High Resolution Aerospace Applications using the NASA Columbia Supercomputer

Dimitri J. Mavriplis *
Department of Mechanical Engineering, University of Wyoming, Laramie WY 82071, USA
Michael J. Aftosmis †
NASA Advanced Supercomputer Division, NASA Ames Research Center, Moffett Field, CA 94035, USA
Marsha Berger ‡
Department of Computer Science, Courant Institute, New York University, NY, NY 10012, USA

This paper focuses on the parallel performance of two high-performance aerodynamic simulation packages on the newly installed NASA Columbia supercomputer. These packages include both a high-fidelity, unstructured, Reynolds-averaged Navier-Stokes solver, and a fully-automated inviscid flow package for cut-cell Cartesian grids. The complementary combination of these two simulation codes enables high-fidelity characterization of aerospace vehicle design performance over the entire flight envelope through extensive parametric analysis and detailed simulation of critical regions of the flight envelope. Both packages are industrial-level codes designed for complex geometry and incorporate customized multigrid solution algorithms. The performance of these codes on Columbia is examined using both MPI and OpenMP and using both the NUMAlink and InfiniBand interconnect fabrics. Numerical results demonstrate good scalability on up to 2016 cpus using the NUMAlink4 interconnect, with measured computational rates in the vicinity of 3 TFLOP/s, while InfiniBand showed some performance degradation at high CPU counts, particularly with multigrid. Nonetheless, the results are encouraging enough to indicate that larger test cases using combined MPI/OpenMP communication should scale well on even more processors.

Keywords: NASA Columbia, SGI Altix, scalability, hybrid programming, unstructured, computational fluid dynamics, OpenMP

I. Introduction

COMPUTATIONAL fluid dynamics (CFD) techniques have been developed and applied to aerospace analysis and design problems since the advent of the supercomputer. However, in spite of several decades of continuous improvements in algorithms and hardware, and despite the widespread acceptance and use of CFD as an indispensable tool in the aerospace vehicle design process, computational methods are still employed in a very limited fashion in the design process. The full potential of these methods in delivering more optimal designs and in accelerating the design cycle has yet to be approached. On the one hand, computational methods for aerodynamic analysis are only reliable within a narrow range of flight conditions, where no significant flow separation occurs. This is due in part to the extreme accuracy requirements of the aerodynamic design problem, where, for example, changes of less than 1% in the drag coefficient of a flight

*Professor, Department of Mechanical Engineering, University of Wyoming
†NASA Advanced Supercomputing Division, NASA Ames Research Center
‡Professor, Department of Computer Science, Courant Institute, NYU, 251 Mercer St. NY, NY 10012, USA

SC—05 November 12-18, 2005, Seattle, Washington, USA

vehicle can determine commercial success or failure. Additionally, the physics of flight aerodynamics is one which encompasses a wide range of scales, from thin boundary layers in the millimeter range, up to the full aircraft length scales. As a result, computational analyses are generally used in conjunction with experimental methods and only over a restricted range of the flight envelope, where they have been essentially calibrated. A recent series of workshops sponsored by the American Institute of Aeronautics and Astronautics (AIAA) has determined that the accuracy achieved by CFD methods for aerodynamic applications is substantially inferior to that delivered by state-of-the-art wind-tunnel testing, and improvements in accuracy will require, among various items, substantially higher grid resolution than what is generally considered practical in the current environment.[1]

While the analysis problem in itself is difficult, the computational design problem, in which one is interested in modifying the geometry in order to improve some design objective of the vehicle, is much more formidable. One the one hand, the number of design variables, which are the degrees of freedom used to modify and define the optimized geometry, can be extremely large in number (10,000 to 100,000) and the sensitivity of the numerical flow field to these design variables must be computed in order to drive the optimization problem. On the other hand, once a new "optimal" design has been constructed, it must be validated throughout the entire flight envelope. This includes not only the full range of aerodynamic flight conditions, but also all possible control surface deflections and power settings. Generating this aero-performance database not only provides all details of vehicle performance, but also opens up new possibilities for the engineer. For example, when coupled with a six-degree-of-freedom (6-DOF) integrator, the vehicle can be "flown" through the database by guidance and control (G&C) system designers to explore issues of stability and control, or G&C system design.[2] Alternatively, a complete unsteady simulation of a maneuvering vehicle may be undertaken. Ultimately, the vehicle's suitability for various mission profiles or other trajectories can also be evaluated by full end-to-end mission simulations.[3]

Our approach to this seemingly intractable problem relies on the use of a variable fidelity model, where a high fidelity model which solves the Reynolds-averaged Navier-Stokes equations (NSU3D) is used to perform the analysis at the most important flight conditions, as well as to drive a high-fidelity design optimization procedure, and a lower fidelity model based on inviscid flow analysis on adapted Cartesian meshes (Cart3D) is used to validate the new design over a broad range of flight conditions, using an automated parameter sweep database generation approach. In addition to this variable fidelity approach, other enabling factors include the use of custom developed state-of-the-art optimal solution techniques and large scale parallel hardware. Both NSU3D and Cart3D employ multigrid methods, specially developed for each application, which deliver convergence rates which are insensitive to the number of degrees of freedom in the problem. Finally, the use of state-of-the-art large-scale parallel hardware enables improved accuracy in all phases of the process by relying on high resolution meshes, generally employing one or two orders of magnitude higher resolution than current-day standard practices.

II. The NASA Columbia Supercomputer

Figure 1 shows a snapshot of NASA's Columbia supercomputer located at Ames Research Center in Moffett Field CA. This platform is a supercluster array of 20 SGI Altix 3700 nodes. Each node has 512 Intel Itanium2 processors clocked at either 1.5 or 1.6Ghz, and each CPU has 2Gb of local memory. Each processor supports up to four memory-load operations per clock-cycle from L2 cache to the floating-point registers, and are thus capable of delivering up to 4 FLOPS per cycle to the user.

Of the 20 nodes in the system, Columbia 1-12 are Altix 3700 systems, while Columbia 13-20 are actually 3700BX2 architectures. Processor bricks in the 3700BX2's are connected using SGI's proprietary NUMAlink4 interconnect with a peak bandwidth rated at 6.4Gb/sec. The work described here was performed on four of these 3700BX2 nodes (c17-c20). These 2048 CPUs are all clocked at 1.6Ghz

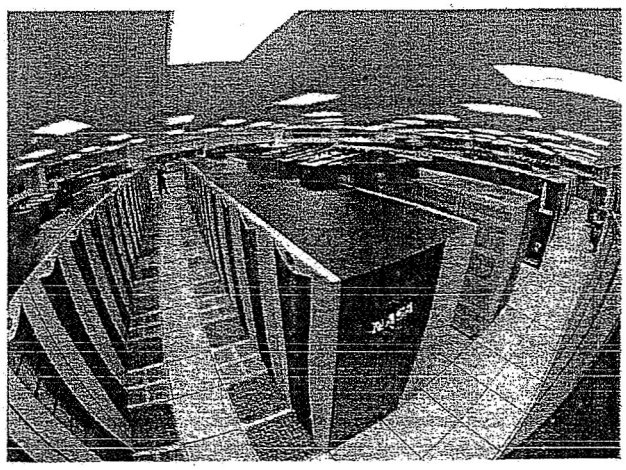


Figure 1. The 10,240 processor Columbia Supercomputer located at NASA Ames Research Center, Moffett Field CA.

and each has 9Mb of L3 cache. While the 1Tb of local memory on each 512 CPU node of the Columbia system is cache-coherent and globally shared, cache-coherency is not maintained between nodes. As a result standard OpenMP codes are currently limited to 512 CPUs.

The entire supercluster is connected using both InfiniBand and 10Gigabit Ethernet networks. The InfiniBand provides low-latency routing among the nodes for system-wide MPI communication, while the 10Gig-E provides user access and I/O.

The large MPI simulations presented here exercised both the NUMAlink and InfiniBand interconnects. While the NUMAlink can directly support MPI on each of the 2048 processors in the "Vortex" system (c17-c20), the same is not true of the InfiniBand connection fabric. As discussed in [4], a limitation in the number of MPI connections available to the InfiniBand cards installed on each 512 node restricts the maximum number of MPI processes to

$$\#MPI_{IB} \leq \sqrt{\frac{N_{IBcards} \times N_{connections}}{n-1}} \tag{1}$$

where n (≥ 2) is the number of Altix nodes, $N_{IBcards} = 8$ per node and $N_{connections} = 64$K per card. In practical terms, this constraint implies that a pure MPI code run on 4 nodes of columbia can have no more than 1524 MPI processes. If more MPI connections are attempted, the system will give a warning message, and then drop down to the 10Gig-E network for communication. Thus, when using the InfiniBand network, hybrid style applications are required (e.g. use 2 OpenMP threads for each MPI processes) when using large numbers of CPUs in several boxes. This is an important point when considering runs on greater than 2048 CPUs. The NUMAlink spans at most 4 boxes, and InfiniBand is the only high-speed interconnect spanning the entire machine. As a result, any plan to use more than 2048 CPUs requires the use of InfiniBand, and therefore will demand hybrid communication to scale to larger problem sizes.

III. High-Fidelity Analysis Model

Our high-fidelity model (the NSU3D code) solves the Reynolds-averaged Navier Stokes (RANS) equations on unstructured hybrid meshes. This code has been under development for over ten years,[5–7] and is currently used in production mode in the aerodynamic design departments of several aircraft manufacturers. This solver has also been a participant in the two recent Drag Prediction Workshops (DPW), sponsored by AIAA, where the predictive ability of various research, production, and commercial CFD codes were evaluated on standard aircraft configurations.[1] By solving the Navier-Stokes equations, NSU3D enables the simulation of viscous flow effects, including boundary layers and wakes, which are not included in the inviscid flow model used by Cart3D. The effects of turbulence for steady-state analyses are incorporated through the solution of a standard one-equation turbulence model,[8] which is solved in a coupled manner along with the flow equations.

The use of unstructured meshes provides the required flexibility for discretizing complex airframe geometries, which may include deflected control surfaces, deployed flaps for landing and take-off,[7] and engine nacelle integration problems.[1] While the solver can handle a variety of element types, high-aspect-ratio prismatic elements are generally used in regions close to the aircraft surfaces, where thin boundary layers and wakes prevail, and isotropic tetrahedral elements are used in outer regions, with pyramidal elements used in transition regions. Because of the high accuracy requirements of aircraft design (drag coefficient values are usually required to 1 part in 10,000: i.e. 1 drag count), and because of the sensitivity of important flow physics such as flow separation to the behavior of the boundary layer, it is common practice in aerodynamic simulations to fully resolve the turbulent boundary layer right down to the very small scale of the laminar sub-layer. This is in contrast to many other CFD applications, where a large part of the boundary layer is approximated using "wall function" boundary conditions. For typical aerodynamic Reynolds numbers (≥ 5 million), fully resolving the boundary layer requires the use of grid cells with a normal height at the wall of 10^{-6} wing chords, where a wing chord corresponds to the streamwise dimension of the wing. In order for the aerodynamic simulation problem to remain tractable, anisotropic resolution must be employed in boundary layer regions by using chordwise and spanwise grid spacings which are several orders of magnitude larger than the normal spacing, but nevertheless adequate for capturing the gradients in these respective directions (see detailed insert in Figure 13 (a)).

