AIAA JOURNAL
Vol. 36, No. 11, November 1998

Parallel Processing Scheme for the Navier–Stokes Equations, Part 2: Parallel Implementation

N. Ghizawi* and S. Abdallah†
University of Cincinnati, Cincinnati, Ohio 45221-0070

Parallel performance of two implicit schemes for solving the compressible Navier–Stokes equations on two parallel computational platforms using the standard (portable) message passing interface is studied. These two schemes are a lower-upper cycle-independent scheme developed by the authors and the symmetric successive overrelaxation scheme (representative of cycle-dependent schemes). The parallel computational platforms used here are the Lewis advanced cluster environment, which is an example of a network of workstations, and the Ohio supercomputer Cray T3D massive parallel computing environment. Two versions of the lower-upper cycle-independent scheme are studied: version I, where one processor per block is used, and version II, where two processors per block are used. However, only version I is possible with the symmetric successive overrelaxation scheme. The competitiveness of the workstation cluster to the Cray T3D is demonstrated. Moreover, parallel performance results indicate that version I of the lower-upper cycle-independent scheme is superior to version II in the range of a moderate number of processors. However, in the future, as the number of processors increases and the communication cost decreases, this scheme will provide a means for using larger computational power (relative to version I) that would otherwise be idle.

Introduction

COMPUTATIONAL fluid dynamics problems of engineering interest are among the most demanding scientific problems in terms of the massive computational resources they require. Only parallel architecture computers offer the promise of providing orders of magnitude greater computational power.

Many workers have tried to implement implicit schemes on parallel computers.[1–4] Generally, these works are based on the approximate factorization method employing either an alternating direction implicit or a lower-upper factorization.[5–7] In either case, the factors (steps) resulting from factoring the multidimensional problem are dependent on each other where the solution from the first factor (step) is needed to be able to solve the second step, and so on. Therefore, these steps have to be performed in series, which limits the parallelization features for these types of factorizations. In fact, the main thrust of these works is directed toward employing domain decomposition procedures where subdomains are assigned to separate processors and solved simultaneously. The boundary conditions at the subdomain interfaces are usually time lagged to enable solving each subdomain separately.

In Part 1 of this work,[8] an implicit lower-upper cycle-independent (LUCI) scheme for solving the compressible Navier–Stokes equations was developed. Using this scheme, the factors (steps) resulting from factoring the multidimensional problem are independent of each other; therefore, parallel processing can be used to enhance computations of large problems.[9,10] The desirable characteristics for the LUCI scheme shown in part 1 (Ref. 8), namely, the unconditional stability, symmetry preserving, accuracy, and robustness in massively parallel environments, motivated us to perform the next natural part of this study (part 2). In this part, the parallel performance of the LUCI scheme on two parallel computational platforms is studied and compared with that of the symmetric successive overrelaxation (SSOR) scheme[7] (a typical cycle-dependent scheme). The two computational platforms used here are the Lewis advanced cluster environment (LACE),[11] which is an example of a network of workstations (NOW),[12] and the Cray T3D massive parallel computing environment.[‡] The standard (portable) message passing interface (MPI)[13] is the parallel library we used to parallelize our code. A staggered biplane configuration employing a hybrid multiblock grid is used as the benchmark case. Details of the parallel implementation of two versions (version I for the SSOR scheme and versions I and II for the LUCI scheme) are given.

In the next section, we briefly discuss the computational platforms used. Then we describe the benchmark problem used for evaluating the parallel performance. The parallelization approach for versions I and II is then discussed, and finally, the parallel performance results are presented and analyzed.

Computational Platforms

A. Network of Workstations

In this work we used a cluster of workstations available at the NASA Lewis Research Center. This cluster is called LACE[12] and is connected via several networks. The LACE cluster gets continuously upgraded. The present configuration has 32 RISC/6000 processors (nodes 1–32) and a RISC/6000 model 990 (node 0), which is the file server. Nodes 1–20 are of the 590 model (the CPU has a 66.5-MHz clock, 256 kilobytes of data, and 32 kilobytes of instruction caches), whereas nodes 21–32 are of the slower 560 model (the CPU has a 50-MHz clock, 64 kilobytes of data, and 8 kilobytes of instruction caches). The main memory capacity of these nodes varies between 128 and 512 megabytes. All of these nodes or subsets of them are connected via several networks [Ethernet, Fiber Distributed Data Interface (FDDI), Asynchronous Transfer Mode (ATM), etc.] with different communication speeds and connection characteristics.

In this study, we used the fast LACE/590 model (nodes 1–20). Furthermore, we chose to use the ATM network due to its high bandwidth and its availability over all of the nodes. The MPI (MPICH implementation version 1.0.13) is the parallel library we used to implement our parallel programs on the LACE cluster.

