IN-02
013170

NASA Contractor Report 202312
AIAA-97-0026

Parallel NPARC: Implementation and Performance

S.E. Townsend
NYMA, Inc.
Brook Park, Ohio

December 1996

Prepared for
Lewis Research Center
Under Contract NAS3-27186

National Aeronautics and
Space Administration

Parallel NPARC: Implementation and Performance

S. E. Townsend [*]
NYMA Inc.
NASA Lewis Research Group
Brook Park, Ohio 44142

Abstract

Version 3 of the NPARC Navier-Stokes code includes support for large-grain (block level) parallelism using explicit message passing between a heterogeneous collection of computers. This capability has the potential for significant performance gains, depending upon the block data distribution. The parallel implementation uses a master/worker arrangement of processes. The master process assigns blocks to workers, controls worker actions, and provides remote file access for the workers. The processes communicate via explicit message passing using an interface library which provides portability to a number of message passing libraries, such as PVM (Parallel Virtual Machine). A Bourne shell script is used to simplify the task of selecting hosts, starting processes, retrieving remote files, and terminating a computation. This script also provides a simple form of fault tolerance. An analysis of the computational performance of NPARC is presented, using data sets from an F/A-18 inlet study and a Rocket Based Combined Cycle Engine analysis. Parallel speedup and overall computational efficiency were obtained for various NPARC run parameters on a cluster of IBM RS6000 workstations. The data show that although NPARC performance compares favorably with the estimated potential parallelism, typical data sets used with previous versions of NPARC will often need to be reblocked for optimum parallel performance. In one of the cases studied, reblocking increased peak parallel speedup from 3.2 to 11.8.

[*]This work was supported by the NASA Lewis Research Center under contract NAS3-27186 with Gary Cole as monitor.

Introduction

Current trends in computational fluid dynamics stress an ever-increasing need for greater computational power from constant or even decreasing resources. In an effort to provide traditional supercomputer performance using lower-cost workstations, version 3 of the NPARC Navier-Stokes code includes support for large-grain parallelism using explicit message passing between a heterogeneous collection of computers.

Since NPARC is a multi-block code, a natural choice for parallel decomposition is by block. This leads to performance being sensitive to the block data distribution. Typical existing NPARC applications have had their blocks designed for primarily geometric reasons. This usually results in a highly unbalanced data distribution between a small number of blocks. By suitably balancing the load between blocks, possibly via a simple block splitting procedure, NPARC applications can now achieve significant performance improvements through the combined computational power and memory of a collection of workstations.

The following sections describe the parallel implementation, how measurements of performance were made, and the performance results for three real-world data sets.

Parallel Implementation

The addition of a parallel processing capability requires addressing issues related to process organization, file input and output, message passing, system startup and shutdown, and parallel vs. serial results.

Process Organization

NPARC uses a master/worker process organization similar to that of the NASTD code[1]. The master process is responsible for worker block assignment and sequencing while worker processes do the actual calculations. The master process also acts as a file server for the worker processes. This avoids requiring a common file system across all processors.

Having a single master process serve multiple worker processes introduces a potential performance bottleneck. It is possible, via a command-line option, to collect statistics on the time workers spend waiting for service by the master process. As expected, the wait time goes up as the number of workers increases. However, in most cases the master process response time is not the limiting factor to performance.

The main determinant to parallel performance is the block data distribution and how those blocks are assigned to worker processes. NPARC does not control block data distribution, but does employ two methods for block assignment in order to improve the load balance across worker processes depending upon the mode of calculation, *interlocked* or *non-interlocked*, which is controlled by the input parameter **ILOCK**.

Interlocked mode, which is the default mode, is required if bit-for-bit identical results are desired between runs using the same input. In this mode all interface data updates are strictly synchronized and deterministic (see Figure 1). When statically balancing an interlocked calculation, NPARC first sorts the blocks according to size, and then assigns blocks in phases related to the synchronization phases which occur at runtime. Worker load within a synchronization phase is balanced, but overall worker load may be quite unbalanced.