While the added cost of computing the physical viscous terms and solving the turbulence modeling equation is relatively modest (25 to 50%), RANS simulations are generally found to be 50 to 100 times more expensive for equivalent accuracy in terms of overall computational time compared to inviscid flow (Euler) simulations such as Cart3D, due mainly to the added resolution required to resolve the viscous regions, and

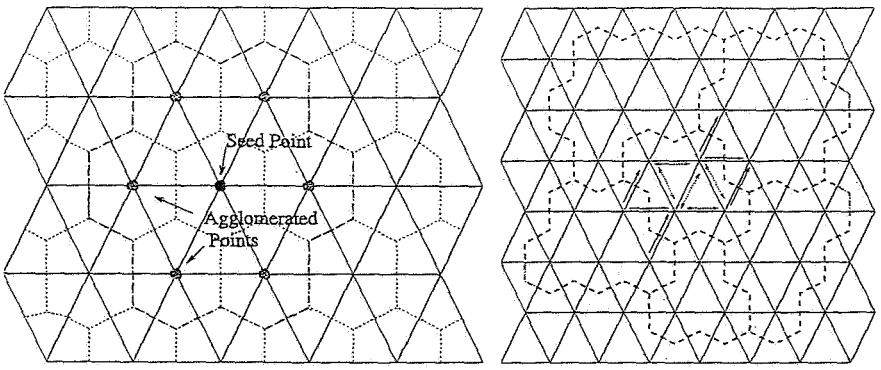


Figure 2. Illustration of agglomeration process for coarse level construction in multigrid algorithm. Median dual control volumes are associated with grid points. Agglomeration proceeds by identifying a seed point and merging all neighboring control volume points with this seed point (a). Resulting coarse level agglomerated control volumes are larger and more irregular in shape (b).

the increased stiffness associated with the highly anisotropic mesh resolution employed in these regions.[9] This makes the use of RANS solvers difficult for rapidly populating large data bases within the context of a broad parameter study. However, due to the more complete modeling of the flow physics, RANS solvers are best suited for use in highly accurate single point analyses, and in design optimization studies.

The NSU3D code employs a second-order accurate discretization, where the unknowns are stored at the grid points. The six degrees of freedom at each grid point consist of the density, three-dimensional momentum vector, energy, and turbulence variable. For the convective terms, the discretization relies on a control volume formulation, achieving second order-accuracy through an extended neighbor-of-neighbors stencil, while the discrete viscous terms are obtained using a nearest neighbor finite-volume formulation.[5]

Using the method of lines, these spatially discretized equations are advanced in time until the steady-state is obtained. Convergence is accelerated through the use of an implicit agglomeration multigrid algorithm.[5,9] The idea of a multigrid method is to accelerate the solution of a fine grid problem by computing corrections using coarser grid levels, where computation costs are lower, and errors propagate more rapidly. For unstructured meshes, the construction of a complete sequence of coarser mesh levels, given an initial fine level, represents a non-trivial task. The agglomeration multigrid approach constructs coarse grid levels by agglomerating or grouping together neighboring fine grid control volumes, each of which is associated with a grid point, as depicted in Figure 2(a). This is accomplished through the use of a graph algorithm, and the resulting merged control volumes on the coarse level form a smaller set of larger more complex-shaped control-volumes, as shown in Figure 2(b). This procedure is applied recursively, in order to generate a complete sequence of fine to coarse grid levels. Figure 3 illustrates the resulting coarse grid levels for a three-dimensional aircraft configuration. In the flow solution phase, each multigrid cycle begins with several

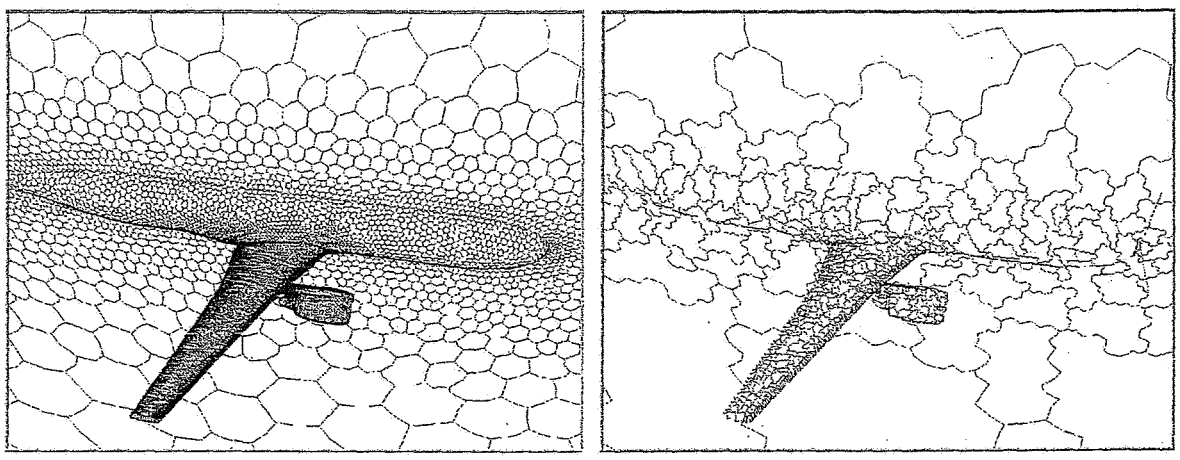


Figure 3. First (a) and third (b) agglomerated multigrid levels for unstructured grid over aircraft configuration.

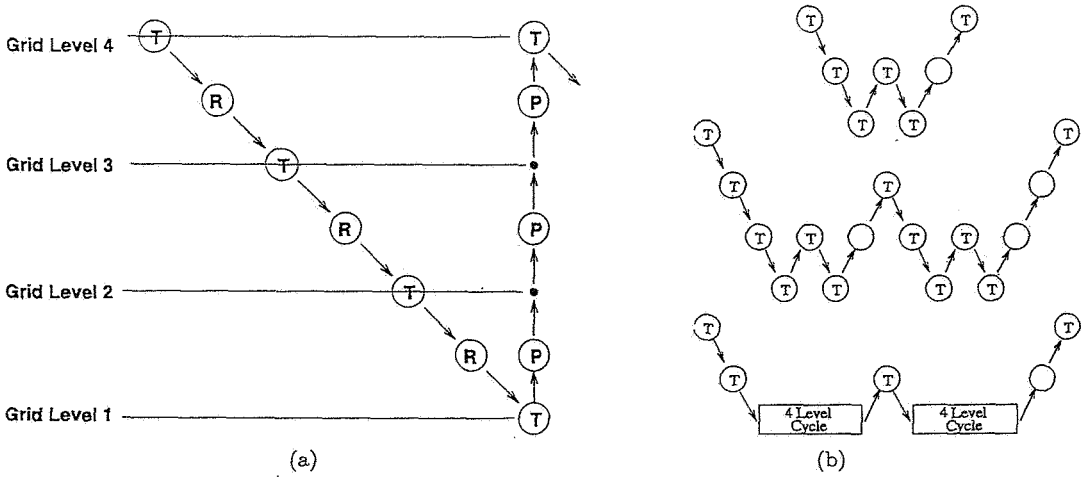


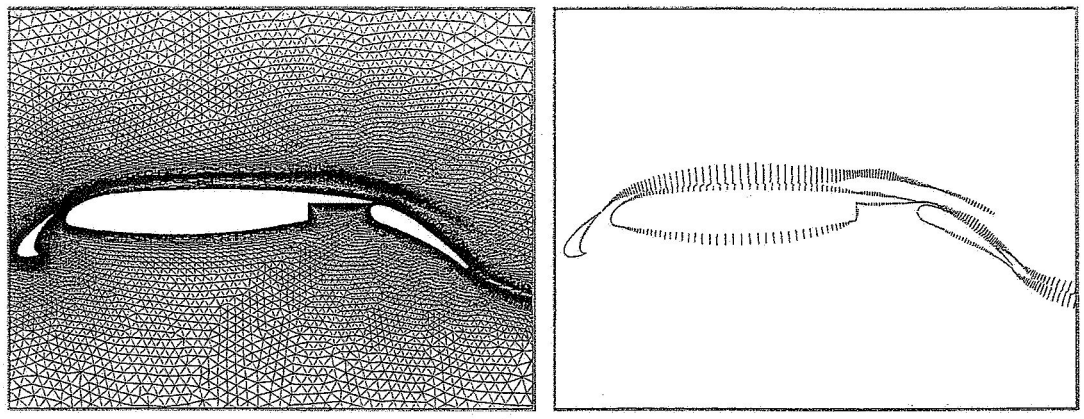
Figure 4. (a): Illustration of multigrid V-cycle. T denotes time step on a particular grid level, R denotes restriction (fine to coarse) transfer, and P denotes prolongation (coarse to fine) transfer. Grid 1 is the coarsest level and Grid 4 denotes the finest level in this depiction. (b): Illustration of the recursive nature of the multigrid W-cycle which performs additional visits to the coarser grid levels. Restriction and prolongation operation symbols have been omitted for clarity.

time steps on the finest level, after which the problem is transfered to the next coarser level, and the process is repeated recursively until the coarsest level of the sequence is reached, at which point the corrections are interpolated back to the finest level, and the process is repeated. The simplest strategy consists of performing one or more time steps on each grid level in the coarsening phase, and no time steps on the refinement phase. This is denoted as the multigrid V-cycle, and is depicted in Figure 4(a). An alternate cycle, denoted as the multigrid W-cycle, is a recursive procedure which performs additional visits to the coarser mesh levels, as shown in Figure 4(b). The multigrid W-cycle has been found to produce superior convergence rates and to be more robust, and is thus used exclusively in the NSU3D calculations.

Rather than performing simple explicit time steps on each grid level within the multigrid sequence, the use of local implicit solvers at each grid point provides a more efficient solution mechanism. This mandates the inversion of dense 6x6 block matrices at each grid point at each iteration. However, in regions of high mesh stretching such as in the boundary layer regions, solution efficiency degrades due to the stiffness induced by the extreme grid anisotropy. This can be overcome be resorting to an implicit line solver in such regions. Using a graph algorithm, the edges of the mesh which connect closely coupled grid points (usually in the normal direction) in boundary layer regions, are grouped together into a set of non-intersecting lines. Figure 5 illustrates the construction of the line set for a two-dimensional mesh with appreciable stretching in near-wall and wake regions. The discrete governing equations are then solved implicitly along these lines, using a block tridiagonal LU-decomposition algorithm for each line. In isotropic regions of the mesh, the line structure reduces to a single point, and the point-implicit scheme described above is recovered. This line-implicit driven agglomeration multigrid algorithm has been shown to produce convergence rates which are both insensitive to the degree of mesh resolution, and to the degree of mesh stretching.[9]

The NSU3D code achieves parallelism through domain decomposition. The adjacency graph of each fine and coarse agglomerated grid level of the multigrid sequence is fed to the METIS partitioner[10] which returns the partitioning information. Each grid level is partitioned independently, and coarse and fine grid partitions are then matched up together based on the degree of overlap between the respective partitions, using a non-optimal greedy-type algorithm. This approach may result in more inter-processor communication when transferring variables between coarse and fine multigrid levels than a fully nested approach - where the coarse level partitions are inferred from the fine level partitions. However, experiments indicate thaat it is more important to optimize the intra-level partitioning process versus inter-level partitioning, since the work required in transferring variables between levels is minimal compared to the work performed by the implicit solver within each level.

