A Two-Dimensional Multigrid-Driven Navier-Stokes Solver for Multiprocessor Architectures

Juan J. Alonso, Todd J. Mitty, Luigi Martinelli, and Antony Jameson
Department of Mechanical and Aerospace Engineering
Princeton University
Princeton, New Jersey 08544 U.S.A.

Abstract

A two-dimensional unsteady Navier-Stokes solver has been parallelized using a domain decomposition approach and the PVM message passing library. Several options for the treatment of multigrid and implicit residual smoothing are examined. Results for the unsteady flow over a pitching NACA 64A010 airfoil are presented.

1 INTRODUCTION

In recent years, computational fluid dynamics (CFD) has been gaining acceptance as a design tool in industry. Advancements in algorithm development and computational hardware have led to more complex modeling of fluid flows. Although current inviscid models can accurately predict the coefficient of lift for an airfoil in transonic flow, viscous effects such as shock wave/boundary layer interaction can significantly modify important aspects of a flow. Furthermore, unsteady phenomena of patent viscous character, such as buffeting, can not be predicted with the help of inviscid models. Such viscous phenomena directly impact the design of engineering configurations, and therefore, it is necessary to enhance the viscous prediction capability of CFD tools.

Increasingly complex fluid flow models require high performance computing facilities. A cost effective solution for problems of this type requiring fast CPUs and large internal memory is the use of a parallel computing paradigm. For computational efficiency, one typically incorporates convergence acceleration techniques such as multigrid and implicit residual smoothing. Message passing becomes necessary in this new environment, and severely limits the performance of processes that are inherently communication intensive.

In this paper we present a parallelized version of a well established Navier-Stokes solver, FLO103 [1]. This computer program has recently been enhanced to include Jameson's implicit multigrid approach [2] for the efficient calculation of unsteady viscous flows. Calculations are performed on an IBM SP1 multiprocessor computer with a domain decomposition approach, and message passing is handled by PVM (Parallel Virtual Machine) software [3]. Several methods for implementing convergence acceleration techniques such as multigrid and implicit residual smoothing are studied.

2 NAVIER-STOKES EQUATIONS DISCRETIZATION

The two-dimensional, unsteady, compressible Navier-Stokes equations may be written in divergence form for a Cartesian coordinate system (x, y) as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = \left(\frac{\partial \mathbf{R}}{\partial x} + \frac{\partial \mathbf{S}}{\partial y} \right), \tag{1}$$

where $\mathbf{w}$ is the vector of flow variables, $\mathbf{f}$ and $\mathbf{g}$ are the convective fluxes, and $\mathbf{R}$ and $\mathbf{S}$ are the viscous fluxes in each of the coordinate directions. With Reynolds averaging, turbulence effects can be taken into account with a turbulence model. In this work, a Baldwin-Lomax model was used. In integral form, Equation 1 can

be applied to each finite volume of a computational domain to yield a set of coupled first-order differential equations of the form

$$\frac{d}{dt}(\mathbf{w}_{ij}\, V_{ij}) + \mathbf{E}(\mathbf{w}_{ij}) + \mathbf{NS}(\mathbf{w}_{ij}) + \mathbf{D}(\mathbf{w}_{ij}) = \mathbf{0}, \tag{2}$$

where $\mathbf{E}(\mathbf{w}_{ij})$ are the convective Euler fluxes, $\mathbf{NS}(\mathbf{w}_{ij})$ are the Navier-Stokes viscous fluxes, and $\mathbf{D}(\mathbf{w}_{ij})$ are the artificial dissipation fluxes added for numerical stability. For unsteady problems, Equation 2 is modified by introducing a pseudo-time formulation to improve computational performance [2].

