NASA Technical Memorandum 105932

Turbomachinery CFD on Parallel Computers

Richard A. Blech and Edward J. Milner
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio

and

Angela Quealy and Scott E. Townsend
Sverdrup Technology, Inc.
Lewis Research Center Group
Brook Park, Ohio

Prepared for the
Symposium on High-Performance Computing for Flight Vehicles
Washington, D.C., December 7–9, 1992

NASA

Turbomachinery CFD on Parallel Computers

Richard A. Blech, Edward J. Milner
NASA Lewis Research Center
Cleveland, Ohio

Angela Quealy and Scott E. Townsend
Sverdrup Technology, Inc.
Brookpark, Ohio

ABSTRACT

The role of multistage turbomachinery simulation in the development of propulsion system models is discussed. Particularly, the need for simulations with higher fidelity and faster turnaround time is highlighted. It is shown how such fast simulations can be used in engineering-oriented environments. The use of parallel processing to achieve the required turnaround times is discussed. Current work by several researchers in this area is summarized. Parallel turbomachinery CFD research at the NASA Lewis Research Center is then highlighted. These efforts are focused on implementing the average-passage turbomachinery model on MIMD, distributed memory parallel computers. Performance results are given for inviscid, single blade row and viscous, multistage applications on several parallel computers, including networked workstations.

INTRODUCTION

The analysis of flows through turbomachinery is one of the most difficult and challenging aspects of advanced propulsion system design. Yet the potential benefit of increasingly detailed codes which can simulate these flows has driven considerable research in this area. Future propulsion system requirements such as reduced noise and emissions, leading to higher pressure ratios and temperatures, can have a large impact on the design of the turbomachinery components. Computer codes which can predict the complex flow patterns and performance of turbomachinery components have proven to be valuable design tools. The problem becomes even more complicated, however, when one considers the turbomachinery components as part of an overall propulsion system. Compressor stall and surge are phenomena which are very much dependent on what goes on both up and downstream of the compressor. The interactions between the propulsion system components must be taken into account.

Unfortunately, limitations in computer speeds have restricted the fidelity of propulsion system analyses. Most system level analyses are performed using lumped-parameter models. Results from these models may prompt further analysis using a more detailed (e.g two or three-dimensional) approach. The results from the detailed analyses are then manually fed back to the lumped-parameter model. This process can be time consuming, especially if the two or three-dimensional analysis requires tens to hundreds of hours of computing time. In addition, the lack of any computer-automated path to feed information back and forth between the different levels of fidelity further limits productivity.

One may conjecture that a full, three-dimensional nose-to-tail simulation of the propulsion system will provide the ultimate answer. This approach, however, does not always make sense depending on the ultimate goal of the analysis to be performed. High resolution may be required to analyze a propulsion system component individually. However, modelling of high-frequency phenomena in one component, from a system point of view, may be wasteful if no other component can respond to those frequencies. A multilevel-of-fidelity approach makes sense in propulsion system simulation.

A multilevel approach has been proposed at the NASA Lewis Research Center as part of the Numerical Propulsion System Simulation (NPSS) project (ref. 1). The technique is called "zooming", where higher resolution can be obtained by "zooming in" from simple lumped-parameter simulations to two or three-dimensional codes. If appropriate, multiple two or three-dimensional codes can be combined. These codes are not necessarily limited to a single discipline, but could involve several disciplines (e.g. fluids and structures). It is at these more complicated levels where parallel processing can play a major role in providing the required computing resources.

Parallel processing will not only allow the complex, multidisciplinary applications to be run in reasonable computing time, but will also allow interactive single discipline codes to be developed. The idea of interactive use of CFD codes is being investigated as part of the Integrated CFD and Experiments (ICE) program (ref. 2) at the NASA Lewis Research Center. The initial focus of the ICE program will be on turbomachinery flow physics research. One of its elements is the exploration of using parallel computing to provide interactive use of CFD codes for experimental setup. Here the term interactive means sufficient turnaround time to impact the outcome of an experiment. Repeatability of experimental conditions for turbomachinery is difficult, making it is desirable to have the outcome of a CFD analysis before the conditions change. Ideally, this would be minutes, but could be as much as a few hours. The ever increasing performance of parallel computer hardware is expected to make the former goal a reality.

USE OF PARALLEL TURBOMACHINERY CODES IN ENGINEERING-ORIENTED ENVIRONMENTS

Two different situations where parallel turbomachinery codes could be useful have been introduced. The first is the large, multidisciplinary application where parallel computing is necessary to perform an analysis in reasonable time. The second is the interactive application, where results from the code are desired within an appropriate time frame to impact a decision, such as experimental setup. Examples of each of these cases are given here.

For the first example, consider the lumped parameter turbofan simulation as shown in Figure 1. The fan, compressor and the two turbines are all modelled by performance maps. The other components are modelled through application of the unsteady equations of conservation of mass, momentum and energy for an inviscid fluid within a control volume. A detailed description of a typical lumped parameter engine simulation is given in reference 3. Suppose that a pressure disturbance leads to the fan component map moving into the stall region. Typically, the stall condition can be handled in one of two ways. The first is to alert the user that a stall condition has arisen and to halt the simulation. The

other possibility is to model the in-stall behavior of the fan, using an approach similar to that of reference 4.

In either case, more detailed information is required, and could be achieved using zooming, as illustrated in Figure 2. A detailed, 3-dimensional analysis would be used to investigate factors leading up to stall, such as tip clearance effects. The analysis would be similar to that of reference 5. A three-dimensional, parallel turbomachinery code would be intialized by the near-stall conditions from the lumped-parameter model. The initial fan inlet velocity, and exit pressure would be used. The three-dimensional code would then be run to examine tip clearance effects. Ideally, the analysis would be multidisciplinary, where structural and thermal effects are taken into account, as these can have a significant impact on the tip clearance gap. The analysis could lead to the evaluation of various casing treatments (ref. 6) in an attempt to improve stall margin. If a particular casing treatment looks promising, a modified fan map with the improved characteristics would be returned to the lumped-parameter model. The system-level model would then be rerun to determine if improved performance is attained.

A fast turbomachinery simulation is also desirable for use in setting up turbomachinery flow physics experiments. One of the considerations in setting up experimental probes is that they be located in a region where accurate measurements can be made with minimum interference. An example of this is the placement of static pressure probes within experimental turbomachinery rigs. The probe at the inlet of the rig must be placed such that it is well upstream of shocks generated by the first rotor. The use of a turbomachinery code with rapid turnaround time would allow this determination to be made at the operating conditions for the current run. The code would also be useful for determining where measurements should be made to capture complex flow phenomena, such as secondary flow vortices. Conceivably, the code could even be used to automatically position the instrumentation (e.g. laser anemometry). As facility conditions are changed, the code could be re-run to determine the new measurement locations of interest. The constraints of facility operation can make rapid turnaround of this code critical to efficient acquisition of data.

It is clear that the above capabilities are desirable. In the case of propulsion system simulation, zooming can reduce propulsion system computational requirements by a large amount. Generation of the database for the simplified models will still require repeated application of the complex component models, such as three-dimensional turbomachinery codes. The effectiveness of the ICE environment will be enhanced by turbomachinery codes with rapid turnaround time. However, current approaches to three-dimensional, multistage turbomachinery simulation demand tens to hundreds of hours of state-of-the art supercomputer time. This has prompted the investigation of using parallel processing to reduce these times. The potential benefits are two-fold. The extremely large turbomachinery applications can benefit through the use of parallel computing on supercomputers such as the Cray Y-MP and C90, which are composed of multiple processors. Parallel computing can also provide cost-effective alternatives to large supercomputers, through the use of dedicated, low-cost parallel processors or networked workstations. The use of parallel processing comes at a price, however. Some of the issues involved in the use of parallel computers are addressed in the next section.

PARALLEL COMPUTING ISSUES

Given the need for parallel turbomachinery codes, one is faced with the formidable task of selecting parallel computing hardware and software. Unfortunately, these two areas are not mutually exclusive. The selection of a parallel programming environment often determines the choice of hardware. The same is usually true if one approaches the problem by first selecting the parallel computing hardware. The user is then limited in the programming tools available.

Parallel computers can generally be classified by two characteristics - control structure and memory interconnection. The two predominant control methodologies are Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD). Likewise, there are two predominant memory interconnection strategies - shared memory and distributed memory.

The distributed memory architecture offers more scalability than the shared memory architecture. However, programming of distributed memory machines tends to be more complex, since the programmer is required to manage the communication of information between processors. Ideally, one would like to program a distributed memory machine in much the same fashion as shared memory machine. Current research (ref.7) is addressing this issue. For the present discussion, however, we will focus on the distributed memory architecture, and its two possible control structures.

The SIMD computer requires that each processor execute the same instruction at the same time. Despite this seemingly restrictive architecture, a large number of applications can be implemented effectively on SIMD computers. These are usually limited to single-discipline applications. The control structure of the SIMD machines prompted the development of software tools which could easily exploit its synchronized, lock-step characteristics. This led to the development of the data parallel method of programming, which allows the same operation to be performed concurrently on multiple data elements. The first example of this was CM Fortran, implemented on the Connection Machine (ref. 8). Ideally, the programmer is shielded from details of low-level programming, load-balancing, and data communication. However, immature versions of data parallel compilers have forced programmers to address these issues (refs. 9,10,11) to achieve reasonable levels of parallel efficiency. Despite these difficulties, a strong case can be made that the programming environment, and not the architecture, has attracted many users to the SIMD machines.

The MIMD architecture does not posses the rigidity of SIMD, allowing multiple threads of control to exist concurently. It's most obvious advantage is that it can truly handle different applications concurrently, such as a CFD and a structures calculation. In addition, the SIMD model of computation can be implemented on MIMD machines. Another practical approach to parallel computing, networked workstations, falls under the MIMD classification. Thus a broader class of parallel computers is encompassed by the MIMD architecture. Programming environments on MIMD machines, due to their more complex nature, are low-level and not as advanced as those on SIMD machines. The programmer is typically responsible for all of the complex issues in parallel computing, such as managing data communication, balancing loads, and mapping the application onto

the processors.

Higher-level programming tools for MIMD computers are required before these machines can be effectively utilized. A significant body of research is currently being conducted in this area. For example, data parallel programming environments on MIMD machines can be considered. FORTRAN D (ref. 12) is a data parallel programming system, which can, in fact, provide portability between SIMD and MIMD machines. However, a fully functional FORTRAN D system does not yet exist. This work is still in the research stage. The use of data parallel compilers on MIMD machines still does not address how one expresses functional parallelism (e.g. CFD and structures) on MIMD machines. Object-oriented approaches (refs. 13,14,15) offer the potential for expressing functional parallelism, but again, are in the developmental stages. There are other approaches to programming MIMD, distributed memory computers, but all must be evaluated from the standpoint of overhead. No clear standard in this area has yet emerged.

A thread of commonality does exist between MIMD computers, which is the communication between processors via messages. It is conceivable to develop a set of message-passing primitives that is portable between various MIMD machines. This is the focus of the Application Portable Parallel Programming Library (APPL) effort (ref. 16). APPL is available on a variety of parallel machines, including networked workstations. APPL provides portability in the coding of communication routines, and allows for the mapping of parallel tasks to parallel hardware external to the application code. Higher-level programming tools, such as FORTRAN D or object-oriented environments, as they mature, could conceivably be implemented on top of software such as APPL.

It should be noted that APPL provides portability between different classes of MIMD machines, but not between MIMD machines and SIMD machines, such as the CM-2. However, the next-generation Connection Machine will support the MIMD, message-passing model of computation as well the SIMD, data-parallel model (ref. 17). APPL is based on previous work in portable message-passing environments at Argonne and Oak Ridge National Laboratories (refs. 18,19).

CURRENT PARALLEL TURBOMACHINERY CFD RESEARCH

The literature contains many citations of various parallel algorithms as applied to the solution of simplified partial differential equations (e.g. convection-diffusion equations, driven cavity, etc.) and linear systems of equations. The number of references shrinks dramatically when one narrows the search to parallel turbomachinery applications. Of these, many consider only a single blade passage or a cascade of airfoils. There are only a few references dealing with the development of parallel codes that can simulate flows through multiple blade rows.

Current research in the development of parallel, multistage, turbomachinery CFD codes reflects the diversity of the hardware and software outlined above. Reference 11 describes the implementation of a time-accurate, two-dimensional, implicit code for solving the compressible Navier-Stokes equations within a turbine stage. The code is third-order accurate, and a zonal grid approach is used to handle the complex, non-stationary geometry. This code was ported to the Connection Machine 2 (CM-2), a SIMD,

distributed-memory massively parallel computer. A fundamental part of the algorithm in the code is an approximate factorization technique which results in tridiagonal systems of equations which must be solved in each of the three coordinate directions. A diagonalization procedure is used, making the systems scalar tridiagonal, although the block-tridiagonal form is retained in wall-normal directions.

A parallel cyclic reduction algorithm (ref. 20) is used to solve the tridiagonal matrices. This algorithm is provided as an optimized library routine on the CM-2. The rest of the code is implemented using CM Fortran. Reference 11 notes that difficulties were encountered in turbulent eddy viscosity calculations and information transfer between grids. Here, lower-level CM Paris coding was used to improve efficiency. It was noted that the rotor and stator blade row calculations could have been done in parallel. However, the parallel blade row approach was not implemented. The achieved performance of the code on the CM-2 approaches that of a single-processor Cray Y-MP version for large grids.

A parallel implementation of a time-accurate, implicit turbomachinery code is discussed in (ref. 21). The code solves the unsteady, 3D Euler equations. An approximate factorization scheme is used which results in a two-pass solution process. The first pass involves the solution of a block, upper-triangular matrix, and the second pass is a lower-triangular matrix solution. Blocked grids are used to handle the complex, non-stationary geometry. A unique grid distortion technique accounts for the rotation between blade rows.

The blocked-grids are exploited as a way to parallelize the code. Individual blocks are assigned to processors and solved in parallel. A MIMD model of computation is used, as well as an object-oriented, portable, programming environment. The portable environment allows the code to be run on various MIMD configurations, from dedicated parallel computers to networked workstations. High parallel efficiency using five processors is achieved, but it must be noted that a fundamental change to the algorithm was made. The original serial algorithm maintained an ordering in which the various blocks were solved. The ordering was done to preserve the implicit way in which the original unblocked system was solved. In order to solve the blocks in parallel, the implicit relationship between blocks must be relaxed. This results in an error between the sequential and parallel algorithms. The absolute value of the error for the test case given in reference 21 did not exceed three percent. The error could vary for different test conditions. For example, the given test case was not transonic, which may have a further impact on stability and accuracy. Future work in this area will include methods to reduce this error, such as additional iterations in the linearization loop.

The final parallel turbomachinery code to be reviewed is based on the 3D average-passage form of the Navier-Stokes equations (ref. 22). Modelling of interactions between blade rows is used to simplify the computations, in contrast to a fully time-accurate calculation. The entire turbomachinery flow path is solved for each blade row, with neighboring blade rows accounted for by a distribution of body forces, energy sources, energy correlations and velocity correlations. The algorithm used within each blade row is based on a finite-volume, four-stage Runge-Kutta technique. Local time-stepping and residual averaging are used to accelerate convergence. The solution process iterates between 3D and

axisymmetric solutions of the flow path. Overall convergence is determined by comparing the axisymmetric average of each blade row's flow field. If the difference between the axisymmetric solutions is greater than a set tolerance, the inter-blade-row terms are updated and the Runge-Kutta procedure is repeated until convergence is achieved.

A parallel version of this algorithm was implemented on the Cray Y-MP, where individual blade rows were solved on separate processors. High efficiencies were obtained in this case, with a sustained performance of about 1.5 Gflops. Even at this rate, the simulation of a four-stage compressor took 2.2 hours of CPU time on a dedicated Cray Y-MP. Current efforts at the NASA Lewis Research Center are devoted to extending the parallel algorithm of this code to a more massively parallel level. These efforts are described in the next section.

CONSIDERATIONS IN PARALLELIZING THE AVERAGE-PASSAGE TURBOMACHINERY CODE

The existing version of the parallel average-passage turbomachinery code exploited the fact that individual blade-rows could be computed in parallel. This model of coarse-grained parallelism is sufficient if only a few processors are available in a parallel machine. However, current distributed-memory parallel computers can consist of more than 1000 processors. Clearly, an extended approach to parallelization is required to take advantage of this number of processors.

One of the first considerations made in the development of the parallel code was the volatility and variety inherent in the parallel computer hardware market. It was decided to focus on MIMD parallel computers and to use a programming approach that would make the parallel code as independent as possible from the target parallel hardware with the least amount of overhead. To accomplish this, the Application Portable Parallel Programming Library (APPL), as described previously, is used.

The partitioning approach must be selected next. The obvious approach to achieving greater parallelism is to partition the grid within each blade row. Now a two-level parallel structure exists as shown in Figure 3(a). A two-dimensional partitioning of the three-dimensional grid in the axial and radial directions is used. These directions were chosen (as opposed to axial and tangential) so that the axisymmetric solution would be effectively partitioned in addition to the 3D solution process. Figure 3(a) shows the division of the computational domain which consists of four blade rows. Each blade row is partitioned into two subdomains both axially and radially. The last subdomain illustrates the partitioning of the axisymmetric calculation which occurs in each blade row. Also shown in the figure is the required communication between the blade rows. Each blade row requires information on the axisymmetric mesh from each of the other blade rows. This can be viewed as a global communication step.

A natural communication pattern exists between each subdomain with the blade rows shown in the illustration. Ideally, this natural communication pattern can be exploited through a mapping of the parallel tasks to the parallel computer architecture. Mapping of the parallel tasks is beneficial for two reasons. The first is the software engineering benefit. The logical association of natural communication in the parallel turbomachinery

model with the physical communication channels in the parallel computer is useful for code organization and debugging.

The second potential benefit is communication performance. The significance of the mapping process in the overall performance of the parallel code will be dependent on the granularity of each task. For very large-grain tasks, where the computation to communication ratio is high, performance will depend very little on the mapping (assuming that routing of messages is supported in the communication kernel). As the granularity decreases, mapping may become more important. The mapping issue can be important even where direct-connect routing is used, such as the Intel iPSC/860 (ref. 23). Random allocation of parallel tasks to processors could lead to network "hot-spots". An excellent discussion on the effects of "hot-spots" on communication performance can be found in reference 24.

Communication performance information is given in reference 25 for a multiblock grid calculation on both the iPSC/860 and the Delta machines. It was shown that the interprocessor bandwidth could vary by as much as 75 percent using different random assignments of blocks to processors. The net effect on the code's performance was not discussed, however. Rather than leave communication performance to chance through random processor assignment, a mapping is used which minimizes interprocessor network traffic. The mapping effectively uses a ring topology for one-dimensional communication and a mesh topology for two-dimensional cases.

The use of the mesh topology for the two-level parallel implementation of the average-passage turbomachinery code is illustrated in Figure 3(b). The mesh interconnection is actually implemented on the Intel Delta Machine (ref. 26) and is a subset of other interconnection topologies such as the hypercube network. A nearest-neighbor interconnection exists within the blade rows. The communication step between blade rows is handled via a contention-free, global communication procedure.

The multi-stage Runge Kutta technique (ref. 27) used within each blade row is highly parallel (refs. 28,29,30). The bulk of the computation in the Runge Kutta algorithms occurs in the computation of fluxes and artificial dissipation. These portions of the code can be parallelized through a straightforward decomposition of the grid. Boundary conditions and convergence acceleration routines require more effort to parallelize, and may even require the use of alternative algorithms. Examples of some of these issues are given in the next section.

INITIAL TEST CASES

Two test codes were developed as a first step toward a massively parallel, 3D, viscous version of the average-passage turbomachinery code. The parallel codes were initially developed and debugged on the Hypercluster parallel computer (ref. 31). The Hypercluster is a test-bed environment for parallel processing research. As such, it provides a robust environment for parallel code development. The use of APPL on the Hypercluster allows porting of the codes to other parallel machines. Various features of the Hypercluster environment proved useful in the development of the test codes described here. The first is a time-out mechanism built into the communication routines. The time-out feature

interrupts a processor waiting to receive information after a default time period. An error message is then sent to the user's console defining which processor timed-out. This allows rapid detection of communication bugs in the code. Another useful feature is an array bounds checking facility. This allows run-time error messages to be generated if an array access goes beyond the defined array dimension. This feature is extremely useful in detecting bugs related to partitioning of arrays on parallel computers. The Hypercluster also provides built-in performance monitoring tools, where graphical displays identify time spent in computation, communication, etc.

The first test code is a medium-grain parallel version of ISTAGE, a 3D Euler code configured for a single blade row. Details on this code can be found in reference 32. A one-dimensional, axial partitioning of the grid is used, since the grid is the longest in this direction. This implies that boundary conditions at the hub and tip regions, as well as the blade surfaces, are parallelized. The inlet and exit boundary conditions are computed serially. Each subdomain is mapped into a ring configuration, on machines where it is appropriate. A convergence acceleration technique, implicit residual averaging (ref. 33), is used in this code. The recursive nature of the Thomas algorithm used to solve the resulting tridiagonal matrices requires special attention when implemented on a parallel computer. The initial version of the parallel code leaves the Thomas algorithm as a serial calculation. Each processor waits for the data it requires in the recursion relation before computing it's part of the sequence.

The initial results obtained using this approach are shown in Table 1, and are reasonable for up to four processors. After this point, the use of additional processors provided little additional benefit. This was due to the fact that now the unparallelized residual averaging routine represents a significant serial fraction of the overall computation. Increasing the size of the problem (i.e. the size of the grid), consistent with the idea of scaled speedup (ref. 34), would certainly improve the speedup results. Results for a larger grid are shown in Table 2. Indeed, the speedup improves for this case. However, scaling of the problem may not always be interpreted as increasing the size of the grid. One method of scaling, in propulsion simulation, could be the coupling of multiple propulsion system component simulations. Ideally, one desires the best performance for each simulation for the given grid sizes. Therefore, a residual averaging procedure that is more amenable to parallel processing is required.

There are several options to consider in parallelizing the residual averaging routine. Two are described here. The first method essentially decouples the full matrix solution between processors in the axial direction. This is called the decoupled residual averaging technique. Each processor solves an independent tridiagonal system resulting from the grid points within it's own subdomain. This approach works well for up to 20 processors. After this point, the solution loses the benefits of the increased stability margin afforded by the residual averaging process, and becomes unstable. It should be noted that this is due, in part, to the low Mach number application for which ISTAGE was configured. Decreasing the CFL number, and hence, the integration time step solves this problem. However, this results in a decrease in the rate of convergence. It is also expected that higher speed flow applications will not be as sensitive to the decoupled residual averaging technique.

The second method employs an iterative, Jacobi relaxation scheme on the original,

unfactored residual averaging equations. This technique was more robust, and allowed runs using up to 32 processors with the given grid size. Four iterations of the Jacobi scheme yielded a better convergence rate than the direct solution technique, and required about the same amount of cpu time as the decoupled tridiagonal solution approach. It should be pointed out that this technique was tested for only one set of conditions, and additional iterations may be required for other operating conditions.

Figure 4 illustrates performance results obtained on the Hypercluster, the iPSC/860 and the Delta Machine. The figure shows a logarithmic plot of time versus number of processors. Also shown, as a straight line across the plot, is the single-processor Cray Y-MP performance for the same code. The best performance is achieved with the iterative residual averaging technique and compiler optimizations on the iPSC/860. Similar performance can be expected on the Delta machine. Roughly half of the Y-MP performance is achieved using 32 processors.

The decoupled residual averaging technique is used in the other cases. Several interesting observations can immediately be made for these latter cases. The achieved speedup on each of the three machines is about the same. This indicates that the dominant effects must be other than hardware communication time. This only makes sense when one considers the Delta machine, with measured communication performance about 3.5 times (ref. 25) that on the iPSC/860. If hardware communication were the dominant factor, one would expect a better speedup on the Delta machine than on the iPSC/860. Since this is not the case, other factors must be contributing to the loss in efficiency.

Use of the serial fraction metric (ref. 35) confirms this hypothesis. The serial fraction of the code is that part which cannot be computed in parallel. It can be determined experimentally from speedup information using the following equation:

$$f = \frac{1/s - 1/p}{1 - 1/p}$$

where f is the serial fraction, s is the speedup, and p is the number of processors. The serial fraction metric is shown in Table 3 for the three different machines. From the table, it is evident that the serial fraction is relatively constant after eight processors on all three machines. This is expected, since the same code was used on all three machines. The serial fraction of the code should be the same in each case. If hardware related communication costs were causing the inefficiency, one would expect an increasing serial fraction as the number of processors increase.

There is a large jump in the serial fraction for the three processor case on all of the machines. Load balancing is suspected in this case. The serial fraction then decreases and settles out at a value of about four percent on all three machines. This trend makes sense, as decreasing the granularity of the problem minimizes the load balancing effects. The load balancing issue is discussed in more detail shortly.

Another interesting effect can be observed which illustrates the sensitivity of the serial

fraction metric. The two, three and four processor cases have a larger serial fraction on the Hypercluster than on the other machines. The serial fraction then drops off as more processors are used. The Hypercluster uses a multiprocessor node, where four processors are connected by a high-speed bus to shared memory. The multiprocessor nodes are then interconnected in a hypercube configuration. The parallel code runs entirely within a node for up to four processors. Bus contention would be expected to increase as more processors within the node are used. This would explain the increase in serial fraction for these cases. The hypercube network is then used when more than four processors are required. The parallel communications offered by the hypercube connection then dilute the effects of the intranode bus contention.

A detailed analysis of the results reveals some of the causes of the inefficiency. Performance information for an eight-processor version of ISTAGE on the iPSC/860 was gathered using Intel's Performance Analysis Tools (PAT) (ref. 36). A histogram showing the composition of the time spent in each processor is shown in Figure 5. The processor numbers are ordered by their location in the ring mapping sequence. The performance data clearly show a load imbalance among the processors. This is evidenced by the variation in the height of the bars indicating actual computation time. The ordering of the bars correlates with the axial position in the computational domain. It can be seen that the amount of computation in each processor increases as the center of the domain is approached, peaks, and then begins to decrease.

The boundary conditions are the largest contributor to the load imbalance. Processors computing the first and last subdomains must also compute inlet and exit boundary conditions. Also, the grid extends upstream and downstream of the blade surface. The processors assigned to this region of the grid do not have to compute the solid surface boundary conditions. The load imbalance due to the blade surface boundary appears to be the major problem. This is due to the fact that both the first and last processors finish computations before the other processors. This is despite the fact that these processors have additional work that the others do not (due to inlet and exit boundary conditions).

The amount of time spent in the communication routines is also shown. There is currently some debate as to the quantitative accuracy of these performance analysis tools. However, they do give useful qualitative information. They can also give an indication of relative performance improvements for different versions of the code.

The effects of the load imbalance and communication are not too serious for the eight-processor case. A speedup of better than 6 is achieved on all machines tested. The effects are more pronounced as more processors are used. Communication time does play more of a role as increasing the number of processors reduces overall computation time. There are other factors related to communication which also contribute to inefficiency. For example, information from the triple indexed arrays must be gathered to and scattered from communication buffers. This does not show up directly as a communication cost, but is extra work that the processors must do over the serial code.

The second test case was a coarse-grain partitioning of the viscous version of the average-passage code, MSTAGE. Each blade row was computed on a separate processor. The information transferred between blade rows is small compared to the overall calculation,

and was communicated through message passing. The parallel version of MSTAGE was implemented on the Hypercluster and a network of IBM RS/6000-560 workstations. The workstations were connected to a shared ethernet using the TCP/IP protocol. APPL is again used for the interprocessor communication.

The Hypercluster version executes 15 cycles of the Runge-Kutta integration procedure before information is transferred between blade rows. The number of cycles is selected in this case to allow reasonable run times. The high speed of the RS/6000 processor allowed 45 cycles to be executed. The latter number of cycles is more typical of a turbomachinery calculation. The large computation to communication ratio coupled with the high performance of the RS/6000 workstation resulted in impressive performance, as shown in Table 4. Note that the achieved speedup on the Hypercluster is higher than on the RS/6000's. This is probably due to the faster internode communication and file I/O on the Hypercluster. A common file system, accessed via the Network File System (NFS), is used for all I/O on the networked RS/6000's.

An interesting observation is that both of the test cases shown above do not perform equally well on all architectures. The medium-grain parallel code, ISTAGE, was implemented on the network of RS/6000 workstations. The two workstation case took almost twice as long to execute than the single workstation case. In contrast, the current version of the parallel MSTAGE code cannot run on the iPSC/860 due to lack of sufficient memory for the grid sizes deemed reasonable for this code. A medium-grain version of MSTAGE is currently under development which should alleviate this problem. It is anticipated that the lower performance of the i860 microprocessor will require more processors to be utilized than in the RS/6000 case. The above experience indicates that one must carefully match a parallel application to a hardware platform. This will continue to be the case until convergence is reached in parallel computing hardware and software design.

FUTURE PLANS AND CONCLUDING REMARKS

Work is currently in progress to implement a medium-grain version of the MSTAGE code, as mentioned earlier. This will allow the code to run on machines such as the iPSC/860. It will also allow the determination of a "crossover point", where a sufficient number of processors on the iPSC/860 meets or exceeds the performance of the networked RS/6000's. The highest performance version of MSTAGE will be incorporated into the ICE environment described earlier. This will allow runs of MSTAGE to be performed for experimental setup and data analysis. The availability of experimental data in the ICE environment will allow use of the data for initializing boundary conditions and initial flow fields in the MSTAGE code. The ideal interactive use of MSTAGE will not be likely with current parallel hardware, but at least the infrastructure will be in place. Interactive use may be possible with a two-dimensional viscous or three-dimensional Euler code, however. These options will be investigated.

The networked RS/6000 version of MSTAGE will be used to demonstrate the zooming concept in NPSS. A lumped parameter propulsion system simulation will be modified to request compressor map information from the parallel version of MSTAGE. Map information is typically required for several different operating points (e.g speed and

pressure ratio). The information for each point requires a separate run of the parallel MSTAGE code. Each operating point can essentially be computed in parallel on a "bank" of workstations. The number of workstations used in each bank is selected as a function of the number of map points required. For example, if parallel MSTAGE operates at an efficiency of 95% on four workstations, and eight map points are required, 32 workstations could be utilized. Performance for this configuration, based on current timing estimates for RS/6000-560's, would reach 464 MFLOPs. This is over three times the performance of MSTAGE on a single Cray Y-MP processor.

If the number of points required exceeds the number of workstations available, it may be better to run a serial version of MSTAGE on each workstation. This is due to the fact that the serial version runs at 100% efficiency. The decision would most likely be based on the amount of memory available on each workstation. One of the benefits of distributed-memory parallel computing is the reduced memory required on each processor. The serial version of MSTAGE could exceed the core memory capacity of the workstation. Excessive page swapping could result, reducing performance, and again make the parallel version of MSTAGE attractive. Considerations such as these illustrate the need for sophisticated resource allocation software on parallel computers.

The use of networked workstations, in particular the RS/6000's, has proven attractive for a number of reasons. First, the power of the RS/6000 processor allows good performance to be achieved using only coarse-grain partitioning. This would also be the case on the "traditional" parallel computers, such as the iPSC/860, if the single-processor performance were improved. Future improvements in processor and compiler technology should allow this to occur. Second, unused cycles on idle workstations could potentially be utilized. Ideally, software would be provided to manage the allocation of workstations and balance the work distribution.

Parallel computing offers high potential for providing the teraflops performance required for future applications. Hardware and ,especially, software, are as yet immature. This is expected to improve in the future. The high performance/cost ratios available with parallel computers will make the interactive use of digital computers a reality for applications such as ICE and NPSS.

REFERENCES

1. R. Claus, et. al.,"Numerical Propulsion System Simulation," Computing Systems in Engineering, Vol. 2, No.4, pp. 357-364, 1991.

2. J. Szuch, et. al., "Enhancing Aeropropulsion Research With High-Speed Interactive Computing," NASA TM 104374, 1991.

3. C. Daniele, "A Generalized Computer Code for Developing Dynamic Gas Turbine Engine Model (DIGTEM)," NASA TM 83508, 1984.

4. Y. Sugiyama, "J85 Surge Transient Simulation," J. Propulsion, Vol. 5, No. 3, 1989

5. J. Adamczyk, et. al., "The Role of Tip Clearance in High-Speed Fan Stall," ASME Paper 91-GT-83,1991.

6. A. Crook, "Numerical Investigation of Endwall/Casing Treatment Flow Phenomena," M.S. Thesis, MIT, 1989.

7. Kendall Square Research Corporation, "Kendall Square Product and Company," 1990.

8. Thinking Machines Corporation, "Getting Started in CM Fortran," 1990.

9. J. Myczkowski, et. al., "Extremely Fast Finite Difference Techniques for the Connection Machine," AIAA paper AIAA-91-0436, 1991.

10. K. P. Jacobsen, "Computational Fluid Dynamics Application Note," MasPar Computer Corporation, 1990.

11. N. K. Madavan, "Massively Parallel Computing for the Simulation of Unsteady Flows in Turbomachinery," 10th AIAA Computational Fluid Dynamics Conference Technical Papers, pp. 961-962, 1991.

12. S. Hiranandani, et. al., "An Overview of the Fortran D Programming System," Rice University Center for Research in Parallel Computation, CRPC-TR91121, 1991.

13. W. J. Leddy, et. al., "The Experimental Systems Software Environment for Distributed Object Execution," MCC Technical Report, ACT-ESP-015-91, 1990.

14. J. K. Lee and D. Gannon, "Object-Oriented Parallel Programming Experiments and Results," Proceedings of Supercomputing '91, IEEE Computer Society Press, 1991.

15. D. Reese and E. Luke, "Object Oriented Fortran for Development of Portable Parallel Programs," Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 608-615, 1991.

16. A. Quealy, "Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library (APPL)," to be published.

17. Thinking Machines Corporation, "The Connection Machine CM-5 Technical Summary," 1991.

18. E. L. Lusk, et. al., "Portable Programs for Parallel Processors," Holt, Rinehart and Winston, Inc., 1987.

19. G. A. Geist, et. al., "PICL: A Portable, Instrumented Communcation Library," ORNL TM-11130, 1990.

20. D. Heller, "A Survey of Parallel Algorithms in Numerical Linear Algebra," SIAM Review, Vol. 20, pp.740-777, 1978.

21. G. Henley and J. M. Janus, "Parallelization and Convergence of a 3D, Implicit, Unsteady Turbomachinery Flow Code," Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, 1991.

22. R. A. Mulac and J. J. Adamczyk, "The Numerical Simulation of a High-Speed Axial Flow Compressor," ASME Paper 91-GT-272, 1991.

23. Intel, Inc., "Concurrent Supercomputing, the Second Generation: A Technical Summary of the iPSC/2 Concurrent Supercomputer," 1988.

24. G.F. Pfister and V. A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," Proceedings of the 1985 International Conference on Parallel Processing, 1985.

25. J. Hauser and R. Williams, "Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines," Journal of Numerical Methods in Fluids, Vol. 15, No. 1, pp. 51-58, 1992.

26. Intel Corp., "Delta System Description," 1991.

27. A. Jameson, et. al., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes," AIAA-81-1259, 1981.

28. E. Barszcz, et. al., "Performance of an Euler Code on Hypercubes," The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers and Applications, Vol. 2, 1989.

29. G. Chesshire and A. Jameson, "FLO87 on the iPSC/2: A Parallel Multigrid Solver for the Euler Equations," The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers and Applications, Vol. 2, 1989.

30. R. K. Agarwal, "Development of a Navier-Stokes Code on a Connection Machine," The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers and Applications, Vol. 2, 1989.

31. R. A. Blech, "The Hypercluster: A Parallel Processing Test-Bed Architecture for Computational Mechanics Applications," NASA TM-89823, 1987.

32. M. L. Celestina, et. al., "A Numerical Simulation of the Inviscid Flow Through a Counterrotating Propeller," Journal of Turbomachinery, Vol. 108, 1986.

33. A. Jameson, "Solution of the Euler Equations for Two-Dimensional Transonic Flow by a Multigrid Method," Appl. Math. and Comput., Vol. 13, 1983.

34. J. L. Gustafson, G. R. Montry, and R. E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," SIAM Journal on Scientific and Statistical Computing, Vol. 9, No. 4, pp. 609-638, 1988.

35. A. H. Karp and H. P. Flatt, "Measuring Parallel Processor Performance," Communications of the ACM, Vol. 33, No. 5, 1990.

36. Intel, Inc., "iPSC/860 Parallel Performance Analysis Tools Manual," 1991.

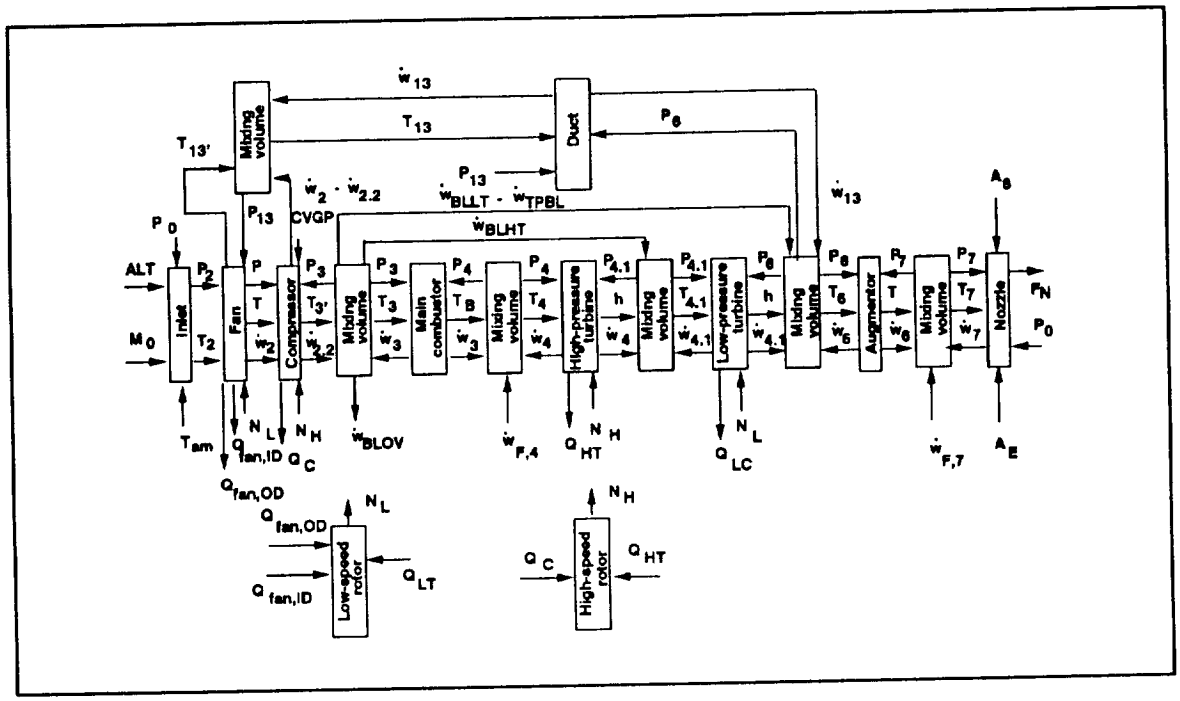

Figure 1 - Lumped-parameter propulsion system model

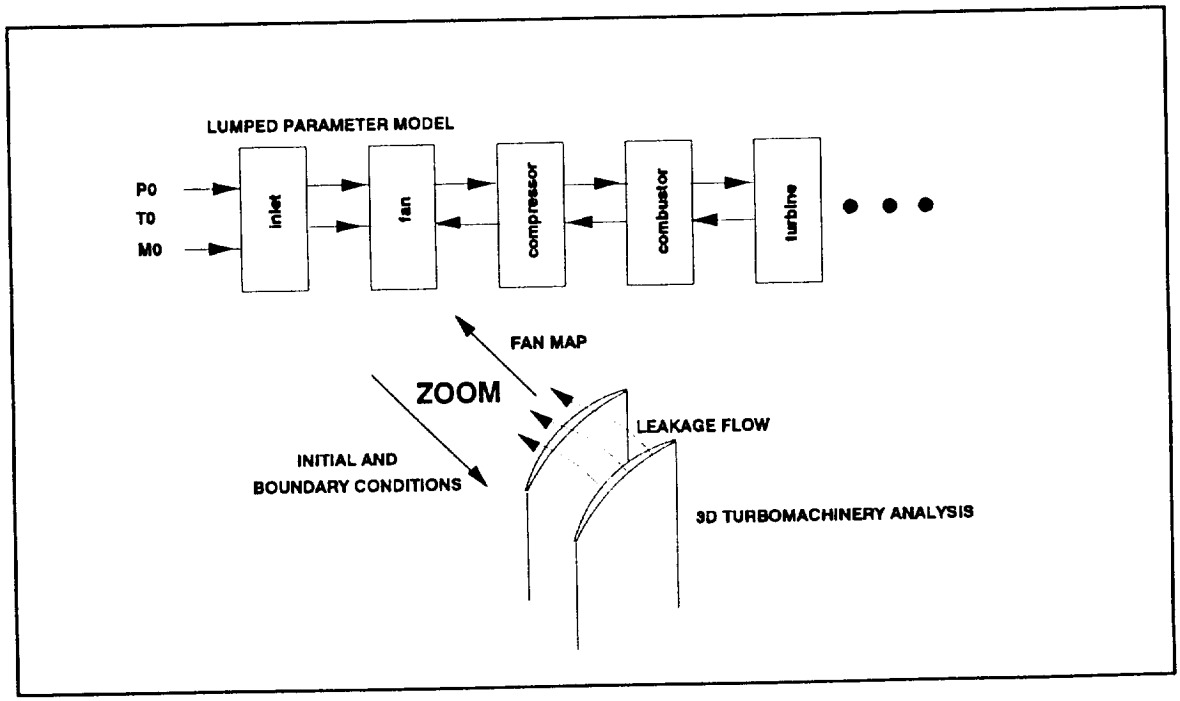

Figure 2 - Illustration of zooming concept

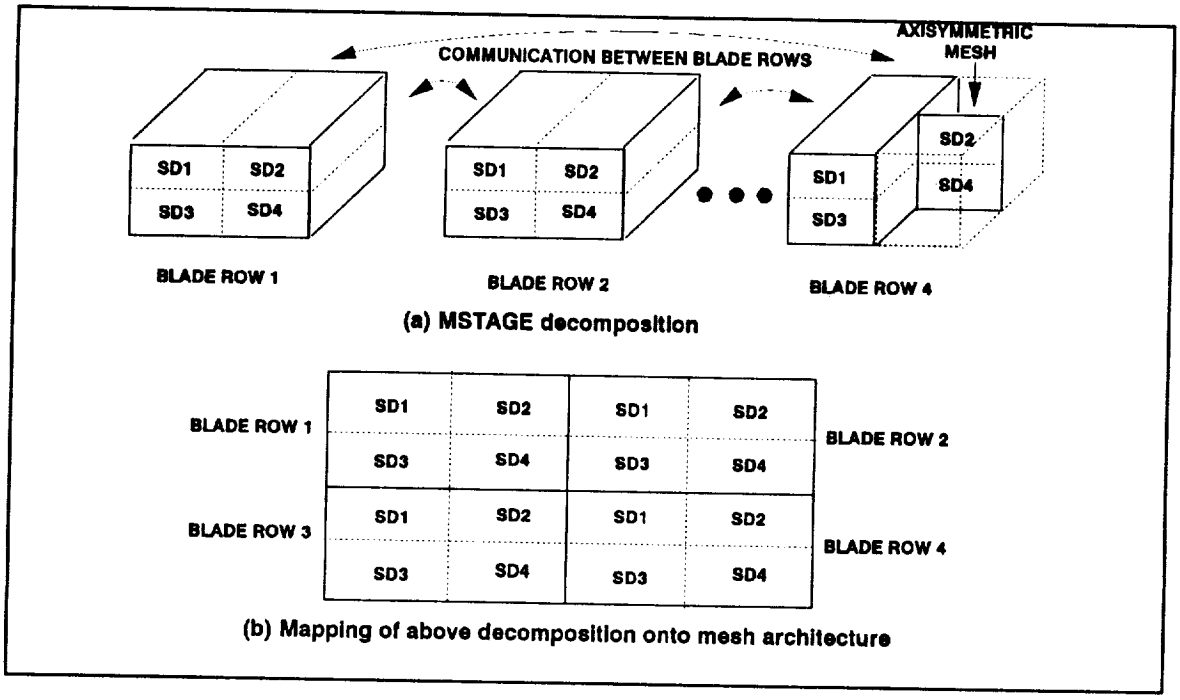

Figure 3 - Decomposition and mapping for MSTAGE

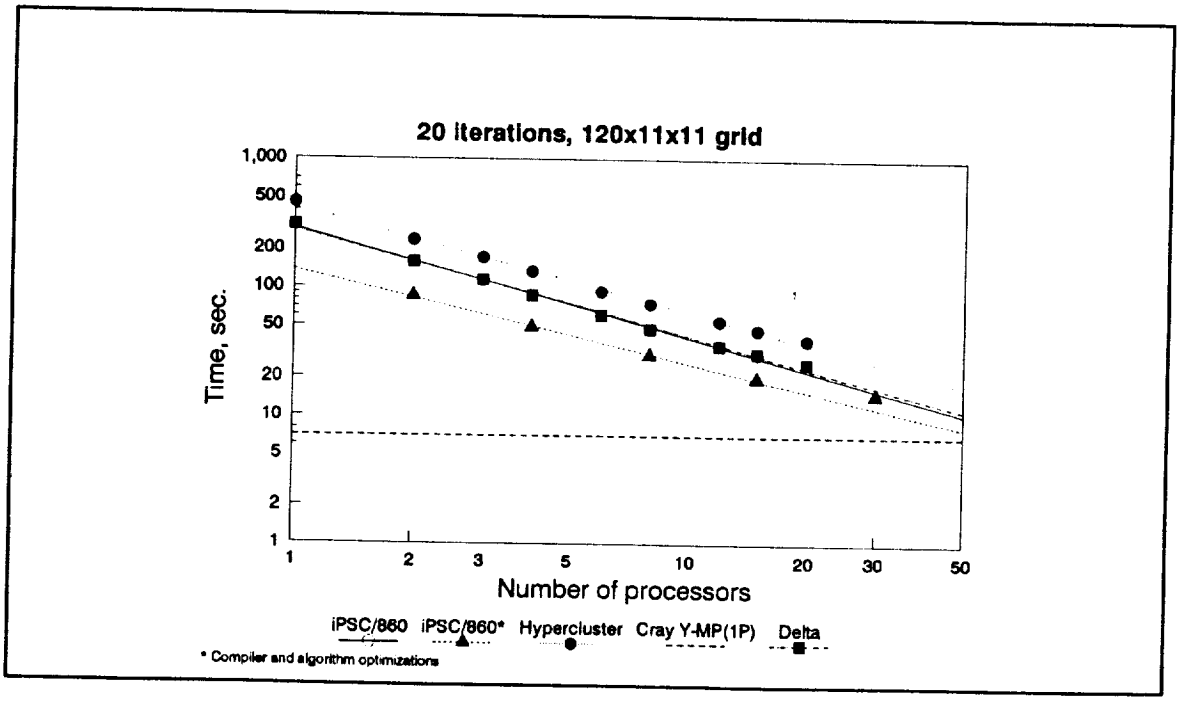

Figure 4 - Parallel ISTAGE performance

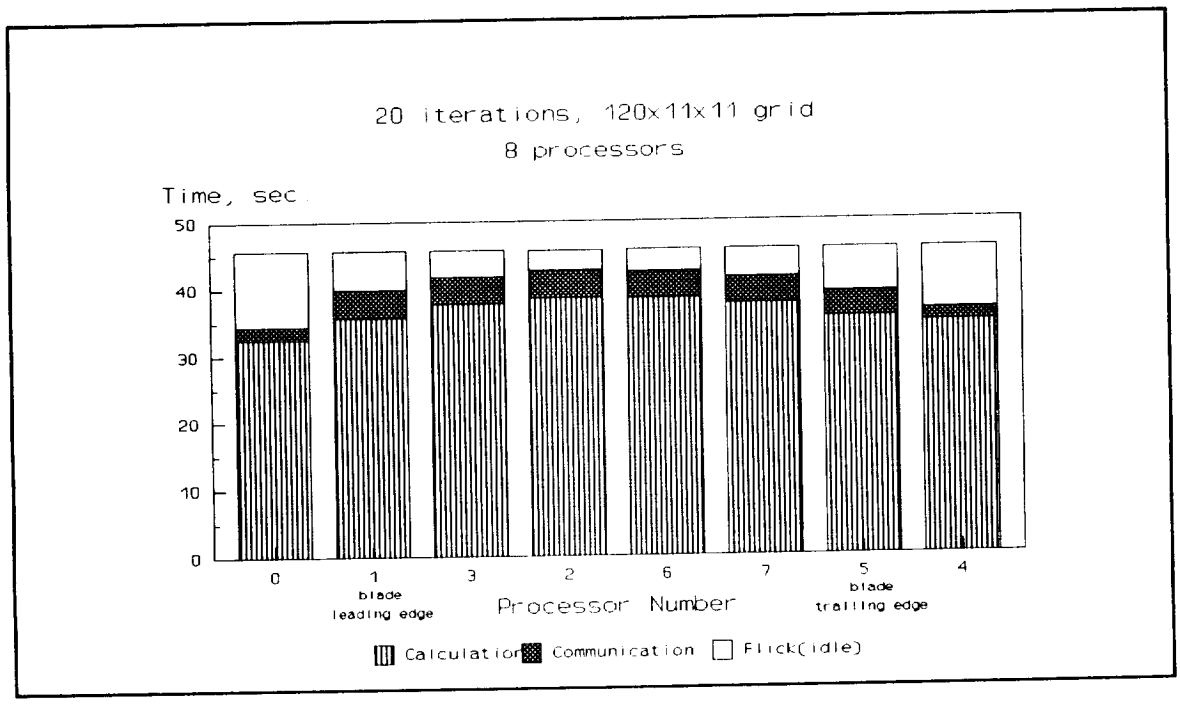

Figure 5 - Performance statistics for parallel ISTAGE

Number of Processors	Hypercluster		iPSC/860	
	Time, sec.	Speedup	Time, sec.	Speedup
1	109.72	-	154.80	-
2	64.66	1.70	90.12	1.72
4	42.58	2.58	59.26	2.61
7	33.98	3.23	44.19	3.50

Table 1 - Performance of ISTAGE with serial Thomas algorithm 42x11x11 grid, 20 iterations

Number of Processors	Hypercluster	
	Time, sec.	Speedup
1	359.79	-
2	194.44	1.85
3	149.11	2.41
4	116.96	3.08
6	89.57	4.02
8	75.41	4.77
12	63.20	5.69

Table 2 - Performance of ISTAGE with serial Thomas algorithm 96x11x11 grid, 20 iterations

Number of Processors	iPSC/860		Hypercluster		Delta	
	Speedup	Serial Fraction	Speedup	Serial Fraction	Speedup	Serial Fraction
2	1.92	.042	1.93	.036	1.92	.042
3	2.67	.062	2.65	.066	2.67	.062
4	3.51	.046	3.41	.058	3.51	.046
6	4.95	.042	4.90	.045	5.00	.040
8	6.28	.039	6.06	.046	6.32	.038
12	8.45	.038	8.24	.041	8.53	.037
15	9.70	.039	9.61	.040	9.69	.039
20	11.27	.040	11.47	.039	11.57	.038

Table 3 - Serial fractions for parallel ISTAGE

Number of Processors	Hypercluster		RS6000		Cray Y-MP
	Time (sec)*	Speedup	Time (sec)**	Speedup	Time (sec)**
1	40,185	-	16,780	-	1,714
2	20,231	1.99	8,760	1.92	-
4	10,272	3.91	4,443	3.78	-

* 15 cycles
** 45 cycles

Table 4 - Parallel MSTAGE Performance, 218x31x31 grid

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1992	Technical Memorandum

4. TITLE AND SUBTITLE

Turbomachinery CFD on Parallel Computers

5. FUNDING NUMBERS

WU-505-62-52

6. AUTHOR(S)

Richard A. Blech, Edward J. Milner, Angela Quealy, and Scott E. Townsend

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-7443

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

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM-105932

11. SUPPLEMENTARY NOTES

Prepared for the Symposium on High-Performance Computing for Flight Vehicles, Washington, D.C., December 7–9, 1992. Richard A. Blech and Edward J. Milner, NASA Lewis Research Center, Cleveland, Ohio. Angela Quealy and Scott E. Townsend, Sverdrup Technology, Inc., Lewis Research Center Group, 2001 Aerospace Parkway, Brook Park, Ohio 44142. Responsible person, Richard A. Blech, (216) 433-3657.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62

12b. DISTRIBUTION CODE

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

The role of multistage turbomachinery simulation in the development of propulsion system models is discussed. Particularly, the need for simulations with higher fidelity and faster turnaround time is highlighted. It is shown how such fast simulations can be used in engineering-oriented environments. The use of parallel processing to achieve the required turnaround times is discussed. Current work by several researchers in this area is summarized. Parallel turbomachinery CFD research at the NASA Lewis Research Center is then highlighted. These efforts are focused on implementing the average-passage turbomachinery model on MIMD, distributed memory parallel computers. Performance results are given for inviscid, single blade row and viscous, multistage applications on several parallel computers, including networked workstations.

14. SUBJECT TERMS

Parallel computing; Turbomachinery; Computational fluid dynamics

15. NUMBER OF PAGES

22

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	