For each partitioned level, the edges of the mesh which straddle two adjacent processors are assigned to one of the processors, and a "ghost vertex" is constructed in this processor, which corresponds to the vertex originally accessed by the edge in the adjacent processor (c.f. Figure 6(a)). During a residual evaluation, the fluxes are computed along edges and accumulated to the vertices. The flux contributions accumulated at the



(a) (b)

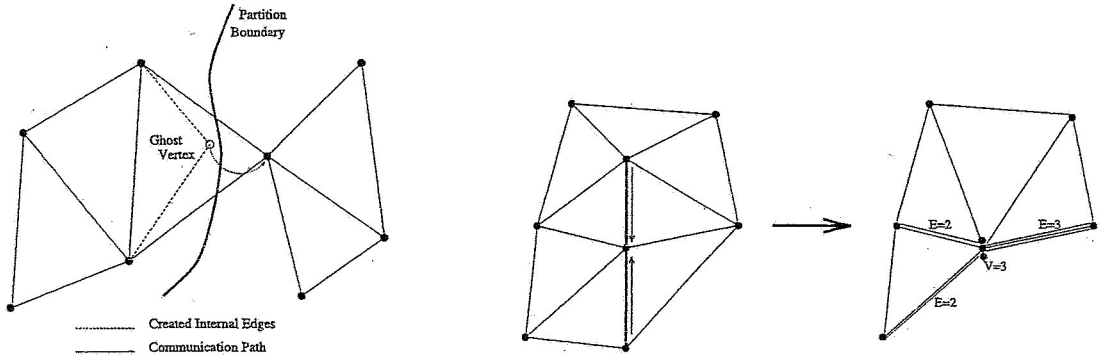
Figure 5. (a): Illustration of two-dimensional unstructured grid with high stretching in near body and wake regions and (b) resulting set of lines used for implicit line solver.

ghost vertices must then be added to the flux contributions at their corresponding physical vertex locations in order to obtain the complete residual at these points. This phase incurs interprocessor communication. In an explicit (or point implicit) scheme, the updates at all points can then be computed without any interprocessor communication once the residuals at all points have been calculated. The newly updated values are then communicated to the ghost points, and the process is repeated.

The use of line solvers complicates the parallel implementation, since the block tridiagonal line solver is an inherently sequential algorithm. The partitioning procedure is thus modified to avoid breaking the implicit lines across inter-partition boundaries. This is achieved by contracting the adjacency graph along the implicit lines, effectively collapsing each line to a single point in the graph as shown in Figure 6(b). Using the appropriate vertex and edge weights which result from the contraction process, this new weighted graph is fed to the METIS partitioner, resulting in a set of partitions which never breaks an implicit line structure.

The NSU3D solver employs a hybrid MPI/OpenMP approach for parallel execution. In general, each partition is associated with an individual processor, and inter-processor communication can be performed using MPI alone, using OpenMP alone, or using a hybrid approach where each MPI process is responsible for several partitions/processors, which communicate among themselves using OpenMP. For MPI-alone cases, communication is executed by packing messages from all ghost points on a given processor that are to be sent to another processor into a buffer that is then sent as a single message. This standard approach to inter-processor communication has the effect of reducing latency overheads by creating fewer larger messages.

For shared memory architectures using OpenMP, the multiple partitions are run simultaneously using one thread per partition, and parallelization is achieved at the partition loop level. In such cases, a potentially more efficient communication strategy is to simply copy (or copy-add) the values from the individual ghost points into the locations which correspond to their real images, since the memory on different partitions is



(a) (b)

Figure 6. (a): Illustration of creation of internal edges and ghost points at inter-processor boundaries; (b): Illustration of line edge contraction and creation of weighted graph for mesh partitioning. V and E values denote vertex and edge weights respectively.

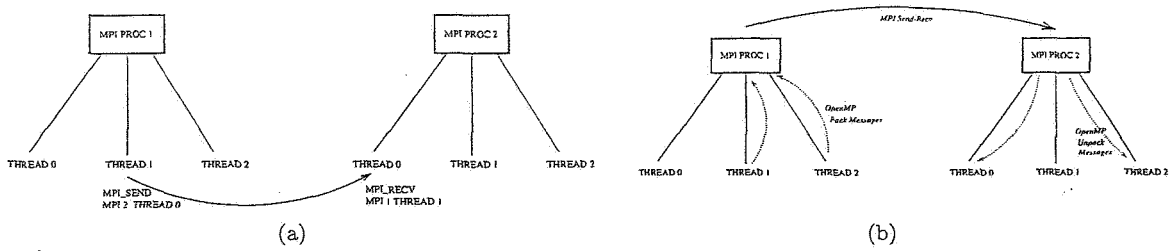


Figure 7. (a): Illustration of thread to thread MPI communication for a two-level hybrid MPI-OpenMP implementation; (b): Illustration of Master-Thread controlled MPI communication for a two-level hybrid MPI OpenMP implementation.

addressable from any other partition.

For hybrid MPI/OpenMP cases, each MPI process is responsible for various partitions, which are executed in parallel using one OpenMP thread per partition. Communication between partitions shared under a single MPI process proceeds as in the pure OpenMP case. In order to communicate between partitions owned by different MPI processes, two programming models have been considered. A communication strategy which can be executed entirely in parallel consists of having individual threads perform MPI calls to send and receive messages to and from other threads living on other MPI processes, as illustrated in Figure 7(a). In this case, the MPI calls must specify the process identifier (*id number*) as well as the thread id to which the message is being sent (or received). While the specification of a process id is a standard procedure within an MPI call, the specification of a thread id can be implemented using the MPI send-recv tag.[11] An alternate approach, illustrated in Figure 7(b), consists of having all threads pack their messages destined for other threads of a particular remote MPI process into a single buffer, and then having the MPI process (i.e., the master thread alone) send and receive the message using MPI . The received messages can then be unpacked or scattered to the appropriate local subdomains. This packing and unpacking of messages can be done in a thread-parallel fashion. However, the MPI sends and receives are executed only by the master thread, and these operations may become sequential bottlenecks since all other threads remain idle during this phase. One way to mitigate this effect is to overlap OpenMP and MPI communication. Using non-blocking sends and receives, the master thread first issues all the MPI receive calls, followed by all the MPI send calls. After this, while the MPI messages are in transit, the OpenMP communication routines are executed by all threads, after which, the master thread waits until all MPI messages are received. Thread-parallel unpacking of the MPI messages then proceeds as usual. This approach also results in a smaller number of larger messages being issued by the MPI routines, which may be beneficial for reducing latency on the network supporting the MPI calls. On the other hand, there is always a (thread-) sequential portion of communication in this approach, which may degrade performance depending on the degree of communication overlap achieved.

Previous experience has shown that the thread parallel approach to communication scales poorly due to the MPI calls "locking" and thus executing serially at the thread level.[12] Thus, the master thread communication strategy is used exclusively in this work.

Within each partition, single-processor performance is enhanced using local reordering techniques. For cache-based scalar processors, such as the Intel Itanium on the NASA Columbia machine, the grid data is reordered for cache locality using a reverse Cuthill McKee type algorithm. For vector processors, coloring algorithms are used to enable vectorization of the basic loop over mesh edges, which accumulate computed values to the grid points. Because the line solver is inherently scalar, the lines are sorted based on their length, and grouped into sets of 64 lines of similar length, over which vectorization may then take place at each stage in the line solver algorithm. These techniques have been demonstrated on the CRAY SV-1 and NEC SX-6.

IV. Optimization and Parametric Studies for Performance Prediction

The outcome of design optimization is a modified vehicle whose performance is known only at the design points. Even with data from several additional cases as provided by NSU3D, this only provides a small portal into understanding overall vehicle performance. Current research into automated parametric studies is aimed at broadening this snapshot of vehicle performance by rapidly producing the entire performance database for a new design as delivered by the shape optimizer. These parametric studies consider not only a range of flight conditions, but also include all possible control surface deflections and power settings.

Large numbers of aerodynamic and shape parameters can easily result in aero-performance databases

with 10^4-10^6 entries. Automatically computing this performance envelope is the goal of NASA's Cart3D analysis package.[2,13,14] This package permits parametric sweeps of not only flight conditions (Mach number, angle-of-attack and sideslip), but also deployment of control surfaces. The geometry comes into the system as a set of watertight solids, either directly from the optimizer or from a CAD system. These solids are automatically triangulated and positioned for the desired control surface deflections.[13,15,16] With the new analysis model in-hand, the embedded-boundary Cartesian method automatically produces a computational mesh to support the CFD runs.[2,13]

The parameter studies consider changes in both the geometry (control surface deflection) and "wind parameters" (Mach, angle-of-attack, sideslip). A typical analysis may consider three "Configuration-Space" parameters (e.g. aileron, elevator and rudder deflections) and examine three "Wind-Space" parameters (Mach number, angle-of-attack, and sideslip angle). In this six-dimensional parametric space, ten values of each parameter would require 10^6 CFD simulations; 1000 wind-space cases for each of the 1000 instances of the configuration in the config-space. The job control scripts arrange the jobs hierarchically such that different instances of the geometry are at the top level with wind parameters below. For a particular instance of the geometry, the jobs exploring variation in the Wind-Space all run using the same mesh and geometry files. This approaches amortizes the cost of preparing the surface and meshing each instance of the geometry over the hundreds or thousands of runs done on that particular instance of the geometry. On Columbia's Itanium2 CPUs the Cartesian mesh generator[13] typically produces 3-5 million cells-per-minute, and mesh sizes for realistically complex vehicles generally contain 3-10 million cells. Moreover, when multiple instances of a configuration need to be produced (e.g. for each of several elevator settings) these mesh generation jobs are all executed in parallel.[2,17] This architecture, combined with the underlying speed of the mesh generation and geometry manipulation processes implies that the speed of the flow solver is the primary driver in the total cost of producing the aerodynamic database.

In typical database fills, hundreds or thousands of cases need to be run. Under these circumstances, computational efficiency dictates running as many cases simultaneously as memory permits, and this strategy maps well to the Columbia system. The 3-10 million cell cases typically fit in memory on 32-128 CPUs, making it possible to run several cases simultaneously on each 512 CPU node of the system. Such cases can be run using either OpenMP or MPI communication.

V. Cart3D Flow Solver and Parallelization

Despite the "embarrassingly parallel" nature of database fills, there is still a strong demand for the ability to run extremely large cases, or individual cases extremely rapidly. Obviously when running 10^5 or 10^6 cases, there is little demand to thoroughly peruse the results of each simulation, and in general, the only data stored for these cases are surface pressures, convergence histories and force and moment coefficients. If, during review of the results, the database shows unexpected results in a particular region, those cases are typically re-run on-demand. The ability to re-create the full solution extremely rapidly by spreading

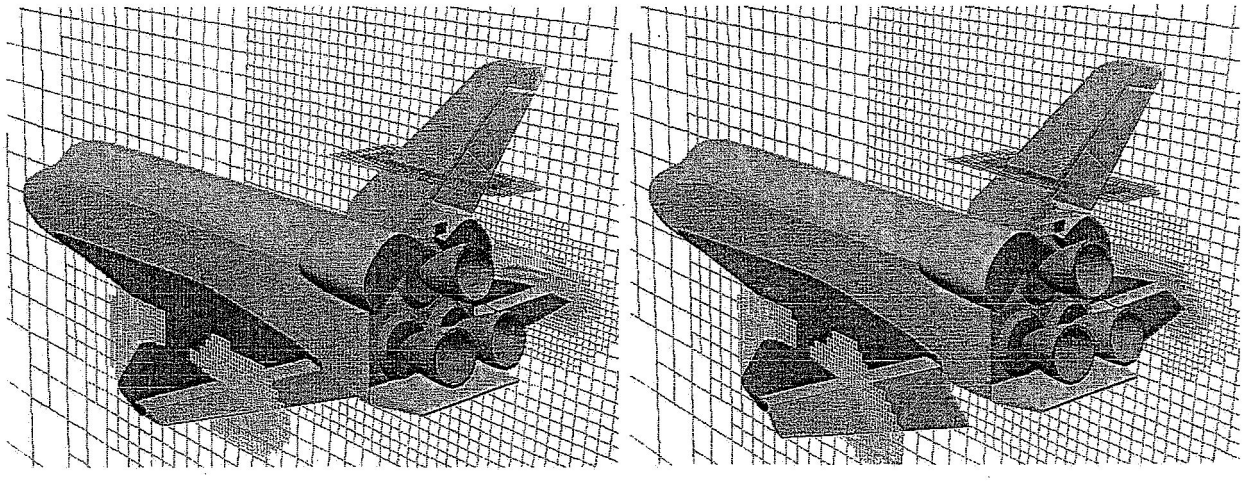


Figure 8. Embedded-boundary Cartesian mesh around two instances of space shuttle orbiter configuration showing automatic mesh response to capture deflection of the elevon control surface.

