
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23681

R. E. Vermeland
Cray Research, Inc.
Eagen, MN 55121

ABSTRACT

An efficient three dimensional unstructured Euler solver is parallelized on a Cray Y-MP C90 shared memory computer and on an Intel Touchstone Delta distributed memory computer. This paper relates the experiences gained and describes the software tools and hardware used in this study. Performance comparisons between the two differing architectures are made.



[1]This research was supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-18605 and NAS1-19480 while the first, second, and third authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

1. Introduction

In the past ten years, supercomputer performance has steadily increased more than one hundred fold. This has allowed computational aerodynamicists to simulate increasingly more complex mathematical models of fluid flow and compute the flow over more complicated geometries. Most aircraft manufacturers are today solving the inviscid form of the Navier-Stokes equations in a production environment. The production use of such codes is limited, however, by the difficulty of generating a suitable mesh and by the speed and size of the computation.

The mesh generation issue has been addressed by many researchers with varying degrees of success.[1,2] One of the more promising approaches is to discretize space into tetrahedral elements.[3,4,5,6] This provides a great deal of geometric flexibility so that highly complex shapes can be modeled accurately. Several automatic mesh generation methods are being developed which will allow engineers to more easily construct a mesh around a complex aircraft configuration.[7,8,9] As these methods become more robust, this current production bottleneck will be removed and engineers will require fast solution times to keep project data flowing.

The time to solution is influenced by the efficiency of the algorithm, and the sustained computation rate of the supercomputer. The unstructured Euler solver used in this study, EUL3D, is numerically efficient. It has been designed to minimize memory overhead, minimize the amount of gather/scatter which results from the use of indirect addressing on vector machines, and provide a rapid convergence to the steady state solution. The first two items are achieved by using a compact, edge based data structure, which minimizes the amount of indirect memory access required in the compute intensive routines. Accelerated converge rates have been achieved through the use of a multigrid algorithm specifically designed to work effectively on unstructured grids.[4]

EUL3D was developed on a Cray Y-MP shared memory vector/parallel computer and ported to an Intel Touchstone Delta distributed memory parallel computer. These machines allow solutions of large models to be computed in a matter of minutes, making production use viable and attractive. In fact, solution times are currently fast enough to effectively use this code in a design loop, allowing engineers to optimize aircraft shapes for best performance. In the next few years, as supercomputers again increase sustained performance levels, such codes may be employed as design tools in a production environment.

This paper relates the experience gained in parallelizing EUL3D on shared memory and distributed memory platforms. The software tools and hardware used in this study are also described and performance comparisons between the Intel Touchstone Delta and the Cray Y-MP C90 supercomputers are given.

2. Three Dimensional Unstructured Solver

2.1. Data structure

Complex aerodynamic shapes require high resolution meshes, and consequently, large numbers of grid points. In order to keep massive problems such as this tractable, one must avoid incurring excessive memory and CPU overheads by using efficient data structures that map effectively onto the machine architecture. Parallelization issues become challenging, since unstructured solvers operate on random data sets, which result in large sparse matrices.

EUL3D uses a compact vertex based scheme, with an edge based data structure. The flow variables are stored at each vertex in the mesh, and the residuals are assembled using loops over the list of edges that define the connectivity of the vertices. By partitioning the mesh or by ordering the lists of edges appropriately, work can be spread effectively over multiple processors.

2.2. Single grid solver

Since a complete mathematical derivation and description of the solver has been previously documented by Mavriplis[4] we will present an abbreviated description of the base solver that drives the multigrid algorithm.

A Galerkin finite element approach, using piecewise linear flux functions over each individual tetrahedra, is used to spatially discretize the domain. This type of discretization corresponds to a central differencing approach often employed on structured meshes, and therefore requires additional artificial dissipation to maintain stability. This is constructed as a blend of Laplacian and biharmonic operators on the conserved variables. The biharmonic operator acts everywhere in the flow field except near shock waves, where the Laplacian operator is turned on to prevent oscillations in the solution.

The spatially discretized equations form a system of coupled ordinary differential equations which are then integrated in time to obtain a steady state solution. A five stage Runge-Kutta scheme is used for the time integration, where the convective terms are evaluated at each stage of the time stepping scheme, and the dissipative terms are evaluated only at the first two stages and then frozen for the remaining stages. A complete multistage time-step, in which the solution is advanced from time level n to $n+1$ can be written as

$$
\begin{aligned}
w^{(0)} &= w^n \\
w^{(1)} &= w^{(0)} - a_1 Dt[Q(w^{(0)}) - D(w^{(0)})] \\
w^{(2)} &= w^{(0)} - a_2 Dt[Q(w^{(1)}) - D(w^{(1)})] \\
w^{(3)} &= w^{(0)} - a_3 Dt[Q(w^{(2)}) - D(w^{(1)})] \\
w^{(4)} &= w^{(0)} - a_4 Dt[Q(w^{(3)}) - D(w^{(1)})] \\
w^{(5)} &= w^{(0)} - a_5 Dt[Q(w^{(4)}) - D(w^{(1)})] \\
w^{n+1} &= w^{(5)}
\end{aligned}
\tag{1}
$$