In contrast to interlocked mode, non-interlocked mode is non-deterministic (see Figure 2). Results from runs using the same input will not necessarily be bit-for-bit identical. This is due to the lack of synchronization between interface data updates. Non-interlocked mode is typically suitable for steady-state calculations. When statically balancing a non-interlocked calculation, NPARC first sorts the blocks according to size and then assigns blocks to workers in order to balance overall worker load. This method of balancing results in higher performance than interlocked mode and is preferred when the determinism of interlocked mode is not required. The lack of interface synchronization in this mode also has the effect of allowing worker processes to be working on different iterations, depending upon how imbalanced the blocks are. The input parameter **ISYNC** is provided to limit this effect. All worker processes are required to be within **ISYNC** iterations of each other. All worker processes are also required to have completed the same iteration before a restart file is written. To use this mode for time-accurate calculations, **ISYNC** must be set to zero.

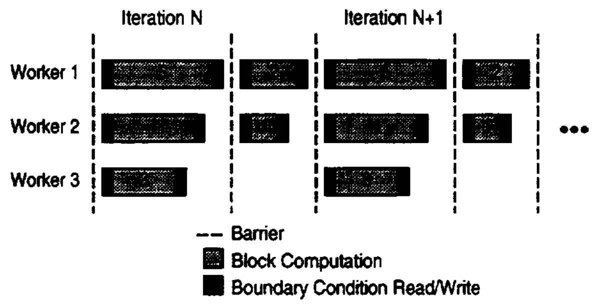

Figure 1: Interlocked computation sequence.

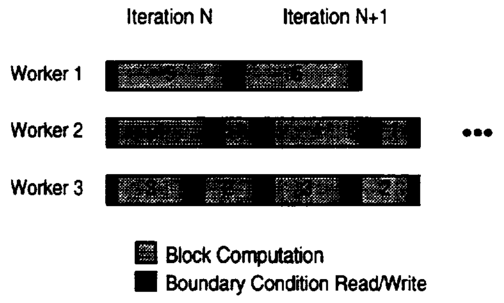

Figure 2: Non-interlocked computation sequence.

The master/worker process organization chosen is not the only way a multi-block code can be run in parallel. For instance, all processes could be peers, maintaining a synchronized view of the system state via various mechanisms. However, the master/worker arrangement is likely to be more flexible for future enhancements such as interactive steering or coupling with other codes in a multidisciplinary environment.

Input/Output Methods

In a distributed environment file handling can sometimes be a problem. As stated above, NPARC's master process acts as a file server for the worker processes. Thus, for a worker to read (or write) a file it sends a read (or write) request to the master process which then returns (or accepts) the data. This file server arrangement is used for restart, interface, and interpolation files. Worker output files such as the convergence history are collected at the end of a run by the **runnparc** script to be described later. All other files have temporary lifetimes and are handled independently by each worker process in typical non-distributed fashion.

File placement may be controlled via the `NPARC_TMP` environment variable. This is particularly useful in clustered environments where files default to an NFS file server. An appropriate setting of `NPARC_TMP` can then cause temporary files to be placed on locally attached disks, avoiding NFS overheads and network contention.

In NPARC 2.0, setting the compile-time parameter `MDISK` appropriately would avoid disk I/O overhead when accessing block data by keeping this data in memory. NPARC 3.0 adds a similar capability for interface and interpolation data via the `MIFACE` and `MINTERP` compile-time parameters, respectively.

Message Passing Interface Layer

A layer of low overhead subroutines are used to isolate the NPARC code from the differences between various message passing library interfaces. These routines also provide a library-independent mechanism for process and messaging statistics.

By using this interface layer, NPARC has been run over MPI (Argonne MPICH and SGI implementations), IBM MPL, and PVM (Oak Ridge and Cray T3D implementations).

