On Parallel Implementations of Dynamic Overset Grid Methods

Andrew M. Wissink
Research Scientist, MCAT Inc.

Robert L. Meakin
Research Scientist, U.S. Army Aeroflightdynamics Directorate, AMCOM

NASA Ames Research Center, Mailstop 258-1
Moffett Field, CA 94035-1000
[wissink,meakin]@nas.nasa.gov

Abstract

This paper explores the parallel performance of structured overset CFD computations for multi-component bodies in which there is relative motion between component parts. The two processes that dominate the cost of such problems are the flow solution on each component and the inter-grid connectivity solution. A two-part static-dynamic load balancing scheme is proposed in which the static part balances the load for the flow solution and the dynamic part re-balances, if necessary, the load for the connectivity solution. This scheme is coupled with existing parallel implementations of the OVERFLOW flow solver and DCF3D connectivity routine and used for unsteady calculations about aerodynamic bodies on the IBM SP2 and IBM SP multi-processors. This paper also describes the parallel implementation of a new solution-adaption scheme based on structured Cartesian overset grids.

1.0 Introduction

Unsteady prediction of viscous flows with relative movement between component parts remains an extremely challenging problem facing Computational Fluid Dynamics (CFD) researchers today. While unstructured-grid methods have shown impressive results for inviscid moving-grid simulations [1], dynamic overset structured-grid schemes based on the “Chimera” [2] approach have demonstrated the most success for viscous moving-grid problems that experience arbitrary motion. There are a number of difficult problems for which dynamic overset grid schemes have been exclusively applied, including space shuttle booster separation [3], missile store separation [4], and helicopters with moving rotor blades [5]. Navier-Stokes calculations with these methods require high computational resources that overwhelm vector-based machines like the Cray C90. Large multi-processor such as the IBM SP and Cray T3E offer the potential for dramatic increases in available computing power but scalable implementations of existing applications software must be devised if the power of these machines is to be fully realized.

A number of researchers have investigated the parallel performance of overlapping grid schemes for steady-state CFD calculations [6-8] but only the works of Weeratunga and Barszcz [9,10] directly address their parallel performance when applied to moving-grid problems. In addition to the flow calculation, moving-grid applications require that interpolation coefficients between the various overlapping grid components be updated at each timestep. The connectivity solution can constitute a significant fraction (10-50%) of the total computational work and can be difficult to parallelize efficiently. A partitioning strategy that gives optimal parallel performance in the flow solution does not necessarily give optimal performance in the connectivity solution, and vice-versa. Therefore, alternative parallel implementation approaches must be considered to attain scalable performance on large numbers of processors for moving-grid problems.

Although Weeratunga and Barszcz introduced novel parallel implementations of both the flow and connectivity solutions, they did not directly address optimal load balancing techniques for parallel efficiency of the connectivity solution. This paper seeks to extend their work by proposing a load balancing approach to improve the parallel efficiency for moving-grid applications. The paper is divided into six sections. The second section gives details about the parallel implementations for the flow solver and connectivity routine. The third section introduces a static and a dynamic load balancing approach. Section four presents results of the parallel performance on the IBM SP2 and IBM SP for unsteady Navier-Stokes calculations about three different aerodynamic configurations. Section five discusses a new approach for overset grid problems which will be followed in future work. Finally, some concluding remarks are given in section six.

2.0 Dynamic Overset Grid Scheme

The overset scheme used in this work utilizes a "Chimera" [2] style of domain decomposition in which the overall flowfield domain is divided into a system of grids which overlap one another by one or more grid cells. "Holes" are cut in grids which intersect solid surfaces and data is interpolated between the subdomains. The solution proceeds by updating, at each step, the boundary conditions on each grid with the interpolated data. This approach offers many advantages for CFD calculations about geometrically complex shapes because the complex domain can be broken into groups of relatively simple overlapping body-fitted grids and topologically simple background grids (Fig. 1).

Further, unsteady moving-grid calculations can be performed without stretching or distorting the respective grid systems. The structured system of grids allows use of the many efficient implicit solution techniques developed for structured grids, which generally do not suffer from the high memory and poor vectorized or RISC-based performance typically associated with unstructured grid solvers. Lastly, the domain decomposition nature of the approach offers a high degree of coarse-grained parallelism that can be exploited on distributed computing environments.

The need to interpolate between the various overlapping grids necessitates use of a connectivity routine. The role of the connectivity routine is to perform searches for the points that lie in the region of overlap to determine the correct interpolation coefficients from the background grid. Static-grid simulations perform the connectivity solution as a preprocessor step and use the information throughout the flow calculation. Moving-grid simulations, on the other hand, require a new connectivity solution to be performed at each timestep because the connectivity solution changes as the grids move with respect to one another.

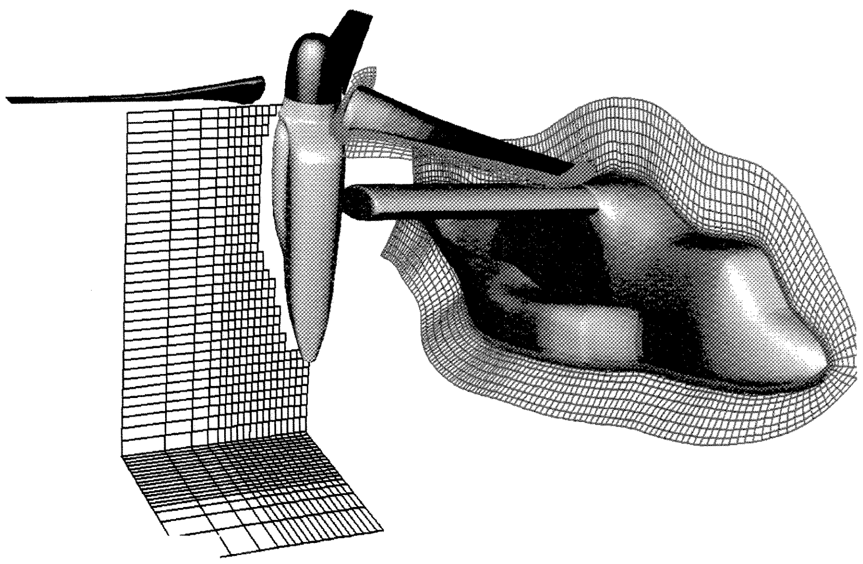


Figure 1. Overset grid system for V-22 tiltrotor

Three main pieces of software are used in the parallel dynamic overset scheme explored in this work. The flow solution on overlapping grid components is performed using NASA's structured-grid flow solver OVERFLOW by Buning [11]. Grid motion is determined by a six degree of freedom model (SIXDOF) [4], and connectivity between the overlapping grids at each timestep is performed using the code DCF3D of Meakin [12]. The parallel implementations of OVERFLOW and DCF3D previously introduced by Weeratunga [9] and Barszcz [10], respectively, are used. The load balancing scheme proposed here is incorporated and the entire package is bundled into a single code called OVERFLOW-D1.

An unsteady flow calculation using OVERFLOW-D1 involves three main steps carried out at each timestep: 1) solve the nonlinear fluid flow equations (Euler/Navier-Stokes) subject to flow and intergrid boundary conditions, 2) move grid components associated with moving bodies subject to applied and aerodynamic loads (or according to a prescribed path), 3) re-establish domain connectivity for the system of overlapping grids. Each step of the solution proceeds separately and barriers are put in place to synchronize each of the solution modules.

2.1 OVERFLOW Parallel Implementation

The parallel implementation approach of OVERFLOW uses both coarse-grained parallelism between grids and fine-grained parallelism within grids. Processor groups are assigned to each grid component and each processor executes its own version of the code for the grid or portion of the grid assigned to it (Fig. 2). The spatial accuracy is second order and temporal accuracy is first order. The solution is marched in time using a diagonalized approximate factorization scheme. Implicitness is maintained across the subdomains on each component so the solution convergence characteristics remain unchanged with different numbers of processors. More details about the flow solver and parallel implementation are available in Refs. [8,9]. The approach is designed for execution on a Multiple Instruction Multiple Data (MIMD) distributed-memory parallel computer. Communication between the processor subdomains is performed with MPI [13] subroutine calls.

