JOURNAL OF PROPULSION AND POWER
Vol. 17, No. 3, May–June 2001

Development and Validation of a Massively Parallel Flow Solver for Turbomachinery Flows

Jixian Yao,* Antony Jameson,† and Juan J. Alonso‡
Stanford University, Stanford, California 94305
and
Feng Liu§
University of California, Irvine, California 92697

The development and validation of the unsteady, three-dimensional, multiblock, parallel turbomachinery flow solver TFLO is presented. The unsteady Reynolds-averaged Navier–Stokes equations are solved using a cell-centered discretization on arbitrary multiblock meshes. The solution procedure is based on efficient explicit Runge–Kutta methods with several convergence acceleration techniques such as multigrid, implicit residual smoothing, and local time stepping. The solver is parallelized using domain decomposition, a single program multiple data strategy, and the message passing interface standard. Details of the communication scheme and load balancing algorithms are discussed. A general and efficient procedure for parallel interblade row interfacing is developed. The dual-time stepping technique is used to advance unsteady computations in time. The focus is on improving the parallel efficiency and scalability of the flow solver, as well as on its initial validation of steady-state calculations in multiblade row environment. The result of this careful implementation is a solver with demonstrated scalability up to 1024 processors. For validation and verification purposes, results from TFLO are compared with both existing experimental data and computational results from other computational fluid dynamics codes used in aircraft engine industry.

Nomenclature

E	=	absolute total energy per unit mass
$\boldsymbol{F}$	=	mean-flow flux vector
k	=	turbulence kinetic energy per unit mass
$\mathcal{L}_{\text{IRS}}$	=	implicit residual smoothing (IRS) operator
$\boldsymbol{q}$	=	heat flux vector
$\mathcal{R}^*$	=	total residual of the Navier–Stokes equations
$\mathcal{S}$	=	bounding surface of control volume
$\boldsymbol{S}$	=	source-term vector
T_H	=	multigrid forcing term
t	=	time
$\mathcal{V}$	=	control volume
$\boldsymbol{V}$	=	absolute velocity vector
$\boldsymbol{V}_b$	=	mesh velocity vector
$\boldsymbol{W}$	=	conservative dependent-variable vector
α_k	=	$\{\frac{1}{4}, \frac{1}{6}, \frac{3}{8}, \frac{1}{2}, 1\}$ for a five-stage Runge–Kutta scheme
Δt	=	physical time step
Δt^*	=	fictitious time step
$\bar{\lambda}$	=	$3/2\Delta t$ or $11/6\Delta t$ for second-/third-order time accuracy
ρ	=	density
$\bar{\bar{\sigma}}$	=	total stress tensor
$\boldsymbol{\Omega}$	=	angular velocity vector
ω	=	specific dissipation rate

Introduction

COMPUTATIONAL fluid dynamic (CFD) simulations play an essential role in the design of modern gas turbine engines, providing engineering predictions of aerodynamic performance, heat transfer, and flow behavior. Steady-state flow predictions are commonplace for problems ranging in size from the design of an individual compressor or turbine blade, to large or whole sections of a complete component, such as a combustor or a low-pressure turbine. Examples of steady, multistage turbomachinery flow prediction capability include those presented by Adamczyk et al.,[1] Dawes,[2] Denton and Singh,[3] Ni and Bogoian,[4] Ni and Sharma,[5] Rhie et al.[6] and LeJambre et al.[7] Adamczyk,[8,9] Hall,[10] and others have extended these steady flow prediction schemes to model various unsteady flow effects in multistage turbomachinery. Today, multiblade row steady flow simulations have become routine in the design process.

The use of unsteady flow simulations in the design process remains limited to small sections of the engine, such as a single stage of a compressor or turbine. The main reasons for this lack of unsteady numerical results are the large computational requirements necessary to calculate the flow solution and the long integration times necessary to achieve meaningful time averages. As the size of an unsteady flow turbomachinery simulation increases beyond two or three blade rows, the overall required computer memory and solution time rapidly grow to the point where the simulation is no longer feasible with most available computer systems. This rapid growth is due to varying numbers of airfoils in each blade row and to the requirement to maintain a constant and equal circumferential section along the entire axial length of the domain for the sake of circumferential periodicity. Examples of current unsteady, multistage turbomachinery flow prediction procedures include those of Arnone and Pacciani,[11] Dorney et al.,[12] Giles,[13,14] Gundy-Burlet,[15] Jorgenson and Chima,[16] Lewis et al.,[17] Rai,[18] and Rao and Delaney.[19] Other components in the engine have similar requirements that make it just as difficult to utilize unsteady flow simulations as a design tool.

Shared and distributed memory parallel computers have helped to extend greatly the feasible size and to reduce the solution time of large-scale gas-turbine design and analysis problems. Results of turbomachinery parallel simulations have appeared since 1996 including the work of Eulitz et al.,[20] Cizmas and Subramanya,[21] Cizmas and Dorney,[22] and Sgarzi and Toussaint.[23] New advances in computer hardware, communication networks, and simulation software are required, however, to bring large-scale simulations into practical, everyday use. The design and analysis of gas turbine jet engines

Presented as Paper 2000-0882 at the AIAA 38th Aerospace Sciences Meeting and Exhibit, Reno, NV, 10–13 January 2000; received 28 April 2000; revision received 21 November 2000; accepted for publication 22 November 2000.

*Research Associate, Department of Aeronautics and Astronautics. Member AIAA.
†T. V. Jones Professor of Engineering, Department of Aeronautics and Astronautics. Fellow AIAA.
‡Assistant Professor, Department of Aeronautics and Astronautics. Member AIAA.
§Associate Professor, Department of Mechanical and Aerospace Engineering, Member AIAA.

is not the only engineering or scientific arena that has encountered these bottlenecks. As a result, in 1997, the Department of Energy launched the Accelerated Strategic Computing Initiative (ASCI) to promote the development of massively parallel computer systems and simulation software that can properly utilize them.

As part of this initiative, the current effort has been focused on developing a new massively parallel CFD solver. Although the simulation software has been written to allow for the computation of steady and unsteady flows in a variety of applications, the target of the current effort has been gas-turbine turbomachinery flows. The long-term goal is to develop the capability to simulate both steady and unsteady flows through entire compressors or turbines. Before reaching this goal, the fundamental ability of our program to predict some of the most basic turbomachinery flows with demonstrated scalability to tackle the larger problems needs to be addressed thoroughly. The purpose of this paper is to document the numerical, data structure, and parallel computing techniques used in the baseline solver, as well as the results of a series of validation test cases used to verify the scalability and prediction accuracy for different flow regimes.

Overview of TFLO

The unsteady Reynolds-averaged Navier–Stokes equations are solved using a cell-centered discretization on arbitrary multiblock meshes. The solver is parallelized using domain decomposition, a single program multiple data (SPMD) strategy and the message passing interface (MPI) standard.

The solution procedure is based on efficient explicit modified Runge–Kutta methods with several convergence acceleration techniques such as multigrid, residual averaging, and local time stepping. These techniques, multigrid in particular, provide excellent numerical convergence and fast solution turnaround. Two numerical dissipation schemes have been implemented: the Jameson–Schmidt–Turkel (JST) switched scheme[24] and the more refined convective upwind split pressure (CUSP) dissipation model,[25,26] which provides sharper resolution of shock waves and contact discontinuities at a moderate increase in computational cost.

The multiblock strategy facilitates the treatment of arbitrarily complex geometries using a series of structured blocks with point-to-point matching at their interfaces. This point-to-point matching ensures global conservation of the flow variables. The structure of the mesh is specified via a connectivity file that allows for arbitrary orientations of the blocks. Two layers of halo cells are used for interblock information transfer, and an efficient communication scheme is implemented for the halo cell data structures. The load of each processor is balanced on the basis of a combination of the amount of computation and communication that each processor performs. Communication of halo cell values is conducted at every stage of the Runge–Kutta integration and in every level of the multigrid cycle to guarantee fast convergence rates. A general and parallel efficient procedure has been developed to handle the interblade row interface models, while maintaining the flexibility necessary to implement different models. The dual-time stepping technique[27–30] is used for time-accurate simulations that account for the relative motion of rotors and stators as well as other sources of flow unsteadiness.

The resulting preprocessor and flow solver combination have been ported to a variety of today's most advanced parallel computers, including CRAY T3E at the Pittsburgh Supercomputing Center, SGI Origin 2000 systems at both Stanford University and the Los Alamos National Laboratory, and the latest IBM-SP systems at the Lawrence Livermore National Laboratory.