System Startup and Shutdown

NPARC uses a Bourne shell script **runnparc** to handle issues related to system startup, fault recovery, and shutdown. A single script is used to support the various message passing environments NPARC can use, providing the user with a single uniform interface. Use of a script separate from the NPARC code itself allows site and/or user customization of how the code is run without affecting the code itself.

During system startup hosts specified by the user or queueing system are scanned to ensure that NPARC and the message passing environment are available. This avoids various difficult-to-interpret errors when starting a distributed application. Once the host scanning phase is complete, any initialization files required by the message library (i.e. the PVM hostfile) are created and any environment initialization (such as starting PVM daemons) is performed. Finally, the master NPARC process is invoked.

During the run, if a fault occurs (such as worker process abort) the NPARC master process will terminate. Upon master process termination, the **runnparc** script interrogates a file to determine why the master process stopped and the number of completed iterations. If the reason for stopping was not a normal completion of the calculation and not a known unrecoverable error (such as an erroneous input file), then the script will automatically restart the calculation from the last restart file written. To avoid constantly restarting in situations which repeatedly fail, **runnparc** requires that the number of completed iterations increase within a configurable number of restart attempts. This simple form of fault tolerance allows NPARC to recover from various problems (such as host failure) automatically. Obviously, loss of the **runnparc** host is fatal.

Once the calculation completes normally (or too many failures have occurred) **runnparc** will perform any necessary message library cleanup (such as PVM daemon shutdown), collect output files, and remove worker temporary files.

Result Comparison

Since block interface data is exchanged differently when running in parallel, the results of serial and parallel calculations are not expected to be bit-for-bit identical. The differences are the result of differing block evaluation orders. This affect also occurs with the serial code when the block order is explicitly changed via the NPARC input file.

Figure 3 shows pressure contours calculated after converging to a steady-state solution with an L_2 residual of 1×10^{-8} for the "Case 4" example from the NPARC user's guide[2]. For this calculation, the blocks are solved serially in normal (1, 2, 3) order. Performing the same calculation, but using a reversed (3, 2, 1) block evaluation order results in a maximum deviation from the original calculation of 6.79×10^{-6}. Performing the same calculation again, but this time evaluating the blocks in parallel results in a maximum deviation of 4.65×10^{-6}.

This example shows that while the parallel results are not bit-for-bit identical to results from a serial calculation, the differences are the same order of magnitude as those resulting from a serial calcu-

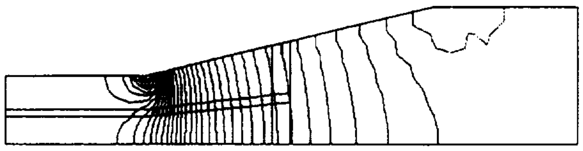

Figure 3: Case 4 pressure contours after converging to an L_2 residual of 1×10^{-8}, block evaluation order: 1, 2, 3.

lation using a different block evaluation order.

Performance Measurement

Metrics

The primary metric used in this study is *Iterations Per Hour (IPH)*. This is a more concrete and application-oriented metric than a simple speedup ratio. Speedup ratios can be misleading when comparing different block data distributions since better parallel block distributions can adversely affect serial performance.

IPH ignores startup and shutdown overheads which are typically a small percentage of total runtime for a real application, but are often a large percentage for the short runs used in performance analyses such as described here. IPH also ignores the effect that block data distribution has on convergence rate. Time to convergence would be a better metric, but the time required would preclude studies such as that described here.

The traditional parallel speedup metric is easily derived as the ratio of parallel IPH to serial IPH. Processor efficiency can then be obtained as the ratio of parallel speedup to the number of processors used.

Also reported below is *estimated speedup*. This metric is directly based upon the block data distribution. It is the ratio of the total number of grid points assigned to the most heavily loaded worker process to the total number of grid points in all blocks. This is a measure of the potential parallelism. It neglects all parallel overheads.

Measurement Procedure

Performance was measured using IBM RS/6000 model 590 workstations using either an Ethernet or ATM network. The message passing library used was PVM version 3.3.11. Non-interlocked mode runs were configured with NPARC input parameter `ISYNC=2`.