with

$$a_1 = \frac{1}{4},\ a_2 = \frac{1}{6},\ a_3 = \frac{3}{8},\ a_4 = \frac{1}{2},\ a_5 = 1$$

where w represents the conserved variables, $Q(w)$ is the convective operator, $D(w)$ is the dissipative operator and Dt represents the discrete time step. The convective and dissipative terms are computed separately. $Q(w)$ is computed in a single loop over the edges, while $D(w)$ requires a two pass loop over the edges to assemble the biharmonic dissipation.

To accelerate convergence of the base solver, locally varying time steps and implicit residual averaging are used.

The above scheme has been designed to rapidly damp out high frequency error components, which is a necessary attribute for a good multigrid driving scheme.

2.3. Multigrid solver

The multigrid solver uses a set of progressively coarser meshes to calculate corrections to a solution on the fine mesh. The advantages of time stepping on the coarser meshes are twofold: first, the permissible time step is much greater, since it is proportional to the cell size; and secondly, the computational work is much smaller due to the decrease in the number of tetrahedra.

On the finest grid, the flow variables are updated using the five stage scheme of equations (1). The residuals and flow variables are then transferred to the next coarser grid. If R' represents the transferred residuals and w' the transferred flow variables, then a forcing function on the coarse grid can be defined as

$$P = R' - R(w'). \tag{2}$$

Now on the coarse grid, time stepping proceeds as follows

$$w^{(q)} = w^{(q-1)} - a_1 Dt[R(w^{(q-1)}) + P] \tag{3}$$

for the q-th stage. In the first stage, $w(q-1)$ reduces to the transferred flow variable w'. Therefore, the computed residuals on the coarse grid are canceled by the second term in the forcing function P, leaving only the R' term. This indicates that the solution on the coarse grid is driven by the residuals on the fine grid, so that as the residuals are driven to zero on the fine grid, no corrections will be generated by the coarse grid. This procedure is repeated successively on coarser grids. When the coarsest grid is reached, the corrections are transferred back to the finer grids.

EUL3D uses a sequence of completely unrelated coarse and fine grids. In this manner, coarse grids can be designed to optimize the speed of convergence, whereas the fine grid can be constructed to provide the most accurate solution. Furthermore, since no relation is assumed between the various meshes in the multigrid sequence, new finer meshes can be introduced by adaptive refinement.

Information is interpolated between the fine and coarse grids by use of four interpolation addresses and four interpolation weights for each vertex. Since these values are static, they are calculated in a pre-processing phase using an efficient graph traversal search algorithm. The cost of pre-processing is roughly equivalent to one or two flow solution cycles on the finest mesh.

The storage overhead incurred by the multigrid strategy corresponds to roughly a 33% increase in memory over the single grid scheme, which includes the storage of all the coarser grid levels, and the inter-grid transfer coefficients. Various multigrid strategies are possible. In this work, both V and W multigrid cycle strategies have been examined. The two cycles are illustrated in Figure 1.

In the case of a V-cycle, a time-step is first performed on the finest grid of the sequence. The flow variables and residuals are then transferred to the next coarser grid, where a new time-step is performed. The process is repeated until the coarsest grid of the sequence is reached, and the resulting corrections are then interpolated back down to the finest grid. Thus, within a multigrid V-cycle, a single time-step is performed on each grid level. The W-cycle strategy, as depicted in Figure 1, is a recursive approach which weights the coarse grids more heavily. The convergence histories of all three solution strategies for the problem described in the next section are displayed in Figure 2.

Each multigrid W-cycle requires more operations than a V-cycle, since more coarse grid visits are effected. In a purely sequential environment, a W-multigrid cycle requires approximately 90% more CPU time than a single grid cycle, while the multigrid V-cycle requires 75% more CPU time. However, both multigrid strategies provide close to an order of magnitude increase in convergence, as can be seen from Figure 2, thus greatly outweighing their increased cost per cycle.

The W-cycle has most often been found to provide sufficient increases in convergence over the V-cycle strategy in order to justify its extra cost. However, in a distributed memory parallel environment, the extra coarse-grid work is accompanied by an increased ratio of communication to computation, since the coarser grids represent smaller data sets spread over an equally large number of processors. The issue of which multigrid cycle constitutes the most efficient overall solution strategy may then become an architecture-dependent problem.