The main goal in the development of TFLO is to ensure that the solver can be scaled to large numbers of processors (in the thousands range) so that problems of far larger size than ever attempted before can be computed efficiently. These problems will involve more data, higher complexity, and a succession of more powerful computing platforms. The issue is not simply bigger and faster, but rather a fundamental shift in the way problems are solved. For this purpose, the solver has to be both robust and efficient and must be able to predict the proposed problems with an acceptable degree of accuracy. However, our intention has also been to develop a code that can be used for small routine calculations that are repeatedly encountered during the process of component design.

Intensive validation has been an intrinsic part of TFLO's development. Some fundamental test cases have been computed[31] to validate the characteristics of the basic solver and turbulence models in TFLO. Several turbomachinery test cases have been carefully selected to validate the predictions of multistage compressor and turbine flows. These test cases include the Virginia Polytechnic Institute (VPI) turbine cascade, the 1.5-stage Aachen turbine, and the Pennsylvania State University Research Compressor (PSRC).

Numerical Methods

Governing Equations

The governing equations solved by TFLO are formulated in a general moving coordinate system. Let $\mathcal{V}(t)$ be a moving control volume with bounding surface $\mathcal{S}(t)$. The functional dependence of $\mathcal{V}$ and $\mathcal{S}$ on t implies that the control volume and its boundary can be time dependent. Let ρ, $\boldsymbol{V}$, E, and $\boldsymbol{V}_b$ be the density, absolute velocity, absolute total energy per unit mass, and mesh velocity, respectively. When a coordinate system that rotates with angular velocity $\boldsymbol{\Omega}$ is used and body forces are neglected, the integral form of the unsteady Navier–Stokes equations can be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathcal{V}(t)} \boldsymbol{W}\,\mathrm{d}\mathcal{V} + \oint_{\mathcal{S}(t)} \boldsymbol{F}\cdot\boldsymbol{n}\,\mathrm{d}S = \oint_{\mathcal{V}(t)} \boldsymbol{S}\,\mathrm{d}\mathcal{V} \tag{1}$$

where

$$\boldsymbol{W} = \begin{bmatrix} \rho \\ \rho\boldsymbol{V} \\ \rho E \end{bmatrix} \tag{2}$$

$$\boldsymbol{F} = \begin{bmatrix} \rho(\boldsymbol{V}-\boldsymbol{V}_b) \\ \rho\boldsymbol{V}(\boldsymbol{V}-\boldsymbol{V}_b) - \bar{\bar{\sigma}} \\ \rho E(\boldsymbol{V}-\boldsymbol{V}_b) - \boldsymbol{V}\cdot\bar{\bar{\sigma}} + \boldsymbol{q} \end{bmatrix} \tag{3}$$

$$\boldsymbol{S} = \begin{bmatrix} 0 \\ \rho\boldsymbol{\Omega}\times\boldsymbol{V} \\ 0 \end{bmatrix} \tag{4}$$

Discretization

A cell-centered finite volume scheme is used to discretize the governing equations. Upwind biasing is achieved with the addition of numerical dissipation. The current version of TFLO uses either a switched scalar dissipation scheme (JST) or the more sophisticated CUSP scheme, coupled with an essentially local extremum diminishing formulation. Details of these techniques and extensive validation studies for both inviscid and viscous flows can be found in the work of Jameson,[25,26] Liu et al.,[32] and Tatsumi et al.[33]

Dual-Time Stepping

When unsteady flows are considered, the Navier–Stokes equations must be integrated forward in time. Jameson's dual-time stepping scheme[27] combines the advantages of both implicit methods and the fast solution techniques that have been developed for steady-state solutions (multigrid, implicit residual smoothing, etc). The stability and robustness of the dual-time stepping scheme has also been verified by other researchers.[28,30,34–39]

Note that, although the overall formulation for the physical time integration is implicit in nature, advancement in time is driven by an explicit inner iteration. It is this characteristic of the dual-time stepping algorithm that allows for the use of well-tested explicit convergence acceleration techniques, such as multigrid and residual averaging, to obtain faster convergence of the pseudotime iterations. The computational advantages of the dual-time stepping scheme are due, in large part, to the use of the multigrid technique; without it, a large number of iterations would be required to converge the flow at each physical time step. Moreover, because the inner iteration is

fully explicit, scalability to large numbers of processors for arbitrarily complex geometries is more easily obtained than with implicit techniques.

Time-Stepping Scheme and Multigrid

The modified m-stage Runge–Kutta scheme, combined with multigrid acceleration, implicit residual smoothing (IRS), and dual-time stepping is formulated as follows:

$$\boldsymbol{W}^{(0)} = \boldsymbol{W}_H^l$$

$$\left(1 + \alpha_k \bar{\lambda} \Delta t^*\right)\boldsymbol{W}^{(k)} = \boldsymbol{W}^{(0)} + \alpha_k \bar{\lambda} \Delta t^* W^{(k-1)} - \alpha_k \Delta t^* \mathcal{L}_{\text{IRS}}\left[\mathcal{R}^*(\boldsymbol{W}) + T_H\right]$$

$$\boldsymbol{W}^{l+1} = \boldsymbol{W}^{(m)} \tag{5}$$

The residual $\mathcal{R}^*$ includes contributions from the convective and dissipative residuals, as well as from the time-derivative discretization. The coefficients α_k are chosen to maximize the stability region along the imaginary axis. Δt^* and Δt are the fictitious and real time steps, respectively. The term $\bar{\lambda}$ originates from the discretization of the time-derivative term in dual-time stepping and is treated implicitly within the Runge–Kutta integration because it is only a diagonal term, and a simple division is required. T_H is a forcing term defined as the difference between the aggregated residuals transferred from the fine mesh and the residual calculated on the coarse mesh with the transferred solution. The accumulated correction at each coarse level is passed to the finer level using trilinear interpolation.[27–29]

Wilcox k–ω Turbulence Model

The basic numerical formulation and solution algorithm for the k–ω equations follow those from Liu and Zheng[40] except that a nonstaggered grid is used in the current paper.

The k–ω equations are solved explicitly at every stage of the Runge–Kutta scheme and are closely coupled with the Navier–Stokes equations. The $k-\omega$ equations are also solved on all multigrid levels to accelerate convergence. Communication of k and ω is performed simultaneously with the conservative flow variables at all block interfaces. Implicit residual smoothing is also applied to the $k-\omega$ equations. The k and ω variables are defined at cell centers in a similar fashion to the main flow quantities. The velocity derivatives, which are used to calculate production and dilation terms at cell centers, are obtained at the vertices of the cell using the values in all of the eight cell centers that surround a given node. The convection terms are discretized with a second-order MUSCL-type upwinding scheme. The update of the k and ω equations within each stage of the Runge–Kutta scheme is modified to treat part of the source terms point implicitly and, therefore, obtain increased numerical stability.

Multiblock Domain Decomposition and Parallelization

To apply the finite volume technique to the solution of flows around complex configurations, we have chosen to implement a multiblock strategy. In a multiblock environment, a series of structured blocks of varying sizes is constructed such that these blocks fill the complete space and conform to the surface of the geometry of interest. This segmentation of the complete domain into smaller blocks avoids topological problems encountered in constructing grids around complex configurations and multiply connected regions. The general strategy in the solution procedure of the multiblock flow solver is to construct a halo of cells that surrounds each block and contains information from cells in the neighboring blocks. This halo of cells, when updated at appropriate times during the numerical solution procedure, allows the flow calculation inside each block to proceed independently of the others.

This approach requires the identification of halo cells adjacent to block boundaries and the construction of lists of halo cells and their internal counterparts in other portions of the global mesh. In TFLO, we have chosen to carry out these setup procedures as part of a preprocessing module. During the preprocessing step, a two-level halo of cells is added around each block. The requirement for this double halo results from the need to calculate all of the necessary fluxes for the internal cells of each block without reference to additional cell locations outside the block in question. In particular, the second differences used for the third-order artificial dissipation terms require the values of the flow variables in the two neighboring cells on all six sides of any given cell.

The conservation laws [Eq. (1)] are applied to all cells in each block. The time integration scheme follows that used in the single-block solver.[24] The solution proceeds by performing the cell flux balance, updating the flow variables, and smoothing the residuals at each stage of the time-stepping scheme and at each level of the multigrid cycle. The main difference in the multiblock integration strategy is the need to loop over all blocks during each stage of the process. The addition of the double halo of cells around each block permits standard single-block subroutines to be used, without modification, for the computation of the flowfield within each individual block. This includes the single-block subroutines for convective and dissipative flux discretization, viscous discretization, multistage time stepping, and multigrid convergence acceleration.

A connectivity file is used to specify the topology of all of the blocks in the domain. The orientation of the indices I, J, K for each block can be arbitrarily specified. Blocks and surfaces are grouped for boundary condition treatment, postprocessing, and visualization.