In an effort to keep serial vs. parallel performance comparisons fair, all serial tests were performed on a machine with sufficient memory to avoid inflicting the serial test with additional paging overhead. In addition, to isolate computational performance from possible NFS overheads and network contention, all files were located on locally attached disks.

The IPH metric was obtained by enabling the NPARC `-trace iterations` command-line option which reports elapsed time per iteration. An average of the reported elapsed times per iteration was taken after dropping anomalous values from startup and shutdown phases of the calculation. While no other users were allowed on the systems during the measurements, various system programs would occasionally be executed for automated system maintenance. Because of this, the exact performance measured for any specific data point may not be reproducible. However, the trends in performance are reproducible, and are expected to be valid for other system configurations as well.

Performance Results

F/A-18 Inlet

The first data set used for performance analysis is based on an F/A-18 inlet flow study[3]. The default Beam-Warming pentadiagonal solver is used with the Spalart-Allmaras turbulence model. The data has been modified from the original in order to conform to NPARC 3.0 input requirements. The block data distribution for this case is shown in Figure 4. Note the considerable variation in block size, and the large number of block interfaces. These factors result in a maximum estimated speedup of only 6.0 and over 900 messages per iteration.

Figure 5 shows the performance when running in non-interlocked mode with various memory allocation options in effect. All runs were performed using an Ethernet network. Peak performance occurs with seven processes, with a measured speedup of 4.9 compared to an estimated speedup of 5.9.

The relative insensitivity of performance with respect to memory allocation options is likely due to a combination of using a locally attached disk and disk buffer caching in the operating system. Performance when accessing files via NFS is likely to be much more sensitive to what data is resident in memory

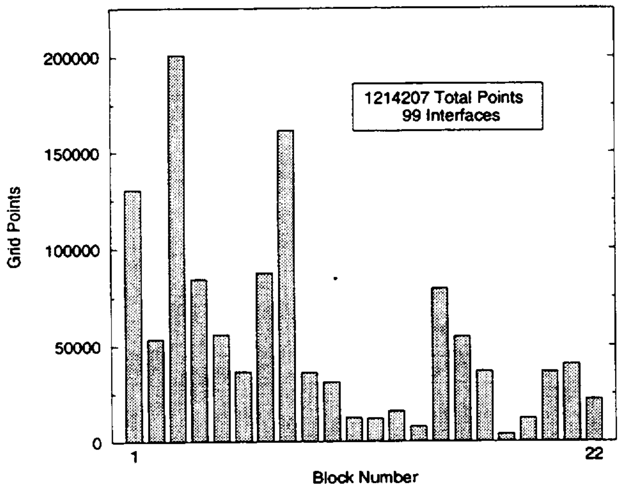

Figure 4: F/A-18 block data distribution.

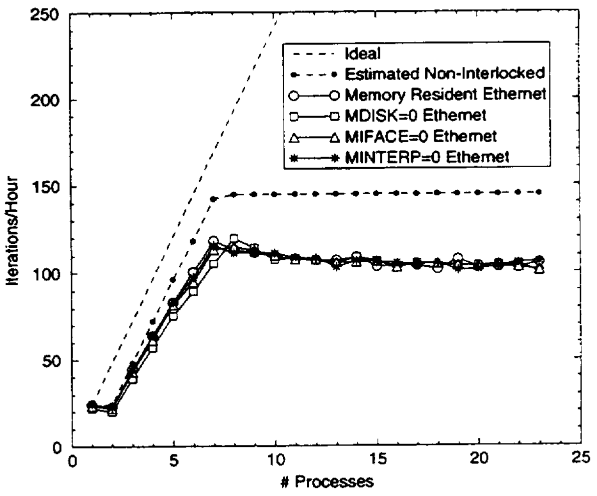

Figure 5: F/A-18 non-interlocked solution performance using an Ethernet network.

and what must be brought in from disk. Regardless of disk access method, it is most important to have block data resident (when there are less worker processes than blocks). Next in importance is keeping interface data resident. Keeping interpolation data resident appears to be fairly unimportant to application performance, though there is a measurable effect.