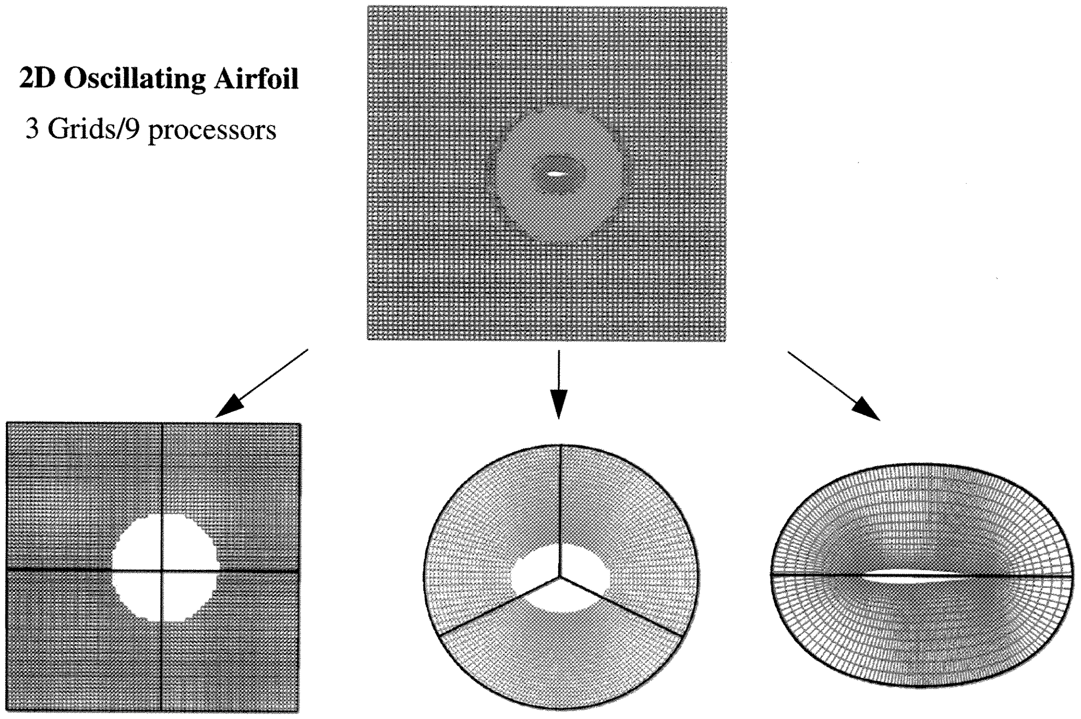


Figure 2. Grid-based parallel implementation approach for OVERFLOW

2.2 DCF3D Parallel Implementation

The parallel approach for the donor search routine in DCF3D was originally devised by Barszcz [10]. The user provides input for each grid and a list of component grids that should be searched to find donor points for each of the inter-grid boundary points (IGBPs) on the processor. The grids are listed in hierarchical manner with the corresponding grids searched in the order they are listed. The donor search procedure performed with in DCF3D proceeds as follows at each timestep:

- Each processor consults the search list to determine the component grid from which to request a search. It then checks the bounding boxes of each of the processors assigned to that grid. This can be determined locally since the bounding box information is broadcast globally at the beginning of the solution.
- Based on the bounding-box information, the processor determines which processor to send its search request to. A list of inter-grid boundary points (IGBPs) is sent to that processor which performs the search for the donor locations on its grid.
- Once all locations have been searched, the information is sent back to the calling processor and the IGBPs which have donors (i.e. those for which the search was successful) are tagged.

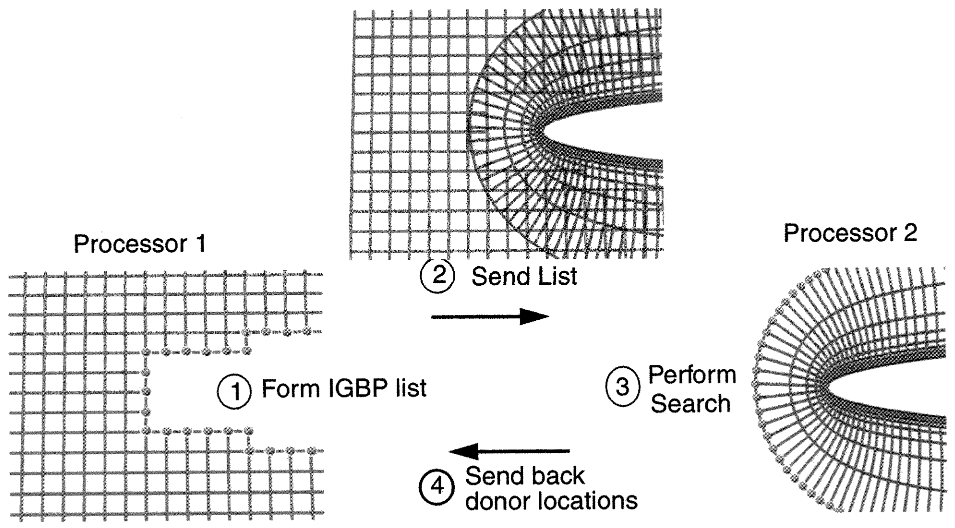


Figure 3. DCF3D parallel implementation approach - distributed donor search procedure

If all IGBPs have not received a donor, the procedure is repeated for the next grid in the hierarchy. If the search happens to hit a processor boundary, the search request is forwarded to the neighboring processor on the grid and the search is continued. This process is illustrated in Fig. 3.

The messages requesting searches are all sent asynchronously so that processors can be performing searches simultaneously. For example, after processor 1 sends a list of its IGBPs for a non-local search to processor 2, it checks if any other processors have requested searches of it. If any have, processor 1 performs the search and returns the result to the calling processor before checking if processor 2 has returned the results of its search request.

The most computationally intensive part of the connectivity solution algorithm above is performing the donor search requests (step 3). There is nearly always a load imbalance in this step because certain processors hold data where many searches take place while others hold data where few searches are required.

In performing the connectivity solution at the first timestep, nothing is known about the possible donor location and the solution must be performed from scratch. In subsequent timesteps, however, the known donor locations from the previous timestep can be used as a starting point for the searches at the new timestep. This idea, referred to as "nth-level-restart", was proposed by Barszcz [10] and was found to yield a considerable reduction in the time spent in the connectivity solution. The maximum timestep used in the problem is most often governed by stability conditions of the flow solver and this timestep is generally small enough that the donor cell locations do not move by more than one cell of the receiving grid per timestep. It is for this reason that the nth-level-restart idea tends to be quite effective at reducing the search costs.

3.0 Load Balance

Poor load balance is the primary source of parallel inefficiency using the parallel moving-body overset scheme. The two dominant computational costs in the calculation are the flow solution and domain connectivity solution and the efficiency is best if both are load balanced. Unfortunately, devising a scheme that can effectively load balance both the flow solution and connectivity solution is difficult.

The load balance of the flow solution is proportional to the volume of gridpoints on each processor. While some variation arises from differences in the computational work per gridpoint (e.g. some grids may be viscous while others are inviscid, turbulence model may be employed on some grids but not others, etc.), the differences are not substantial for the cases presented in this work.

A static load balancing scheme is applied initially which seeks to distribute the volume of gridpoints in the subdomains as evenly as possible. The routine takes as input the number of grids, their indices, and the total number of processors to be applied to the problem and determines the number of processors that should be applied to each grid to distribute the gridpoints as evenly as possible. The algorithm used to achieve this result is as follows:

Algorithm 1: Static Load Balance Routine

```
$g(n)$ = number of gridpoints in component grid $n$
$G$ = total number of gridpoints from all grids   $G = \sum g(n)$
$np(n)$ = number of processors applied to component $n$
$NP$ = total number of processors
$\tau, \Delta\tau$ = tolerance factor, increment

1. Initialize: Compute $g(n)$, $G$; set $\varepsilon = \frac{G}{NP}$ and $\tau = 0$; set $\Delta\tau$ to small value (~0.1)

2. DO until $\sum np(n) = NP$

    • $np(n) = int\left[\frac{g(n)}{\varepsilon}\right]$, with condition that $np(n) \geq 1$

    • $\tau = \tau + \Delta\tau$

    • $\varepsilon = \varepsilon \cdot (1 + \tau)$
   END DO
```

In Algorithm 1, ε is initially the most efficient load balance possible, equal to the total number of gridpoints over the total number of processors. Then, the number of subdomains *np(n)* is computed for grid *n* by dividing the number of gridpoints *g(n)* by ε. The *g(n)* value is inclusive of the points which are eventually blanked out in the connectivity solution. Unless the grids are perfectly divisible by ε, the resulting total number of subdomains is less than the number of processors. Thus, a tolerance factor τ is incremented by Δτ and the process is repeated until the number of subdomains equals the number of processors. If τ=0, the problem is perfectly load balanced. Increasingly greater values of τ indicates higher degrees of load imbalance. Thus, the tolerance factor is a measure of the degree of load imbalance.