After partitioning the complete flow domain into a series of connected blocks that can handle arbitrarily complex geometries, the computational workload has to be divided evenly among the processors participating in the calculation. The multiblock decomposition provides a natural approach to parallelization by assigning complete blocks to different processors in the parallel computer. This coarse-grained parallelism is easily handled using the double-halo construct mentioned earlier, an SPMD strategy, and the MPI libraries. Within each processor, fine-grain parallelism can be obtained using compiler-assisted techniques, such as OpenMP or multithreading, which are available on most platforms. This paradigm of parallel implementation used in TFLO maps extremely well to current and future generations of high-performance parallel computers.

Because entire blocks are assigned to any of the N processors participating in the computation, if our mesh has a number of blocks equal to M, we are restricted to using a maximum number of processors $N = M$. This limitation has not typically presented any problems because our preprocessing software has the ability to decompose the original mesh into very large numbers of blocks depending on the size of the complete mesh. In practice, each and every one of the N processors participating in the calculation is assigned several blocks so that each processor virtually runs a copy of a multiblock flow solver.

Multigrid Multiblock Solution Procedure

The solution procedure of the multigrid method with a multiblock scheme relies on the depth within the algorithm at which the block loop is placed and how often the communication takes place. In TFLO, Jameson's multigrid method[27] naturally allows the block loop be placed as deep as into the stage loop of the Runge–Kutta scheme. Each multigrid cycle in the solution procedure consists of a triple-nested loop, the innermost block loop, then the communication layer, and the outermost stage loop of the modified Runge–Kutta integration. It is necessary to communicate at every stage of the Runge–Kutta scheme and at every multigrid level. Reducing the times of communication by only communicating at the finest grid level and/or only at the first Runge–Kutta stage often results in the degradation in the robustness and the convergence rate of the flow solver.[41–43]

Our solution procedure keeps the multiblock scheme virtually equivalent to the single-block scheme. The only difference is in the implementation of the residual averaging technique. In the single-block solution context, tridiagonal systems of equations are set up and solved using flow information from the entire grid. Thus, each residual is replaced by a weighted average of itself and the residuals of its neighbors in the entire grid. In the multiblock strategy, the support for residual smoothing is reduced to the extent of each block. This eliminates the need to solve scalar tridiagonal systems spanning

the blocks, which would incur a severe penalty in communication costs. Depending on the topology of the overall mesh, the setup of tridiagonal systems that follow coordinate lines may lose the physical interpretation it had in the single-block implementation. This change has no effect on the final converged solution and in all applications of the solver has not led to any significant reduction in the rate of convergence.

Communication Scheme and Load Balancing

Achieving high-level scalability in the implementation of a numerical solution procedure relies heavily on three main issues. For large-scale scalability, the computation must contain a fraction of parallelizable work that is close to 100% of the total problem. Clearly, the portion of parallelizable work typically grows with problem size, and therefore, larger problems are typically more amenable to parallelism. If the fraction of parallelizable work in a given problem falls far below the 100% mark, strict limitations on the maximum achievable speedup and efficiency are imposed by Amhdahl's law. In addition, the overhead of the communication scheme must be minimized so that the work performed is useful work. Finally, the total work must be evenly distributed among the participating processors so that no one processor is idle while the others are still computing.

TFLO is based on efficient multigrid, Runge–Kutta explicit algorithms that can be parallelized efficiently. Because of the explicit nature of the solution procedure, a large amount of the computational work (close to 100%) is indeed parallel work. Although high parallel efficiencies can also be obtained with implicit methods, these often come at the expense of increasing the total amount of computational work when compared to the serial algorithm: This is not the case in TFLO.

Once the numerical algorithm is selected, there are two main ways to minimize the communication overhead. These ways include limiting the number of messages that are sent during a single calculation (latency impact) and the total amount of data to be transferred between processors (bandwidth impact). However, in the parallel implementation of TFLO, we have avoided communicating less often than is consistent with the serial scheme so that the convergence rate of the numerical scheme is not adversely affected. This results in a larger number of messages that complicate obtaining high parallel speedups. The communication between processors inside TFLO uses MPI asynchronous constructs, which avoid deadlocks and contention in the network and can take advantage of special purpose communication hardware that the parallel platform may have.

As described earlier, each block in the multiblock mesh is surrounded by two layers of halo cells that are used to decouple the computational procedure within each of the blocks. The requirement for double halo cells (in the finest mesh of the multigrid sequence only) results from the need to preserve the conservative discretization of the governing equations. To fill in the values of these halo cells, donor cells are identified during a preprocessing step. The values of the flow variables in these donor cells are sent to the corresponding halos cell every time that the flow solution values change. This involves a three-step communication procedure for the transfer of information across faces, edges, and corners of the block halos because the block connectivity list only contains information about the blocks that abut with the faces of the current block. Because of this restriction, there would typically be redundant communication for the edge and corner cells. Moreover, the second and third step have to wait for the completion of the previous step, which imposes redundant synchronization as well. The source of redundancy is shown in Fig. 1 with a simple example. The four blocks in Fig. 1 are assumed to reside in four different processors, and they share an internal corner. The donor cell for P3,C is clearly P1,A. However, to get this information across processors using the connectivity list, P1,A has to first be transferred to P2,B before it is sent to P3,C. Alternatively, P1,A could first be moved upward to P4,D and then to P3,C. Actually, both path P2,B $\rightarrow$ P3,C and path P4,D $\rightarrow$ P3,C happen during the communication for the edge cells. Similarly, the redundancy of communication across corners is threefold.

Table 1 Communication matrix of five processors, based on the first load balancing model

p	1	2	3	4	5	r
1	1,488	18,255	14,215	10,712	13,627	56,809
2	18,255	7,660	11,182	15,943	7,913	53,293
3	14,215	10,988	1,154	12,097	15,517	52,817
4	10,712	15,950	12,104	5,122	13,817	52,583
5	13,627	7,899	15,517	13,623	4,204	50,666
s	56,809	53,092	53,018	52,375	50,874	266,168

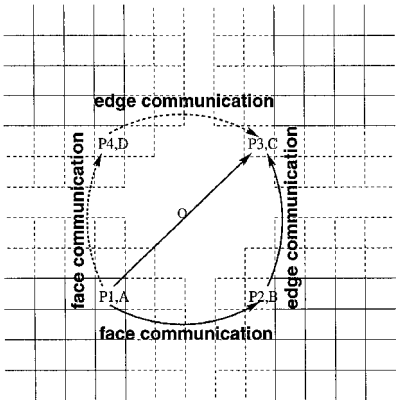


Fig. 1 Communication scheme.

The removal of redundant data transfers leads to a compact communication scheme involving a single step that yields exactly the same final state of the three-step communication procedure. Not only are we reducing the total number of messages and the total amount of data to be transferred, but we are also decreasing the communication overhead by removing the synchronization calls between the three steps of the original procedure. This reduction to a single-step communication procedure is accomplished in the flow solver using a coloring scheme. Although the preprocessor creates the necessary data structures for the three-step communication scheme, as soon as the flow solver is started, every processor encodes its cells with information about the block they come from and their cell indices. The three-step communication procedure is then followed with the result that the halos of all blocks now contain information regarding the original provenance of their donor cells. Once this three-step procedure has been performed once, a new set of data structures is created that contains a single communication step. This single communication step data structure is the one used repeatedly during the process of the calculation. As shown in Fig. 1, the double (triple in three dimensions) paths for the communication of corner cells is replaced by the direct link between P1,A and P3,C.

Table 1 shows a typical communication table for the Aachen axial turbine geometry using a mesh with 2.15×10^6 cells partitioned into 58 blocks and distributed among 5 processors. Each number in Table 1 represents that total number of cells that need to be communicated between the processors indicated by the corresponding row and column numbers. The diagonal elements represent the information that a processor needs to send to itself because it contains more than one block. Direct memory-to-memory copy operations are used in this situation, rather than the traditional MPI messages. In our implementation, this memory-to-memory copy operation is used to hide the latency of the interprocessor communication as well.

The problem of load balancing is that of assigning a set of M arbitrarily connected blocks of varying sizes to N processors of a parallel machine in such a way that all processors use approximately the same amount of CPU time between communication updates. In TFLO, load balancing is performed statically during the preprocessing step. We have experimented with two basic models: The essence of the first model is to assign sequentially the largest unassigned block to the processor with the lowest current load. As a result, the total number of unknowns per processor is as evenly distributed as

Table 2 Communication matrix of five processors, based on the second load balancing model

p	1	2	3	4	5	r
1	9,900	12,476	6,201	7,686	9,442	35,805
2	12,476	17,798	4,090	5,885	14,534	36,985
3	6,201	4,145	15,852	11,697	11,285	33,328
4	7,686	5,885	11,558	17,460	7,278	32,407
5	9,241	14,534	11,285	7,278	43,923	42,338
s	35,604	37,040	33,134	32,546	42,539	180,863

possible. This model has performed fairly well as long as the scalability limits are not pushed too strongly. Table 1 is based on this model yielding a maximum load imbalance of 2%.