2.4. Pre-processing operations

Prior to the flow solution operation, an unstructured mesh must be generated. In the event that a multigrid solution strategy is to be employed, additional coarse grids must also be generated. These are constructed using an advancing front grid generator[9] run sequentially on a single CRAY Y-MP processor. Each grid must then be transformed into the appropriate edge based data structure for the flow solver, which entails constructing a list of edges with the addresses of the two end vertices for each edge, and a set of coefficients associated with each edge. For use on vector architectures, a coloring algorithm is then employed to divide the edge loop into multiple non-contiguous groups, such that within each group no data recurrences occur (i.e. no two edges access the same vertex). For use on distributed memory parallel architectures, the mesh must be partitioned and each partition assigned to an individual processor. The partitioning strategy must ensure load balancing and minimize communication by creating partitions of approximately equal size, and by minimizing the partition surface-to-volume ratios. In the multigrid strategy, the patterns for transferring data between the various meshes of the multigrid sequence must be determined. This is done using an efficient graph traversal search routine in a pre-processing operation. The result is a set of four addresses and four weights for each vertex of the mesh determining the interpolation of data from the current mesh to the next mesh in the sequence.

All of these preprocessing operations are performed sequentially on a single CRAY Y-MP processor. Apart from the grid generation and the partitioning problem, all operations are relatively inexpensive when implemented appropriately, usually requiring no more than the equivalent of one or two flow solution cycles. However, the particular partitioning strategy currently employed[10] was found to require CPU times comparable to the amount of time required for the entire flow solution procedure. Furthermore, the sequential implementation of all these preprocessing operations will inevitably lead to a bottleneck as the flow solution procedure becomes increasingly efficient with machines involving higher degrees of parallelism.

On the other hand, the preprocessing may be amortized over a large number of flow solutions. A set of grids may be generated, preprocessed and partitioned or colored, and

then employed to solve the flow over the particular geometry for a whole range of Mach number and incidence conditions, as is sometimes required in an industrial setting.

3. Shared Memory Implementation

3.1. Approach

The majority of the computations made in EUL3D are in loops over the edges of the mesh and there are typically well over one million edges in a mesh around a complex geometry. These loops move randomly through memory using indirect addressing. On a shared memory, vector/parallel machine like the Cray Y-MP C90, it is easiest to split the loops into groups or colors such that within each group, no recurrences occur. Each group can then be vectorized by either adding an argument to the compile statement, or by inserting a compiler directive at the beginning of each loop.

A simple parallelization strategy is to further divide the colorized groups into subgroups that can be computed in parallel. This is automatically done at compile time by the auto-tasking compiler. The subgroups are then distributed over all processors, taking advantage of the complete vector and parallel power of the machine.

For the problem used in this work, the number of fine grid edges was about 5.5 million. Since the typical number of groups is not high, say 20 to 30, the vector lengths within each subgroup are still large enough to fully realize the vector speedup of the machine. However, as the number of processors continues to increase, the vector lengths decrease, and this method becomes less efficient for a fixed problem size. For the case run in this study, the hypothetical use of 128 processors would still yield vector lengths of the order of 2000 elements, which is sufficient to mask slave CPU start-up overhead while achieving good vector performance.

3.2. Performance results

Figure 3 illustrates an unstructured mesh generated over a three dimensional aircraft configuration. The mesh contains a total of 106,064 points and 575,986 tetrahedra and is the second finest mesh used in the multigrid sequence. The finest mesh, which is not shown due to printing resolution limitations, contains 804,056 points and approximately 4.5 million tetrahedra. The inviscid flow was calculated using EUL3D on a 16 processor Y-MP C90 with 256 MWords of memory. Four meshes were used in the multigrid sequence.

The freestream Mach number for this case was 0.768 and the angle of attack was 1.116 degrees. The computed Mach contours are shown in Figure 4. Good shock resolution is observed, due to the large number of grid points employed. The convergence rates for this case using the single grid and the two multigrid strategies are shown in Figure 2.

The W-cycle multigrid strategy yields the fastest convergence rate on a per cycle basis. After 100 W-cycles, the residuals were reduced by nearly six orders of magnitude. This run took 242 seconds of wall clock time running in dedicated mode, including the time to read all grid files, write out the solution, and monitor the convergence by summing and printing out the average residual throughout the flow field at each multigrid cycle. The run required 94 million words of memory. The average speed of the calculation was 3.1 GFlops, as measured by the Cray hardware performance monitor.[11]

These results are documented in Table 1c, including the performance for runs using 1, 2, 4, and 8 processors. In Tables 1a and 1b similar statistics are documented for the single grid and the V-cycle multigrid runs. In all cases, a high degree of parallelism is achieved yielding on the average a CPU to wall clock time ratio of 15.4 for 16 processors. This indicates that the algorithm has achieved greater than 99% parallelism. However, total CPU time increases are observed as the number of concurrent CPUs increases (approximately 20% increase for 16 CPUs). This is due to the overhead associated with multitasking. The overall speedup achieved on 16 CPUs is thus 12.4 times the single CPU speed for the W-cycle in Table 1c.