Because Algorithm 1 uses integer arithmetic, there arises the possibility of having infinite solutions with certain grid and processor partition choices, for which the algorithm will not converge. For example, if three processors are applied to two equally-sized grids (i.e. same *g(n)*), the algorithm cannot decide which grid should receive two processors and which should receive one and will continue iterating. A special condition has been put in place for these cases; If the method proceeds for many iterations and does not converge, the value of the grid index *n* is added to *g(n)* and the method is repeated. Since *n* is generally very small relative to *g(n)*, this effectively adds a small perturbation to the system and the method will subsequently converge.

Once the number of processors applied to each grid is determined, the routine divides the grids into subdomains with the smallest surface area based on the index space of the grid. The routine forms subdomains based on the prime factors of *np(n)*. For example, if *np(n)=12*, the prime factors are 3, 2, and 2. It divides the largest dimension in the domain by 3, then subdivides the largest dimension in each subdomain by 2, and further subdivides the largest dimension in each sub-subdomain by 2. In this way, the algorithm forms subdomains which have index spaces that are as close to cubic as possible, thereby minimizing the surface area in order to minimize communication between the subdomains (Fig. 4).

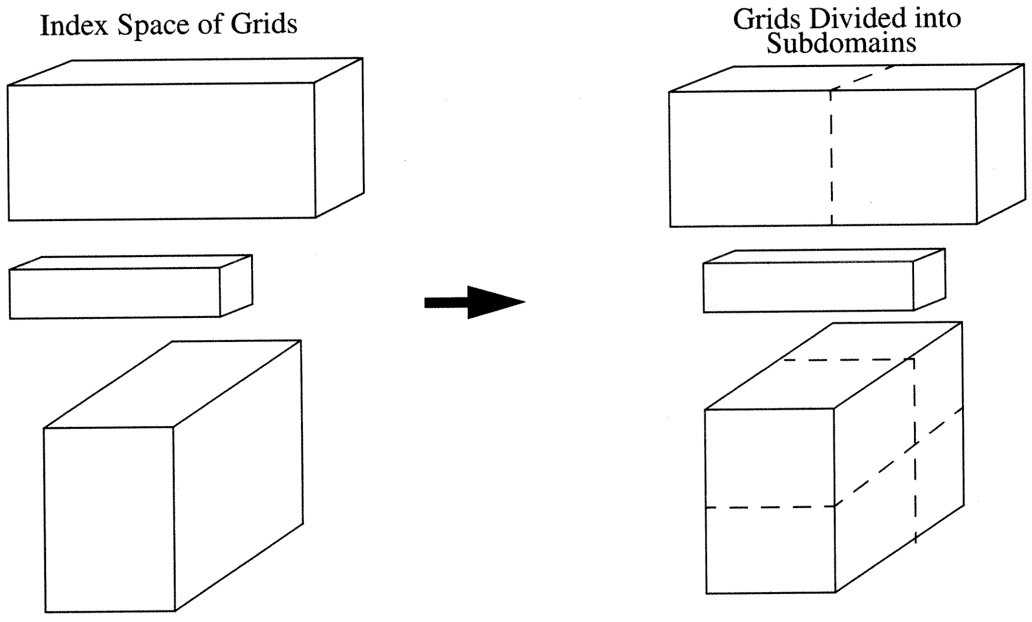


Figure 4. Static load balance routine

The static load balancing scheme is effective at load balancing the flow solution but makes no effort to load balance the connectivity solution. In some cases, namely those which have a large number of inter-grid boundary points relative to the number of gridpoints, the connectivity solution can represent a significant proportion of the computational cost. It is therefore desirable to have in place a means of load balancing the connectivity solution for use with these cases.

The primary computational costs in the connectivity solution are incurred in facilitating the donor search requests sent by other processors (step 3 in Fig. 3). The donor search costs are proportional to the number of non-local IGBPs, located on other processors, sent to the local processor in the search request. Thus, an efficient load balancing strategy for the connectivity solution should seek to equally distribute the number of non-local IGBPs sent to each processor.

A dynamic scheme is proposed for load balancing the connectivity solution. The necessity of a dynamic scheme is twofold. First, determining a prediction locally of the number of IGBPs that will be sent by other processors is difficult to do apriori because it requires all non-local data. It is generally easier to begin the calculation and determine the degree of imbalance after a few steps of the solution than it is to predict the imbalance before the start of the solution. Second, the number of search requests will change throughout a moving-grid calculation as grids pass through each other so a dynamic scheme will adapt the load balance as the problem changes. The proposed dynamic scheme is given in Algorithm 2.

Algorithm 2: Dynamic Load Balance Scheme

```
$f_o$ = user specified load balance factor
1. Begin solution - apply static load balance routine
2. Check solution after specified number of timesteps:
    • Determine $I(p)$ = #IGBPs received on processor $p$
    • Compute $\bar{I} = \frac{\sum I(p)}{NP}$ (global average over all processors)
    • Compute $f(p) = \frac{I(p)}{\bar{I}}$ for each processor. If $f(p) > f_o$, set $np(n) = np(n) + 1$,
      where $n$ is the grid component to which processor $p$ is assigned.
    • Rerun static load balance routine with above $np(n)$ condition enforced for grid $n$.
```

The role of the above scheme is to reassign the processor distribution so that a better degree of load balance will evolve in the connectivity solution. The drawback is, of course, that the number of subdomains assigned to each component grid will change according to the connectivity solution needs so that it will not optimally load balance the flow solution. The goal of the routine is to achieve a reasonable tradeoff for those problems in which the connectivity solution is a severe bottleneck.

The user-specified value of f_o acts as a weight to control the desired degree of load balance in either the flow solution or connectivity solution. If $f_o \approx \infty$, the dynamic scheme retains the static partition regardless of the degree of load imbalance in the connectivity solution and the load balance will be optimized for the flow solution only. If $f_o \approx 1$, the dynamic routine will continue trying to optimally load balance the connectivity solution. It should be noted, however, that the dynamic routine will never *completely* load balance the connectivity solution because there is no mechanism in place to subdivide the grids in such a way that the number of IGBPs searched is distributed evenly across the grid. We assume that, by continuing to apply more processors to the grid and performing the grid subdivision in such a way as to achieve the smallest surface area in each subdomain, the number of IGBPs searched will be distributed more-or-less evenly, although this is not always the case. In practice, the "best" value of f_o is problem dependent.

4.0 Results

The parallel moving-body overset grid scheme has been tested for three baseline test problems on the IBM SP2 at NASA Ames Research Center, and the IBM SP at the U.S. Army Corps of Engineers Waterways Experiment Station (CEWES). Both are distributed memory machines with a switch-based interconnect network. The SP2 is a slightly older model, using IBM's RISC-based RS/6000 POWER2 chips (66.7 MHz clock rate) at each node with a peak interconnect speed of 40 MB/sec. The SP uses the POWER2 Super Chip (P2SC, 135 MHz clock rate) at each node with a maximum interconnect speed of 110 MB/sec.

The test problems include 1) a two-dimensional oscillating airfoil, 2) a descending delta-wing configuration, and 3) a finned-store separation from a wing/pylon configuration. The following subsections provide details on the measured performance for these cases.

4.1 2D Oscillating Airfoil

This case computes the viscous flowfield about a NACA 0012 airfoil with freestream Mach number M_∞ =0.8 and Reynolds number Re=10^6 whose angle of attack goes through the following sinusoidal motion:

$$\alpha(t) = \alpha_o \sin \omega t$$

where $\alpha_o = 5°$ and $\omega = \pi/2$. The three grids used to resolve the flowfield, which are shown in Fig. 2, are a near-field grid that defines the airfoil and extends to a distance of about one chord, an intermediate-field circular grid that extends to distance of about three chords from the airfoil, and a square Cartesian background grid that extends seven chords from the airfoil. The three grids have roughly equal numbers of gridpoints with a composite total of 64K gridpoints. The ratio of the number of IGBPs to gridpoints is about 44×10^{-3}. The airfoil grid rotates with the above specified angle of attack condition while the other two grids remain stationary.

Table 1 gives a summary of the measured parallel performance for this moving case on the IBM SP2 and SP. The statistics shown are the average Megaflop/sec/node rate measured using

Table 1. Performance of 2D Oscillating Airfoil Case

Nodes	Average Gridpoints/node	Avg. Mflops/ Node		Parallel Speedup		% Time in DCF3D	
		SP2	SP	SP2	SP	SP2	SP
6	10088	23.1	31.3	1	1	10%	7%
9	6726	18.3	24.6	1.45	1.43	11%	8%
12	5044	18.6	27.7	2.07	2.23	14%	11%
18	3363	14.6	21.9	2.83	3.11	15%	11%
24	2522	11.3	15.6	3.65	3.75	14%	11%

IBM's PHPM (Parallel Hardware Performance Monitor) software on the SP2, the measured parallel speedup, and the percentage of time spent in the connectivity solution. Because the SP does not have PHPM available, its Mflop rate is derived by comparing the solution time to that of the

SP2. Figure 5 shows the overall parallel speedup and a breakdown of the parallel speedup in the flow solution (OVERFLOW) and connectivity (DCF3D) routines. The statistics shown do not include effects such as preprocessing steps or I/O costs.

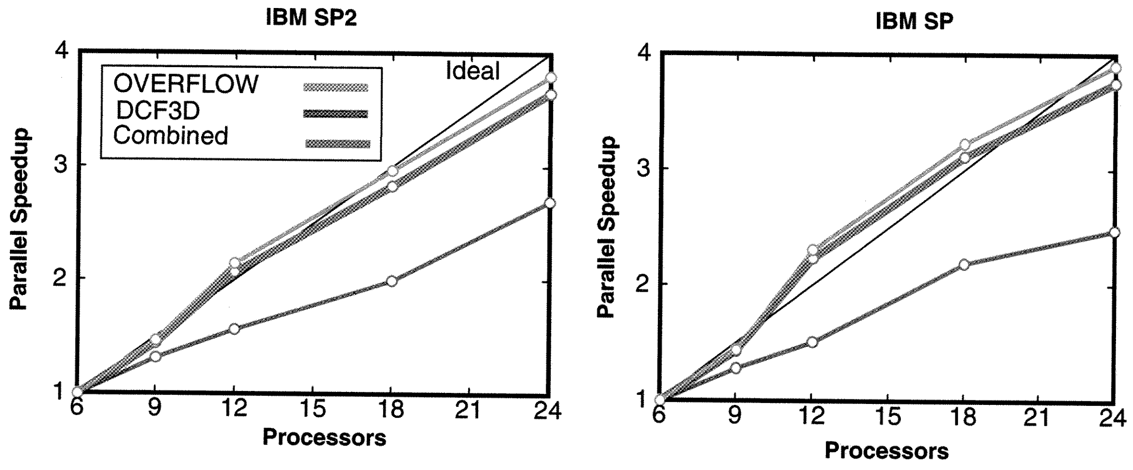


Figure 5. Parallel speedup - 2D oscillating airfoil

The case shows good parallel speedups overall and, although the parallel speedup of DCF3D is significantly lower than that of OVERFLOW, it uses a relatively small percentage of the total solution time (i.e. less than 15%) and remains roughly the same for all processor partitions. The Megaflop rate per node drops off significantly but this is most likely a consequence of the low number of gridpoints for this small problem on large numbers of processors. Some degree of super scalar speedups are observed on both machines but most-notably on the SP. These super scalar results are most-likely caused by an improvement in the cache performance as a result of the shorter loop lengths in the 12 processor case.

The dynamic load balance parameter f_o is set to ∞ for this case, which means the dynamic routine is effectively ignored and no effort is made to rebalance the solution based on the connectivity performance. Because of the low percentage of time spent in the connectivity solution, any effort to rebalance the solution to improve the performance of the connectivity solution causes the optimal performance for the flow solver to be lost and the performance becomes worse overall.

A scaleup study is performed with this case to determine how well the method can be scaled for larger problem sizes. The original grids are coarsened by removing every other gridpoint so that the number of gridpoints is reduced by a factor of four. The original grids are also refined by adding a gridpoint between the others so the number of gridpoints is increased by a factor of four. The three resulting problem sizes of the coarsened, original, and refined cases are 15.3K, 63.6K, and 240.4K gridpoints, respectively. The IGBPs/gridpoints ratio stays roughly the same for all three cases at 44×10^{-3}. The coarsened case is run on 3 processors, the original case on 12 processors, and the refined case on 48 processors. Table 2 shows results of the scaling study. Preprocessing and I/O costs were not included in determination of the statistics.

Table 2. 2D Oscillating Airfoil Scaling Study

Case	Average Gridpoints/node	Time/ Timestep(sec)		% Time in DCF3D	
		SP2	SP	SP2	SP
Coarsened - 3 nodes	5117	0.255	0.175	10%	8%
Original - 12 nodes	5299	0.285	0.195	14%	11%
Refined - 48 nodes	5008	0.365	0.231	23%	17%

The scaleup study indicates that there is a noticeable drop in efficiency as we move from the smallest to the largest case. This is due to an increase in communication costs across the larger numbers of processors and to a relative lack of scalability in DCF3D. The lower scalability in the connectivity solution is indicated by the percentage of time spent in DCF3D which increases by about 2.2 times from the coarsened 3 node case to the refined 48 node case. This appears to indicate that the connectivity solution may become a more dominant parallel cost for larger problems with many gridpoints.

4.2 Descending Delta Wing

This case was originally studied with the baseline serial version of the software by Chawla and Van Dalsem [14]. The case consists of four grids, shown in Fig. 6, that have a composite total of about 1 million gridpoints with a IGBPs/gridpoints ratio of $33x10^{-3}$. Three curvilinear grids make up the delta wing and pipe jet, and the fourth is a Cartesian background grid. The three curvilinear grids move at a relatively slow speed of M_{∞} =0.064 with respect to the background grid. The viscous terms are active in all directions on all four grids and no turbulence models are used.

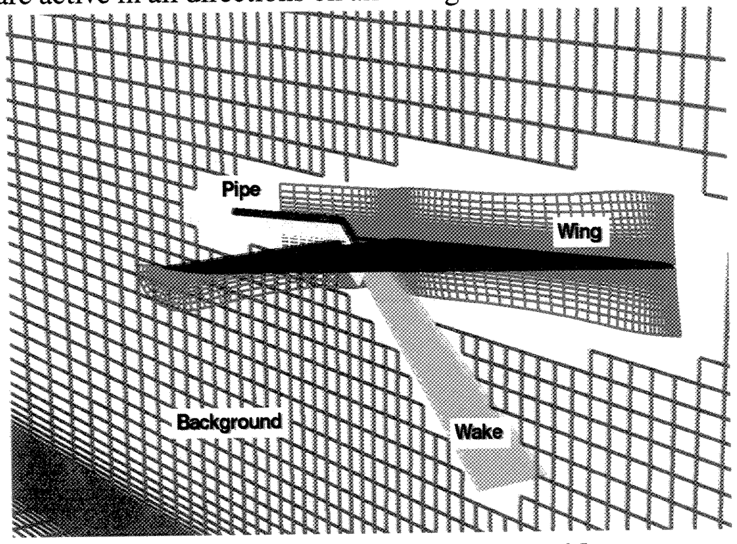


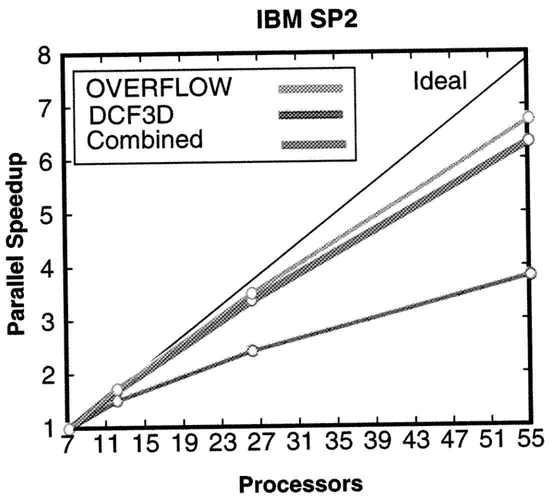
Figure 6. Descending delta-wing grids.