The essence of the second load balancing model is to distribute the blocks among participating processors in such a way that the differences between the amount of computation and communication that the processors must perform is a minimum. This algorithm has a tendency to place approximately the same number of cells in each processor (this is not always possible because of varying block sizes), but, in addition, attempts to place blocks that share a physical interface (and, therefore, must communicate with each other) on the same processor, thus decreasing the cost of communication. This model proceeds by taking the largest block yet to be distributed and temporarily assigning it to every processor. For each processor, a temporary increase of the load is updated by the addition of the current block. This update is then rewarded by a decrease in the equivalent size if its neighboring blocks are already assigned to that processor. After all of the temporary assignments, the block permanently settles in the processor with the lowest load. This model requires some performance parameters of the parallel computer and the code itself, such as latency, bandwidth, floating point operations per second and number of floating point operations per cell required to complete a single iteration. Table 2 is the communication matrix that results from this load balancing model for the same problem described earlier. The effectiveness of this model can be seen by comparing the two communication matrices. The matrix in Table 2 is more diagonally dominant than the matrix in Table 1, implying that more data transfer can be done through memory-to-memory copying. The total amount of message passing using MPI is reduced from 93% of the total in the first model to 63.3% using the second model; the maximum load imbalance, however, remains exactly at 2%.

Further reductions in the amount of message passing is achieved by taking advantage of our modified five-stage Runge–Kutta scheme and a first-order artificial dissipation implementation in the coarser meshes of the multigrid sequence. As already mentioned, the second layer of halo cells is only used for the construction of the artificial dissipation flux, and this flux term is only updated on the odd stages, that is, first, third, fifth, of the Runge–Kutta scheme and is not required on coarser meshes. Therefore, we only pass a single layer halo of cells during the even stages of the Runge–Kutta integration and on all stages on the coarser meshes of the multigrid sequence. This decreases significantly the total amount of communication to be performed during the solution procedure.

Parallel and Scalable I/O

In some of the most recent large-scale calculations that we have performed within the ASCI project, we have encountered a bottleneck in parallel speedup that is not usually paid much attention. For meshes with more than 100×10^6 cells and computations using more than 512 processors, the I/O time required can be as large as 30% of an 8-hour run. Although parallel computers have greatly increased their peak performance, the I/O bandwidth has not kept pace. In any case, we have modified TFLO in two different ways to minimize the I/O time. The modifications to TFLO have included the following:

1) Before the flow solver is started, the preprocessor creates one image file per processor containing communication tables, boundary condition, domain decomposition, and load balancing information. The preprocessor also splits the mesh domain to generate a single mesh file per processor. This provides naturally parallel and scalable file input to TFLO; of course, it has the drawback that there will be as many output files as there are processors.

2) During the runtime, each processor dumps and/or reads its own restart image. This also maintains the natural parallelism for restarting I/O, although, compared to single restart files, the user loses the flexibility to change the number of processors for restart.

These simple modifications to the flow solver have been sufficient to speed up the I/O process. It is not, however, the best way of approaching the problem. Several standards for parallel I/O have been created that allow a number of processors to write to and read from a single file using the hardware that may be available on the parallel computer. Within TFLO, the MPI-2 standard is now used for all significant I/O tasks. On the ASCI Blue Pacific (IBM SP2) with general parallel file system, the parallel I/O time (wall clock) was reduced to about 25% of that required by serial I/O when using over 256 processors, thus making the I/O expense much more tolerable.

Interblade Row Interface

For multiple blade-row calculations, we have chosen to use meshes that are attached to the individual blades (either rotors or stators) in the calculation. Because of the relative motion of rotors and stators, by design TFLO is required to handle areas of the overall mesh that share a common surface but slide over each other as the rotors turn. These surfaces are termed sliding mesh interfaces, and a special boundary condition module was developed in TFLO to handle this situation in a completely general multiblock, multiprocessor environment. The sliding mesh interface implementation is responsible for the exchange of all necessary information (flow variables, turbulence variables, etc.) across these surfaces in a way that does not decrease the parallel efficiency of the overall computation. However, in a complex, general, multiblock-parallel environment, there is the added difficulty that the sliding mesh interface will usually be composed of faces from many different blocks with potentially arbitrary orientations and that will most likely reside in different processors across the parallel computer.

After the investigation of these concerns, we have introduced the structure of data pool, which is defined to abstract the concept of the sliding mesh interface in the presence of multiple blocks and processors. A data pool is simply a two-dimensional arrangement of data that has an identity provided by the fact that the totality of the data is ordered and lies on a sliding mesh interface. Data pools can be both physical and fictitious (if they correspond to halo cells) (see Fig. 2). Together with the data pool, a particular processor (usually the one that has a light load) is selected to perform the interface model for one or more interfaces and is referred as master processor. Data pools are only allocated on master processors. Because the master processor has the global and distinct view of the interface, it provides the clarity and flexibility for the implementation of different interface models, including the mixing models for steady flow calculations and interpolation for rotor/stator interactions. The

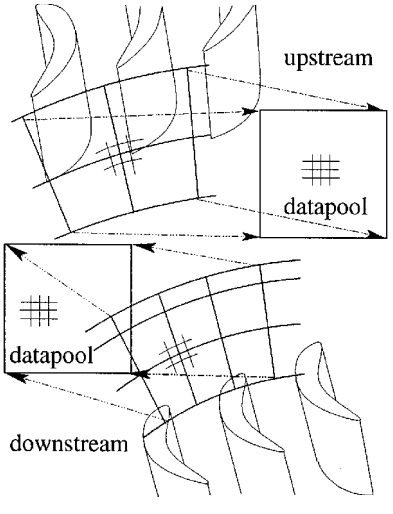


Fig. 2 Data pools of an inter- blade row interface.

implementation of those models is then exactly the same as in serial calculations; hence, it is independent of both the domain decomposition topology at upstream and downstream of an interface and the changing connectivity due to grid rotating in the cases of rotor/stator interactions.

This strategy is implemented in the following manner: The preprocessor provides the address in the data pool of each cell on an interface before the flow solver starts and distributes them onto each processor along with the decomposition information. In the flow solver, the involved processors send the pieces in their workload to report duty to the master processors. Meanwhile, the master processors collect the relevant pieces from the member processors of a certain interface, fill the data pool, call the interface model, and then deliver the results back to the senders. The amount of communication is at the same order as that involved with regular neighboring block communication, and the computation overhead can be similar to the boundary condition treatment.

This interfacing strategy can also be applied to radial interfaces for casing treatment applications and secondary flow systems in turbomachinery. When larger sizes of the interfaces are involved, dedicated master processors can be used to avoid degradation of parallel performance.

Boundary Conditions

Because each processor in the calculation is typically responsible for more than one block and because similar boundary conditions can be imposed on more than one face of each of these blocks, an unstructured approach to the implementation of the boundary conditions is followed. In the preprocessor, one-dimensional lists for each boundary condition type are constructed for each processor. These lists are later used in TFLO such that each processor can impose all boundary conditions of the same type, independently of the number of blocks, in a single DO-loop. This is an efficient way, in the sense of programming and computational cost, for the flow solver to deal with arbitrarily *IJK*-multiblock oriented domain.

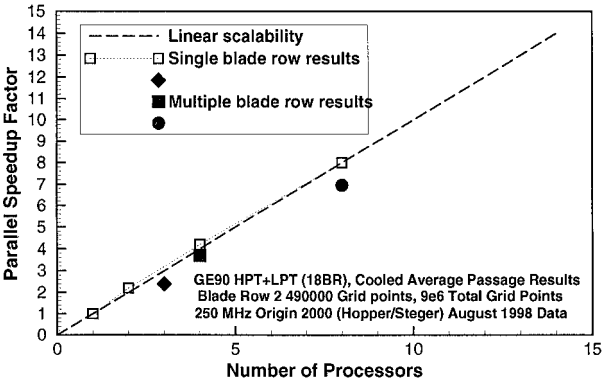
Parallel Performance of TFLO

To demonstrate the parallel performance and scalability of TFLO, we have performed a number of benchmark timings using a variety of problem sizes and numbers of processors. Ideally, the parallel speedup of a program is a linear function of the number of processors used in the problem. This measure of parallel performance is clearly limited by the existence of communication overhead and by the amount of load imbalance in the partition of the problem. Because of issues of load imbalance alone (assuming zero communication cost), parallel speedup may be severely limited. In fact, given a number of blocks of varying sizes in a given mesh, and a prespecified number of processors, it is often impossible to perfectly balance a calculation. It can be shown that maximum theoretical speedup using N processors, S_N, can be expressed as $S_N = (N/2)(1 + 1/Imb)$, where Imb is the ratio of maximum to minimum load, $Imb \geq 1$ in a problem. Because we are able to compute this imbalance from the result of our preprocessor, we can set realistic target to the maximum achievable speedup of TFLO. We present these results together with the scalability curves later in this section.