Another characteristic of these runs is the relative insensitivity of the overall computational rates to the solution strategy. The single grid and the two multigrid strategies all achieve similar computational rates on 16 CPUs. This is attributed to the high memory bandwidth capacity of the CRAY Y-MP C90. Under these circumstances, just as in the sequential case, the W-cycle multigrid strategy is the most effective overall. A solution converged to within six orders of magnitude is obtained in 242 seconds using all 16 processors. A similar level of convergence using the V-cycle would require roughly 360 seconds, and the single grid strategy would require approximately 1 hour.

4. Distributed Memory Implementation

4.1. Approach

The implementation of EUL3D on the distributed memory MIMD architecture of the Intel Touchstone Delta machine was carried out using a set of software primitives known as PARTI (Parallel Automated Runtime Toolkit at ICASE). These tools have been designed to ease the implementation of computational problems on parallel architecture machines by relieving the user of low-level machine specific issues. The design philosophy has been to leave the original (sequential) source code essentially unaltered, with the exception of the introduction of various calls to the PARTI primitives which are imbedded in the code at the appropriate locations. These primitives allow the distribution and retrieval of data from the numerous processor local memories. Eventually, a parallel compiler is planned which should

be capable of automatically imbedding the primitives at the appropriate locations in the source code.[12] This implementation formed part of a research project aimed at demonstrating the effectiveness of these tools, while providing valuable input to the design and formulation of such tools.[13]

In distributed memory machines the data and the computational work must be divided between the individual processors. The criteria for this partitioning is to reduce the volume of interprocessor data communication and also to ensure good load-balancing. For the case described in this paper, this corresponds to partitioning each mesh of the multigrid sequence and assigning each partition to a particular processor. Since the majority of the computation is performed as loops over edges of the mesh, an edge which has both end points inside the same partition (processor) requires no outside information. On the other hand, edges which cross partition boundaries require data from other processors at each loop.

In this work, partitioning is done sequentially using a recursive spectral approach.[10] This method is known to deliver good load balancing and to minimize inter-partition surface area (and thus communication requirements). However, the expense of the partitioning operation has been found to be comparable to the cost of a sequential flow solution. If multiple flow solutions are required on the same mesh, this work can be amortized over a large number of flow solutions, since this is a preprocessing operation. The development of more efficient partitioning strategies is still an important concern. After the input data has been partitioned, a data file is created for each processor to read. Although the processors execute the same code, the partitioning of the input data causes each of the processors to perform the computation on a separate part of the mesh.

In distributed memory MIMD architectures, there is typically a non-trivial communications latency or startup cost. For efficiency reasons, information to be transmitted should be collected into relatively large messages. The cost of fetching array elements can be reduced by precomputing what data each processor needs to send and to receive. In irregular problems, such as those resulting from unstructured mesh problems, this is inferred by the subset of all mesh edges which cross partition boundaries. The communications pattern depends on the input data (i.e. the mesh). In this case, it is not possible to predict at compile time what data must be prefetched. We work around this problem by transforming the original loop into two constructs called inspector and executor.[14] During program execution, the inspector examines the data references made by a processor, and calculates what off-processor data needs to be fetched. The executor loop then uses the information from the inspector to implement the actual computation. The PARTI primitives can be used directly by programmers to generate inspector/executor pairs. Each inspector produces a communications schedule, which is essentially a pattern of communication for gathering or scattering data.

The executor has embedded PARTI primitives to gather or scatter data. The primitives are designed to minimize the effect on the source code, such that the final parallel code remains as close as possible to the original sequential code. Latency or start-up cost is reduced by packing various small messages with the same destinations into one large message.

We performed two types of optimization, both of which contribute to improve the total computational rate. We improve the single processor computation rate by reordering both the nodes and the edges which constitute the mesh. Next, we perform communication optimizations to reduce the volume of data that must be transmitted between processors. The communication optimizations are built into the software primitives.

4.2. Node and edge reordering

When the data access pattern is irregular, as it is in this case, the i860 (the Delta processors) memory hierarchy causes low computational rates. The i860 has three levels of memory. The first level are the registers, followed by the data cache and in the end the main memory. If the data access pattern is such that most of the time the data residing in the registers and the cache is utilized then very high computational rates can be achieved. Irregular data access patterns cause excessive cache misses which results in performance degradation.

Most of the computational work in EUL3D appears as loops over mesh edges. The edge list was therefore reordered such that all the edges incident on a vertex are listed consecutively. In this manner, once the data for a vertex is brought into the cache it can be used a number of times before it is removed. Clearly, this causes better cache utilization. We also performed node renumbering which causes data associated with nodes linked by mesh edges to be stored in nearby memory locations. These optimizations alone improved the single node computational rate by a factor of two.

4.3. Communications optimizations

In EUL3D, we encounter a variety of situations in which the same data is accessed by several consecutive loops. For instance, consider a step of the Runge Kutta integration. Flow variables are used in sequence of three loops over edges followed by a loop over boundary faces. The flow variables are only updated at the end of each of the Runge Kutta steps. We can obtain all of the off-processor flow variables needed at the beginning of the step. This makes it advantageous to develop methods that avoid bringing in the same data more than once.