Figure 6 shows the performance when running in interlocked mode. These runs have all data resident in memory. Estimated performance for both interlocked and non-interlocked modes are shown to indicate the effect of the different block assignment strategies in the two modes. Runs using Ethernet and ATM networks are shown, displaying the sensitivity of application performance on network performance. Peak Ethernet performance occurs with twelve processes, with a measured speedup of 3.0 compared to an estimated speedup of 5.1. Peak ATM performance occurs with sixteen processes (no more ATM hosts were available for testing), with a measured speedup of 4.0 compared to an estimated speedup of 5.4.

With the poor load balance between blocks and the large number of interfaces, the F/A-18 data set provides an example of how NPARC performs in a near worst case scenario. Non-interlocked mode performance is fair, but interlocked mode performance is quite poor. Even with a high-performance network such as ATM, application performance is disappointing, indicating that code modifications are required. Such modifications have begun, and pre-

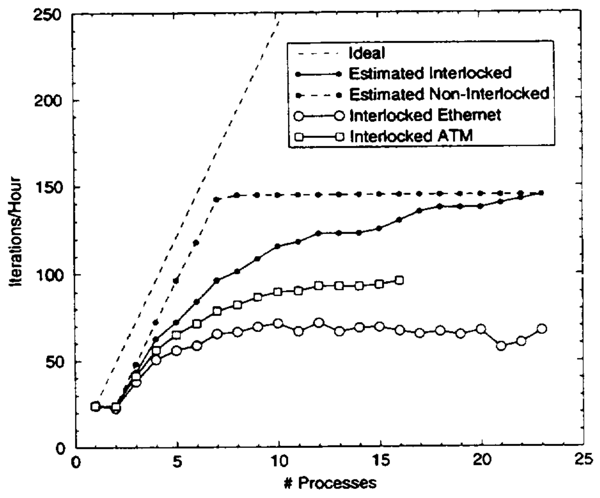

Figure 6: F/A-18 interlocked solution performance using Ethernet and ATM networks.

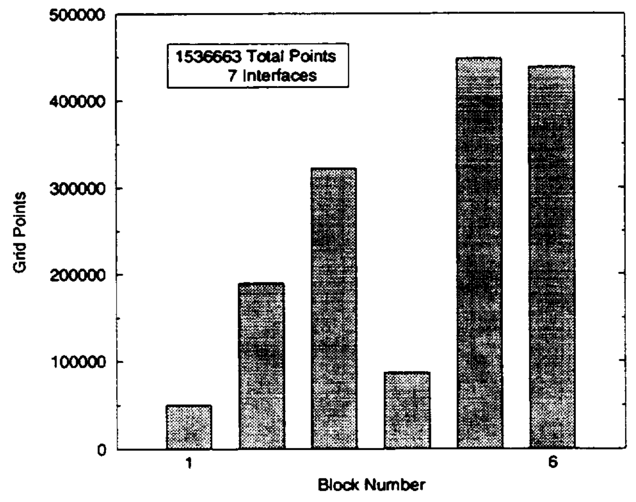

Figure 7: Original RBCC block data distribution.

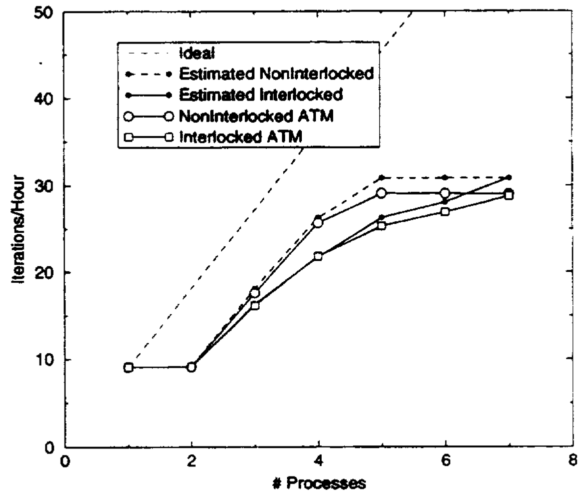

Figure 8: Original RBCC solution performance using an ATM network.

liminary results are shown at the end of this paper.