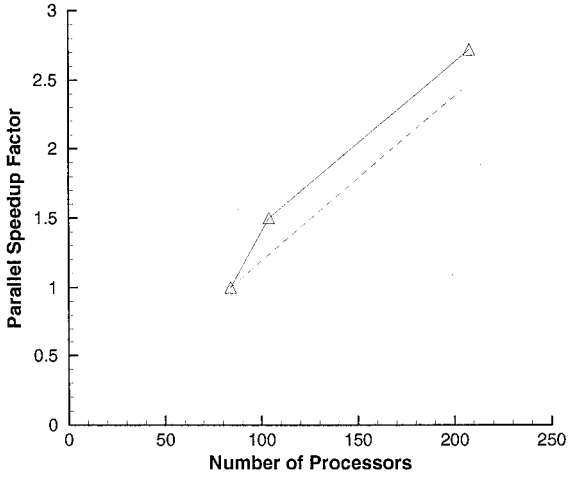
The parallel performance of the widely used turbomachinery code APNASA has been recently reported at the numerical propulsion system simulation review[44] and is presented here in Figs. 3a and 3b for reference. In Fig. 3b, performance data are presented for 104 and 208 processors. Scalability factors are based on the assumption of linear speedups up to 84 processors. The problem size is not reported, and neither is whether the timing include I/O performance.

In the range of 1–32 processors, the parallel speedup of TFLO is shown in Fig. 4. These results were obtained on a 32-processor SGI Origin-2000 with a problem size of 4.16×10^6 grid points in 240 blocks. Relative to the ideal linear trend, TFLO reached 87.5% in efficiency at 32 processors with less than half of the problem size that APNASA used, and still performed roughly the same as APNASA did at only 8 processors for its multiple blade-row results.

On the ASCI Blue-Pacific (IBM SP2), we have pushed the number of processors to 1024, which is the total number of processors



a) Number of processors up to 8 processors



b) Number of processors up to 208 processors

Fig. 3 Parallel performance of APNASA on High Performance Computing and Communications NAS Origin 2000.[44]

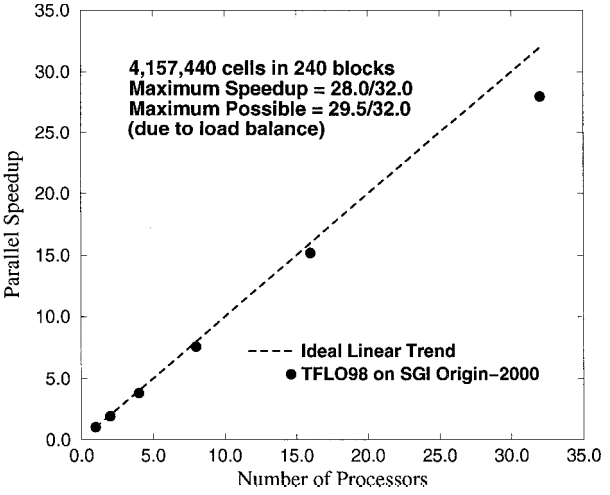


Fig. 4 Parallel performance of TFLO on Origin 2000.

available, solving for unsteady flow in a 9 blade-row turbine with 93.8×10^6 grid points in 110 blade passages. The TFLO speedup factor is shown in Fig. 5, and the actual wall clock time per iteration per grid point is in Fig. 6. The performance in Figs. 5 and 6 is based on 60 processors (for speedup and ideal curves). File I/O time is included in Figs. 5 and 6 as well. The dashed line in Fig. 5 is the ideal trend based on 84 processors for comparison purposes to the performance of APNASA in Fig. 3b. Note that, in Fig. 5, the higher point at 512 processors was obtained by having the whole machine dedicated but only half of the processors were used (1024 processors total). Other runs were performed using the regular queuing system when the machine was heavily loaded. At 1024 processors,

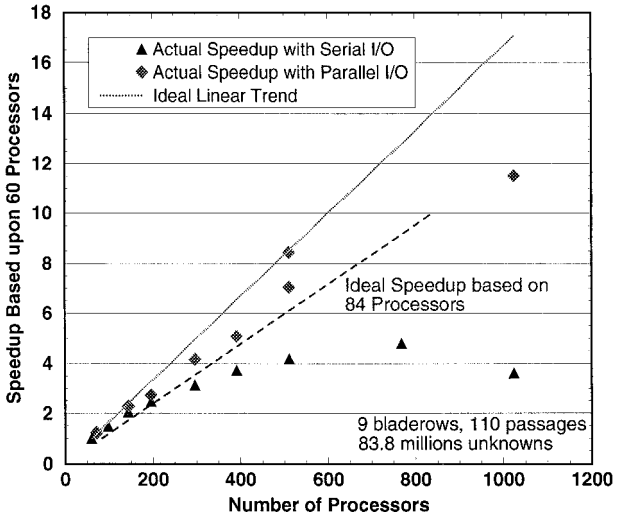


Fig. 5 Speedup factor of TFLO on ASCI Blue-Pacific.

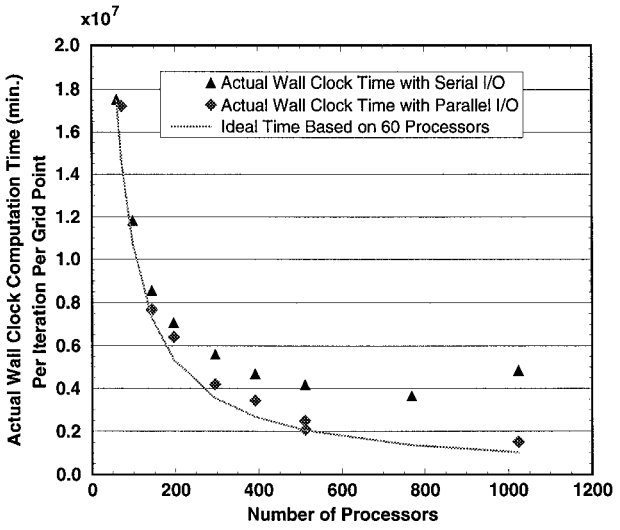


Fig. 6 Wall clock time (per iteration per grid point) of TFLO on ASCI Blue-Pacific.

by considering the load imbalance (the possible speedup is 15.2), TFLO reaches reasonably good efficiency of 76% with I/O time included.

With serial I/O, the speedup slows down when more than 500 processors are used and drops at 1024 processor. This sets the optimal number of processors to about 400. When the file I/O is parallized using the described strategy, considerable amount of speedup is regained, especially for larger number of processors. This pushes the optimal number of processors far beyond the one with serial I/O.

Validation

VPI Cascade

The VPI cascade is a linear transonic turbine nozzle. The viscous flow was calculated using both the Wilcox k–ω and the Baldwin–Lomax models. Two options were used for the exit boundary condition: tangentially uniform back pressure and a nonreflecting boundary condition. Results for the surface pressure distribution produced by TFLO and APNASA-V5[45] (NASA John H. Glenn Research Center software run by General Electric personnel), are compared in Fig. 7 together with the experimental data. All numerical calculations were carried out using exactly the same grid, whose dimensions were $145 \times 17 \times 81$. Figure 7 shows satisfactory pressure comparisons between both codes and the experimental data. In addition, it indicates the improvement in the prediction of the suction surface isentropic Mach number when the nonreflecting exit boundary conditions are used.