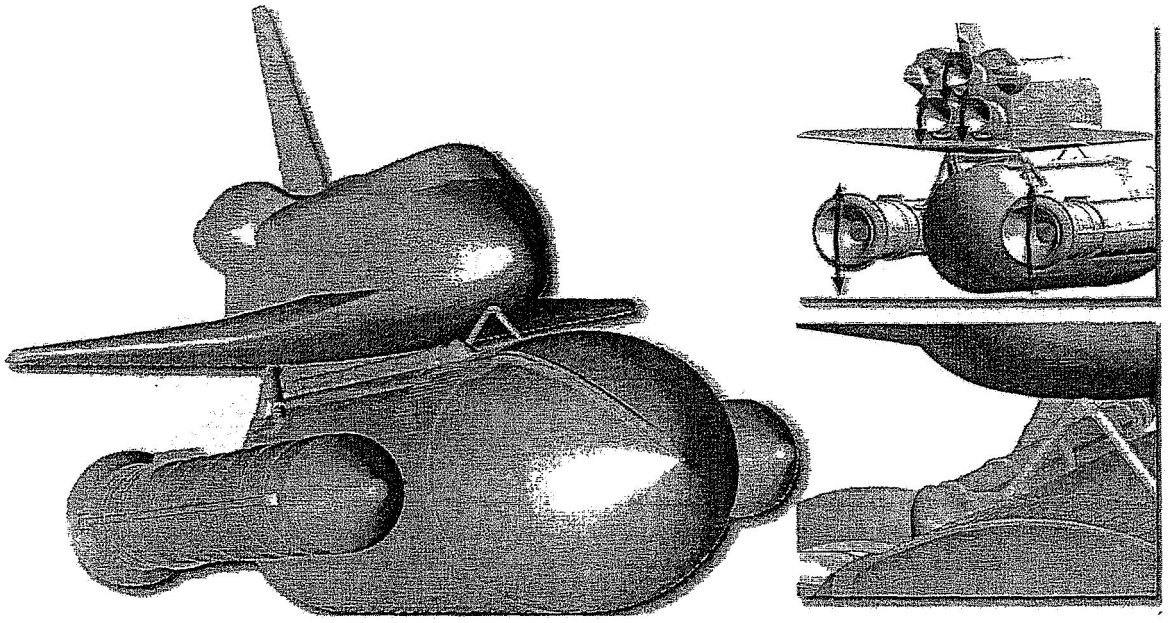


Figure 9. Surface triangulation of SSLV geometry including detailed models of the orbiter, solid rocket boosters, external tank, attach hardware and five engines with gimbaling nozzles. 1.7 million elements.

the job to thousands of processors provides a "virtual database" of the full solution data. In many cases, it is actually faster to re-run a case than it would be to retrieve it from mass storage. In addition to these on-demand detailed queries, there is often a need to compute a case on a much larger mesh than the relatively small meshes used in database fills. This need may be triggered by the desire to compare detailed flow structures with the Navier-Stokes design code, or in performing mesh refinement studies to establish meshing parameters or to verify results. Thus there is great pressure for the same solver to perform well on thousands of CPUs.

Cart3D is a simulation package targeted at conceptual and preliminary design of aerospace vehicles with complex geometry. It is in widespread use throughout NASA, the DoD, the US intelligence industry and within dozens of companies in the United States. The flow simulation module solves the Euler equations governing inviscid flow of a compressible fluid. Since these equations neglect the viscous terms present in the full Navier-Stokes equations, boundary-layers, wakes and other viscous phenomena are not present in the simulations. This simplification removes much of the demand for extremely fine meshing in the wall normal direction that Navier-Stokes solvers must contend with. As a result, the meshes used in inviscid analysis are generally smaller and simpler to generate than those required for viscous solvers like NSU3D. This simplification is largely responsible for both the degree of automation available within the Cart3D package and the speed with which solutions can be obtained. Despite this simplification, inviscid solutions have a large area of applicability within aerospace vehicle design as there are large classes of problems for which they produce excellent results. Moreover, when significant viscous effects are present, large numbers of inviscid solutions can often be corrected using the results of a relatively few full Navier-Stokes simulations.

Cart3D's solver module uses a second-order cell-centered, finite-volume upwind spatial discretization combined with a multigrid accelerated Runge-Kutta scheme for advance to steady-state.[14] Figure 8 shows how the package automatically adapts the embedded-boundary Cartesian grid to capture control surface deflection of a particular geometry. This flexibility is a key ingredient in the automation of configuration-space parameters.

Like NSU3D, Cart3D uses a variety of techniques to enhance its efficiency on distributed parallel machines. It uses multigrid for convergence acceleration and employs a domain-decomposition strategy for subdividing the global solution of the governing equations up among the many processors of a parallel machine.[14,18,19] The same multigrid cycling strategies as shown in Figure 4 are used by Cart3D's solver module, and as with NSU3D, W-cycles are preferred. Rather than relying upon agglomeration and METIS, both the mesh coarsener and mesh partitioner in Cart3D take advantage of the hierarchical nesting of adaptively refined Cartesian meshes. As detailed in reference [18], the techniques are based upon a Space-Filling-Curve (SFC)

reordering of the adapted meshes. Figure 10 illustrates this ordering using a 2D example mesh around a NACA 0012 airfoil. For illustration purposes this 2D example shows the cells ordered using the Morton SFC, however in 3D the Peano-Hilbert SFC is generally preferred.[18] The construction rules for these SFC's are such that a cell's location on the curve can be computed by one-time inspection of the cell's coordinates, and thus the reordering process is bound by the time required to quicksort the cells.

Examining the ordering in Figure 10, the coarse mesh generation process becomes clear. Tracing along the SFC, cells that collapse into the same coarse cell ("siblings") are collected whenever they are all the same size, and the corresponding coarse cell is inserted into a new mesh structure. This process builds the coarse mesh cell-by-cell. An additional benefit of this single-pass construction algorithm is that the coarse mesh is automatically generated with its cells already ordered along the SFC. Thus, this coarse mesh is immediately available for further coarsening by the same traversal algorithm. Numerical experiments with this coarsening procedure show that it achieves coarsening ratios in excess of 7 on typical examples.[18] Figure 11 shows a coarsening sequence around a re-entry vehicle geometry. The fine grid is on the left, and the coarsest mesh is on the right. Each frame in this figure shows the mesh partitioned into 2 subdomains using the SFC as a partitioner. The mesh partitioner actually operates on-the-fly as the SFC-ordered mesh file is read. The locality properties of the SFC ordering are such that a good partitioning strategy is to simply distribute different segments of the SFC among the various processors. For example, if the mesh in figure 10 were to be divided into 2 subdomains, dividing the SFC in half would result in two subdomains which split the mesh vertically down the center. Quartering the SFC would result in 4 square subdomains with the airfoil at the center. Results in reference [18] indicate that the surface-to-volume ratio of these SFC-derived partitions track that of an idealized cubic partitioner.

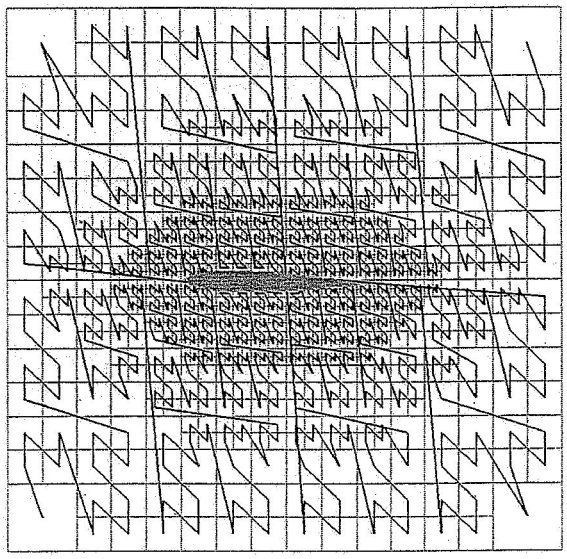


Figure 10. Space-Filling-Curve (Morton order) illustrating reordering of adaptively-refined Cartesian mesh around a 2-D airfoil. In three dimensions, Cart3D uses either Peano-Hilbert or Morton SFC's for both mesh coarsening and domain-decomposition.

Figure 11 shows each mesh in the multigrid hierarchy partitioned into two subdomains. All meshes in

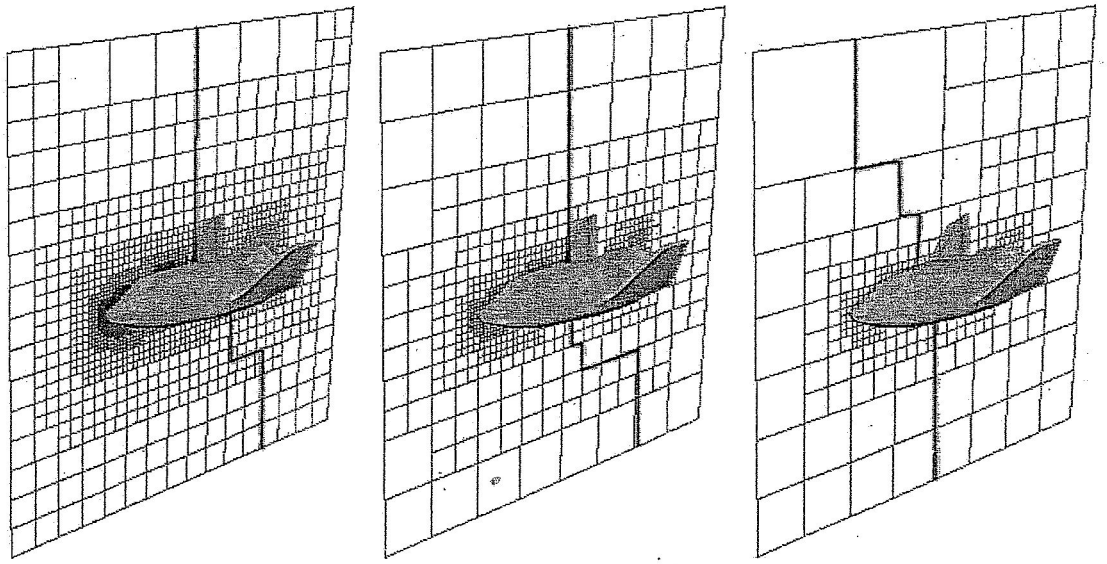


Figure 11. Example multigrid mesh hierarchy for re-entry vehicle geometry, fine mesh on left, coarsest on right. The partitioning is shown for 2 subdomains and each mesh in the hierarchy is partitioned independently using the same SFC.

the hierarchy are partitioned independently using the same SFC. This implies that although there will be generally very good overlap between corresponding fine and coarse partitions, they are not perfectly nested. The slight changes in partition boundaries in the figure make this clear. While most of the communication for multigrid restriction and prolongation in a particular subdomain will take place within the same local memory, these operators will incur some degree of off-processor communication as well. As with NSU3D, this approach favors load-balancing the work on each mesh in the hierarchy at the possible expense communication.

Figure 12 shows an example of an adapted Cartesian mesh around the full SSLV configuration. This mesh contains approximatly 4.7M cells with 14 levels of adaptive subdivision. The mesh is illustrated with a single cutting plane through the domain. The grid in in this figure is painted to indicate its partitioning into 16 subdomains using the Peano-Hilbert SFC. Partition boundaries in this example were chosen for perfect load-balancing on homogeneous CPU sets and cut-cells were weighted 2.1 times more heavily than un-cut Cartesian hexahedra. The partitions in this example are all predominantly rectangular as is characteristic of subdomains generated with SFC-based partitioners.

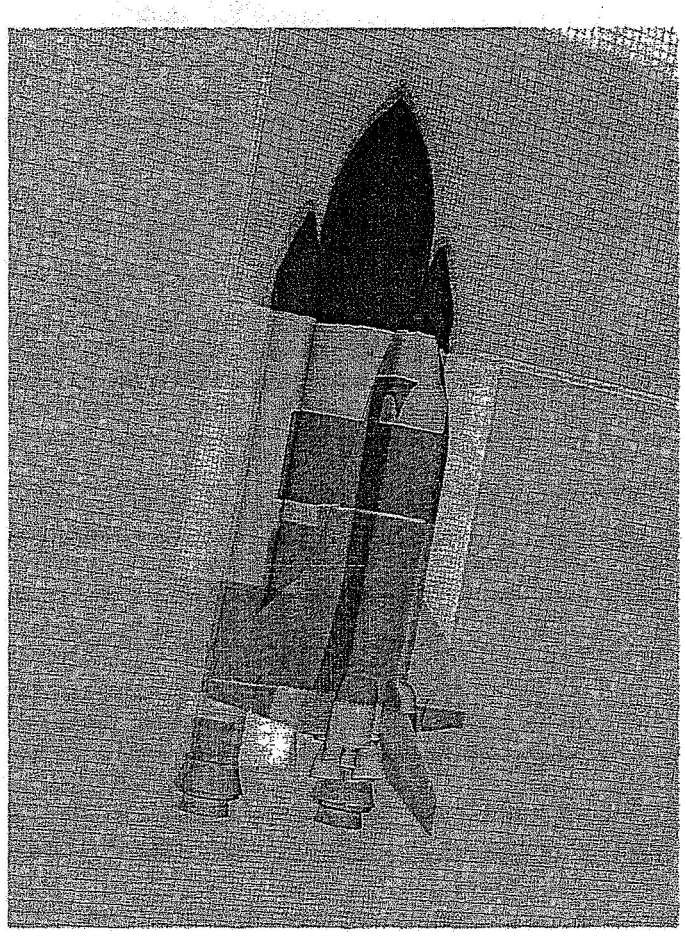
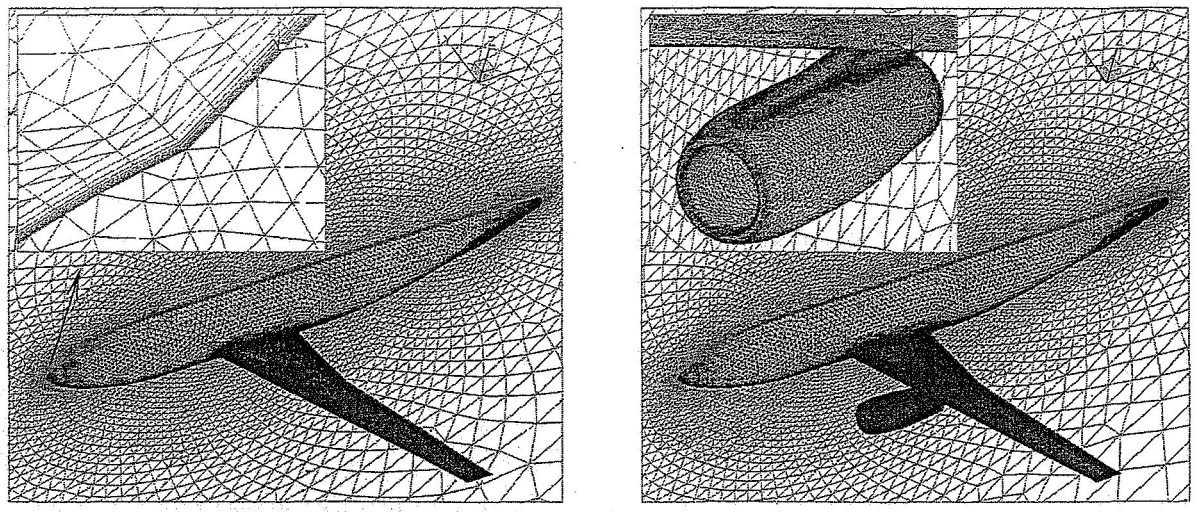


Figure 12. Cartesian mesh around full SSLV configuration including orbiter, external tank, solid rocket boosters, and fore and aft attach hardware. Mesh color indicates 16-way decomposition of 4.7M cell using the SFC partitioner in reference.[18]

VI. Performance and Scalability of NSU3D for High-Fidelity Analysis

Figure 13 illustrates a coarse unstructured mesh over two aircraft configurations, similar to the finer mesh used in the benchmark NSU3D simulations on the NASA Columbia supercomputer. While the displayed mesh in Figure 13 (a) contains a total of 1 million grid points, the fine benchmark mesh contains a total of 72 million points on the same configuration. The displayed meshes and configurations are taken from the AIAA drag prediction workshop study.[1] Subsequent studies comparing various CFD codes on this configuration have shown that this level of grid resolution, and even additional levels of refinement (leading to a total of 9 million grid points) are inadequate for the level of accuracy desired in the aircraft design process.[20,21] Therefore, a finer grid of 72 million points (315 million cells) was generated and used for the benchmarks. The accuracy of the results computed on this mesh is examined in detail in reference[22] by comparing these results with results obtained on the coarser grid levels, and with computations performed on an alternate mesh of 65 million points, using a different mesh topology in the wing trailing edge region. The results show substantial differences remain even at these high resolution levels, and make the case for the use of even finer grids, or at least for a better distribution and topology of mesh points in critical regions of the domain.

Figure 14 (a) depicts the convergence to steady-state achieved on the 72 million point grid using four, five, and six agglomerated multigrid levels, using a multigrid W-cycle in all cases. This problem contains a total of 433 million degrees of freedom, since each fine grid point contains 6 quantities. The flow conditions are determined by a freestream Mach number of 0.75, an incidence and sideslip angle of 0 degrees, and a Reynolds number of 3 million (based on the mean aerodynamic chord). For the five and six multigrid-level runs, the solution is adequately converged in approximately 800 multigrid cycles, while the four-level

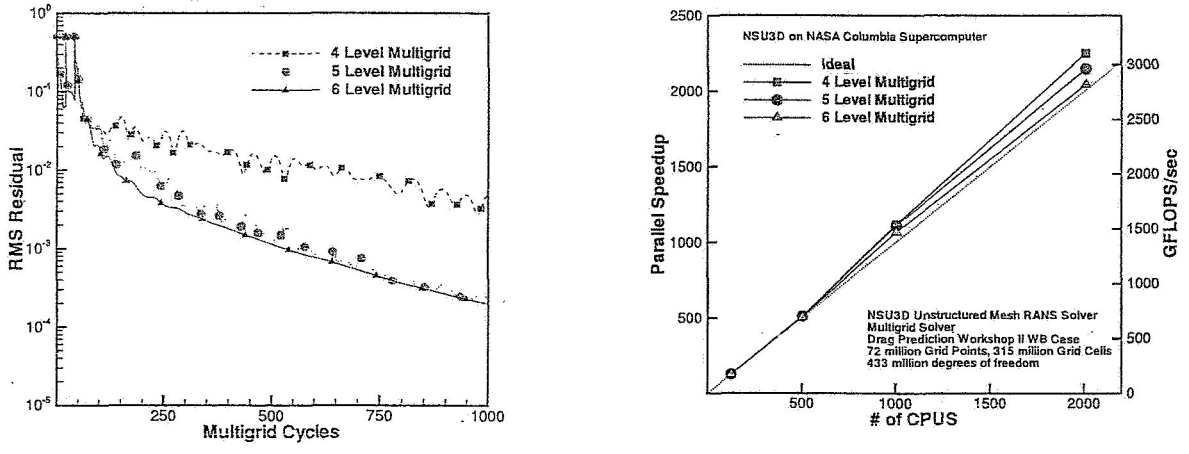


(a) (b)

Figure 13. (a): Illustration of coarse mesh about aircraft without and with engine nacelle configuration showing detail of anisotropic prismatic grid layers near aircraft surface (a), and details near engine nacelle (b). Mesh (a) contains 1 million grid points. Mesh (b) contains 1.9 million grid points. Fine mesh test case (not shown) contains 72 million grid points.

multigrid run suffers from slower convergence. Note that the single grid case (i.e. fine grid only without multigrid) would be very slow to converge, requiring several hundred thousand iterations for a mesh of this size. Figure 14 (b) depicts the parallel speedup and total number of floating point operations achieved for this case on the NASA Columbia supercomputer, using up to 2008 CPUs. The identical problem was run on 128, 256, 502, 1004, and 2008 CPUs. Assuming a perfect speedup on 128 CPUs, the four-level multigrid run achieves a superlinear speedup of 2250 on 2008 CPUs, while the six-level multigrid run achieves 2044 on 2008 CPUs. Note that the single grid case (not shown) achieves even higher speedup (2395) than the four-level multigrid case, but does not constitute a practical solution strategy.

The superlinear speedup in these cases is likely due to favorable cache effects for the decreased partition sizes on large CPU counts. On the other hand, the reduction in scalability with additional multigrid levels is due to the increased communication requirement of the coarsest levels, which contain minimal amounts of computational work, but span the same number of processors as the finest grid level. In fact, the coarsest



(a) (b)

Figure 14. (a): Multigrid Convergence Rate using 4, 5, and 6 grid levels for NSU3D Solution of Viscous Turbulent Flow over Aircraft Configuration. (b): Scalability and Computational Rates Achieved on NASA Columbia Supercomputer for NSU3D Solution of Viscous Turbulent Flow over Aircraft Configuration.

(6th) level contains only 8188 vertices, and in the case of the 2008 CPU run, the average partition contains only 4 or 5 coarse grid points, with some of the coarsest level partitions being empty (i.e. containing no grid vertices at all) due to minor imbalances in the partitioning process. Note also that within the context of a multigrid W-cycle (c.f. Figure 4(b)), the coarsest level is visited $2^{n-1} = 32$ times for a six-level multigrid cycle (which corresponds to a single fine grid visit). Nevertheless, all three multigrid cases achieve better than ideal speedup in going from 128 to 2008 CPUs, due to the fact that the majority of the work and communication is performed on the finest grid levels. Considering that even on the finest grid, the average partition contains only approximately 36,000 grid points for 2008 CPUs, this level of scalability is rather impressive.

The number of floating point operations (FLOPS) was measured using the Itanium hardware counters through the "pfmon" interface. The difference in the number of FLOPS recorded for a five multigrid-cycle run and a six multigrid-cycle run were recorded, in order to get a FLOP number for a single multigrid cycle. This number was then divided by the amount of wall-clock time required for a single multigrid cycle for the various runs on different numbers of processors. In this approach, the FLOP count was determined by disabling the MADD feature on the compiler, while the timings were obtained with the MADD feature enabled, thus resulting in the counting of MADD operations (combined Multiply-Add) as 2 FLOPS. Using this approach, the single grid run achieved a computational rate of 3.4 Tflops on 2008 CPUs, while the four, five and six level multigrid runs achieved 3.1 Tflops, 2.95 Tflops, and 2.8 Tflops, respectively. When taking into account the speed of convergence of these different runs (c.f. Figure 14 (a)), the five level multigrid scheme represents the overall most efficient solution scheme. However, for robustness reasons, we prefer to use the six level multigrid scheme which delivers the most consistent convergence histories over a wide range of flow conditions. On 2008 CPUs, a six level multigrid cycle requires 1.95 seconds of wall clock time, and thus the flow solution can be obtained in under 30 minutes of wall clock time (including I/O time). The fact that the multigrid runs with fewer grid levels deliver better scalability but lower numerical convergence illustrates the importance of balancing floating point performance with numerical algorithmic efficiency in order to obtain the most efficient overall solution strategy.

Since the 72 million point grid case can run on as few as 128 CPUs (as determined by memory requirements) and because of the demonstrated speed of this same case on 2008 CPUs, it should be feasible to run much larger grids on the four node (2048 CPU) sub-cluster of the NASA Columbia machine. For example, a case employing 10^9 grid points can be expected to require 4 to 5 hours to converge on 2008 CPUs. At present, the main issues holding back the demonstration of such large cases involve the grid generation and preprocessing operations, which are mostly sequential in nature, and the resulting file sizes. The grid input file for the flow solver in the 72 million point case measures 35 Gbytes, and increasing the grid size by another order of magnitude will certainly produce I/O bottlenecks particularly considering the transfer rates typically encountered between the compute servers and the mass storage system.

On the other hand, there are compelling reasons to seek further speedup of the existing 72 million grid point case, by going to even higher processor counts. For example, in the case of a design optimization problem, multiple analysis runs are required throughout the design process. Even for relatively efficient adjoint-based design-optimization approaches,[23–25] as many as 20 to 50 analysis cycles may be required to reach a local optimum, which would require up to 24 hours on the 72 million point grid running on 2008 CPUs. We are thus interested in examining the speedup achievable for the 72 million point case on even higher processor counts, using up to 4016 CPUs. However, in order to run a case on more than 2048 CPUs, we are faced with certain hardware limitations of the NASA Columbia machine. Notably, the current NUMAlink interconnect only spans 2048 CPUs, and therefore the InfiniBand interconnect must be used to access larger numbers of processors. Additionally, the limitation on the number of MPI processes under the InfiniBand interconnect (c.f. eq. (1)), which corresponds to a total of 1524 MPI processes, results in the requirement of using a combined OpenMP/MPI approach for accessing the required number of processors.

In order to study the effects of these limitations, we begin with a study of the performance of the 72 million point grid case on 128 CPUs, using the hybrid OpenMP/MPI communication strategy, and comparing the observed performance for the same cases using the NUMAlink and InfiniBand interconnects. The baseline case consists of the 6 level multigrid problem running on 128 CPUs, using MPI exclusively within one compute node (512 CPUs). This case was also run using 128 CPUs across two compute nodes (using 64 CPUs in a node), and across four compute nodes (using 32 CPUs in a node), making use of the NUMAlink interconnect between the nodes. In all cases, the timings were essentially indistinguishable, and averaged 31.3 seconds per multigrid cycle. Using this as the reference time, Figure 15 compares the relative efficiency

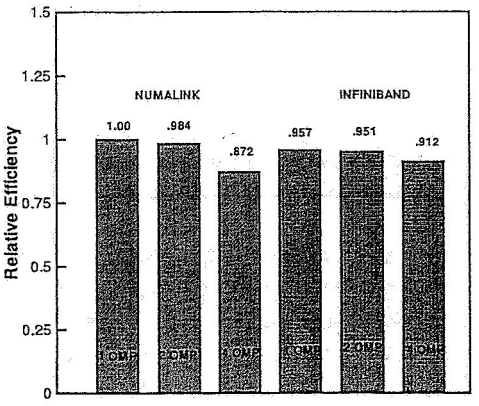


Figure 15. Relative parallel efficiency for 72 million point six-level multigrid case on 128 processors distributed over four compute nodes, using the NUMAlink interconnect versus the InfiniBand interconnect and using from 1 to 4 OpenMP threads per MPI process.

using the InfiniBand interconnect for four compute nodes, and using 128 MPI processes with 1 thread per process, 64 MPI processes with 2 OpenMP threads per MPI process, and 32 MPI processes with 4 OpenMP threads each. In all cases, the degradations in performance from the baseline case are relatively minor. Using 2 and 4 OpenMP processes with the NUMAlink interconnect the efficiency decreases to 98.4% and 87.2% respectively (i.e. time per cycle increases by the inverse of the efficiency). This penalty may be due to the loss of local parallelism (at the OpenMP thread level) during the MPI to MPI communication, which is carried out by the master thread on each MPI process.[12] The InfiniBand results show similar behavior, although the degradation in performance in going from NUMAlink to InfiniBand is minimal (95.7% efficiency for the pure MPI case, with InfiniBand, actually outperforming the NUMAlink for the 4 thread OpenMP/MPI case). Only the results using four compute nodes are shown, since the timings using two and four nodes are essentially identical.

Figure 16 (a) depicts the scalability using NUMAlink and InfiniBand for the combined OpenMP/MPI code using 1 or 2 OpenMP threads, for the single grid (no multigrid) case from 128 up to 2008 CPUs. Note that on 2008 CPUs, the InfiniBand case can only be run using 2 OpenMP threads per MPI process, due to the limitation on the number of MPI processes (i.e. 1524) under InfiniBand. These results mirror those observed on 128 CPUs, showing only slight degradation in overall performance between the NUMAlink and

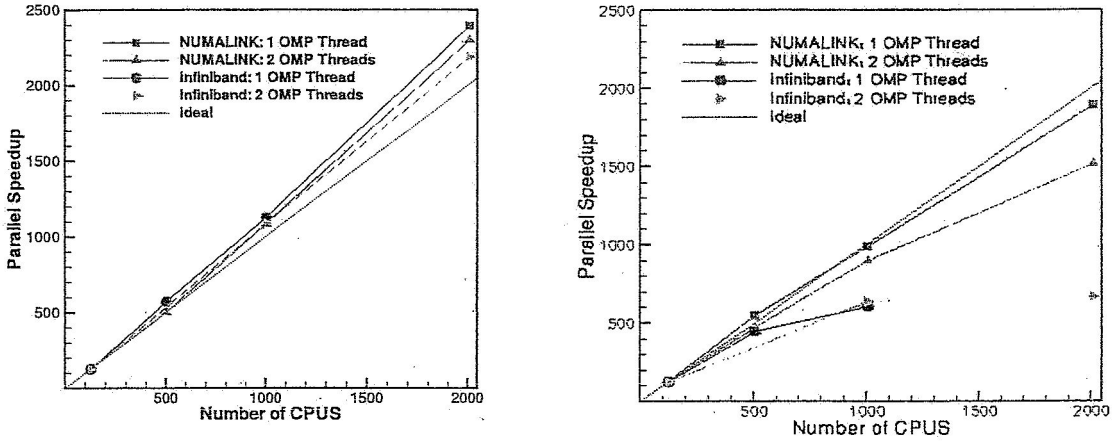
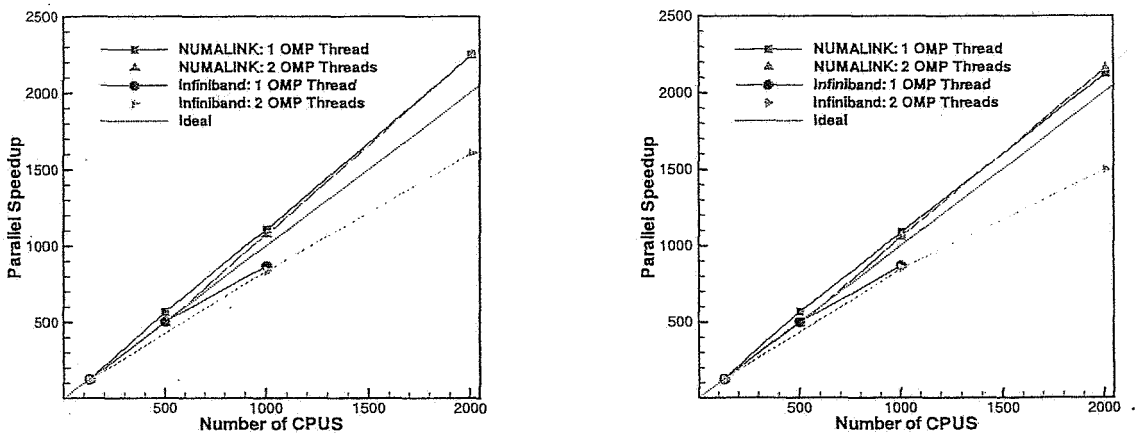


Figure 16. Parallel speedup observed for 72 million point grid comparing NUMAlink versus InfiniBand interconnect, and using 1 or 2 OpenMP threads per MPI process for single grid case (a), and for six-level multigrid case (b).

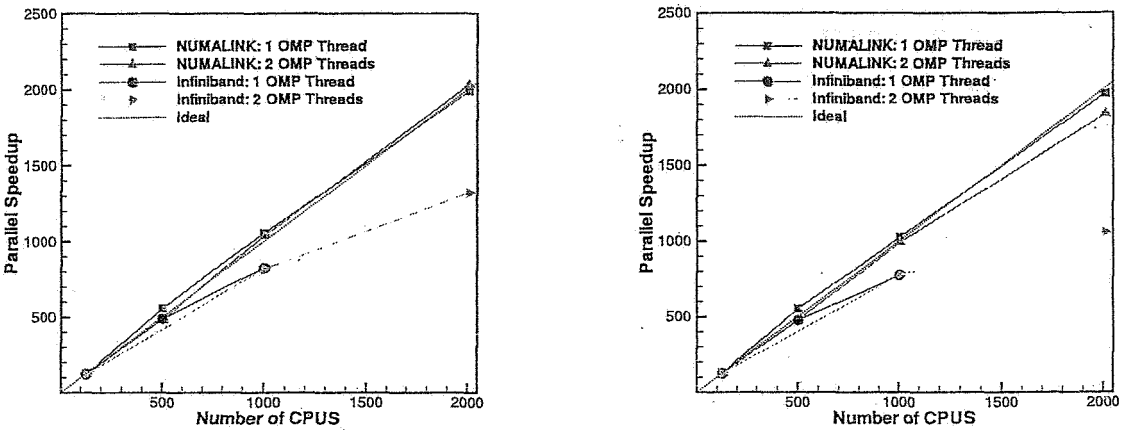


(a) (b)

Figure 17. (a): Parallel speedup observed for 72 million point grid comparing NUMAlink versus InfiniBand interconnect, and using 1 or 2 OpenMP threads per MPI process for two-level multigrid case (a), and for three-level multigrid case (b).