Rocket Based Combined Cycle Engine

The second data set used for performance analysis is based on a Rocket Based Combined Cycle Engine (RBCC) inlet study[4]. The default Beam-Warming pentadiagonal solver is used with the Chien k-ε two-equation turbulence model. The data has been modified from the original in order to conform to NPARC 3.0 input requirements. The block data distribution for this case is shown in Figure 7. Note the considerable variation in block size. This results in a maximum estimated speedup of only 3.4.

Figure 8 shows the performance when running in both interlocked and non-interlocked modes. These runs have all data resident in memory and use the ATM network. Peak non-interlocked performance occurs with 5 processes, with a measured speedup of 3.2 compared to an estimated speedup of 3.4. Peak interlocked performance occurs with 7 processes, with a measured speedup of 3.2 compared to an estimated speedup of 3.4. The significantly lower overall performance shown here compared to the F18 performance is likely due to the more complex turbulence model.

The limited number of blocks and poor balance of this data set is likely to be representative of many current NPARC applications. Parallel performance with respect to the potential parallelism is good, primarily due to the limited number of block interfaces. The limiting factor is the imbalance in the block data distribution.

Reblocked RBCC

The third data set used for performance analysis is taken from the RBCC case described above. However, the data has been reblocked using a simple block-splitting routine in order to approximately balance the amount of data in each block. The blocks were split using a cutting plane perpendicular to the main flow path. The resulting block data distribution is shown in Figure 9. Note that although the balance between blocks has been improved, both the total number of points (due to block overlaps) and the number of block interfaces has increased. This reblocking results in an increase in maximum estimated speedup from 3.4 to 12.6 and an increase in messages per iteration from 83 to 191.

Figure 10 shows the performance when running in both interlocked and non-interlocked modes. These runs have all data resident in memory and use the ATM network. Peak non-interlocked performance occurs with 16 processes, with a measured speedup of 11.8 compared to an estimated speedup of 12.6. The temporary plateau in performance at 8 processes appears to be the result of the block assignment strategy and master/worker process arrangement, but this phenomenon needs further investigation. Peak interlocked performance occurs with 16 processes, with a measured speedup of 10.6 compared to an estimated speedup of 12.6. Beyond 8

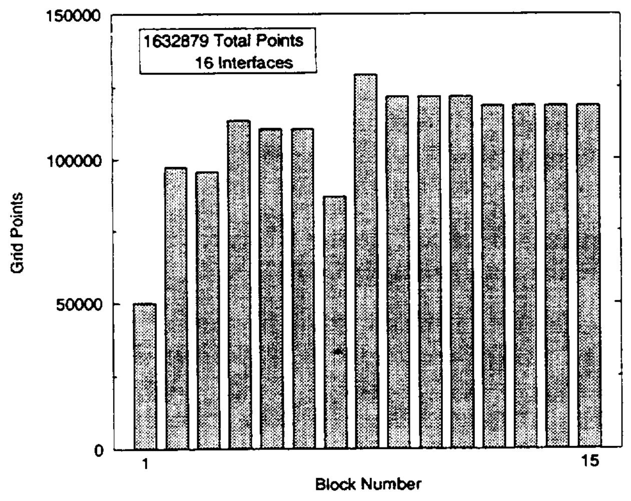

Figure 9: Reblocked RBCC block data distribution.