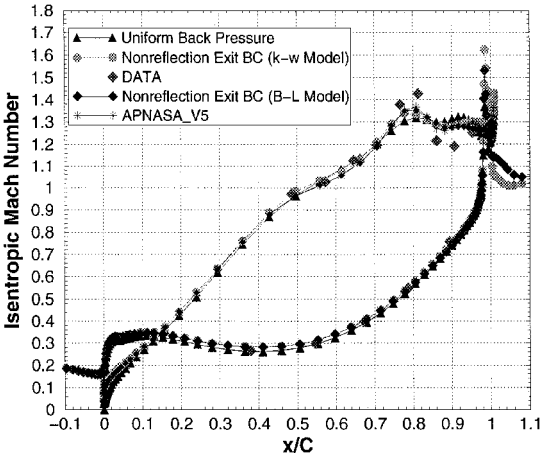
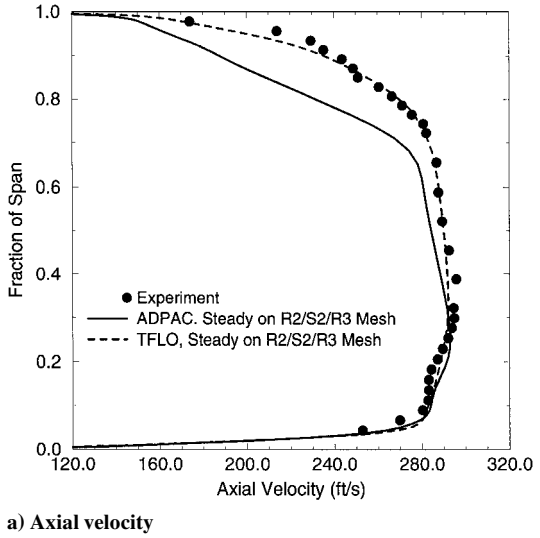
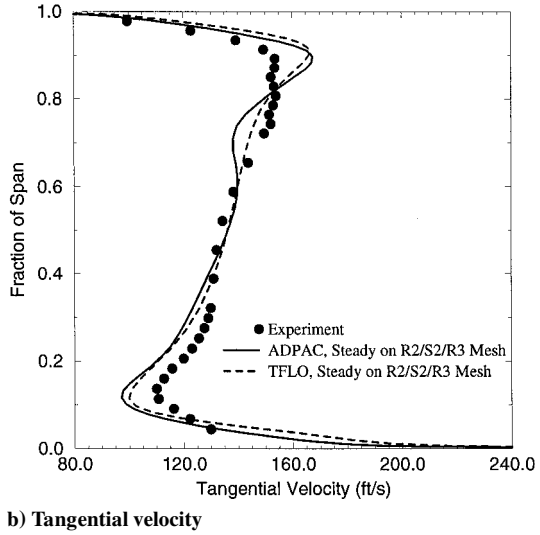


Fig. 7 Isentropic Mach number on blade surfaces of the VPI cascade.



a) Axial velocity



b) Tangential velocity

Fig. 8 Circumferentially averaged parameters along spanwise direction at 5.6% chord after the trailing edge of stator-2.

PSRC

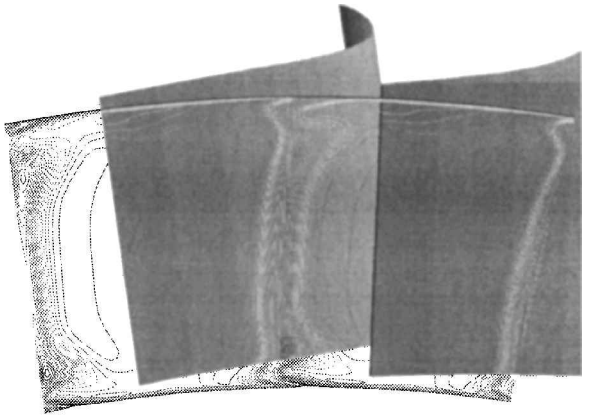
The second test case is the PSRC. The PSRC facility is an axial flow compressor consisting of an inlet guide vane row and three stages of rotor and cantilever-mounted stator with a rotating hub. The results presented here are for a 1.5-stage subset of the complete configuration which consists of rotor2–stator2–rotor3. The blade counts for these three blade rows are 72, 73, and 74. The tip gap of the rotor blade rows and the hub gap of the stator blade row

were modeled in this calculation using TFLO's open-gap boundary condition. The inflow boundary conditions for this test case were obtained from a full rig simulation performed with ADPAC. The experimental data, computational grid, inlet conditions, and ADPAC results were provided by the Allison Engine Company.

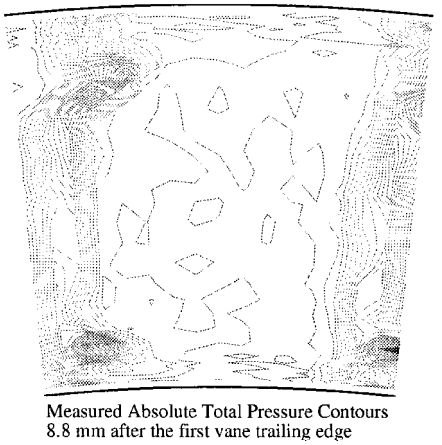
The steady-state flow through the three blade rows was calculated. The two stators used meshes with $125 \times 65 \times 81$ nodes, whereas the rotor used a mesh with $105 \times 65 \times 81$ nodes. All calculations were performed using 72 processors of a CRAY-T3E supercomputer. The flow conditions for this test case are those described in Refs. 46 and 47. Note that all measurements were taken in the full rig environment. Figure 8 shows comparisons of the circumferentially averaged axial and tangential velocity components along the blade spanwise direction at a distance of 5.6% of the chord behind the trailing edge of stator-2. TFLO predicts similar results to ADPAC, and both sets of results agree well with the experimental data. The axial velocity profile predicted by TFLO is closer to the experimental data.

Aachen 1.5-Stage Turbine

The third test case is the 1.5-stage Aachen Turbine. The geometry and experimental data were provided by the European Research Community on Flow, Turbulence, and Combustion. This facility is an axial flow turbine consisting of three blade rows: the first vane, the blade, and the second vane. The geometry of the second vane is exactly the same as that of the first vane. The blade counts for this case are 36, 41, and 36, respectively. The steady-state flow through the turbine was calculated with the following mesh sizes: Inlet Guide Vane, $137 \times 65 \times 81$; blade, $113 \times 65 \times 81$ for the blade passage and $89 \times 17 \times 17$ for the tip gap; and stator, $153 \times 65 \times 81$. One blade passage per blade row is included in the mesh. Calculations were run on an SGI Origin 2000 computer.



a) Predicted contours of total pressure (TFLO)



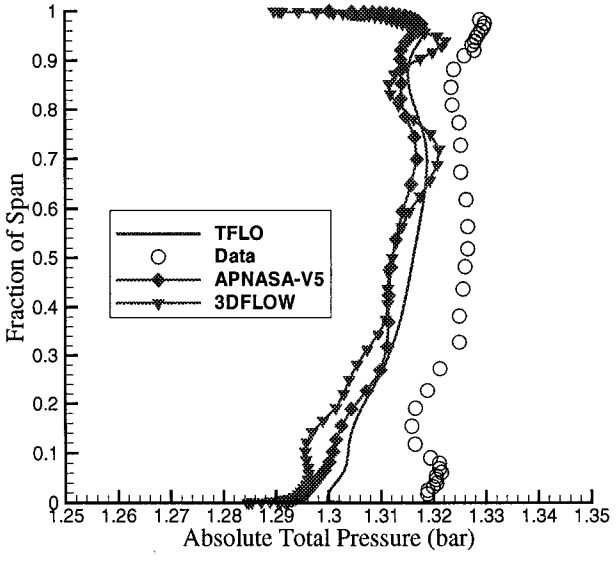
b) Measured contours of total pressure

Fig. 9 Comparison of total pressure contours at the measurement plane, 8.8 mm behind trailing edge of the first vane.

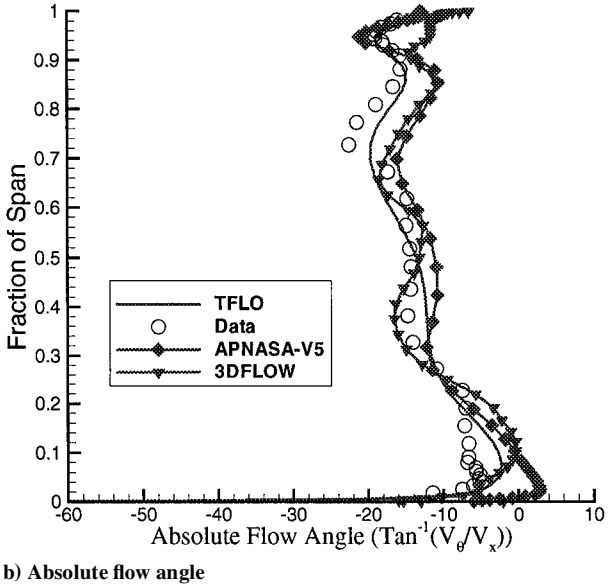
Comparisons are presented between the results of TFLO, United Technologies' solver 3DFLOW,[4,5] NASA/General Electric's solver APNASA-V5, and the experimental data for the low mass flow rate condition. The calculated mass flow rate was 7.1 kg/s whereas the measured mass flow rate was in the range 6.6–6.9 kg/s. The computational grids were generated by personnel of the United Technologies Research Center. These grids have a separate block for the tip gap region of the rotor. The results from TFLO and 3DFLOW were calculated using the same grids. The grid used by APNASA-V5 was similar in size, although the tip gap region was not resolved; the open-gap boundary condition was applied instead.