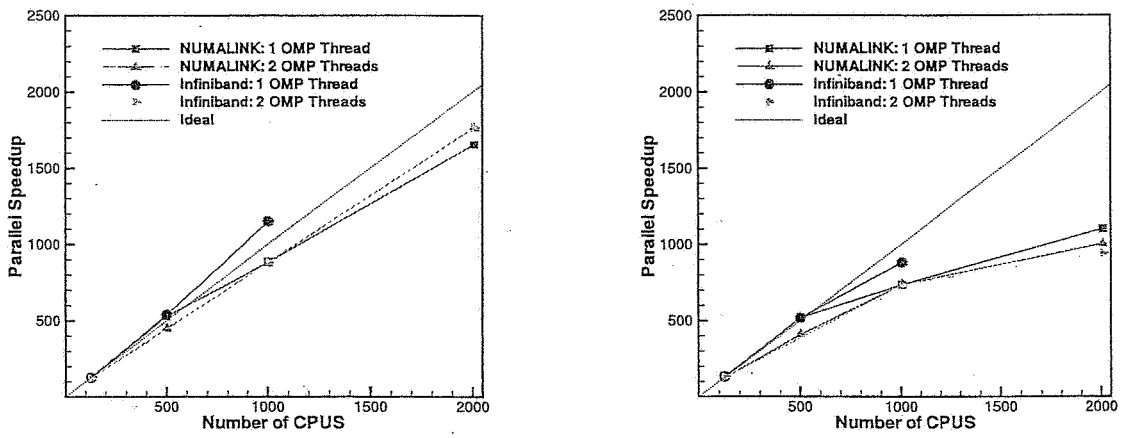
the InfiniBand interconnects, and an additional slight degradation in going from 1 to 2 OpenMP treads per MPI process. Note that in all cases, superlinear speedup is still achieved at 2008 CPUs.

Figure 16 (b) depicts the same scalability results for the six-level multigrid solver, which is the preferred solution algorithm for the 72 million point case. The performance degradation due to the use of 2 OpenMP threads is somewhat larger than in the single grid case, although it is still modest. (Note that the scalability of the baseline case, NUMAlink with 1 OpenMP thread for six-level multigrid, is slightly lower than that observed in Figure 14 (b). This may be due to different compiler options used to invoke OpenMP, and/or to variations in the state of the hardware, since these test were performed several weeks apart). However, the degradation in performance due to the use of InfiniBand over NUMAlink is dramatic, particularly at the higher processor counts. This may be attributable to the lower bandwidth of the InfiniBand for the increased communication required by the coarser levels of the multigrid sequence. In order to further investigate this behavior, scalability studies have been run for the two-level, three-level, four-level, and five-level multigrid solvers, as shown in Figures 17 and 18. As expected, a gradual degradation of performance is observed as the number of multigrid levels is increased. However, even the two level multigrid case shows substantial degradation between the NUMAlink and InfiniBand results. In Figure 19 (a) the second grid in the multigrid



(a) (b)

Figure 18. (a): Parallel speedup observed for 72 million point grid comparing NUMAlink versus InfiniBand interconnect, and using 1 or 2 OpenMP threads per MPI process for four-level multigrid case (a), and for five-level multigrid case (b).



(a) (b)

Figure 19. Parallel speedup observed for second coarse multigrid level alone (9 million grid points) (a) and for third multigrid level alone (1 million grid points) (b) comparing NUMAlink versus InfiniBand interconnect, and using 1 or 2 OpenMP threads per MPI process.

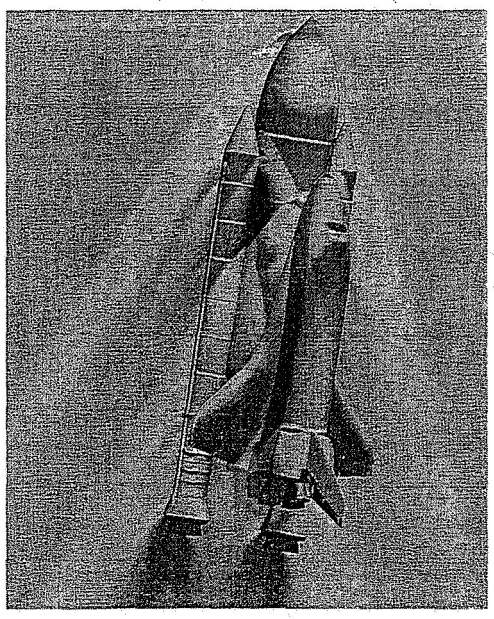
sequence, which contains approximately 9 million points, is run by itself, without the finer grid, or any coarser multigrid levels, to examine the scalability on this grid alone.As expected, this coarser grid level does not scale as well as the finer 72 million point grid. However, both the NUMAlink and InfiniBand results degrade at similar rates, and deliver similar performance even on 2008 CPUs. Analogous results are found for the next coarser multigrid level (which contains approximately 1 million points) in Figure 19 (b). These findings suggest that the increased communication generated by the coarser multigrid levels is not responsible for the differences observed between the NUMAlink and InfiniBand scalabilities of the full multigrid algorithm.

The other main source of communication in the multigrid algorithm occurs in the inter-grid transfer phase, when transferring solution quantities from fine to coarse (restriction operation) and from coarse to fine (prolongation operation) grids. Although the volume of communication data in these operations is estimated to be lower than in the intra-grid communication routines, because the coarse and fine levels are non-nested, these communication operations may be less local than those performed on each level, although the number of neighbors in the communication graph is approximately the same in both cases (i.e. the maximum degree of the fine grid communication graph is 18, while the maximum degree of the inter-grid communication graph is 19). In reference,[4] severe degradation of the InfiniBand latency and bandwidth was observed for a Random Ring communication benchmark, and we speculate that the performance of the inter-grid multigrid communication operations may be related to this effect.

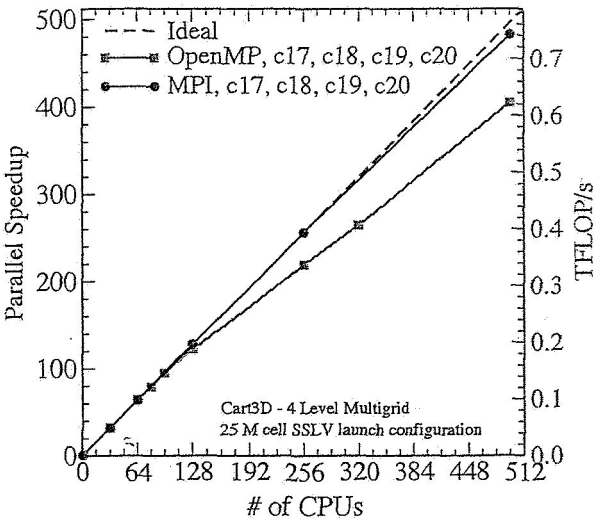
Given the results of Figure 16 (a), we may expect the single grid case for 72 million points to scale relatively well on 4016 CPUs, using the InfiniBand interconnect, and 4 OpenMP processes per MPI process (as dictated by the available number of MPI processes under InfiniBand). However, the multigrid algorithm using any number of grid levels will most likely perform no better on 4016 CPUs, than on 2008 CPUs using the NUMAlink. However, the results obtained on 128 CPUs (c.f. Figure 15) suggest that a larger multigrid case (of the order of 10^9 grid points with 7 multigrid levels) would perform adequately on 4016 CPUs, delivering of the order of 5 to 6 Tflops. In order to obtain good performance with the 72 million point multigrid case, the exact cause of the InfiniBand performance degradation must be determined and resolved if possible.

VII. Performance and Scalability of Cart3D on Large Problems

To assess performance of Cart3D's solver module on realistically complex problems, several performance experiments were devised examining scalability for a typical large grid case. The case considered is based on the full Space Shuttle Launch Vehicle (SSLV) example shown earlier (Fig. 12). For scalability testing the mesh density was increased to 25M cells, which is about twice as fine as that shown in Figure 12. Cart3D's solver module solves five equations for each cell in the domain giving this example approximately 125M degrees-of-freedom. The geometry includes detailed models of the orbiter, solid rocket boosters, external



(a)



(b)

Figure 20. (a):Pressure contours around full SSLV configuration including orbiter, external tank, solid rocket boosters, and fore and aft attach hardware for benchmarking case described in text. (b): Parallel scalability of Cart3D solver module on Columbia using SSLV example on 25M cell mesh. Runs conducted on single 512 CPU node of Columbia system.

tank, five engines, and all attach hardware. The geometry in this example also includes the modifications to the external tank geometry as part of NASA's Return-to-Flight effort. Figure 20(a) shows pressure contours of the discrete solution at Mach = 2.6, angle-of-attack = 2.09 deg. and 0.8 degrees sideslip. The surface triangulation contains about 1.7M elements. An aerodynamic performance database and virtual-flight trajectories using this configuration with power on was presented in 2004.[2]

This example was used for several performance experiments on the Columbia system. These experiments included comparisons of OpenMP and MPI, the effects of multigrid on scalability, and comparisons of the NUMAlink and InfiniBand communication fabrics. The baseline solution algorithm used 4 levels of multigrid, and unless otherwise stated, all results are with this scheme.

As discussed earlier, Cart3D's solver module can be built against either OpenMP or MPI communication libraries. On the Columbia system, the 1Tb of each 512 CPU node is globally sharable to any process within the node, but cache-coherent shared memory is not maintained between nodes. Thus, pure OpenMP codes are restricted to, at most, the 512 CPUS within a single box. Figure 20(b) shows scalability for the same problem using both OpenMP and MPI. These cases were run on CPU sets with 32 to 504 processors on Columbia node c18. In computing parallel speedup, perfect scalability was assumed on 32 CPUs. Performance with both programming libraries is very nearly ideal, however while the MPI shows no appreciable degradation over the full processor range, the OpenMP results display a slight break in the slope of the scalability curve near 128 CPUs. Beyond this point the curve is again linear, but with a slightly reduced slope. This slight degradation is most probably attributable to the routing scheme used within the Altix nodes. The 512 CPU nodes are built of four 128 CPU double cabinets, within any one of these, addresses are dereferenced using the complete pointer. More distant addresses are dereferenced using "coarse mode" which drops the last few bits of the address. On average, this translates into slightly slower communication when addressing less local memory. Since only the OpenMP uses this global address space, the MPI results is not impacted by this pointer swizzling.

The right axis of the speedup plot in Figure 20 is scaled in TFLOP/s for the baseline solution algorithm. As with NSU3D, FLOP/s were counted by interrogating the Itanium2's hardware counters using Intel's "pfmon" interface. Operations were counted for a single multigrid cycle and then divided by the time per iteration on various numbers of processors to provide this scale. In establishing this scale, MADD operations were counted as two operations. Substantial work on optimizing single CPU performance with this code has resulted in somewhat better than 1.5 GFLOP/s on each CPU. When combined with linear parallel speedup, this produces around 0.75 TFLOP/s for the code on 496 processors of a single Columbia node.

With single node performance solidly in the same range as that of NSU3D, our investigations now focus on performance across multiple nodes of the Columbia system. These experiments were carried out on nodes c17-c20, all of which are part of the Columbia's "Vortex 3700" subsystem. They use the BX2 routers, have double density processor bricks, and are connected using NUMAlink, InfiniBand, and 10Gig-E. Since the system is not cache-coherent across all 4 of these nodes and the solver module does not have a hybrid OpenMP+MPI build mode, performance was evaluated using MPI only.