We have developed optimizations which make it possible to track and reuse off-processor

data copies. We do this by modifying our software so that we are able to generate incremental communications schedules. Incremental schedules obtain only those off-processor data not requested by a given set of pre-existing schedules. Hash-tables are used omit duplicate off-processor data references. Using these incremental schedules we can significantly reduce the volume of communication.

4.4. Performance results

The flow calculations performed on the Cray Y-MP C90 were repeated on the Intel Touchstone Delta machine. The single grid and V-cycle multigrid strategies were run on 256 and 512 processors. The W-cycle multigrid results are scaled from experience on a smaller grid. The solution and convergence rates obtained were, of course, identical to those displayed in Figures 2 and 4. Table 2a, 2b, and 2c depict the performance statistics obtained for these runs. The total wall clock time required to run 100 cycles for each solution strategy is given. This time is then broken down into computation and communication time. The computational rate (MFlops) obtained by counting the number of operations in each loop are also given. These rates are about 10% more conservative than those based on the CRAY hardware performance monitor[11] (using a simple time scaling of the CRAY performance numbers).

The single grid solution strategy yields the highest computational rates achieving 1.5 GFlops on 512 Delta processors. However, this method is also the slowest to converge. The multigrid V-cycle procedure exhibits a degradation in computational rates of about 10 to 15% over the single grid case, while the W-cycle rates are estimated to be 25 to 30% lower. This is due to the increased amount of work performed on the coarse grid levels, which represent smaller data-sets distributed over the same number of processors, thus increasing the communication to computation ratio. The communication required for inter-grid transfers (between coarse and fine grids of the multigrid sequence) has been found to constitute a small fraction of the total communication costs.

The reduced computational efficiency of the multigrid strategies and the additional work required at each cycle are more than outweighed by the faster convergence rates of these methods over the single grid strategy. A single grid solution converged to 6 orders of magnitude on 512 Intel Delta processors would require approximately 1 hour of wall clock time, while the V and estimated W-cycle multigrid strategies would require 1083 and 843 seconds respectively. For certain cases the V-cycle may be the most effective strategy for the Intel Delta.

5. Shared vs. Distributed Memory – A Comparison

From the preceding sections, it is evident that the performance of EUL3D on both machines is comparable with the Y-MP C90 outperforming the Touchstone Delta by roughly a factor of two. The 512 Intel Delta machine appears to be roughly equivalent to a 5 processor CRAY Y-MP C90. The full CRAY Y-MP C90 achieved roughly 21% of its peak rated performance, while the Intel Delta achieved 5% of its theoretical peak. Both machines miss the mark on peak performance, mainly due to indirect addressing and the random nature of the data-sets. Such low utilization on the Intel i860 processors is rather common, and can be attributed to the small cache and low memory bandwidth of the processors. More significantly perhaps, is the ratio of computation to communication achieved on the 512 processor Delta machine, which is of the order of 50% for this problem, thus implying a relatively efficient implementation. This ratio, however, varies significantly with the size of the problem, the number of processors employed, and the particular solution strategy chosen. On the other hand, the computational rates achieved on the CRAY Y-MP C90 are relatively insensitive to problem size and solution strategy, a fact which is attributable to the shared memory architecture of the machine, and the large bandwidth to memory.

Parallelizing EUL3D on the CRAY Y-MP C90 was a relatively simple task, while the implementation on the Intel Touchstone Delta machine formed the basis of a research project.[13] The main reason for this disparity in efforts is the existence of sophisticated software tools such as automatic vectorizing and parallelizing compilers for the CRAY Y-MP series machines. While such tools are currently unavailable for distributed memory architectures, the current implementation was carried out using a set of experimental tools (i.e. the PARTI primitives) with the aim of demonstrating the effectiveness of such tools, as well as aiding in their formulation and development. The situation can be likened to the early days of vector supercomputing, when considerable programming effort was required to achieve the full vector potential of such machines. We believe that software tools will be critically important in determining the success of various parallel architectures in the future.

6. Conclusions

We have shown that a numerically efficient computational fluid dynamics code can be parallelized on both shared memory and distributed memory machines. Both machines yield comparable performance rates. However, the availability of sophisticated software tools enabled the parallelization of EUL3D on the shared memory vector/parallel CRAY Y-MP C90 with minimal user input. On the other hand, the implementation on the distributed memory massively parallel architecture of the Intel Touchstone DELTA machine is considerably more

involved. As massively parallel software tools become more mature, the task of developing or porting software to such machines should diminish.

We have also shown that with today's supercomputers, and with efficient codes such as EUL3D, the aerodynamic characteristics of complex vehicles can be computed in a matter of minutes, making design use feasible.