Figure 9 shows a comparison between the results of TFLO and the experimental data for the total pressure at a measurement plane located behind the first vane. The strong secondary flow near the endwalls and in the wake behind the trailing edge cause major losses in total pressure. TFLO appears to capture the main features of the total pressure map.

The circumferentially averaged total pressure and absolute flow angle behind the trailing edge of the blade are compared in Fig. 10. Predictions of the different solvers seem to agree well with each other and the trends in the data, but differ somewhat in the measured

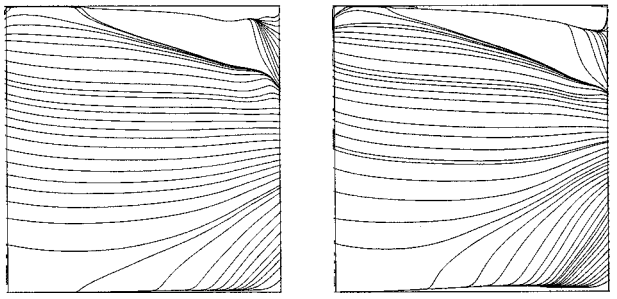


a) Absolute total pressure



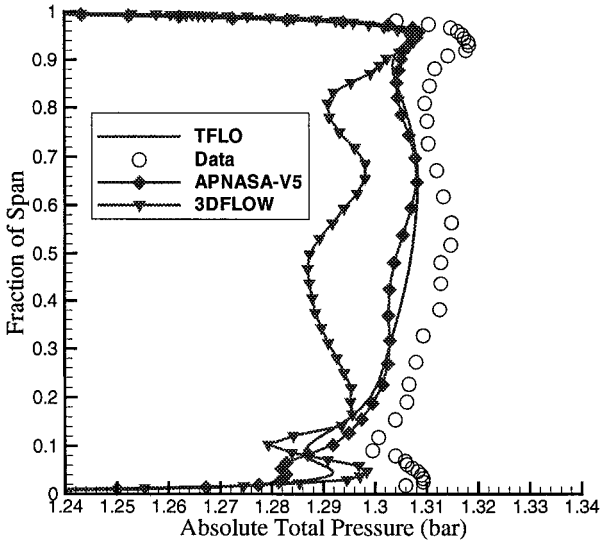
b) Absolute flow angle

Fig. 10 Circumferentially averaged total pressure, total temperature, and flow angle at a station 8.8 mm downstream of the blade trailing edge.

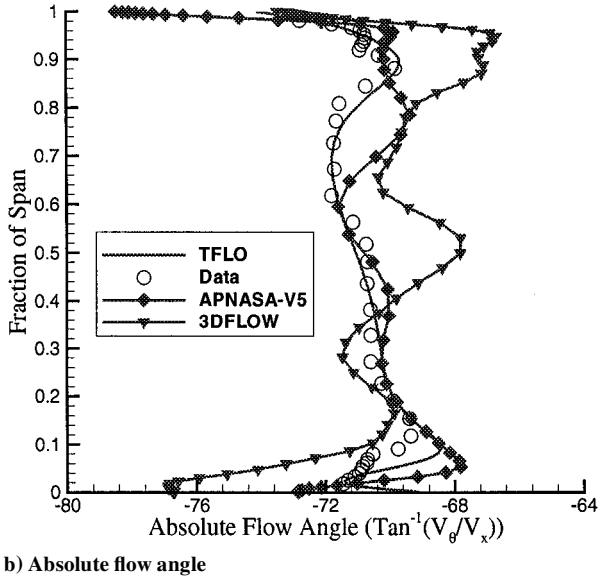


a) TFLO b) 3DFLOW

Fig. 11 Predicted limiting streamlines on suction surface of the blade.



a) Absolute total pressure



b) Absolute flow angle

Fig. 12 Circumferentially averaged total pressure and absolute flow angle at a station 8.8 mm downstream of the trailing edge of the second vane.

level, especially for the total pressure. The limiting streamlines on the suction surface of the blade predicted by TFLO and 3DFLOW are shown in Fig. 11. These flow patterns are the signature of the hub secondary flow and blade tip vortex. The dividing streamline patterns in the area of the blade tip look quite similar for both solvers.

The circumferentially averaged total pressure and absolute flow angle behind the trailing edge of the second vane are compared in Fig. 12. TFLO produces similar results to APNASA-V5 and 3DFLOW though the differences tend to be larger for the second vane. For the prediction of the flow angle at the exit of this turbine, TFLO distinctly shows the closest agreement with the measurement both in the spanwise trend and absolute level.

Concluding Remarks

The initial development and validation of the general turbomachinery flow solver TFLO has been completed. The result is a versatile program that can handle a range of general, multiple blade-row geometries in both steady and unsteady flows. Having achieved high parallel performance, TFLO is able to obtain results that are very similar to those from other state-of-the-art simulation codes used in both government laboratories and industry. All of the basic elements of TFLO (domain decomposition, parallel implementation, communication scheme, load balancing models, and procedure for interblade row interface, as well as the basic numerical schemes and algorithms) have been described, tested, and demonstrated.

Emphasis has been placed on the ability of TFLO to scale to large numbers of processors so that the more interesting problems of complete compressor/turbine unsteady calculations can be tackled. It is in this area where we expect that TFLO can make a contribution to the field, because, using computational resources from the ASCI project, we will be in a position to tackle more complex calculations than have been previously attempted. However, TFLO can also be used to perform smaller calculations, which require moderate resources.

Acknowledgments

The authors would like to thank the U.S. Department of Energy for its generous support under the Accelerated Strategic Computing Initiative Program. We are also deeply indebted to Roger L. Davis from the United Technologies Research Center, who, during his stay at Stanford University has made invaluable contributions to the completion of this work. The authors would also like to thank Lyle D. Dailey from General Electric Aircraft Engines Company, and Edward J. Hall and Kurt Weber from the Rolls–Royce Allison Engine Company for providing the Pennsylvania State University Research Compressor case. We thank the Pittsburgh Super-Computing Center and Lawrence Livermore National Laboratory for the use of their parallel platforms. The European Research Community on Flow, Turbulence, and Combustion is gratefully acknowledged for providing the experimental data for the Aachen turbine.

References

[1] Adamczyk, J. J., Celestina, M. L., Beach, T. A., and Barnett, M., "Simulation of 3-Dimensional Viscous Flow Within a Multistage Turbine," *Journal of Turbomachinery*, Vol. 112, No. 3, 1990, pp. 370–376.

[2] Dawes, W. N., "Simulation of Unsteady Blade Row Interaction with CFD: Applications," Rept. VKI-LS-1998-02, von Karman Inst. for Fluid Mechanics, 1998.

[3] Denton, J. D., and Singh, U. K., "Time Marching Methods for Turbomachinery Flow Calculations," Rept. VKI-LS-1979-07, von Karman Inst. for Fluid Mechanics, 1979.

[4] Ni, R. H., and Bogoian, J. C., "Predictions of 3-D Multi-Stage Turbine Flow Fields Using a Multiple-Grid Euler Solver," AIAA Paper 89-0203, Jan. 1989.

[5] Ni, R. H., and Sharma, O. P., "Using 3-D Euler Flow Simulations to Assess Effects of Periodic Unsteady Flow Through Turbines," AIAA Paper 90-2357, July 1990.

[6] Rhie, C. M., Gleixner, A. J., Spear, D. A., Fischberg, C. J., and Zacharias, R. M., "Development and Application of a Multistage Navier–Stokes Solver: Part I—Multistage Modeling Using Body Forces and Deterministic Stresses," *Journal of Turbomachinery*, Vol. 120, No. 2, 1997, pp. 205–214.

[7] LeJambre, C. R., Zacharias, R. M., Biederman, B. P., Gleixner, A. J., and Yetka, C. J., "Development and Application of a Multistage Navier–Stokes Flow Solver: Part II—Application to a High-Pressure Compressor Design," *Journal of Turbomachinery*, Vol. 120, No. 2, 1998, p. 215.

[8] Adamczyk, J. J., "Model Equation for Simulating Flows in Multistage Turbomachines," American Society of Mechanical Engineers, ASME Paper 85-GT-226, 1985.

[9] Adamczyk, J. J., "A Model for Closing the Inviscid Form of the Average Passage Equation System," *Journal of Turbomachinery*, Vol. 108, No. 2, 1986, p. 180.

[10]Hall, E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields," American Society of Mechanical Engineers, ASME Paper 97-GT-344, 1997.

[11]Arnone, A., and Pacciani, R., "Rotor-Stator Interaction Analysis Using the Navier–Stokes Equations and a Multigrid Method," American Society of Mechanical Engineers, ASME Paper 95-GT-117, 1995.

[12]Dorney, D. J., Davis, R. L., Edwards, D. E., and Madavan, N. K., "Unsteady Analysis of Hot Streak Migration in a Turbine Stage," AIAA Paper 90-2354, July 1990.

[13]Giles, M. B., "Stator/Rotor Interaction in a Transonic Turbine," AIAA Paper 88-3093, July 1988.

[14]Giles, M. B., "UNSFLO: A Numerical Method for Unsteady Inviscid Flow in Turbomachinery," MIT Gas Turbine Lab. Rept. 195, Massachusetts Inst. of Technology, Cambridge, MA, 1988.

[15]Gundy-Burlet, K. L., "Computations of Unsteady Multistage Compressor Flows in a Workstation Environment," American Society of Mechanical Engineers, ASME Paper 91-GT-336, 1991.

[16]Jorgeson, P. C. E., and Chima, R., "An Explicit Runge–Kutta Method for Unsteady Rotor/Stator Interaction," AIAA Paper 88-0049, Jan. 1988.

[17]Lewis, J. P., Delaney, R. A., and Hall, E. J., "Numerical Prediction of Turbine Vane-Blade Aerodynamic Interaction," *Journal of Turbomachinery*, Vol. 111, No. 4, 1989, pp. 387–393.

[18]Rai, M. M., "Navier–Stokes Simulations of Rotor/Stator Interactions Using Patched and Overlaid Grids," *Journal of Propulsion and Power*, Vol. 3, No. 5, 1987, pp. 387–396.

[19]Rao, K., and Delaney, R., "Investigation of Unsteady Flow Through Transonic Turbine Stage, Part I: Analysis," AIAA Paper 90-2408, July 1990.

[20]Eulitz, F., Engel, K., and Gebing, H., "Numerical Investigation of the Clocking Effects in a Multistage Turbine," American Society of Mechanical Engineers, ASME Paper 96-GT-26, 1996.

[21]Cizmas, P., and Subramanya, R., "Parallel Computation of Rotor-Stator Interaction," *Proceedings of the Eighth International Symposium on Unsteady Aerodynamics and Aeroelasticity of Turbomachines*, edited by T. H. Fransson, Kluwer Academic Pub., Dordrecht/Boston/London, 1997, pp. 633–643.

[22]Cizmas, P., and Dorney, D. J., "Parallel Computation of Turbine Blade Clocking," AIAA Paper 98-3598, July 1998.

[23]Sgarzi, O., and Toussaint, C., "A Parallel Computation of 3D Unsteady Flows in a Full Stage of Transonic Turbine," *Proceedings of the Fourteenth International Symposium on Air Breathing Engines*, ISABE Paper 99-7105, Sept. 1999.

[24]Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods with Runge–Kutta Time Stepping Schemes," AIAA Paper 81-1259, Jan. 1981.

[25]Jameson, A., "Analysis and Design of Numerical Schemes for Gas Dynamics 1, Artificial Diffusion, Upwind Biasing, Limiters and their Effect on Multigrid Convergence," *International Journal of Computational Fluid Dynamics*, Vol. 4, 1995, pp. 171–218.

[26]Jameson, A., "Analysis and Design of Numerical Schemes for Gas Dynamics 2, Artificial Diffusion and Discrete Shock Structure," *International Journal of Computational Fluid Dynamics*, Vol. 5, 1995, pp. 1–38.

[27]Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," AIAA Paper 91-1596, June 1991.

[28]Alonso, J. J., Martinelli, L., and Jameson, A., "Multigrid Unsteady Navier–Stokes Calculations with Aeroelastic Applications," AIAA Pape 95-0048, Jan. 1995.

[29]Belov, A., Martinelli, L., and Jameson, A., "Three-Dimensional Computations of Time-Dependent Incompressible Flows with an Implicit Multigrid-Driven Algorithm on Parallel Computers," *Proceedings of the 15th International Conference on Numerical Methods in Fluid Dynamics*, edited by P. Kutler, J. Flores, and J.-J. Chattot, Springer, Berlin/New York, June 1996, pp. 430–437.

[30]Alonso, J. J., "Parallel Computation of Unsteady and Aeroelastic Flows using an Implicit Multigrid-Driven Algorithm," Ph.D. Dissertation, Dept. of Mechanical and Aerospace Engineering, Princeton Univ., Princeton, NJ, June 1997.

[31]Yao, J., Jameson, A., Alonso, J. J., and Liu, F., "Development and Validation of a Massively Parallel Flow Solver for Turbomachinery Flows," AIAA Paper 2000-0882, Jan. 2000.

[32]Liu, F., Jameson, A., and Jennions, I., "Computation of Turbomachinery Flow by a Convective-Upwind-Split-Pressure (CUSP) Scheme," AIAA Paper 98-0969, Jan. 1998.

[33]Tatsumi, S., Martinelli, L., and Jameson, A., "A New High Resolution Scheme for Compressible Viscous Flows with Shocks," AIAA 33rd Aerospace Sciences Meeting, AIAA Paper 95-0466, Jan. 1995.

[34]Liu, F., and Ji, S., "Unsteady Flow Calculations with a Multigrid Navier–Stokes Method," *AIAA Journal*, Vol. 34, No. 10, 1996, pp. 2047–2053.

[35]Belov, A., Martinelli, L., and Jameson, A., "A New Implicit Algorithm with Multigrid for Unsteady Incompressible Flow Calculations," AIAA Paper 95-0049, Jan. 1995.

[36]Melson, N. D., Sanetrik, M. D., and Atkins, H. L., "Time-Accurate Navier–Stokes Calculations with Multigrid Acceleration," *Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods*, edited by N. D. Melson, T. A. Manteuffel, and S. F. McCormick, July 1993, pp. 423–437; also in NASA CP 3224.

[37]Arad, E., and Martinelli, L., "Large Eddy Simulation of Compressible Flow Using a Parallel, Multigrid Driven Algorithm," AIAA Paper 96-2065, June 1996.

[38]Davis, R. L., Shang, T., Buteau, J., and Ni, R. H., "Prediction of 3-D Unsteady Flows in Multi-Stage Turbomachinery Using an Implicit Dual Time-Step Approach," AIAA Paper 96-2565, July 1996.

[39]Arnone, A., Liou, M. S., and Povinelli, L. A., "Multigrid Time-Accurate Integrations of Navier–Stokes Equations," AIAA Paper 93-3361, July 1993.

[40]Liu, F., and Zheng, X., "A Strongly Coupled Time-Marching Method for Solving the Navier–Stokes and k–ω Turbulence Model Equations with Multigrid," *Journal of Computational Physics*, Vol. 128, 1996, pp. 289–300.

[41]Alonso, J. J., Mitty, T. J., Martinelli, L., and Jameson, A., "A Two-Dimensional Multigrid Navier–Stokes Solver for Multiprocessor Architectures," *Parallel Computational Fluid Dynamics: New Algorithms and Applications*, edited by N. Satofuka, J. Periaux, and P. Ecer, Elsevier Science B. V., 1994, pp. 435–442.

[42]Jameson, A., and Alonso, J., "Automatic Aerodynamic Optimization on Distributed Memory Architectures," AIAA Paper 96-0409, Jan. 1996.

[43]Reuther, J., Alonso, J. J., Vassberg, J. C., Jameson, A., and Martinelli, L., "An Efficient Multiblock Method for Aerodynamic Analysis and Design on Distributed Memory Systems," AIAA Paper 97-1893, June 1997.

[44]*Proceedings 2000 Numerical Propulsion System Simulation Review*, NASA John H. Glenn Research Center at Lewis Field, OH, Computing and Interdisciplinary Systems Office, Oct. 2000, pp. 79, 84.

[45]Kirtley, K. R., Turner, M. G., and Saeidi, S., "An Average Passage Closure Model for General Meshes," American Society of Mechanical Engineers, ASME Paper 99-GT-077, 1999.

[46]Hall, E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields—Part 1: Analysis of Rotor/Stator/Rotor Aerodynamic Interaction," Allison Engine Co., Indianapolis, IN, ASME 97-GT-344, June 1997.

[47]Hall, E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields—Part 2: Modeling Deterministic Stresses," Allison Engine Co., Indianapolis, IN, ASME 97-GT-345, June 1997.