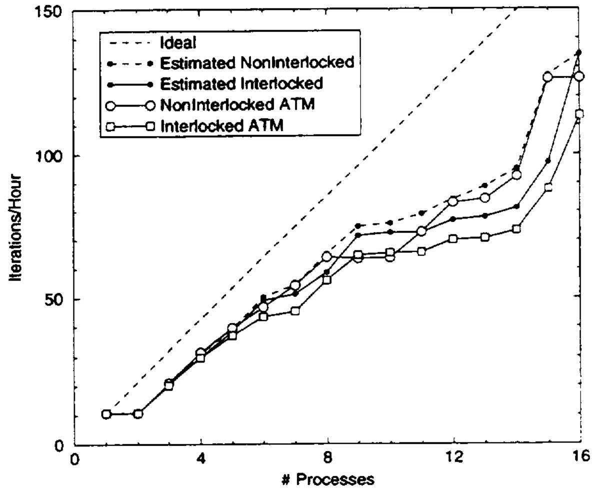

Figure 10: Reblocked RBCC solution performance using an ATM network.

processes parallel overheads become evident. This is primarily the result of worker synchronization.

As can be seen from comparing figures 8 and 10, simply splitting existing blocks in a manner to approximately balance their size can have dramatic effects on parallel performance. Note that even when the original block distribution is performing well (such as at 4 processes) the reblocked case has better performance. This increased performance is not entirely free of drawbacks. The increased number of blocks will increase the number of iterations required for a given level of convergence. However, the performance advantage of the reblocked case is expected to outweigh the disadvantage of the slightly slower convergence rate.

Future Releases

Various modifications to the released NPARC 3.0 code have been made to improve its performance. These include a less restrictive interlock barrier technique, better message packing, and tracking which worker is running the slowest so that its messages may be given a higher priority. Figure 11 shows the before and after results of these changes, using the F/A-18 interlocked calculation as the test case. Both the Ethernet and ATM performance have been considerably improved, with the ATM performance reasonably close to the estimated speedup curve. The majority of the performance improvement is likely due to the improved interlock scheme. Figure 12 shows the before and after results when running the reblocked RBCC non-interlocked calculation. In this case performance is essentially unchanged, the variations shown are not significant given the uncertainty in the measurements. Note that the temporary plateau in performance at 8 processes noted previously is still evident. This shows that the phenomenon is not the result of inefficient message packing or message servicing.

Additional changes which may be included in future releases include enabling worker processes to exchange interface data directly rather than via the master process, and performing multiple block solution iterations between block interface updates.

Concluding Remarks

The new parallel capability of NPARC can provide a considerable boost in performance over the serial version provided a reasonably balanced block distribution is used along with a good network such as ATM. The computed results are different than those calculated by the serial code, but the differences are qualitatively the same as those resulting from an altered block order in a serial computation.

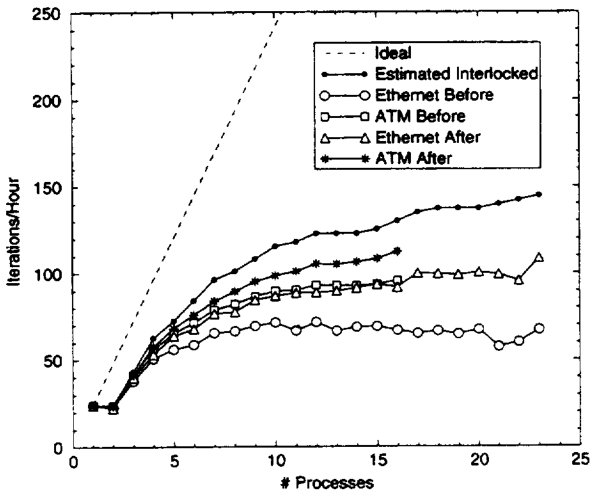

Figure 11: Improved NPARC interlocked solution performance on F/A-18 data set using Ethernet and ATM networks.