Table 3 shows the parallel performance statistics of this case with static load balancing on the SP2 and SP. The method appears to scale well for this problem, showing good parallel speedup with relatively small dropoff in the Megaflop/sec/processor with the increasing number of processors. The percentage of time spent in DCF3D grows larger with increasing numbers of processors but remains a relatively low percentage of the time overall. The overall parallel speedup is plotted in Fig. 7 along with a breakdown in the speedup of the flow solution and connectivity solution.

Overall, the parallel performance for this case is quite good. Although the connectivity solution shows worst parallel speedup than the flow solver, it comprises a small percentage of the total cost (less than 15%) so the performance is degraded only slightly. Because the connectivity costs represented a small fraction, there is no performance gain by use of the dynamic load balance scheme.

Table 3. Performance of Descending Delta Wing Case

Nodes	Average Gridpoints/node	Avg. Mflops/ Node		Parallel Speedup		% Time in DCF3D	
		SP2	SP	SP2	SP	SP2	SP
7	140480	27.3	43.8	1	1	9%	6%
12	81947	27.0	51.8	1.72	2.00	10%	9%
26	37822	26.3	51.1	3.37	3.88	12%	11%
55	17879	23.5	45.3	6.31	7.08	15%	13%



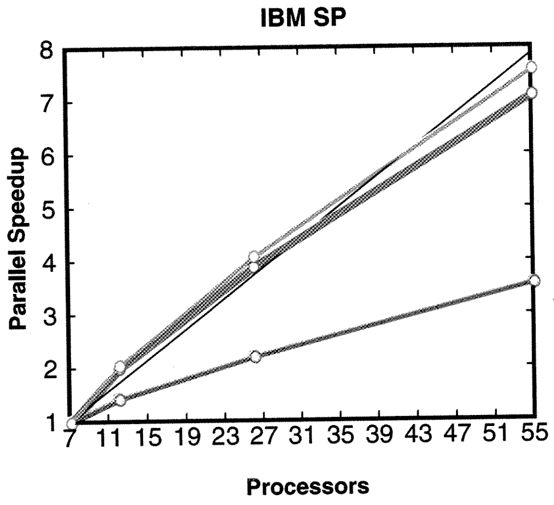


Figure 7. Parallel speedup - descending delta-wing case

4.3 Finned-Store Separation from Wing/Pylon

The third test case investigates the performance of the method for computing the unsteady viscous flow in a Mach 1.6 store separation event. The case was studied previously by Meakin [4] with vectorized versions of the software. A total of 16 grids are used (Fig. 8) with ten curvilinear grids defining the finned store, three curvilinear grids defining the wing/pylon configuration, and three background Cartesian grids around the store. A composite total of 0.81 million gridpoints are used and the IGBPs/gridpoints ratio is about 66x10^{-3}. Viscous terms are active in all curvilinear grids with a Baldwin-Lomax turbulence model. The three Cartesian background grids are all

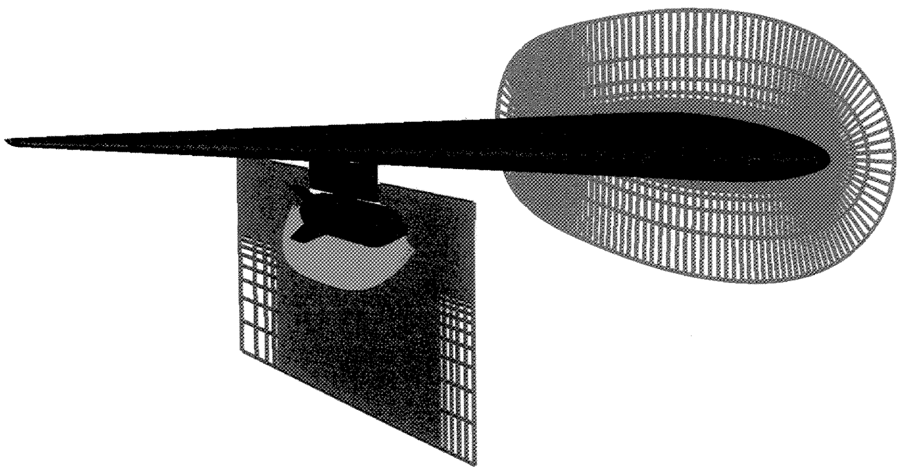


Figure 8. Grids for Finned-Store Separation Problem

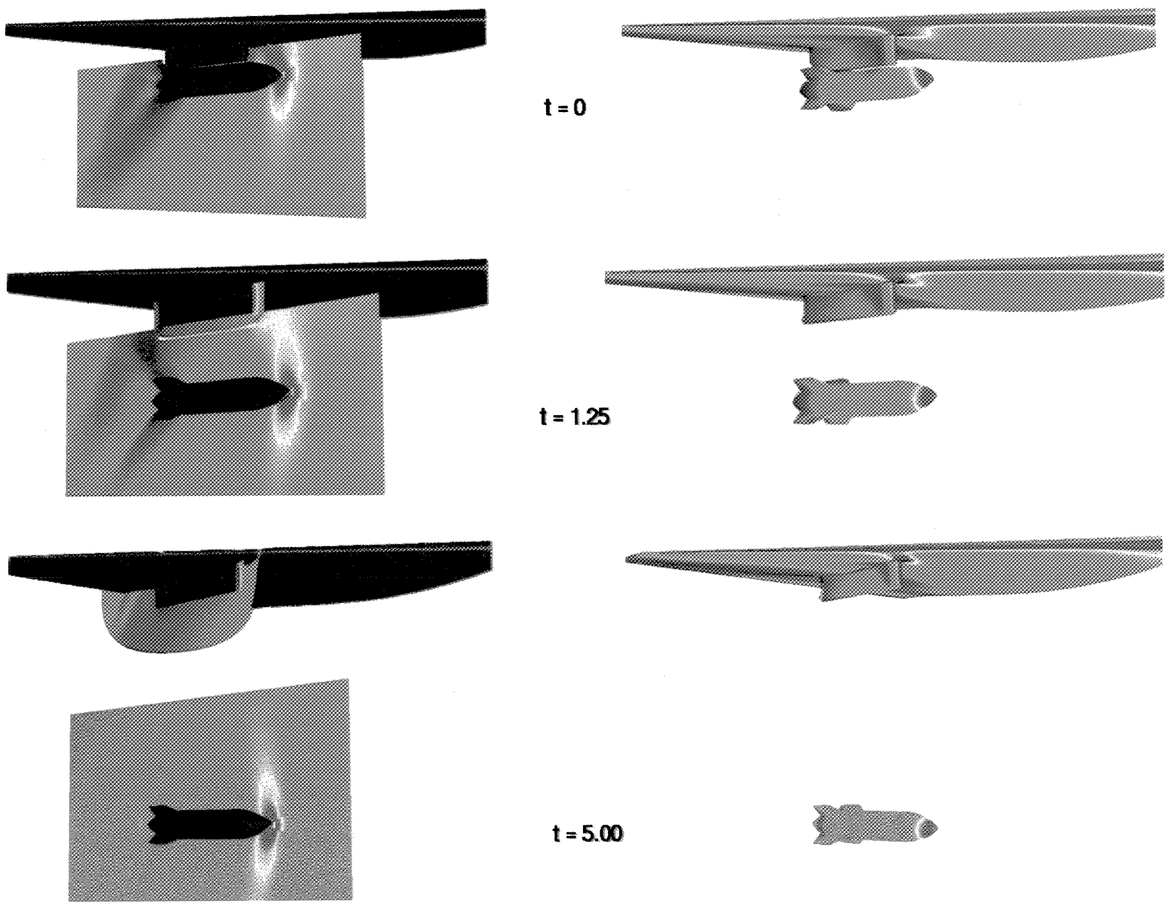


Figure 9. Computed Mach contours (left) and surface pressure (right)

inviscid. The motion of the store is specified in this case rather than computed from the aerodynamic forces on the body. However, the free motion can be computed with negligible change in the parallel performance of the code. Figure 9 shows the computed solution through 1000 timesteps.

Table 4 gives the measured performance statistics for this calculation on the SP2 and the SP using static load balancing. The time spent in DCF3D is noticably higher for this case than either of the previous two cases because the ratio of intergrid boundary points to gridpoints is larger in this grid system. The average Megaflop/processor rate is also slower than the descending delta-wing case, for two reasons. First, the problem size is smaller so the number of gridpoints per processor is less. Second, the lack of load balance in the connectivity solution results in a relatively low FLOP count so the larger proportion of time spent in this routine leads to a drop in the overall Megaflops/sec rate.

Table 4. Performance for Finned-Store Separation Case

Nodes	Average Gridpoints/node	Avg. Mflops/ Node		Parallel Speedup		% Time in DCF3D	
		SP2	SP	SP2	SP	SP2	SP
16	50462	10.3	16.4	1	1	17%	15%
18	44855	15.0	23.3	1.65	1.61	28%	21%
22	36699	18.2	29.2	2.54	2.48	24%	19%
28	28836	19.2	32.6	3.52	3.74	32%	28%
35	23068	17.5	29.7	4.11	4.38	28%	23%
42	19223	16.7	29.5	5.17	5.70	33%	29%
52	15526	16.5	28.4	6.57	7.08	34%	29%
61	13236	14.5	25.4	7.56	8.33	32%	28%

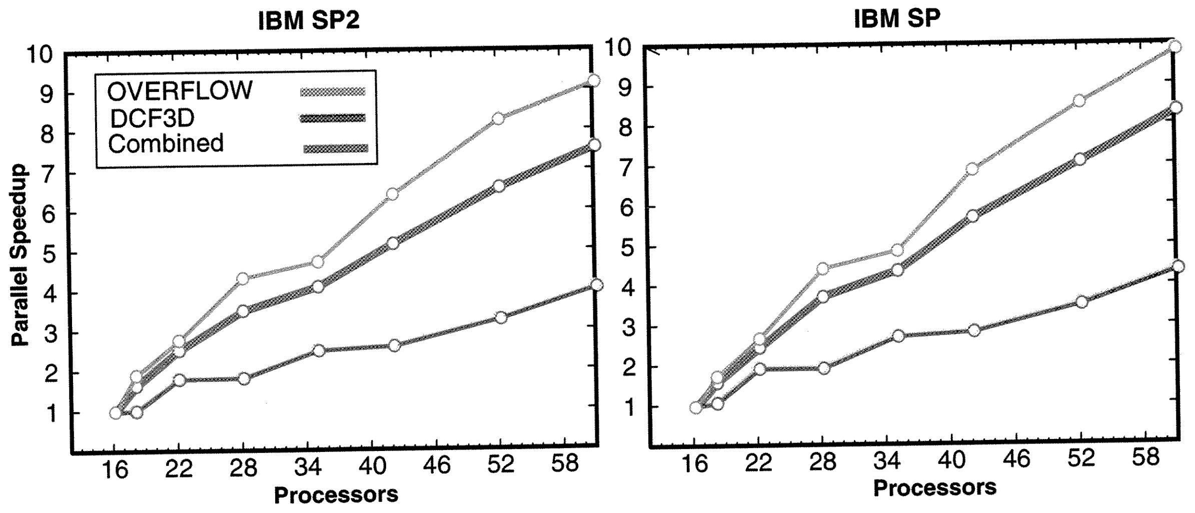


Figure 10. Parallel speedup of finned-store separation case with static load balancing

The Mflop/processor rate remains relatively constant between 18 and 52 processors, however, indicating a relatively good degree of scalability. The execution rate improves in the range of 16 to 28 processors because the problem is achieving a better degree of static load balance by increasing the number of processors applied to the problem. The overall parallel speedup along with a breakdown in the speedup of OVERFLOW and DCF3D is shown in Fig. 10.

Table 5. Parallel performance of DCF3D with dynamic load balance scheme applied to wing/pylon/finned-store problem.

	% Time in DCF3D		DCF3D Speedup	
Nodes	Dynamic	Static	Dynamic	Static
16	17%	17%	1	1
18	17%	28%	1.31	1.00
28	22%	32%	2.64	1.87
52	24%	34%	4.10	3.28

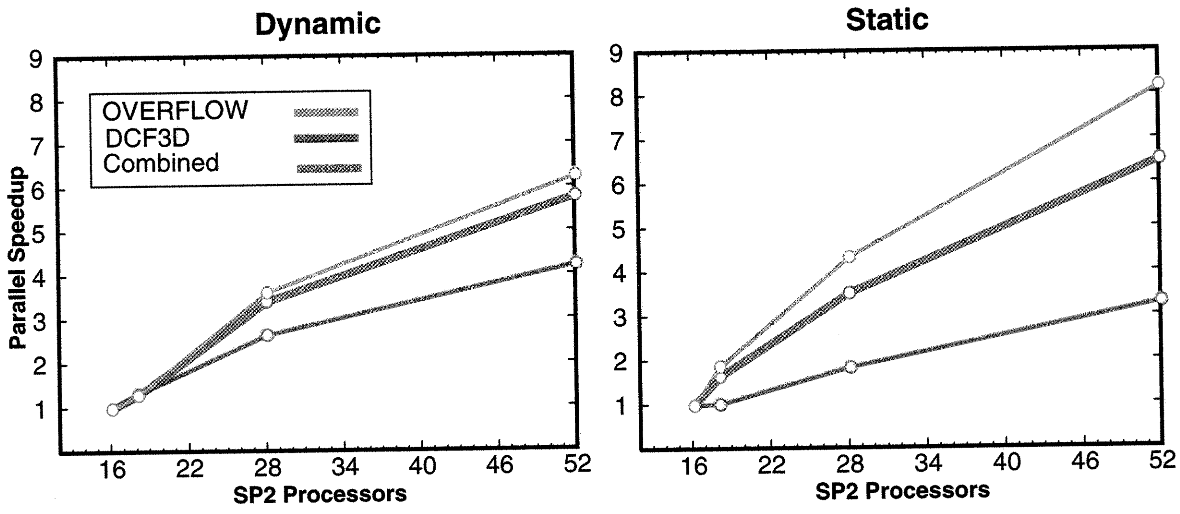


Figure 11. Parallel speedup using static and dynamic load balancing schemes with finned-store separation case.

The IGBPs/gridpoints ratio for the wing/pylon/finned-store case is 1.5-2 times larger than either of the previous two problems investigated and the connectivity solution consequently represents a higher proportion of the total solution costs. It is therefore a good candidate to evaluate the performance of the dynamic load balance scheme. Table 5 shows the performance statistics of DCF3D on various processor partitions of the SP2 using a connectivity load balance factor of f_o=5. The maximum degree of load imbalance in the connectivity solution for this problem was found to be around $f(p)=7$, so the value of f_o=5 is chosen with the goal of reducing this imbalance somewhat without reducing the load balance in the connectivity solution significantly. It is clear from the result in Table 5 that the dynamic scheme is effective at improving the parallel speedup of DCF3D. This translates to a better degree of scaling in the connectivity solution. When increasing the number of processors from 16 to 52, the proportion of time spent in DCF3D increases by a factor of 2.0 with the static load balance scheme. With the dynamic scheme, the proportion of time increases by a factor of 1.35.

Unfortunately, the improvement in parallel performance in DCF3D comes at the cost of a reduction in parallel performance of OVERFLOW (Fig. 11). OVERFLOW constitutes a more significant proportion of the total solution cost, greater than or equal to two-thirds of the total all the processor partitions tested, so the reduction in its parallel performance outweighs the improvement in performance in DCF3D and the overall parallel speedup is diminished for this case. Depending on the number of processors, the combined parallel performance with the static scheme is about 15-25% better than the combined performance with the dynamic scheme. Thus, the static scheme is most effective at achieving optimal parallel performance for this problem.