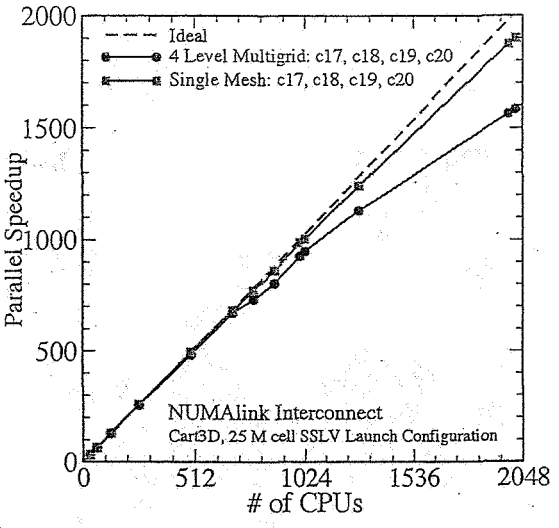


Figure 21. Comparison of parallel speedup of Cart3D solver module using 1 and 4 levels of mesh in the multigrid hierarchy. NUMAlink interconnect.

Figure 21 examines parallel speedup for the system comparing the baseline four level multigrid solution algorithm with single grid. This experiment was carried out exclusively using the the NUMAlink interconnect, and spanned from 32-2016 CPUs. As with the study in Figures 14 - 16 for NSU3D, reducing the number of multigrid levels de-emphasizes communication (relative to floating-point performance) in the solution algorithm. Scalability for the the single grid scheme is very nearly ideal, achieving parallel speedups of about 1900 on 2016 CPUs. Its clear that even on the NUMAlink, communication is beginning to effect scalability of the multigrid. This is not surprising, with only 25M cells in the fine mesh (12000 cells/partition on 2016 CPUs), the coarsest mesh in the multigrid sequence has only 32000 cells giving only about 16 cells per partition on 2016 CPUs. Roll-off in the multigrid results does not become apparent until around 688 CPUSs, and does really not start to degrade until above 1024 CPUs. Given this relatively modest decrease in performance it seems clear that the bandwidth demands of the solver are not greatly in excess of that delivered by the NUMAlink. With 2016 CPUs and 4 levels of multigrid the NUMAlink still posts parallel speedups of around 1585.

The work in Reference 4 includes a study of delivered bandwidth and latency for both the NUMAlink and InfiniBand for a variety of different communication patterns. To understand the implications of this for Cart3D's solver module the baseline four-level multigrid scheme was re-run using the InfiniBand interconnect on the same nodes as the preceding experiment. Figure 22 displays these results plotted against those of the NUMAlink interconnect.

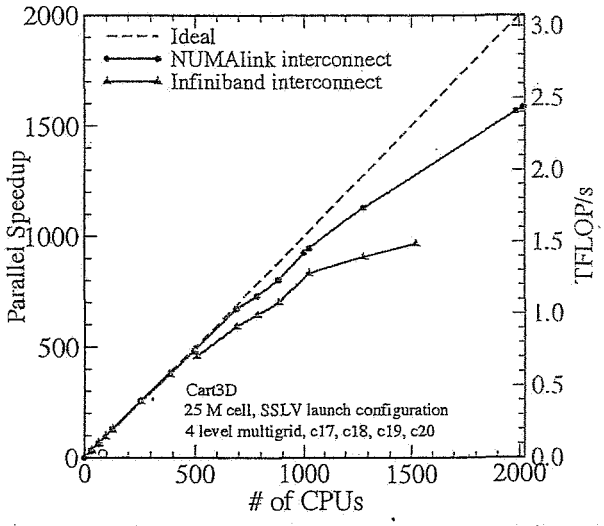


Figure 22. Comparison of parallel speedup of Cart3D solver module with 4 levels of multigrid using the NUMAlink and InfiniBand interconnect.

As before, the identical problem was run on from 32 to 2016 CPUs using MPI. Note that results with the InfiniBand, however, do not extend beyond 1524 CPUs due to the limitation expressed in equation 1. Tracing the results, from 32-496 CPUs the cases were run on a single node and thus there is no difference between the two curves (no box-to-box communication). Cases with 508-1000 CPUs were run spanning two nodes of Columbia and some interesting differences begin to appear. While the InfiniBand consistently lags the NUMAlink, the most striking example is the case at 508 CPUs which actually underperforms the single-box case with 496 CPUs. This is consistent with the observations in reference 4 which quantify the decrease in delivered bandwidth for InfiniBand across two nodes. This work also predicts an increasing penalty when spanning 4 nodes. As expected, cases with 1024-2016 CPUs (run on 4 nodes) show a further decrease with respect to those posted by the NUMAlink. These results are also consistent with the investigations performed with NSU3D, however, the smaller problem size used here emphasizes the communication even more heavily. Performance of the NUMAlink case with 2016 CPUs is slightly over 2.4 TFLOP/s.

VIII. Conclusions and Future Work

This paper examined the parallel performance of two widely used high-performance aerodynamic simulation packages on the newly installed NASA Columbia supercomputer. These packages include both a high-fidelity, unstructured, Reynolds-averaged Navier-Stokes solver (NSU3D), and a fully-automated inviscid flow package for cut-cell Cartesian grids (Cart3D). The combination of these two simulation codes enables high-fidelity characterization of aerospace vehicle design performance over the entire flight envelope. They permit both extensive parametric analysis as well as detailed simulation of critical cases. Both packages are industrial-level codes designed for complex geometry and incorporate customized multigrid solution algorithms. Numerical performance on Columbia was examined using MPI, OpenMP and hybrid (OpenMP & MPI) communication architectures. Experiments focused on scalability to large numbers of CPUs on the Columbia system. In particular, they contrasted the performance of the NUMAlink and InfiniBand interconnect fabrics, and examined the incremental performance degradation incurred by additional communication when including very coarse grids in the multigrid scheme. Numerical results demonstrate good scalability on up to 2016 cpus using the NUMAlink4 interconnect. These examples showed linear parallel speedups and posted measured computational rates in the vicinity of 3 TFLOP/s. Both codes showed modest performance degradation at large CPU counts on the InfiniBand interconnect particularly as ever coarser grids were included in the multigrid hierarchy. These results are important since the NUMAlink spans at most four Columbia nodes and runs using more than 2048 CPUs must rely on the InfiniBand for at least a fraction of their communication. The numerical results in this study are encouraging enough to indicate that larger test cases using combined MPI/OpenMP communication should continue to get good performance improvements well beyond the four Columbia nodes used in this study.

References

[1] "Second AIAA Drag Prediction Workshop," see: http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/.

[2] Murman, S. M., Aftosmis, M. J., and Nemec, M., "Automated parameter studies using a Cartesian method," AIAA-Paper 2004-5076.

[3] Salas, M. D., "Digital Flight: The Last CFD Aeronautical Grand Challenge," Proc. of the Int. Conf. on the Research Trend of PDE Modeling and Computation. To appear: J.of Scientific Computing.

[4] Biswas, R., Djomehri, M. J., Hood, R., Jin, H., Kiris, C., and Saini, S., "An Application-Based Performance Characterization of the Columbia Supercluster," Paper to be presented at the 2005 Supercomputing Conference, Seattle, WA.

[5] Mavriplis, D. J. and Venkatakrishnan, V., "A 3D agglomeration multigrid solver for the Reynolds-averaged Navier-Stokes equations on unstructured meshes," *International Journal for Numerical Methods in Fluids*, Vol. 23, No. 6, 1996, pp. 527–544.

[6] Mavriplis, D. J. and Venkatakrishnan, V., "A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes," *International Journal for Computational Fluid Dynamics*, Vol. 8, 1997, pp. 247–263.

[7] Mavriplis, D. J. and Pirzadeh, S., "Large-Scale Parallel Unstructured Mesh Computations for 3D High-Lift Analysis," *AIAA Journal of Aircraft*, Vol. 36, No. 6, Dec. 1999, pp. 987–998.

[8] Spalart, P. R. and Allmaras, S. R., "A One-equation Turbulence Model for Aerodynamic Flows," *La Recherche Aérospatiale*, Vol. 1, 1994, pp. 5–21.

[9] Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes," *Journal of Computational Physics*, Vol. 145, No. 1, Sept. 1998, pp. 141–165.

[10] Karypis, G. and Kumar, V., "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," Tech. Rep. 95-035, University of Minnesota, 1995, A short version appears in Intl. Conf. on Parallel Processing 1995.

[11] Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, MA, 1994.

[12] Mavriplis, D. J., "Parallel Performance Investigations of an Unstructured Mesh Navier-Stokes Solver," ICASE Report No. 2000-13, NASA CR 2000-210088.

[13] Aftosmis, M. J., Berger, M. J., and Melton, J. E., "Robust and efficient Cartesian mesh generation on component based geometry," *AIAA Journal*, Vol. 36, No. 6, June 1998, pp. 952–960.

[14] Aftosmis, M. J., Berger, M. J., and Adomavicius, G. D., "A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries," AIAA-Paper 2000-0808.

[15] Murman, S. M., Chan, W. M., Aftosmis, M. J., and Meakin, R. L., "An interface for specifying rigid-body motion for CFD applications," AIAA-Paper 2003-1237.

[16] Haimes, R. and Aftosmis, M., "On generating high-quality water-tight triangulations directly from CAD," Proc. of the Internat. Soc. for Grid Generation (ISGG) 2002, Honolulu, HI.

[17] Nemec, M., Aftosmis, M. J., and Pulliam, T., "CAD-based aerodynamic desisgn of complex configurations using a Cartesian method," AIAA-Paper 2004-0113.

[18] Aftosmis, M., Berger, M. J., and Murman, S. M., "Applications of Space-Filling-Curves to Cartesian methods in CFD," AIAA-Paper 2004-1232.

[19] Berger, M. J., Aftosmis, M. J., Marshall, D. D., and Murman, S. M., "Performance of a new CFD flow solver using a hybrid programming paradigm," *J. Parallel Distrib. Comput.*, Vol. 65, 2005, pp. 414–423.

[20]Lee-Rausch, E. M., Buning, P. G., Morrison, J. H., Park, M. A., Rivers, S. M., Rumsey, C. L., and Mavriplis, D. J., "CFD Sensitivity Analysis of a Drag Prediction Workshop Wing/Body Transport Configuration," AIAA Paper 2003-3400.

[21]Lee-Rausch, E. M., Frink, N. T., Mavriplis, D. J., Rausch, R. D., and Milholen, W. E., "Transonic Drag Prediction on a DLR-F6 Transport Configuration using Unstructured Grid Solvers," AIAA Paper 2004-0554.

[22]Mavriplis, D. J., "Grid Resolution Study of a Drag Prediction Workshop Configuration using the NSU3D Unstructured Mesh Solver," AIAA-Paper 2005-4729, presented at the 23rd AIAA Applied Aerodynamic Conference, Toronto, Canada.

[23]Jameson, A., "Aerodynamic Shape Optimization using the Adjoint Method," VKI Lecture Series on Aerodynamic Drag Prediction and Reduction, von Karman Institute of Fluid Dynamics, Rhode St Genese, Belgium.

[24]Nielsen, E. J., Lu, J., Park, M. A., and Darmofal, D. L., "An Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids," AIAA Paper 2003-0272.

[25]Mavriplis, D. J., "Formulation and Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," AIAA-Paper 2005-0319, accepted for publication in AIAA Journal.

[26]Nielsen, E. J. and Park, M. A., "Using an adjoint approach to eliminate mesh sensitivities in computational design," AIAA Paper 2005-0491.

[27]Yang, Z. and Mavriplis, D. J., "Unstructured Dynamic Meshes with Higher-order Time Integration Schemes for the Unsteady Navier-Stokes Equations," AIAA-Paper 2005-1222.