With the availability of rapid solution procedures, grid generation and preprocessing operations, which are presently executed sequentially, become the bottlenecks. In particular, the partitioning strategy employed for the distributed memory parallel implementation , although effective, is excessively costly. More research is required in this area in order to develop more efficient and parallel partitioners. Finally, the issues involved in parallel mesh generation and parallel adaptive mesh refinement must also be investigated in order to develop a complete and effective solution package.

References

[1] Thompson, J. F., "A Composite Grid Generation Code for General Three-Dimensional Regions," AIAA Paper 87-0275, January, 1987.

[2] Benek, J. A., Buning, P. G., and Steger, J. L., "A 3-D Chimera Grid Embedding Technique," AIAA Paper 85-1523-CP, July, 1985.

[3] Jameson, A., Baker, T. J., and Weatherill, N. P., "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA Paper 86-0103, January, 1986.

[4] Mavriplis, D. J., "Three Dimensional Unstructured Multigrid for the Euler Equations," Proc. of the AIAA 10th Comp. Fluid Dyn. Conf., AIAA Paper 91-1549, June, 1991.

[5] Smith, W. A., "Multigrid Solution of Transonic Flow on Unstructured Grids," Recent Advances and Applications in Computational Fluid Dynamics, Proceedings of the ASME Winter Annual Meeting, Ed. O. Baysal, November, 1990.

[6] Peraire, J., Peiro, J., and Morgan, K., "A 3D Finite Element Multigrid Solver for the Euler Equations," AIAA Paper 92-0449, January, 1992.

[7] Baker, T. J., "Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets," Proc. of the AIAA 8th Comp. Fluid Dyn. Conf., AIAA Paper 87-1124, June, 1987.

[8] Weatherill, N. P., "The Delaunay Triangulation," In Advances in Numerical Grid Generation, Mississippi State University Grid Courses, August, 1990.

[9] Gumbert, C., Lohner, R., Parikh, P., and Pirzadeh, S., "A Package for Unstructured Grid Generation and Finite Element Flow Solvers," AIAA Paper 89-2175, June 1989.

[10] Pothen, A., Simon, H. D., and Liou, K. P., "Partitioning Sparse Matrices with Eigenvectors of Graphs," *SIAM J. Math Anal. Appl.*, 11:430-452, 1990.

[11] UNICOS Performance Utilities Reference Manual, SR-2040 6.0, Cray Research, Inc.

[12] Saltz, J., Berryman, H., and Wu, J., "Runtime Compilation for Multiprocessors," *Concurrency, Practice and Experience*, 3(6):573-592, 1991.

[13] Das, R., Mavriplis, D. J., Saltz, J., Gupta, S., and Ponnusamy, R., "The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives," AIAA Paper 92-0562, January, 1992.

[14] Mirchandaney R., Saltz J. H., Smith R. M., Nicol D. M., and Crowley K., "Principles of Runtime Support for Parallel Processors," Proceedings of the 1988 ACM International Conference on Supercomputing, St. Malo France, pages 140–152, July, 1988.

CPUs	Wall Clock	CPU sec.	MFlops
1	1916	1878	252
2	974	1909	495
4	508	1957	966
8	273	2038	1856
16	156	2185	3252

Table 1a: Y-MP C90 speeds for *EUL3D* running *100* single grid cycles.

CPUs	Wall Clock	CPU sec.	MFlops
1	2586	2557	247
2	1326	2611	485
4	698	2572	945
8	380	2805	1804
16	223	3085	3161

Table 1b: Y-MP C90 speeds for EUL3D running 100 multigrid cycles using the V cycle.

CPUs	Wall Clock	CPU sec.	MFlops
1	3041	2992	249
2	1552	3048	484
4	815	3146	939
8	444	3323	1790
16	268	3709	3136

Table 1c: Y-MP C90 speeds for EUL3D running 100 multigrid cycles using the W cycle.

Nodes	Seconds per 100 cycles			Rate
	Communication	Computation	Total	MFlops
256	121	326	448	778
512	95	170	265	1496

Table 2a: Touchstone Delta speeds for EUL3D running 100 single grid cycles.

Nodes	Seconds per 100 cycles			Rate
	Communication	Computation	Total	MFlops
256	536	427	963	680
512	374	231	605	1252

Table 2b: Touchstone Delta speeds for EUL3D running 100 multigrid cycles using the V cycle.

Nodes	Seconds per 100 cycles			Rate
	Communication	Computation	Total	MFlops
256	787	596	1383	573
512	565	278	843	1030

Table 2c: Estimated Touchstone Delta speeds for EUL3D running 100 multigrid cycles using the W cycle.

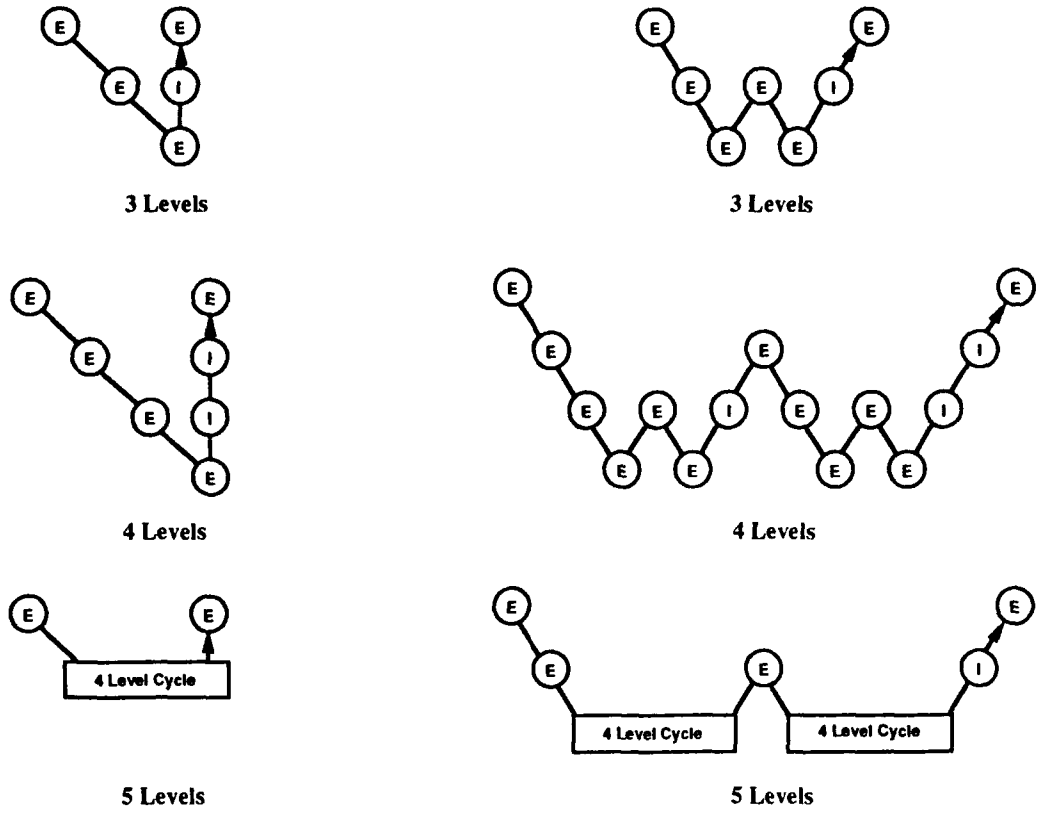


Figure 1: Multigrid V and W-cycles. Euler time steps are depicted by E, interpolations are depicted by I.

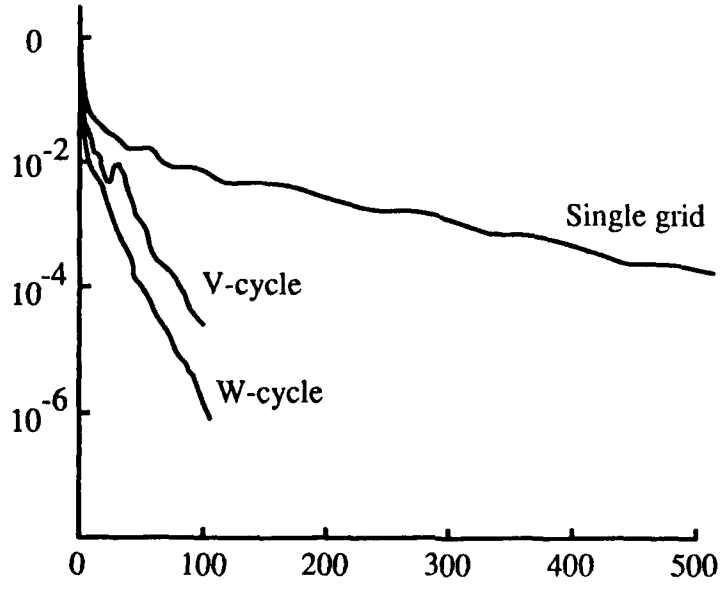


Figure 2: Convergence history for single grid and for V and W multigrid cycles.

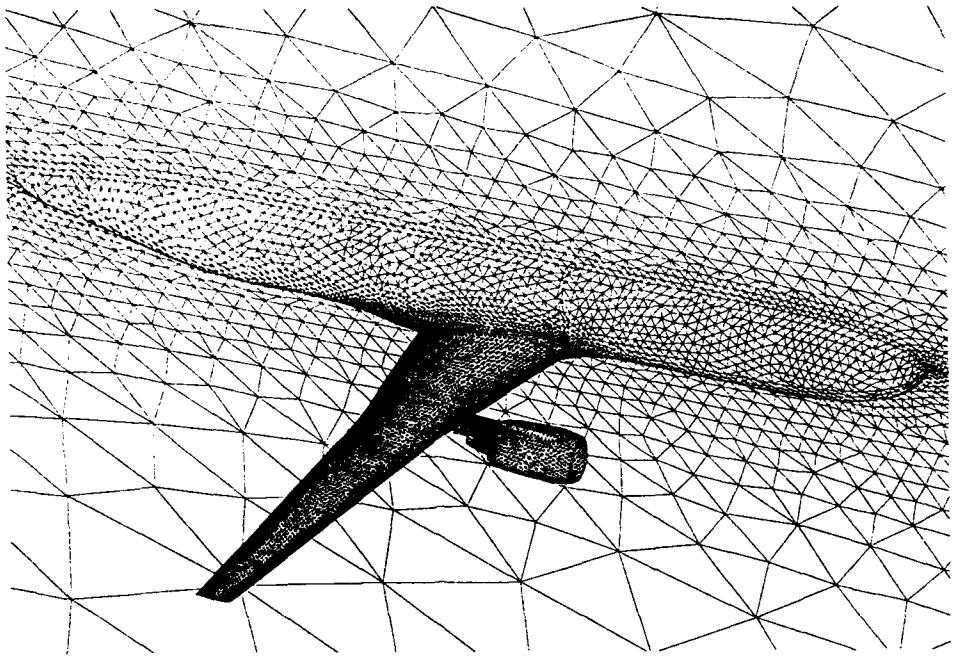


Figure 3: Unstructured mesh about a three dimensional aircraft configuration. The mesh shown is the second finest in the multigrid sequence and contains 106,064 nodes and 575,986 tetrahedra. The finest mesh, which is not shown due to printing limitations, contains 804,056 modes and approximately 4.5 million tetrahedra.

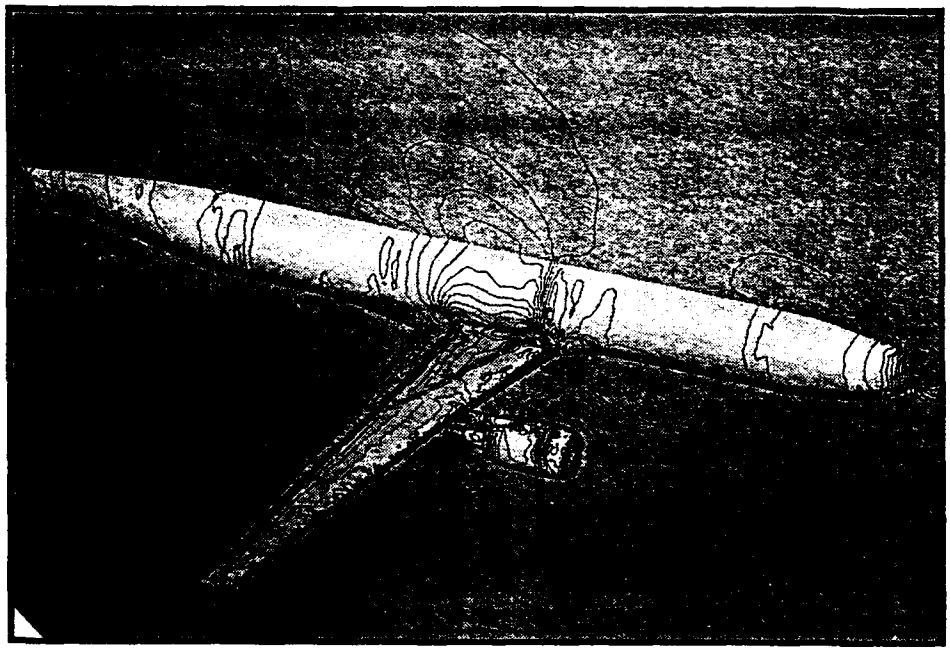


Figure 4: Computed Mach contours of Transonic Flow over Aircraft Configuration.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Contractor Report

4. TITLE AND SUBTITLE
IMPLEMENTATION OF A PARALLEL UNSTRUCTURED EULER SOLVER ON SHARED AND DISTRIBUTED MEMORY ARCHITECTURES

5. FUNDING NUMBERS
C NAS1-18605
C NAS1-19480
WU 505-90-52-01

6. AUTHOR(S)
D.J. Mavriplis, Raja Das, Joel Saltz and R.E. Vermeland

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 92-68

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-189742
ICASE Report No. 92-68

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Michael F. Card
Final Report

Appeared in Supercomputing '92 Proc. -- Submitted to the Journal of Supercomputing

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 02, 34

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

An efficient three dimensional unstructured Euler solver is parallelized on a Cray Y-MP C90 shared memory computer and on an Intel Touchstone Delta distributed memory computer. This paper relates the experiences gained and describes the software tools and hardware used in this study. Performance comparisons between two differing architectures are made.

14. SUBJECT TERMS
unstructured; parallel; shared memory; CRAY

15. NUMBER OF PAGES
20

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		