3 PARALLELIZATION STRATEGY

FLO103P is parallelized using a domain decomposition model, a SIMD (Single Input Multiple Data) strategy, and the PVM Library for message passing. Flows were computed on a C-mesh of size $n_i \times n_j = 1024 \times 64$. This domain was decomposed into subdomains containing $\frac{n_i}{N_p} \times n_j$ points, where N_p is the number of subdomains used. Communication between subdomains was performed through halo cells surrounding each subdomain boundary. A two-level halo was sufficient to calculate the convective, viscous, and dissipative fluxes for all cells contained in each processor. In the coarser levels of the multigrid sequence, a single level halo suffices since a simplified model of the artificial dissipation terms is used.

For problems with a low task granularity (ratio of the number of bytes received by a processor to the number of floating point operations it performs), large parallel efficiencies can be obtained. Unfortunately, convergence acceleration techniques developed in the 1980s base their success on global communication in the computational domain. Thus, current multigrid and implicit residual smoothing techniques [4] are bound to hinder parallel performance in traditional mesh sizes. In order to effectively deal with the parallelization of these two techniques, we propose several ideas.

In this paper, static load balancing is performed at the beginning of the calculation. Since the number of cells remains constant throughout the calculations, the domain can be partitioned into subdomains with an equal number of cells. Except for some domains where an additional message across the wake is required, this provides for perfect load balancing.

4 PARALLEL MULTIGRID

The full approximation multigrid technique enhances the convergence rate of a scheme by performing computations on a series of increasingly coarser meshes. The calculations performed in each of these meshes are driven by the residuals at the previous finer level, and the results obtained are interpolated to the corresponding finer mesh. The explicit time-stepping acts as a smoothing operator for the high frequency errors in each level, thus damping the dominant error mode at the finest level and accelerating convergence. A more detailed discussion of this procedure can be found in [4]. If domain decomposition is used for parallel calculations, the size of the meshes contained in each processor at the coarser levels of the multigrid cycle is quite small (typically 8 or 16 cells). Most of the CPU time is then wasted sending information back and forth between processors. This situation worsens for multigrid W-cycles, where equilibrium in the coarser meshes is established repeatedly before traversing the series back to the finest level. Previous authors [5] have attempted to deal with this problem in the Euler equations by limiting message passing to only some stages in the Runge-Kutta time-stepping scheme, or passing the boundary data exclusively at the finest level in the series. This usually led to a decrease in the convergence rate of the numerical algorithm, presenting a clear tradeoff between the improved parallel performance and the increasing number of cycles required for a similar level of convergence. In this work, we examine three different approaches to the implementation of the multigrid algorithm. First, the full multigrid algorithm is implemented with message passing at all required points such that the parallel program exactly recovers the results of the serial code. Second, multigrid is applied independently within each subdomain to completely avoid inter-processor communication. Third, and last, at coarse multigrid levels where communication overhead dominates CPU time, a single processor will be used to gather, compute, and scatter information. Parallel speed-up curves for these three cases are also presented.

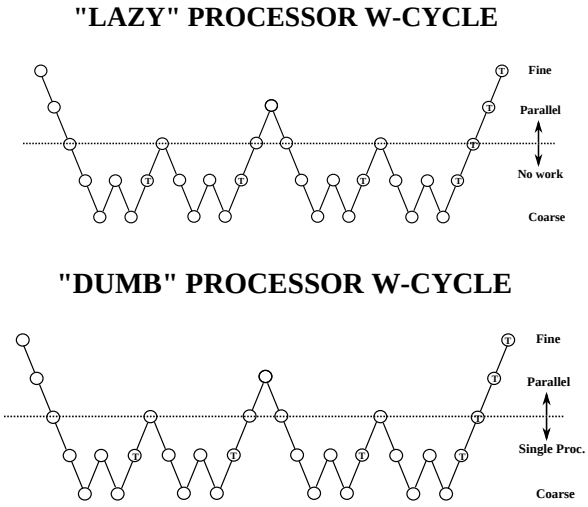

Figure 1: "Lazy"-"Dumb" Multigrid Approach.

4.1 Full multigrid approximation

This first approach was an attempt to implement a parallel program that exactly reproduced the output of the single processor code in which message passing is not necessary. In order to achieve this goal, boundary information was passed among processors on all multigrid levels at the beginning of all five stages of the Runge-Kutta time-stepping. Additional messages were required in order to process convergence information, calculate force coefficients, and compute the eddy viscosity in the turbulence model. This procedure recovers the serial version convergence rate, at the expense of poor parallel performance.

4.2 Implicit multigrid within subdomains

The second approach used to deal with the multigrid algorithm was to completely decouple the subdomains on the coarser levels of the series in order to minimize the number of messages passed when very little computation is being performed. As expected, this approach restores parallel efficiency to higher levels, but it suffers from a decreased convergence rate. Clearly, the success of the multigrid technique is due to the increased communication between different parts of the computational mesh at all levels, and the transfer of this information is limited by the isolation of the coarser levels in each subdomain. The degradation of the convergence rate is especially bad when interdomain boundaries lie close to regions of high gradients (such as shock waves and stagnation points). In some of these cases, a converged solution cannot be obtained.

4.3 "Lazy"-"Dumb" multigrid approach

In order to preserve the convergence rate of the original multigrid algorithm, and somehow improve the parallel efficiency of the first approach we propose the following procedure: calculations on the finer levels are performed as in 4.1; when the algorithm shifts the solution to a user-specified coarse mesh, all the processors in the calculation pass their flow variables and grid locations to a single processor which computes the coarser levels of multigrid without the need for message passing. This processor is termed "dumb" since it performs everyone else's work. The rest of the processors in the computation wait until the calculation needs to be performed on the finer levels, at which point they receive the information from the "dumb" processor and proceed once more in parallel. These processors are called "lazy" since they avoid carrying out part of the work that was, in theory, assigned to them. In our program, the level at which the transfer is done can be specified as an input, allowing for investigations of the optimal location of this transition point. Figure 1 presents this procedure graphically. Note that this construction can be extended to a hierarchy of "dumb" processors for larger calculations involving more processors.

5 IMPLICIT RESIDUAL SMOOTHING

Implicit residual smoothing (or averaging) is a technique that couples the residuals at any given point with those of all the other cells in the domain, increasing the support of the scheme, and allowing a larger time step than that permitted by the Courant-Friedrichs-Lewy (CFL) restriction. Once more, this is a communication intensive procedure that negatively impacts parallel performance in a multiprocessor environment.

5.1 Fully implicit residual smoothing

Serial implementations of this technique usually split the problem into the implicit coupling along each of the two coordinate directions. The direction which is normal to the airfoil surface presents no difficulties since all the required data resides in the appropriate processors (in this decomposition), allowing calculations to proceed in parallel. In the coordinate direction that is parallel to the airfoil surface, the residuals that are to be coupled reside in different processors. Moreover, the solution procedure (Thomas algorithm for a tridiagonal system) is inherently serial. A brute force method allows only one processor to be active at a time in the forward elimination and back substitution phases, while the remaining processors are idle. Since the residual smoothing procedure consumes about 30% of the time of the total calculation, this idle time causes parallel performance to drop-off considerably.

5.2 Implicit residual smoothing within subdomains

In a very similar fashion to the multigrid performed implicitly within blocks, and in an attempt to reduce the amount of messages passed, residual smoothing was performed implicitly within blocks, with no global coupling of the residuals. Once more, as expected, although parallel performance improves, the convergence rate degrades to unacceptable levels. As in the corresponding multigrid procedure, this lack of coupling between domains often led to instability in the calculations.

5.3 Iterative implicit residual smoothing

An alternative to the previous approach is to perform an iterative solution of the smoothing problem. Messages containing the boundary residuals are passed at the beginning of each iteration, and the relaxation process proceeds in parallel. It has been observed in practice that in order to obtain the increase in CFL number provided by the fully implicit version (from 3 to about 6), at least two iterations are required. In these calculations, three iterations were performed and a CFL number of 6 was used without stability problems. The matrix problem is setup in each subdomain and a Jacobi relaxation procedure is performed on all subdomains concurrently.

6 RESULTS

A summary of the results for the different multigrid approaches is presented in Figure 2. The full multigrid approach produces results with an optimum convergence rate, but with a parallel performance that degrades for a large number of processors. One must take into account, that for these meshes ($n_i \times n_j = 1024 \times 64$) granularity becomes too large as the number of processors increases. This makes efficient parallel calculations not viable for a number of processors larger than 8. The multigrid performed implicitly within blocks (see Section 4.2) exhibits better parallel performance, but at a very high cost. When the total cost to achieve a solution converged to the same level through these two procedures is computed, the second technique requires about twice the number of cycles making this approach less than desirable. Finally, the "lazy"-"dumb" version of the algorithm performs slightly better when the "dumb" processor computes the two coarsest levels of the sequence. When the three coarsest levels are assigned to the "dumb" processor, parallel performance degrades since the amount of time required by the "dumb" processor exceeds that needed for all processors (with poor parallel performance). Notice that wall time was used to compute the parallel efficiencies in all cases. If the unused CPU time of the "lazy" processors were to be factored in, the gain would be larger. It must also be mentioned that the implementation of this approach introduces a higher level of complexity to the multigrid algorithm, since the memory management on both types of nodes differs

considerably from the traditional one. As a payoff for the additional work, this approach exhibits the exact same rate of convergence as the full multigrid algorithm.

Evaluating the performance of multigrid in meshes that are relatively small places a strong restriction on the parallel efficiencies that can be achieved. Calculations using very large meshes (three-dimensional calculations, two-dimensional LES calculations, etc) will benefit from the full multigrid algorithm and will achieve parallel performances on the order of 90% since most of the time will be spent on the finer levels of the sequence. Nevertheless, for engineering calculations, reasonable speed-ups can be obtained through this method.

The performance results of the different methods used to implement residual smoothing are presented in Figure 3. The fully implicit approach is clearly unacceptable even for a small number of processors. While the approach that uses implicit residual smoothing within blocks exhibits a parallel performance of 96% for 8 processors, the degradation of the convergence rate disqualifies it as a possible candidate. Finally, the iterated residual smoothing preserves both the convergence rate and a reasonable parallel speed-up, and is chosen as the candidate for engineering calculations.

Figure 4 presents a detail of the domain decomposition for a calculation involving four processors for a typical airfoil section. Figure 5 shows the pressure contours at different points on the oscillation period of a NACA 64A010 airfoil at a Reynolds number of 10^6, $M_\infty = 0.796$, and a reduced frequency, $k_c = 0.202$. These results were obtained with the O-mesh version of the program. One can clearly see how the shock waves strengthen and weaken as the airfoil pitches up and down.

The lines that intersect the pressure contour lines are the interprocessor boundaries of the O-mesh. Perfect continuity of these contour lines validates the accuracy of the code even for the cases where strong shocks are close to the interprocessor boundaries. For more details, please refer to [6].

7 CONCLUSIONS

A parallelized, two-dimensional, unsteady Navier-Stokes flow solver has been developed. Parallelization was realized using a domain decomposition approach to achieve proper load balancing and computational efficiency. PVM communication software was used for message passing between processors. Strategies for dealing with two convergence acceleration techniques, namely implicit residual smoothing and multigrid, and the performance of each of these techniques have been evaluated.

It is observed that for meshes of sizes typically used in engineering calculations, acceptable parallel performances can be achieved with up to 8 processors. A larger number of processors is not suitable for this type of calculation. For larger meshes, the multigrid technique is still quite favorable even in multiprocessor architectures. Implicit residual smoothing can be performed in an iterative fashion without an observable impact in the convergence rate while retaining good parallel performance. All calculations used the public distribution of the PVM software developed at Oak Ridge National Labs. Preliminary results with the IBM optimized version of this message passing standard (PVMe), on both the SP1 and SP2 platforms, confirm the expected trends in which parallel performances improve considerably. Finally, results for the unsteady pressure field around a pitching NACA 64A010 at $M_\infty = 0.796$ were computed on the IBM SP1 system.

References

[1] Martinelli, L., Jameson, A., Validation of a Multigrid Method for the Reynolds Averaged Equations, AIAA Paper 88-0414, AIAA 26th Aerospace Sciences Meeting, Reno, January 1988.

[2] Jameson, A., Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings, AIAA Paper 91-1546, AIAA 10th Computational Fluid Dynamics Conference, Honolulu, June 1991.

[3] Geist, A., Beguelin A., Dongarra J., Jiang W., Manchek, R., Sunderam V., PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratory, May 1993.

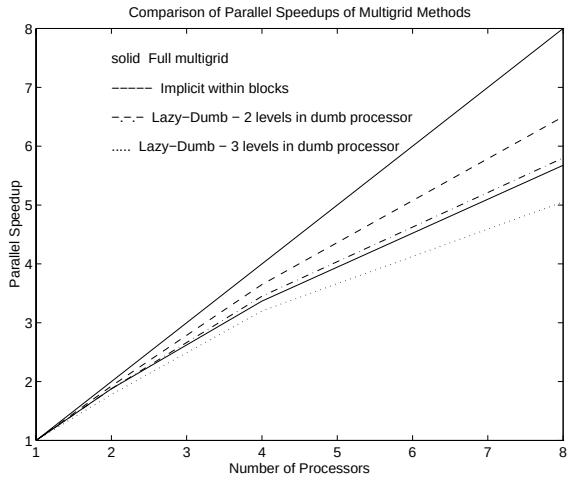

Figure 2: Summary of Parallel Speed-ups for Different Approaches to the Multigrid Technique.

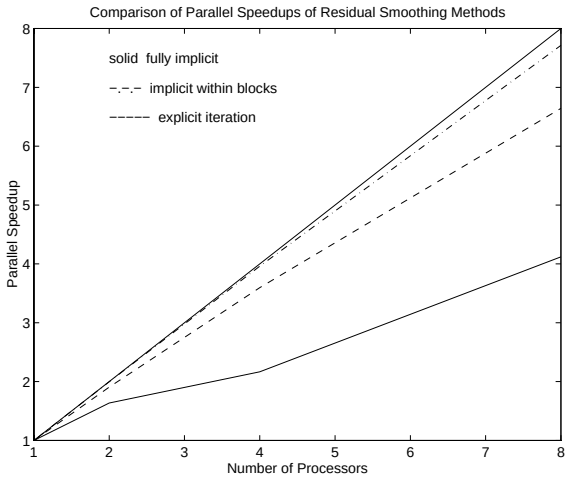

Figure 3: Summary of Parallel Speed-ups for Different Implicit Residual Smoothing Approaches.

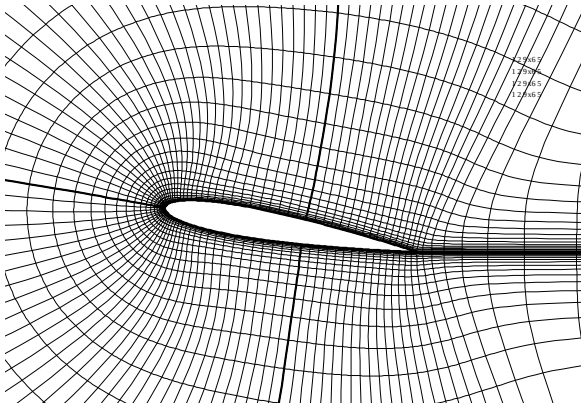

Figure 4: Domain Decomposition for a NACA 0012 Airfoil at a 10 degree Angle of Attack.

[4] Jameson, A., Transonic Flow Calculations, Princeton University Report 1651, March 1984, in Numerical Methods in Fluid Dynamics, edited by F. Brezzi, Lecture Notes in Mathematics, Vol. 1127, Springer-Verlag, 1985, pp. 156-242.

[5] Yadlin, Y. and Caughey, D. A., Block Multigrid Implicit Solution of the Euler Equations of Compressible Fluid Flow, AIAA Journal, 29(5):712-719.

[6] Alonso, J. J., Martinelli, L., and Jameson A., Multigrid Unsteady Navier-Stokes Calculations with Aeroelastic Applications, 33rd AIAA Aerospace Sciences Meeting, AIAA Paper 95-0048, Reno, NV, January, 1995.

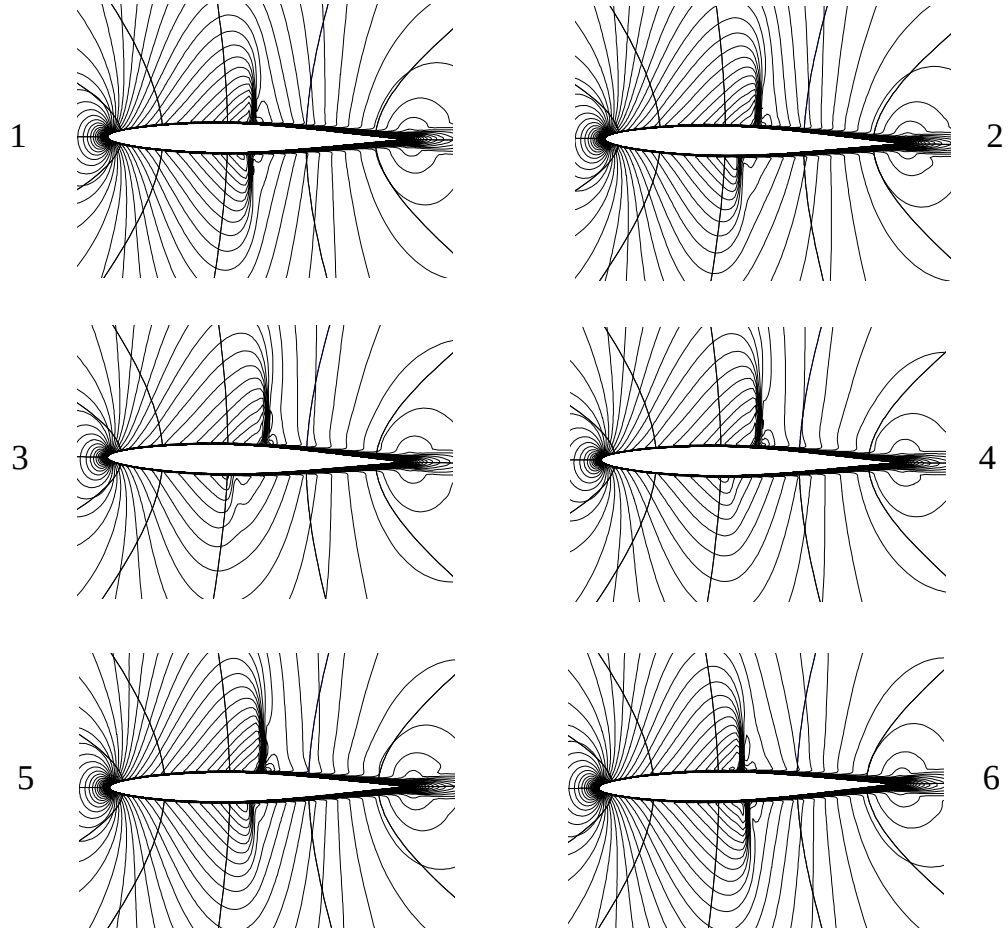

Figure 5: Mach Number Contours. Pitching Airfoil Case. $Re = 1.0 \times 10^6$, $M_\infty = 0.796$, $K_c = 0.202$. Read figures by lines.