We conclude this section with a remark about the effectiveness of parallel processing at reducing the turnaround time for moving-body overset grid calculations. Table 6 gives the wallclock run time for the wing/pylon/finned-store case on the SP2 and SP compared with the run time on a single processor Cray YMP 864 (times reported in [4]). The YMP has a clock rate of 4.2 ns and peak flop rate of 333 Mflops (by comparison, the Cray C90 has a clock rate of 6.0 ns and a peak flop rate of 1 Gflop, so the times recorded on the YMP are probably two to three times slower than what achieved on a C90 head). The overall wallclock time speedup and the average speedup per node are shown in "YMP units", which designates one unit of time on the single processor of the Cray YMP. Speedups in the run time on the order of one to two orders of magnitude are observed for these modern multi-processor supercomputers, and the per node performance is a significant fraction of the performance of the YMP across a wide range of processors.

Table 6. Wallclock speedup in run time (compared to Cray YMP)

Run Time Speedup (in YMP Units[a])				
	Overall		Per Node	
nodes	SP2	SP	SP2	SP
18	9.4	18.5	0.52	1.03
28	19.9	33.8	0.71	1.21
42	29.2	51.5	0.70	1.23
61	43.0	75.3	0.70	1.23

a. Speedup over YMP where 1 YMP unit = 1 unit of time on single processor Cray YMP/864.

5.0 Future Work

The prediction accuracy of calculations about geometrically complex shapes and evolving unsteady flowfields in moving-grid problems can be improved with solution adaption. A class of solution adaption methods based on nested overset Cartesian grids [15,16] have demonstrated favorable results. Use of Cartesian grids offers several advantages over traditional structured curvlinear and unstructured grids. Uniformly-spaced Cartesian grids can be completely defined via knowledge of only their bounding box and spacing, constituting only seven parameters per grid, much lower than traditional methods that require the coordinates (three terms) and the associated metrics (13 terms) for each gridpoint. The connectivity solution with Cartesian grids can be determined very quickly because costly donor searches are avoided. Cartesian grids retain all of the advantages typically associated with unstructured grids, namely low grid generation costs and ability to adapt to the solution, but they also can utilize the large class of efficient implicit solution methods designed for structured grids, which require much less storage and better vectorized and RISC-based processor performance than unstructured-grid solvers. The main restriction faced by Cartesian solvers, which tends to also be faced by unstructured methods, is the inability to resolve fine-scale viscous effects in the near-body regions of the flowfield.

Meakin [17,18] has introduced a novel solution adaption scheme based on structured overset Cartesian grids that incorporates a pure Chimera approach in the near-body region to resolve the viscous fine-scale effects. Curvilinear overset grids are used in the region near the body and the off-body position of the domain is automatically partitioned into a system of Cartesian background grids of variable levels of refinement. Initially, the level of refinement is based on proximity to the near-body curvilinear grids. The off-body domain is then automatically repartitioned during adaption in response to body motion and estimates of solution error, facilitating both refinement and coarsening. Adaption is designed to maintain solution fidelity in the off-body portion of the domain at the same level realizable in the near-body grid systems. Figure 12 shows the near-body curvilinear grids with the boundaries of the a) default off-body Cartesian set of grids and b) the off-body Cartesian set of grids after several refinement steps for the X-38 vehicle (Crew Return Vehicle). The flow solution carried out for this case is shown in Figure 12 c.

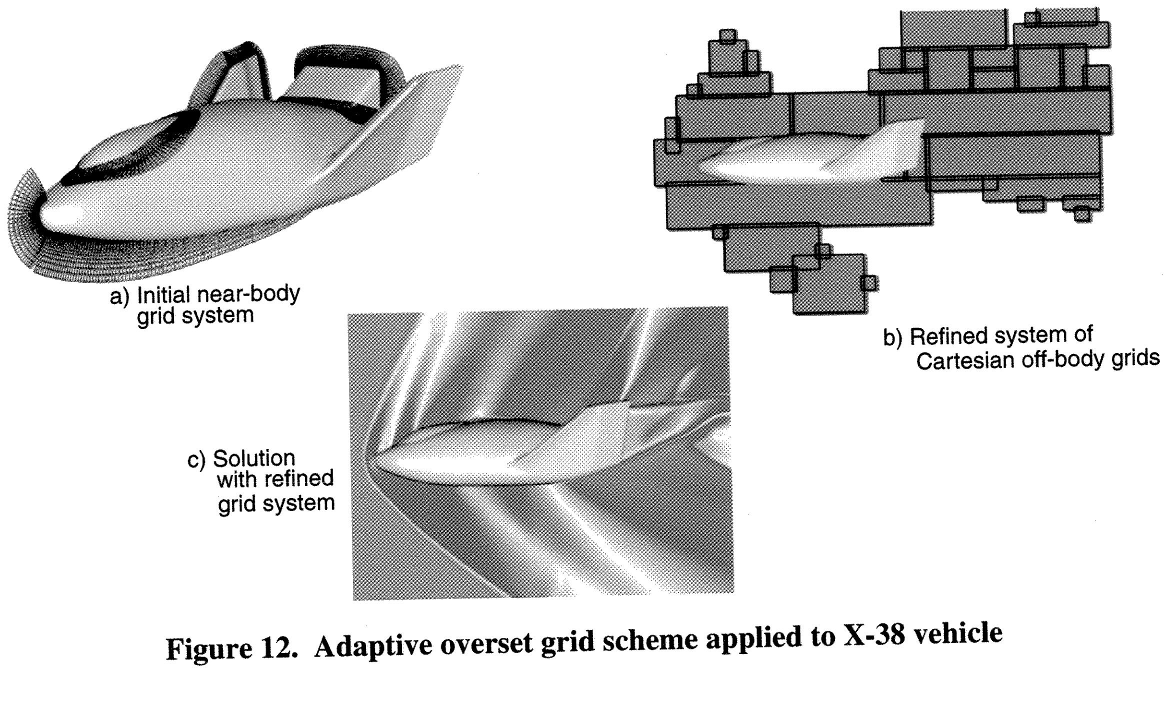
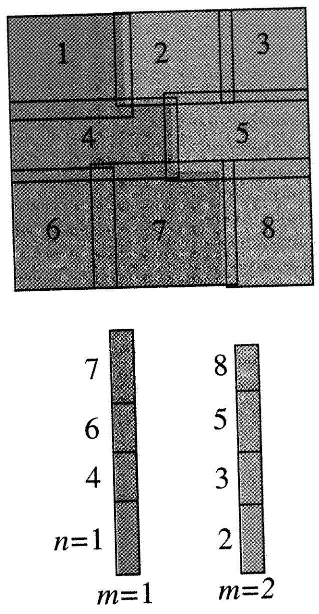

a) Initial near-body grid system
b) Refined system of Cartesian off-body grids
c) Solution with refined grid system

Figure 12. Adaptive overset grid scheme applied to X-38 vehicle

The adaptive scheme is being implemented in parallel through an entirely coarse-grain strategy. Unlike the OVERFLOW-D1 algorithm which is suited for Chimera problems with a relatively small number of grids, the solution adaption scheme generates very large numbers of smaller Cartesian grids (generally hundreds to thousands) and therefore offers a high level of coarse-grain parallelism. A load balancing scheme, described in Algorithm 3, gathers grids into groups and assigns each group to a node in the parallel platform. The load balance approach insures that computational work (i.e. gridpoints) is distributed evenly among the groups and also maintains a degree of locality among the members of each group to maximize the level of intra-group connectivity and, hence, lower communication costs. Each group is assigned to a different node in the parallel architecture and MPI subroutine calls are used to pass overlapping grid information for grids which lie at the edge of the group.

Algorithm 3: Grouping Strategy

```
Loop through N grids (largest-to-smallest), n
    Loop through M groups (smallest-to-largest), m
        IF group m is empty,
            Assign grid n to group m
        Else,
            Check connectivity array to see if grid n is
            connected to any other members of group m.
            If so, assign grid n to group m.
    End loop on M
    If grid n is not connected to any members of
    the groups as currently constituted, assign
    grid n to the smallest of the M groups
End loop on N
```



In addition to the parallelism between the groups, there is the potential for further parallelization at the intra-group level. The multiple grids at each group can be updated simultaneously so the computations within the group can be parallelized. This will be effective for parallel platforms consisting of clusters of shared-memory processors.