References

1. Johnson, J., "Distributed Parallel Processing in Computational Fluid Dynamics, " NPARC Alliance Technical Meeting, Fall 1994.
2. Sirbaugh, J., et al., *A User's Guide to NPARC Version 2.0*, November 1, 1994.
3. Smith, C.F., Podleski, S.D., "Installed F/A-18 Inlet Flow Calculations at 30 Degree Angle-of-Attack: A Comparative Study," AIAA-94-3213, AIAA/ASME/ASEE 30th Joint Propulsion Conference, Indianapolis, IN, June 27-29, 1994. (See also: NASA CR-195297)
4. DeBonis, J.R., Yungster, S., "Rocket-Based Combined Cycle Engine Technology Development—Inlet CFD Validation and Application," AIAA-96-3145, AIAA/ASME/-SAE/ASEE 32nd Joint Propulsion Conference, Lake Buena Vista, Florida, July 1-3, 1996. (See also: NASA TM-107274, ICOMP-96-6)

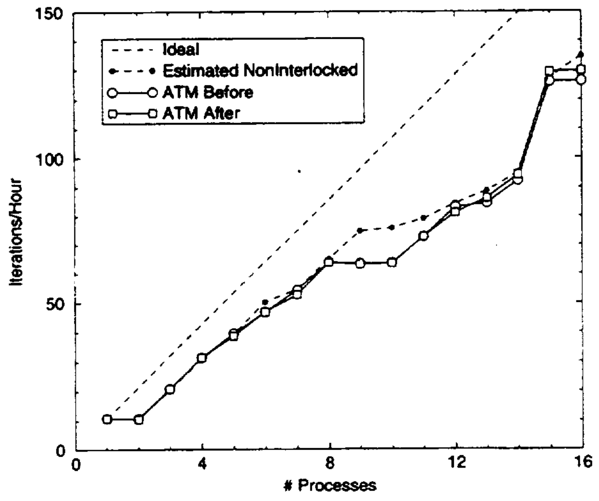

Figure 12: Improved NPARC non-interlocked solution performance on reblocked RBCC data set using an ATM network.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED Final Contractor Report

4. TITLE AND SUBTITLE Parallel NPARC: Implementation and Performance	5. FUNDING NUMBERS WU–509–10–11 C–NAS3–27186
6. AUTHOR(S) S.E. Townsend	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NYMA, Inc. 2001 Aerospace Parkway Brook Park, Ohio 44142	8. PERFORMING ORGANIZATION REPORT NUMBER E–10605
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135–3191	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR–202312 AIAA–97–0026

11. SUPPLEMENTARY NOTES

Prepared for the 35th Aerospace Sciences Meeting and Exhibit sponsored by the American Institute of Aeronautics and Astronautics, Reno, Nevada, January 6–10, 1997. Project Manager, Gary L. Cole, Turbomachinery and Propulsion Systems Division, NASA Lewis Research Center, organization code 5880, (216) 433–3655.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Categories 02 and 61 This publication is available from the NASA Center for AeroSpace Information, (301) 621–0390.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Version 3 of the NPARC Navier-Stokes code includes support for large-grain (block level) parallelism using explicit message passing between a heterogeneous collection of computers. This capability has the potential for significant performance gains, depending upon the block data distribution. The parallel implementation uses a master/worker arrangement of processes. The master process assigns blocks to workers, controls worker actions, and provides remote file access for the workers. The processes communicate via explicit message passing using an interface library which provides portability to a number of message passing libraries, such as PVM (Parallel Virtual Machine). A Bourne shell script is used to simplify the task of selecting hosts, starting processes, retrieving remote files, and terminating a computation. This script also provides a simple form of fault tolerance. An analysis of the computational performance of NPARC is presented, using data sets from an F/A-18 inlet study and a Rocket Based Combined Cycle Engine analysis. Parallel speedup and overall computational efficiency were obtained for various NPARC run parameters on a cluster of IBM RS6000 workstations. The data show that although NPARC performance compares favorably with the estimated potential parallelism, typical data sets used with previous versions of NPARC will often need to be reblocked for optimum parallel performance. In one of the cases studied, reblocking increased peak parallel speedup from 3.2 to 11.8.

14. SUBJECT TERMS Parallel computing; CFD			15. NUMBER OF PAGES 11 16. PRICE CODE A03
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT