NASA Technical Memorandum 106449
AIAA–94–0408
ICOMP–93–49

Preconditioned Implicit Solvers for the Navier-Stokes Equations on Distributed-Memory Machines

Kumud Ajmani
Institute for Computational Mechanics in Propulsion
Lewis Research Center
Cleveland, Ohio

and

Meng-Sing Liou and Rodger W. Dyson
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio

Prepared for the
32nd Aerospace Sciences Meeting and Exhibit
sponsored by the American Institute of Aeronautics and Astronautics
Reno, Nevada, January 10–13, 1994

NASA

Preconditioned Implicit Solvers for the Navier-Stokes Equations on Distributed-Memory Machines

Kumud Ajmani*
Institute for Computational Mechanics in Propulsion
Lewis Research Center
Cleveland, Ohio

Meng-Sing Liou† and Rodger W. Dyson‡
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio

Abstract

The GMRES method is parallelized, and combined with local preconditioning to construct an implicit parallel solver to obtain steady-state solutions for the Navier-Stokes equations of fluid flow on distributed-memory machines. The new implicit parallel solver is designed to preserve the convergence rate of the equivalent 'serial' solver. A static domain-decomposition is used to partition the computational domain amongst the available processing nodes of the parallel machine. The SPMD (Single-Program Multiple-Data) programming model is combined with message-passing tools to develop the parallel code on a 32-node Intel Hypercube and a 512-node Intel Delta machine. The implicit parallel solver is validated for internal and external flow problems, and is found to compare identically with flow solutions obtained on a Cray Y-MP/8. The computational speed on 32 processing nodes of the i860 machines is comparable to the speed on a single processor of the Cray Y-MP. A peak computational speed of 2300 MFlops/sec has been achieved on 512 nodes of the Intel Delta machine, for a problem size of 1024K equations (256K grid points).

Introduction

Parallel machines based on distributed-memory architectures are providing viable alternatives to expensive vector supercomputers for performing numerical integrations of large-scale Computational Fluid Dynamics (CFD) problems. As CFD codes are combined with other codes to perform multidisciplinary work, severe pressures will be exerted on modern supercomputers — both in terms of memory requirements and CPU time.

The computational power of multiprocessor machines with a large number of processors can be harnessed to obtain solutions for the non-linear partial differential equations of fluid flow. CFD can choose between the wide variety of machine architectures currently available — shared-memory machines (e.g. Cray Y-MP), distributed-memory machines (e.g. Intel Hypercube) or machines with hybird architectures. This paper concentrates on using a distributed memory, Multiple Instruction Multiple Data (MIMD), message-passing architecture to obtain solutions to the Navier-Stokes equations of fluid flow.

The Navier-Stokes equations are discretized in space using an upwind, finite-volume, flux-split formulation. The discretized equations are linearized with an Euler-implicit linearization, and integrated in time to obtain a steady-state solution. The Euler-implicit linearization produces a system of simultaneous linear equations characterized by a large, sparse, non-symmetric coefficient matrix. The work described in this paper concentrates on investigating parallel implementations of Krylov-subspace methods, particularly GMRES[1], for solving this linear system of equations at each time-step of the time-integration.

In earlier work on parallel Krylov solvers, Saad & Schultz[2] combined GMRES with ILU preconditionings on shared-memory machines and loosely-coupled linear or mesh-connected arrays. O'Leary[3] implemented the block Conjugate Gradient algorithm on a coarse-grained parallel machine. Anderson & Saad[4] examined the standard ILU(0) and polynomial preconditioners for shared-memory machines. Their work concludes that ILU(0) may

* Research Associate, AIAA Member
† Senior Scientist, AIAA Member
‡ Computer Engineer

outperform polynomial preconditioning if the number of processors is small. Radicati & Robert[5] used overlapped domain decomposition to implement ILU preconditioning on a shared-memory multiprocessor. Their numerical experiments showed that a local ILU factor on overlapping blocks is a good preconditioning strategy. Baxter et. al[6] examined the performance of Krylov solvers with ILU preconditioning on a shared-memory machine and on Hypercube architectures. Application problems from reservoir engineering and mathematics were studied in their work.

In more recent work on machines with large number of processors, Berryman et. al[7] have measured the performance of key interactive kernels (sparse triangular solves, inner products etc.) of preconditioned Krylov solvers on the CM-2 machine. The ideas of multi-level domain decomposition for preconditioned Krylov solvers have been presented by Gropp and Keyes[8]. Shadid and Tuminaro[9] have implemented GMRES and other Krylov solvers on a 1024-node ncube/2 Hypercube. Their work has examined the efficiency of various local and global preconditioners, for several standard, model problems. Desturler[10] has proposed a modified, computationally cheaper version of GMRES for medium to fine grained parallelism on MIMD machines.

In summary, several authors have explored implementations of Krylov solvers on shared-memory and distributed-memory architectures. Most of the works have examined 'model' problems arising from elliptic and hyperbolic PDEs. The ILU factorization seems to be a popular preconditioner, as several efforts have been made to study its performance in the parallel environment. However, a comprehensive study of the performance of preconditioners and Krylov solvers for practical CFD problems appears to be lacking in the literature.

This paper investigates the viability of using the preconditioned GMRES algorithm in a domain-decomposition framework to develop an implicit solver for solving CFD problems on distributed-memory machines. A new local preconditioner, which is much cheaper than the popular ILU preconditioner, has been developed to support efficient implementation of the parallel GMRES solver. The new preconditioner is based on symmetric Gauss-Seidel sweeps across each domain, and shows excellent scalability over a large range of processors. The implicit parallel solver is validated on a 32-node Intel Hypercube and a 512-node Intel Delta machine.

This introduction section is followed by a section describing some of the theory of parallel machines (MIMD architectures in particular), CFD and Krylov solvers. Detailed results of computations on an Intel Hypercube and Intel Delta machines will then be presented. The paper will conclude with some remarks about the present results and some ideas for future work in the area of implicit parallel solvers for CFD codes.

Presentation of Theory

Distributed-Memory Systems

Multiple Instruction Multiple Data (MIMD) or distributed-memory machines are characterized by a grouping of processors which are capable of functioning independently as computational nodes. Each processor has individual memory, computational units and communication units. Information is exchanged amongst processors by sending packets of information or 'messages' from one processor to another. Each processor has its own clock, and there is is no 'global' clock. The processors can be 'synchronized' by performing a 'global' communication. The connectivity between processors defines the topology of the machine and determines the speed at which messages are passed from one processor to another.

The processors in a Hypercube architecture are interconnected with a cube-type connectivity ; each processing node lies on the vertex of an order-n cube. The 32-node Intel Hypercube machine at NASA Lewis Research Center is organized as a cube of order 5. In contrast, the processors for the 512-node Intel Delta machine at CalTech are arranged in a mesh-type connectivity (16*32 mesh). Both machines are based on the Intel i860 microprocessor, with 16 MBytes of memory per node. The i860 is a 40 MHz RISC microprocessor with a peak theoretical rating of 32 MIPS (integer performance) and 60 Mflops (64-bit floating-point performance). The communication networks for the Intel machines are characterized by relatively low communication bandwidths and high communication latencies. This implies that a few long messages are preferable to numerous short messages. Further details of the Hypercube architecture may be found in reference 11.

Navier-Stokes Equations

The governing equations of compressible fluid flow in 2-D are the Navier-Stokes equations written

as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial F_v}{\partial x} + \frac{\partial G_v}{\partial y} \qquad (1)$$

where Q is the vector of independent conserved variables, F and G are inviscid flux vectors and F_v and G_v are viscous flux vectors. The governing equations are solved computationally in their integral, conservation law form in generalized coordinates, using a cell-centered finite volume formulation. The inviscid fluxes are upwinded using Van Leer's[12] flux-splitting scheme. The viscous fluxes are evaluated with second-order accurate central-differences. Additional details of the 'serial' code may be found in reference 13.

The generalized-coordinate form of equation 1 can be written in compact form as

$$\frac{1}{J}\frac{\partial Q}{\partial t} = -R \qquad (2)$$

where J is the jacobian of the transformation from cartesian to generalized coordinates. R is called the residual vector, and equals to zero for a steady-state solution. The accuracy of the computed solution is directly affected by the accuracy of the residual vector computation. A nine-point stencil is used for second-order accurate calculations of the residual vector.

The Euler-implicit time-linearization of equation 2 results in

$$\frac{\Delta Q^n}{J\Delta t} = -R^{n+1} \qquad (3)$$

where ΔQ^n is the incremental change in the cell-centered values of the vector Q between the $n+1^{\text{th}}$ time level and the known n^{th} time level, i.e. $\Delta Q^n = Q^{n+1} - Q^n$. R^{n+1} is linearized in time about the n^{th} time level which results in

$$\left(\frac{I}{J\Delta t} + \frac{\partial R}{\partial Q}\right)^n \Delta Q^n = -R^n \qquad (4)$$

where $\frac{I}{J\Delta t}$ is a block-diagonal matrix and $\frac{\partial R}{\partial Q}$ is a large, sparse, block, banded non-symmetric matrix. In this paper, $\frac{\partial R}{\partial Q}$ is evaluated with a five-point stencil (as compared to a nine-point stencil computationa for the R). This is done to reduce the computational and storage costs associated with a nine-point stencil evaluation of $\frac{\partial R}{\partial Q}$ at the expense of increased time-steps required to reach a converged steady-state solution.

Equation 4 can be rewritten in matrix-vector form as

$$[V^n]\{\Delta Q^n\} = -\{R^n\} \qquad (5)$$

Equation 5 represents the system of simultaneous linear equations which has to be solved for ΔQ^n at each time-step of the time-integration. For small (of order 100), well-conditioned coefficient matrices, the system may be solved exactly by inverting the matrix $[V^n]$ at each time-step. However, for large and/or poorly-conditioned matrices (found in practical CFD applications), an iterative solution of equation 5 becomes more viable. The preconditioned GMRES method, investigated in reference 13, is parallelized to solve equation 5. Some issues related to the development of the parallel solver are now discussed.

Parallel Domain-Decomposition

The discretized Navier-Stokes equations can be solved in a parallel framework by decomposing the original, large uniprocessor domain of grid points into a number of smaller domains which are then distributed to the available processors (one domain per processor). In a distributed-memory system, each processor can only access information from its own local-memory. Thus, information may have to be exchanged across the domain interfaces (or processor boundaries) in order to preserve the characteristics of the original problem.

Recall, that the residual vector computation uses a nine-point stencil. Thus, the flux-evaluation for cell-faces which lie on (and adjacent to) domain boundaries will require information from (a maximum of) two adjacent cells which reside in a neighboring processor. This information exchange is facilitated by creating two layers of 'ghost' cells at each domain boundary. At each time-step, data from the neighboring domains is 'communicated' to these 'ghost' cells before the flux-evaluation routines are invoked. This communication is done prior to the application of the explicit boundary conditions. The flux-balance evaluated by this approach has been validated to be identical to the flux-balance computed for the original uniprocessor domain. Note that each domain must contain at least three cell-faces (in each coordinate direction) for this approach to work successfully.

The implementation of boundary conditions at physical boundaries may also require communication amongst processors. Airfoil calculations on C and O-type meshes require communication between non-neighboring processors in order to effect C and O-type periodicity. This is achieved by communication amongst domains which lie along the wake-cut line of the particular C or O-type grid. Note,

that the boundary condition routines are invoked only for those processors which contain actual physical boundaries corresponding to the inflow, outflow, bottom and top planes of the original uniprocessor domain. This creates an imbalance in the workload across the processors, since the 'interior' processors do not perform boundary condition calculations. This imbalance is not significant since $< 1\%$ of the total computer time is required for the boundary condition computations.

The inviscid and viscous flux vectors, and the respective flux-jacobian matricesare first calculated. The individal flux-jacbian matrices are then assembled into the implicit, left-hand-side coefficient matrix, for each domain. The coefficient matrix is assembled from linear combinations of the flux-jacobian matrices. Each domain assembles its own individual matrix, and no extra communication is required for this computational step. A five-point stencil is used for the implicit operator, providing a sparse, banded, coefficient matrix with five block-diagonals.

The Parallel GMRES solver

The original, large, system of linear equations corresponding to the uniprocessor domain is thus transformed to a series of smaller linear systems, with one linear system for each processor. A preconditioned GMRES solver is used to iteratively solve each linear system. The original GMRES method is designed to iteratively solve linear systems with non-symmetric coefficient matrices. In order to solve a linear system (say $Ax = b$), the method seeks an approximate solution x_k of the form $x_k = x_0 + z_k$, where x_0 is some arbitrary initial guess to the exact solution $\bar{x}$. The vector z_k lies in the Krylov subspace, $K(A, r_0, k)$, defined by the matrix A, the unit-vector $w_1 = r_0/\beta$ ($r_0 = b - Ax_0$, $\beta = ||r_0||$) and the size of the Krylov subspace k.

The GMRES solver will converge the fastest when each successive iterate x_k minimizes the residual norm, $||r_k||$ over the subspace $K(A, r_0, k)$. One major practical difficulty with GMRES is that when k increases, both storage and operation cost increase as $O(k)$ and $O(k^2)$, respectively. If the available storage is limited, the method may be restarted after k sub-iterations, with x_k replacing x_0 in step 1. The restart version is often used in practical problems and is referred to as GMRES(k).

The complete GMRES algorithm can be written as follows:

1. For any starting vector x_0, form $r_0 = b - Ax_0$; $\beta = ||r_0||_2$; $w_1 = r_0/\beta$
2. Perform k steps of Arnoldi's method[14] with w_1 as the initial vector to form k vectors which are successively orthogonal to w_1
3. Find the vector (say z_k) which minimizes $||r_k||$ in the subspace defined by the vectors in step 2. Compute the solution $x_k = x_0 + z_k$

The GMRES algorithm involves three basic linear algebra operations — inner-products of vectors (steps 1 & 2), *saxpy* operations (steps 1 & 3) and matrix-vector products (steps 1 & 2). Evaluation of the inner-products requires inter-processor communications since the local inner-products have to be accumulated across all the processors. This can be done by passing $2log_2 N$ messages across the N processors. The *saxpy* operation can be performed independently by each processor, since only local data needs to be manipulated.

Each matrix-vector operation requires communication of the 'boundary' elements of the particular multiplying vector to neighboring processors. The components that correspond to the cells lying on the the four boundaries of each domain are communicated to 'ghost' cells of the neighboring domains. This is critical in order to reproduce the uniprocessor matrix-vector product, i.e. the product resulting from multiplying the original, single-domain coefficient matrix with a given vector. The multiple-processor matrix-vector product is required to be identical to the uniprocessor matrix-vector product, at each sub-iteration of the GMRES solver. This is to ensure that the parallel GMRES solver (without preconditioning) has the exact convergence rate as the serial GMRES solver.

Preconditioning the Linear System

The rate of convergence of any iterative algorithm depends on the condition number, $\kappa_2(A)$, of the iteration matrix A and the distribution of singular-values of A. If $\kappa_2(A)$ is large and/or the spectrum of singular-values of A is wide and scattered, the matrix A is poorly conditioned, and the underlying algorithm may converge slowly. Preconditioners improve the conditioning of the iteration matrix, and usually have a first-order effect on improving the convergence rate and overall efficiency of solvers based on GMRES-like algorithms. Formally, a preconditioning matrix M transforms the original system $Ax = b$ into

$$M^{-1}Ax = M^{-1}b \quad \Leftrightarrow \quad \tilde{A}x = \tilde{b} \qquad (6)$$

The operation of M^{-1} on any vector (say $u = Ax$) is equivalent to the solution of a linear system

$M\tilde{u} = u$, with M as the coefficient matrix. Such linear systems have to be solved repeatedly for each sub-iteration of the preconditioned GMRES algorithm. Any matrix M which produces easy-to-solve linear systems of the type $M\tilde{u} = u$ (e.g. M = Diagonal of A), is a potentially efficient preconditioner.

The costs associated with preconditioning can be enumerated as (i) Computation of the preconditioning matrix M, (ii) Linear system solves associated with M, and, (iii) Additional storage for the matrix M, which may be of the order of storage requirements for the matrix A. The selection of an 'efficient' preconditioner is motivated by the minimization of the afore-mentioned costs.

Several different preconditioners that can be chosen are diagonal (M = major diagonal of A), block-diagonal, incomplete L-U factorization (ILU) and block-ILU[15] — in increasing order of the cost to calculate and store M. Iterative methods used in existing CFD codes can also be used as effective preconditioners. A variant of the iterative scheme of Yoon and Jameson[16] is used locally in each domain as a parallel preconditioner. This preconditioner, referred to herein as the LUSGS preconditioner, was validated with the serial GMRES algorithm in reference 13, and found to be much more efficient (in terms of CPU time and storage) and extremely competitive (in terms of convergence rate) with the currently popular preconditioners based on ILU factorizations of the matrix A.

The LUSGS preconditioner is applied to solve linear systems like $[M]\{\tilde{u}\} = \{u\}$ as follows :

1. Approximately factor $[M] \approx [L][D]^{-1}[U]$ where $[D]$=major block-Diagonal of $[M]$, $[L]$=($[D]$− lower-triangular part of $[M]$), $[U]$=($[D]$− upper triangular part of $[M]$).
2. Invert the block-diagonal matrix [D]
3. Sweep forward (bottom-left to top-right) to solve $[L]\{\hat{u}\} = \{u\}$ for $\{\hat{u}\}$.
4. Sweep backward (top-right to bottom-left) to solve $[U]\{\tilde{u}\} = [D]\{\hat{u}\}$ for $\{\tilde{u}\}$.

The LUSGS preconditioner described above is used without modification in the parallel implementation. Consequently, the sweeps in steps 3 & 4 (which are now partial sweeps limited to the confines of the individual domains) will not produce the same solutions as the serial algorithm. However, this is found to have only marginal effects on the convergence of the parallel, preconditioned GMRES solver. The preconditioner performs excellently for rectangular or square domains, and the performance deteriorates slightly for high aspect-ratio domains. The LUSGS preconditioned GMRES solver is found to maintain its convergence characteristics to within 5% of the serial solver (for upto 512 processors). This demonstrates the excellent scalability of the new solver. Note, that if the number of processors equals the number of computational cells, the LUSGS preconditioner is equivalent to a fully-scalable block-diagonal preconditioner.

I/O and Memory Considerations

This paper uses the Single Program Multiple Data (SPMD) model of parallel programming to run identical copies of the code on all processors. Each processor performs its input/output operations independently of the other processors. The fastest way of performing I/O operations on both iPSC/860 systems is to read/write from/to the Concurrent File System (CFS). Each processor can access data from the CFS at a peak rate of 1.5 Mbytes per second.

Three data files are required by each processor — an input parameter file, a grid file and a restart file (if restarting). In the current implementation, all processors read from a commonly shared parameter file and grid file, both of which reside on the CFS. Each processor determines its position in the global domain, and correspondingly extracts the relevant information from the parameter file and grid file. Each processor is provided its own unique restart file (if restarting), which it reads directly from the CFS.

On completion of the user-specified time-steps, each processor outputs a solution file directly to the CFS. This invokes multiple writes to the CFS and may cause delays due to contention for the I/O nodes as all processors try and write to the CFS at the same time. Since the global solution is distributed across the various processors, a post-processing program has been written to assemble the global solution from the various output files. The global solution file can also be used to generate restart information for any number of processors. The post-processor can be incorporated into the CFD code itself, but has been avoided in favor of the increased flexibility afforded by the current approach.

It must be mentioned that if the I/O operations are done to/from the front-end (or 'remote host') system (a Sun Sparc10 Workstation), the wall-clock time of each run increases considerably. In addition, if intermittent solution files have to be output to the CFS (e.g. for an unsteady calculation), the I/O time can tend to dominate the overall wall-clock time.

The memory requirements for the implicit code are estimated at 320 words per grid point per processor. This includes storage for 10 search directions of the GMRES solver. 16 MBytes of RAM is available on each processor of the Intel Hypercube and Intel Delta machines. In practice, a maximum of $\approx$ 3000 grid points (corresponding to $\approx$ 10 MBytes of RAM) could be assigned to each processor, when using 64-bit floating-point arithmetic. The remaining memory is assigned for data, performance monitoring tools, system software and communications software. Hence, the maximum problem size is restricted by the total available memory on the parallel machine.

Test Results and Discussion

A parallel, preconditioned GMRES solver has been implemented for implicit solutions of the two-dimensional, upwind, finite-volume, Navier-Stokes equations. The global uniprocessor domain representing the computational grid is partitioned among the processors of a distributed-memory machine. Each processor runs identical copies of the same computational code on different sets of data. The processors communicate with each other at several times during each computational time-step in order to exchange information.

The parallel code has been developed on an Intel Hypercube with 32 processors. All code-development, testing and debugging, and performance optimization has been done on the Hypercube. The parallel code has been validated against the original serial code (which is run on a single processor of the parallel machine). Results from the parallel residual vector computation and parallel GMRES solver have been validated independently over different domain decompositions, and found to be identical to the serial code. This ensures complete scalability of the domain decomposition algorithm and the unpreconditioned GMRES solver. The two problems selected for validation were low-speed flow over a backward-facing step and subsonic flow over a NACA 1406 airfoil. Subsequently, the parallel code was ported to an Intel Delta machine with 512 processors. All performance results presented here are based on data obtained from computations on the Intel Delta machine.

Parallel Code Validation

The problem of computing low-speed flow laminar over a backward-facing step was the first test case to validate the parallel solver for internal flow conditions. This flow problem illustrates the phenomena of flow separation and recirculation in internal flows. All flow variables are second-order accurate, fully-upwinded in the streamwise direction, and third-order accurate, upwind-biased in the normal direction. The implicit (left-hand-side) operator is discretized in a first-order accurate manner. Excellent comparisons with experimental data of Armaly et. al[17] have been obtained for this problem with the serial code[13].

The parallel validation was performed on 16 nodes of the Hypercube, with a $4 * 4$ decomposition of the $61 * 51$ global grid. This partitioning assigns $16 * 14$ grid points to nodes 0–7 and $16 * 13$ points to nodes 8–15. The uniprocessor grid is shown in Figure 1. A freestream Mach number of $M_\infty = 0.1$ was specified. Four different flow conditions with Reynolds number of Re_∞ = 100, 200, 300 and 389 were computed. Adiabatic, no slip boundary conditions were used on the top and bottom walls forming the boundaries of the channel, and on the lower portion (which defines the step) of the inflow boundary. For fully developed subsonic flow at the outflow boundary, three variables (ρ, u and v) were extrapolated and the pressure was determined by fixing the stagnation enthalpy. The parabolic velocity profile at the inflow boundary was simulated by imposing a profile of Reimann invariants.