The approach also allows latency hiding strategies to be employed. By structuring the computations to begin on the grids which lie at the interior of the group, the data communicated at the group borders can be performed asynchronously, effectively overlapping communication with computation.

The two most challenging parts of the new overset solution adaption scheme to implement efficiently in parallel are the adaption step and the connectivity solution. Each adaption step requires interpolation of information on the coarse systems to the refined grids as well as re-distribution of data after the adapt cycle. Efficient techniques for parallelizing this step have been introduced in other work [19] and the new MPI_SPAWN function, which allows new parallel processes to be spawned from an existing process (scheduled to be included in the MPI-2 [20] release), could be an effective tool in the parallel implementation of the adaption cycle. The bulk of the connectivity solution can be performed at very low cost because no donor searches are required when donor elements reside in Cartesian grid components. Points which require a traditional domain connectivity solution will be performed with a parallel version of the DCF3D software adopted from OVERFLOW-D1. The lessons learned from parallelizing the connectivity solution will be useful for efficient parallelization of this step. Since the vast majority of the interpolation donors will exist in Cartesian grid components in this type of discretization, the approach should scale well.

6.0 Concluding Remarks

This paper investigates the parallel performance of moving-body Chimera structured overset grid CFD calculations for computations of the unsteady viscous aerodynamics about multi-component bodies with relative motion between component parts. The predominant costs associated with these calculations are computation of the fluid flow equations and re-establishing domain connectivity after grid movement. A two-part static-dynamic load balancing scheme is introduced in which the static part optimizes the load balance of the flow solution and the dynamic part repartitions, if desired, in an effort to improve load balance in the connectivity solution. Algorithm performance is investigated on the IBM SP2 and IBM SP.

The static scheme alone provides the best parallel performance for the flow solution but gives rather poor parallel performance in the connectivity solution. The dynamic scheme improves the performance of the connectivity solution but does so at the expense of the flow solver, which yields lower parallel performance with the repartitioning. Thus, some tradeoff is necessary. In the problems tested, for which the flow solver was the dominant cost (comprising 67-90% of the total), the static scheme alone demonstrates the best overall performance. The dynamic scheme is effective at improving the parallel performance of the connectivity solution but its cost to the flow solver outweighs its benefits to the connectivity solution and the overall parallel performance is reduced. The dynamic scheme may, however, prove to be effective in cases that are dominated by connectivity costs, where the flow solution makes up a less significant fraction of the total (e.g. Euler calculations).

One conclusion drawn from this work is that the flow solution and connectivity solution require fundamentally different partitions for load balancing and any one choice is a compromise. The use of a dynamic scheme was proposed to allow some degree of tradeoff in the compromise but, while it may ultimately prove successful for some cases, the compromise strategy will never scale to very large numbers of processors (i.e. greater than 100) because of the fundamental difference required in the partitioning. Some form of inter-leaving of the flow solver costs and connectivity costs, with optimal partitions in both, is viewed to be the most successful approach for machines with very large numbers of processors.

Finally, parallel implementation of a new solution adaption scheme based on overset cartesian grids is discussed. The scheme offers multiple levels of coarse-grain parallelism that can be exploited on a variety of parallel platforms.

Acknowledgments

This work was performed as part of the DoD Common HPC Software Support Initiative (CHSSI) funded through NASA contract number NAS2-14109, Task 23. Time on the IBM SP2 was provided by the NASA HPCCP Computational Aerosciences Project. Time on the IBM SP was provided by the Major Shared Resource Center at the U.S. Army Corps of Engineers Waterways Experiment Station (CEWES). The authors wish to acknowledge Eric Barszcz and Dr. Sisera Weeratunga for their assistance with the parallel OVERFLOW and DCF3D. The authors gratefully acknowledge the suggestions of Dr. William Chan, Dr. Roger Strawn, and Dr. Chrisopher Atwood throughout the course of this work. Special thanks to Messrs. Reynoldo Gomez and James Greathouse of NASA Johnson Space Center for providing the grids used for the X-38 vehicle described in section 5.

References

[1] Lohner, R., "Adaptive Remeshing for Transient Problems," *Comp. Meth. Appl. Mech. Eng.*, Vol. 75, 1989, pp. 195-214.

[2] Steger, J.L., Dougherty, F.C., and Benek, J.A., "A Chimera Grid Scheme," Advances in Grid Generation, K.N. Ghia and U. Ghia, eds., ASME FED Vol. 5, June 1983.

[3] Meakin, R., and Suhs, N., "Unsteady Aerodynamic Simulation of Multiple Bodies in Relative Motion," AIAA 89-1996, Preseted at the 9th AIAA Computational Fluid Dynamics Conference, Buffalo, NY, June 1989.

[4] Meakin, R.L., "Computations of the Unsteady Flow About a Generic Wing/Pylon/Finned-Store Configuration," AIAA Paper AIAA-92-4568, Presented at the AIAA Atmospheric Flight Mechanics Conference, Hilton Head Island, NC, Aug. 1992.

[5] Meakin, R., "Unsteady Simulation of the Viscous Flow about a V-22 Rotor and Wing in Hover," AIAA 95-3463, Presented at the AIAA Atmospheric Flight Mechanics Conference, Baltimore, MD, Aug. 1995.

[6] Atwood, C.A., and Smith, M.H., "Nonlinear Fluid Computations in a Distributed Environment," AIAA 95-0224, Presented at the 33rd AIAA Aerosciences Meeting, Jan. 1995.

[7] Brown, D., Chesshire, G., Henshaw, W., and Quinlan, D., "Overture: An Object-Oriented Software System for Solving Partial Differential Equations in Serial and Parallel Environments," LA-UR-97-335, Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, March 14-17, 1997.

[8] Ryan, J.S., and Weeratunga, S., "Parallel Computation of 3D Navier-Stokes Flowfields for Supersonic Vehicles," AIAA Paper 93-0064, Jan. 1993.

[9] Weeratunga, S.K., Barszcz, E., and Chawla, K., "Moving Body Overset Grid Applications on Distributed Memory MIMD Computers," AIAA-95-1751-CP, 1995.

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

[11] K.J. Renze, P.G. Buning, and R.G. Rajagopalan, "A Comparative Study of Turbulence Models for Overset Grids," AIAA-92-0437, AIAA 30th Aerospace Sciences Meeting, Reno, NV, Jan. 6-9, 1992.

[12] Meakin, R.L., "A New Method for Establishing Inter-grid Communication Among Systems of Overset Grids," AIAA Paper AIAA-91-1586, Presented at the 10th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 1991.

[13] MPI: A Message-Passing Interface Standard, Published by the University of Tennessee, Knoxville, TN, 1995. http://WWW.ERC.MsState.Edu/mpi

[14] Chawla, K., and Van Dalsem, W.R., "Numerical Simulation of a Powered-Lift Landing," 72nd Fluid Dynamics Panel Meeting and Symposium on Computational and Experimental Assessment of Jets in Cross Flow, Winchester, U.K., April 19-23, 1993. AGARD Conference Proceedings CP-534, pp. 32.1-32.10.

[15] Berger, M., and Oliger, J., "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *Journal of Computational Physics*, Vol. 53, 1984, pp. 484-512.

[16] Berger, M. and LeVeque, R., "A Rotated Difference Scheme for Cartesian Grids in Complex Geometries," AIAA Paper 91-1602, AIAA 10th CFD Conference, 1991.

[17] Meakin, R.L., "An Efficient Means of Adaptive Refinement within Systems of Overset Grids," AIAA-95-1722-CP, June 1995, pp. 844-857.

[18] Meakin, R.L., "On Adaptive Refinement and Overset Structured Grids," AIAA-97-1858, Proceedings of the AIAA 13th CFD Conference, June 1997, pp. 236-249.

[19] Quinlan, D., "AMR++: A Design for Parallel Object-Oriented Adaptive Mesh Refinement", Published in Proceedings of the IMA Workshop on Structured Adaptive Mesh Refinement, Minneapolis, MN, March 1997.

[20] MPI-2: Extensions to the Message Passing Interface, Published by the University of Tennesse, Knoxville, TN, 1996. http://www.mcs.anl.gov/mpi/mpi2/mpi2.html