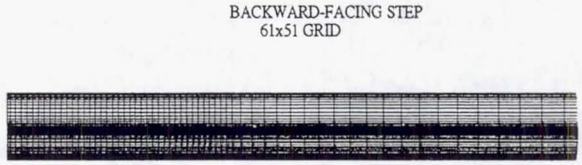

Figure 1. Computational Grid for Backward-Facing Step

Figure 2 shows Mach number contours obtained for the Re = 389 case. The nature and size of the separation and recirculation behind the step closely matches the physical description of the flow as obtained in the experimental data. The ratio of reattachment distance (X_R) to step-height (S) was found to be identical to the uniprocessor calculations, for all the four Reynolds' numbers. The calculated X_R/S ratios also compared very well (to within $\pm 5\%$ accuracy) with the experimental data.

Low Reynolds number, laminar, subsonic flow over a NACA 1406 airfoil was the second validation test case for the parallel solver, for external flow conditions. The flow conditions correspond to a freestream Mach number of M_∞ = 0.6, an-

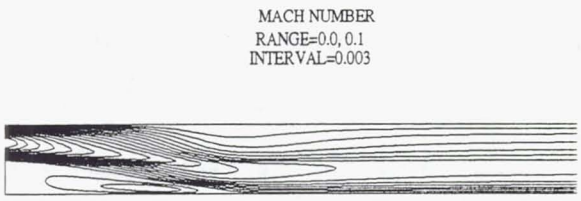

Figure 2. Mach Number Contours for Backward-Facing Step

gle of attack, $\alpha = 1.0°$, and Reynolds number, $Re = 5.0 * 10^3$. The computational grid was a 'C' mesh of $257 * 65$ points, and is shown in figure 3. The far-field boundary was placed five-chords from the airfoil and points are clustered near the airfoil to resolve viscous gradients. All flow variables are third-order accurate, upwind-biased in the streamwise and normal directions.

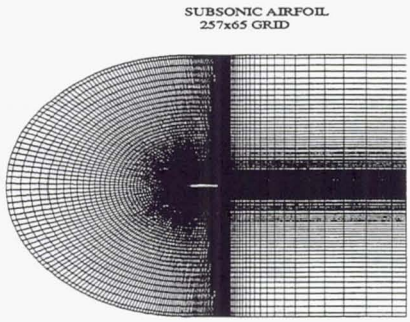

Figure 3. Computational Grid for Subsonic Airfoil

The parallel validation for this case was performed on 32 nodes of the Hypercube with an $8 * 4$ partitioning of the $257 * 65$ grid to yield $65 * 17$ grid points per node. Figure 4 is a plot of the computed steady-state pressure coefficient, C_p, on the surface of the airfoil. The computed lift, drag, and pitching moment coefficients obtained were C_L=0.18148, C_D=0.041703, and C_M=-0.023718, respectively. These coefficients compared exactly with those computed with the serial version of the code.

Convergence Rate Comparisons

A comparison of convergence rates on the serial and parallel machines reveals the effectiveness and accuracy of the parallel implicit solver. A comparison for the backward-facing step test case is shown in figure 5. A constant Courant number of 50 has been used. It is clear that the parallel preconditioned GMRES solver retains the convergence char-

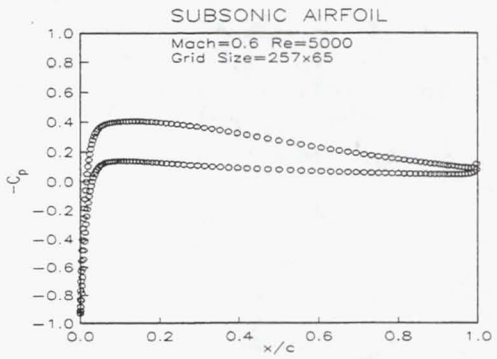

Figure 4. Chordwise Distribution of C_p

acteristics of the serial solver. The differences in the serial and parallel solvers arise because of the local (instead of global) implementation of the parallel LUSGS preconditioner. The LUSGS sweeps were restricted to the individual processor domains, and no additional message-passing was done to reproduce the uniprocessor LUSGS sweeps of the global domain. This seemed to have a negligible impact on the convergence rate of the parallel solver.

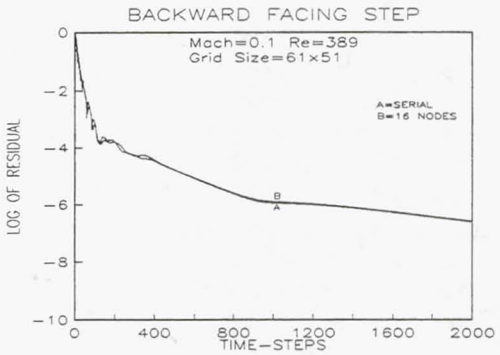

Figure 5. Convergence histories for Backward-Facing Step

Figure 6 traces the convergence rates for the subsonic airfoil calculations, for 1500 time-steps. The parallel preconditioned GMRES solver was tested on 16(8*2), 32(8*4), 64(16*4), 128(16*8) and 256(32*8) nodes. A constant Courant number of 10 was used for the comparisons with the serial solver. It can be seen in figure 6 that the convergence rate of the parallel preconditioned GMRES solver decreases as the number of processors increases. This decrease implies that the number of time-steps required by the parallel solver to attain an eight-order reduction in the l_2 norm of the residual vector, will increase slightly (5–10%) as compared to the serial solver. The decrease in convergence rate is negligible up to a four-order residual reduction, which is usually sufficient for most engineering problems. Thus, it can

be claimed that an implicit parallel code (including the preconditioner, the GMRES solver and the residual-vector computation) has been designed to perform consistently over a large number of processors in a distributed-memory environment.

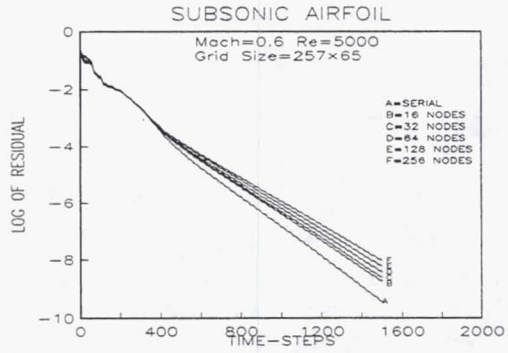

Figure 6. Convergence histories for Subsonic Airfoil

Parallel Performance Results

The parallel code was run on a single node of the Hypercube to determine the single-processor performance. The code was compiled with the maximum available vectorization and optimization options. The 61*51 grid from the backward-facing step calculation was used, as this was the largest number of points that could be accommodated on a single node (in accordance with memory requirements of the CFD code). The preconditiond GMRES solver was run for 100 time-steps, with 5 sub-iterations per time-step. The average single-processor CPU time for the Intel Hypercube was recorded as 340 seconds. The total number of floating-point operations were determined by invoking the hardware performance monitor (hpm) of the Cray Y-MP. The hpm indicated that the code performed 1300 Mflops, which translated to a 1300/340=3.8 Mflops/sec rating for a single processor of the Intel Hypercube. The unpreconditioned GMRES solver performed at a higher rate of 5.9 Mflops/sec on a single node of the Intel Hypercube (1290 Mflops, 220 seconds, 100 time-steps, 10 sub-iterations per time-step).

The slower performance of the LUSGS preconditioned GMRES solver can be attributed in part to the inherent lack of vectorization of the LUSGS preconditioner. However, in practice, the faster vector performance of the unpreconditioned GMRES solver was sufficiently compensated for by the much superior steady-state convergence rate of the preconditioned solver. The use of the LUSGS preconditioner considerably reduced the CPU time to attain a steady-state solution[13]. This suggests that the LUSGS preconditioner can be used profitably in a parallel environment, provided the convergence rate does not suffer as the number of processors is increased.

Recall that the i860 processor is rated at 60 Mflops/sec for 64-bit floating-point operations. Hence, when running at 5.9 Mflops/sec, only 10% of the peak performance is achieved (by the unpreconditioned GMRES solver) on a single node. These performance numbers seem to be very low, but they compare very favorably with other typical CFD applications[18] on machines built around the i860 microprocessor. As a comparison, the unpreconditioned and LUSGS-preconditioned GMRES solvers performed at rates of 120 Mflops/sec and 170 Mflops/sec, respectively, on a single processor of the Cray Y-MP located at the NASA Lewis Research Center.

The CPU times for the subsonic airfoil calculation are plotted in figure 7. This figure demonstrates that a parallel implementation on 32 nodes can match the turnaround time of a serial implementation on a single processor of a Cray Y-MP. In addition, a parallel implementation on 256 nodes is three times faster than a Cray Y-MP implementation, in terms of CPU time to convergence. Note, that for larger problem sizes, the potential gain in CPU time with 256 nodes is much larger. This is because the ratio of computational work to communication work increases with problem size, and each processor is utilized more efficiently.

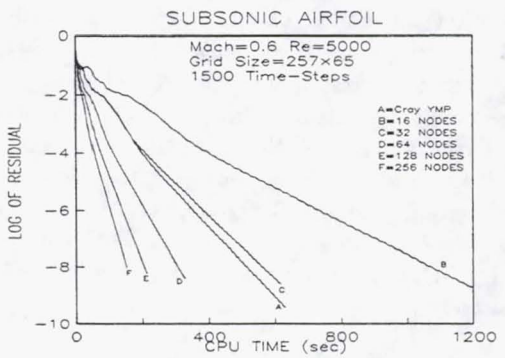

Figure 7. CPU time Characteristics for Subsonic Airfoil

Figure 8 shows a profile of the run-time characteristics for the parallel code on $32(8*4)$ of the Intel Hypercube, for the $257*65$ grid airfoil problem. The computational work is in slight imbalance because the nodes which process the boundary-

condition information perform more work than the 'interior' nodes. The communication load is in slight imbalance because the number and length of messages varies across each node (e.g. boundary nodes have fewer messages). However, this does not have an effect on the overall load balance since the communication time is only a small fraction (≈5%) of the total CPU time.

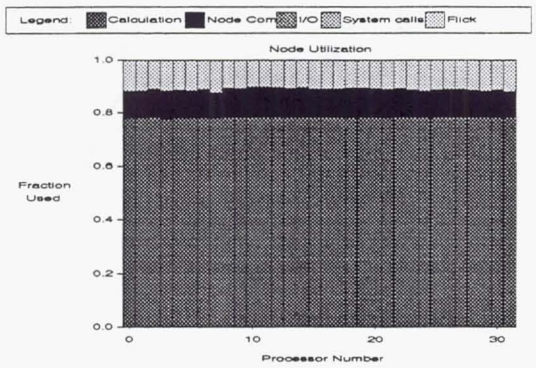

Figure 8. Run-Time Profiling Characteristics (32 nodes)

In this work, three grids of dimensions 193*161, 257*257 and 513*513 were employed to study the effects of computational load on parallel performance. The backward-facing step problem was chosen as the test case. CPU times for 100 time-steps (10 sub-iterations per time-step) of the unpreconditioned GMRES solver were recorded. The performance for each grid is summarized in figure 9. A peak performance corresponding to 2300 Mflops/sec (512 nodes) is achieved for the 513*513 grid. The 256*256 and 193*161 grids have reduced peak performances of 1350 Mflops/sec and 750 Mflops/sec, respectively.

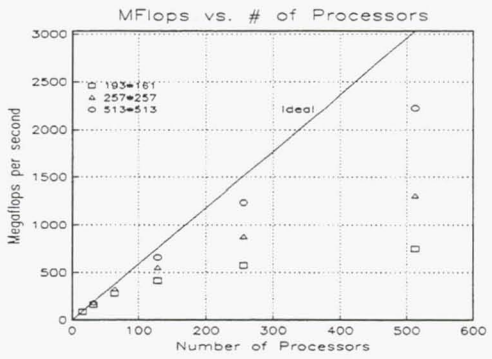

Figure 9. Variation of Performance vs. Load

The 'ideal' performance (figure 9) is based on the single-node performance of 5.9 Mflops/sec for a domain size of ≈ 3000 points. This implies that for a fixed size of the uniprocessor grid, the performance will be less than 'ideal' as the number of processors increases (and the domain size decreases). This is evident from the performance numbers for the 193*161 grid, which deteriorate rapidly from 5.3 Mflops/sec/node (16 nodes) to 2.32 Mflops/sec/node (256 nodes). However, when the grid size increases to 257*257, the performance numbers range from 5.6 Mflops/sec/node (32 nodes) to 3.4 Mflops/sec/node (256 nodes). The 513*513 grid performs in the range of 5.2 Mflops/sec/node (128 nodes) to 4.8 Mflops/sec (256 nodes).

The parallel efficiency for N processors, η_N, is calculated as

$$\eta_N = \frac{T_1}{NT_N}$$

where T_1 is the CPU time for one processor and T_N is the CPU time for N processors. The parallel efficiency characteristics for different grid sizes are shown in figure 10. As expected, η_N decreases as the number of grid points per processor decreases. It is estimated that $\approx$ 1024 grid points per node are required to keep η_N above 80%. Speedup factors for the different grids and processors can be derived from the values of η_N by using the relation $S_N = \eta_N * N$. For example, for the 513*513 grid, when N = 512 and η_N=0.74, S_N = 379. It must be remarked that the maximum speed-up for a fixed-size problem is governed by Amdahl's law[19], and depends on the distribution of sequential and parallel work in the code.

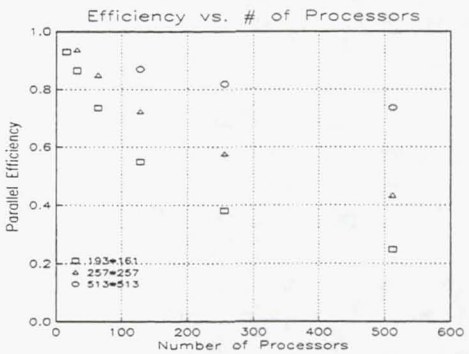

Figure 10. Variation of Efficiency vs. Load

Conclusions and Future Work

A parallel, implicit solver has been developed for distributed-memory parallel machines, for obtaining steady-state solutions of the compressible Navier-Stokes equations with a state-of-the-art CFD

code. The implicit solver is a combination of a parallelized Krylov solver (GMRES) and a scalable, local parallel preconditioner. This paper shows that the parallel, implicit solver provides steady-state convergence rates which compare excellently with serial implicit solvers used in shared-memory implementations. The domain-decomposition strategies adopted in this paper are validated for internal and external flow test cases, on a wide range of processing nodes.

The performance of the parallel CFD code varies as a function of the computational workload and the communication overhead for each processor. The parallel efficiency (defined as ratio of actual speedup to ideal speedup) is found to decrease as the amount of computational workload (or number of grid points) per processor decreases. The parallel CFD code peaks at a computational rate of 2300 Mflops/sec on a $513 * 513$ grid on 512 nodes of the Intel Delta machine. A parallel efficiency of 80% or greater is achieved if each processing node is assigned at least 1024 grid points. The parallel implementation is determined to be memory-bound, as a maximum of 3200 grid points can be accomodated in the 2MW RAM of each processor. The communication overheads are determined to be largely independent of the nature of the domain decomposition and the assignment of domains to processors. The total communication time constitutes roughly 5–7% of the total execution time.

The attainable single-node performance on the Intel machines (Hypercube or Delta) is 30 times lower than that on a single processor of a Cray Y-MP (6 Mflops/sec versus 170 Mflops/sec). However, a subsonic airfoil calculation on 256 nodes is demonstrated to run three times faster than a single-processor Cray Y-MP computation. Considerable improvements in the areas of compilers, data cacheing, memory-access times and I/O operations are required to further enhance the competitiveness of parallel machines for large, three-dimensional, unsteady-flow simulations of fluid-flow problems. Improvements in parallel algorithms, solvers and programming models will also contribute to the acceptability of parallel machines in widespread CFD applications.

The implicit, parallel CFD code developed in this paper is being integrated into a multidisciplinary design environment. Efforts are currently underway to parallelize the turbulence models and sensitivity-analysis algorithms, to obtain design-sensitivities for a large number of design variables in parallel. Recent Krylov solvers (CGS, BiCGSTAB and QMR) are also being studied to evaluate their competitiveness in parallel environments.

Acknowledgements

The authors wish to thank Dr. L. A. Povinelli for his support and encouragement of this research under the ICOMP Program at NASA Lewis Research Center. Computational resources were provided by the Advanced Computational Concepts Lab (ACCL) at NASA LeRC and the Delta Consortium at Caltech. The grid for the airfoil calculations was provided by Dr. A. C. Taylor of Old Dominion University, Norfolk, VA.

References

1. Saad, Y., and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, 1986, pp. 856–869.

2. Saad, Y., and Schultz, M. H., " Parallel Implementation of Preconditioned Conjugate Gradient Methods," Report No. AD-A161988, Yale University, New Haven, CT., 1986.

3. O'Leary, D. P., "Parallel Implementation of the Block Conjugate Gradient Algorithm," *Parallel Computing*, Vol. 5, July 1987, pp. 127–139.

4. Anderson, E., and Saad, Y., "Preconditioned CG Methods for General Sparse Matrices on Shared Memory Machines," in *Parallel Processing for Scientific Computing*, Proceedings of the Third SIAM Conference, Los Angeles, CA, 1987, pp. 88–92.

5. Radicati di Brozolo, G., and Robert, Y., "Parallel Conjugate Gradient-like Algorithms for solving Sparse Nonsymmetric Linear Systems on a Vector Multiprocessor," *Parallel Computing*, Vol. 11, August 1989, pp. 223–239.

6. Baxter, D., Salz, J., Schultz, M., and Eisenstat, S., "Preconditioned Krylov Solvers and Methods for Runtime Loop Parallelization," Report No. AD-A206388, Yale University, New Haven, CT., 1989.

7. Berryman, H., Saltz, J., Gropp, W., and Mirchandaney, R., "Krylov Methods Preconditioned with Incompletely Factored Matrices on the CM-2," NASA-CR-181961, ICASE Report No. 89-54, 1989.

8. Gropp, W. D., and Keyes, D. E., "Domain Decomposition with Local Mesh Refinement," NASA-CR-187528, ICASE Report No. 91-19, 1991.

9. Shadid, J. N., and Tuminaro, R. S., "Iterative Methods for Nonsymmetric Systems on MIMD Machines," Sandia National Labs Report No. 90-2689C, Albuquerque, NM.

10. Desturler, E., "A Parallel Restructured version of GMRES(m)," Delfts University Report No. 91-85, Netherlands.

11. Braaten, M., "Development of a Parallel CFD Algorithm on a Hypercube Computer," *Intl. J. Num. Meth. Fluids*, Vol. 12, 947–963, July 1990.

12. Van Leer, B., "Flux Vector Splitting for the Euler Equations," *Lecture Notes in Physics*, Vol. 170, 1982, pp. 507–512 (also ICASE Report 82-30, September 1982).

13. Ajmani, K., "Preconditioned Conjugate Gradient Methods for the Navier-Stokes Equations," Ph.D Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1991.

14. Arnoldi, W. E., "The Principle of Minimized Iteration in the Solution of the Matrix Eigenvalue Problem," *Quart. Appl. Math.*. Vol. 9, 1951, pp. 17–29.

15. Meijerink, J. A., and Van der Vorst, H. A., "Guidelines for Usage of Incomplete Decompositions in Solving Sets of Linear Equations as they occur in Practical Problems," *Journal of Computational Physics*, Vol. 44, 1981, pp. 134–155.

16. Yoon, S., and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations," AIAA Paper 87-0600, January 1987.

17. Armaly, B. F., Durst, F., Pereira, J. C. F., and Schonung, B., "Experimental and Theoretical Investigation of Backward Facing Step Flow," *Journal of Fluid Mechanics*, Vol. 127, 1983, pp. 473–496.

18. Ryan, J. S., "Parallel Computation of 3-D Navier-Stokes Flowfields for Supersonic Vehicles," AIAA Paper 93-0064, January 1993. Communication, June 1992.

19. Amdahl, G., Validity of the single-processor approach to achieving large scale computing capabilities, *Proc. AFIPS Conf.* (1967), pp. 483–485.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1994	Technical Memorandum

4. TITLE AND SUBTITLE

Preconditioned Implicit Solvers for the Navier-Stokes Equations on Distributed-Memory Machines

5. FUNDING NUMBERS

WU–505–90–5K

6. AUTHOR(S)

Kumud Ajmani, Meng-Sing Liou, and Rodger W. Dyson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E–8302

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM–106449
ICOMP–93–49
AIAA–94–0408

11. SUPPLEMENTARY NOTES

Prepared for the 32nd Aerospace Sciences Meeting and Exhibit, sponsored by the American Institute of Aeronautics and Astronautics, Reno, Nevada, January 10–13, 1994. Kumud Ajmani, Institute for Computational Mechanics in Propulsion, NASA Lewis Research Center (work funded under NASA Cooperative Agreement NCC3–233); Meng-Sing Liou and Rodger W. Dyson, NASA Lewis Research Center. ICOMP Program Director, Louis A. Povinelli, (216) 433–5818.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 64

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The GMRES method is parallelized, and combined with local preconditioning to construct an implicit parallel solver to obtain steady-state solutions for the Navier-Stokes equations of fluid flow on distributed-memory machines. The new implicit parallel solver is designed to preserve the convergence rate of the equivalent 'serial' solver. A static domain-decomposition is used to partition the computational domain amongst the available processing nodes of the parallel machine. The SPMD (Single-Program Multiple-Data) programming model is combined with message-passing tools to develop the parallel code on a 32-node Intel Hypercube and a 512-node Intel Delta machine. The implicit parallel solver is validated for internal and external flow problems, and is found to compare identically with flow solutions obtained on a Cray Y-MP/8. The computational speed on 32 processing nodes of the i860 machines is comparable to the speed on a single processor of the Cray Y-MP. A peak computational speed of 2300 MFlops/sec has been achieved on 512 nodes of the Intel Delta machine, for a problem size of 1024K equations (256K grid points).

14. SUBJECT TERMS	15. NUMBER OF PAGES
Parallel computing; Krylov solvers; Preconditioners	13
	16. PRICE CODE: A03

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

National Aeronautics and
Space Administration

Lewis Research Center
Cleveland, OH 44135-3191