B. Cray T3D

The Cray T3D is a globally distributed but logically shared memory multiprocessor with the topology of a three-dimensional torus mesh network (three-dimensional mesh with edges wrapped around and connected).[14‡] The machine we used in our study is the Ohio Supercomputer Center's Cray T3D. It consists of 128 DEC Alpha 21064 processors with 8 megawords (64 megabytes) of memory

Received Dec. 5, 1997; revision received July 10, 1998; accepted for publication July 26, 1998. Copyright © 1998 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Graduate Assistant, Department of Aerospace Engineering and Engineering Mechanics; currently Aerodynamicist, AlliedSignals Turbocharging Systems, Mail Stop R12, 3201 West Lomita Boulevard, Torrance, CA 90505. Member AIAA.

†Professor, Department of Aerospace Engineering and Engineering Mechanics. Member AIAA.

‡Ohio Supercomputing Center, http://www.osc.edu.

per processor. Finally, the message passing library we used on the T3D is MPI (CRI/EPCC implementation supplied by the Cray Research Incorporated via Edinghburgh Parallel Computing Center, Version 1.5a).

Description of the Test Problem

To evaluate the parallel performance of the LUCI and the SSOR schemes, the staggered biplane configuration shown in Fig. 1 of Part 1 (see Ref. 8) is used as the benchmark case. The two elements are NACA 0012 airfoils, staggered half a chord length in the chordwise and pitch directions. The flow is transonic with a freestream Mach number M_∞ of 0.7 and the configuration is at a zero angle of attack. Even though the flow is subcritical for a single NACA 0012 airfoil at $M_\infty = 0.7$ and zero angle of attack, a shock forms in the passage between the two airfoils.

A hybrid C-H-C three-block grid is used to solve this problem. These blocks are shown in Fig. 1, and the mesh schematics close to the two airfoils are shown in Fig. 2 (only one out of every 10 mesh lines in each direction is shown for clarity). Figure 3 gives the resulting computational domain for this problem.

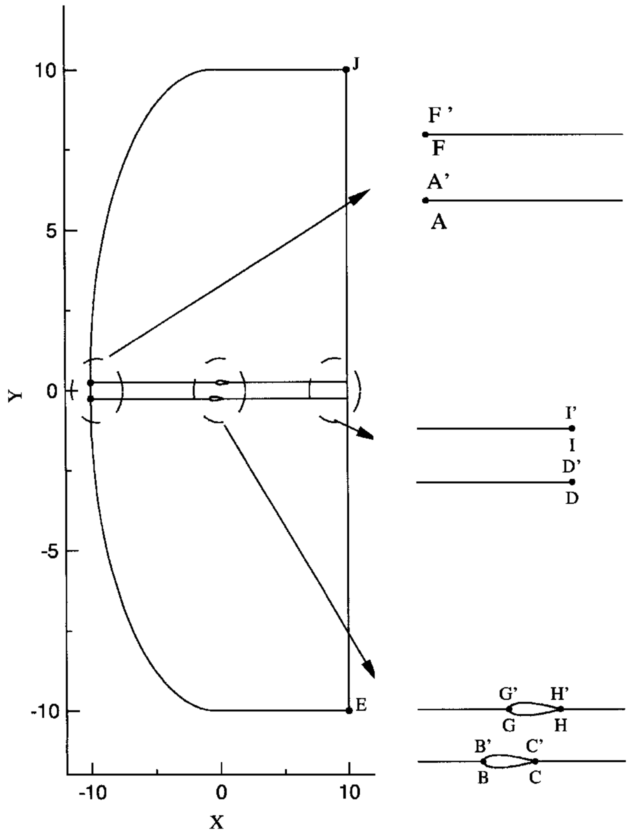

Fig. 1 Physical domain for the staggered biplane configuration.

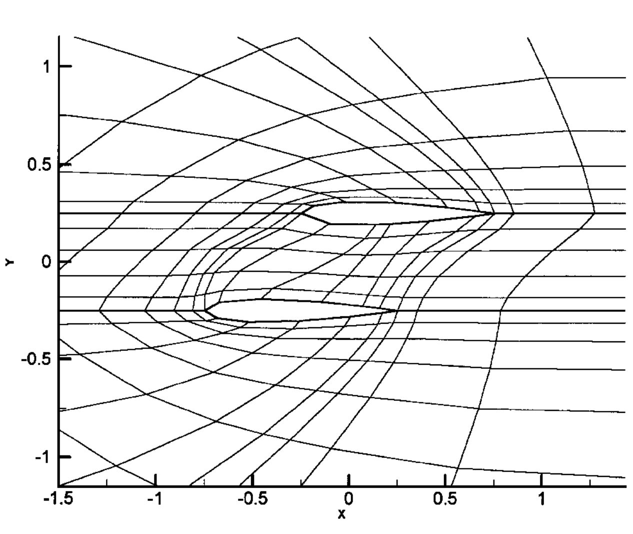

Fig. 2 Hybrid mesh schematics close to the airfoils.

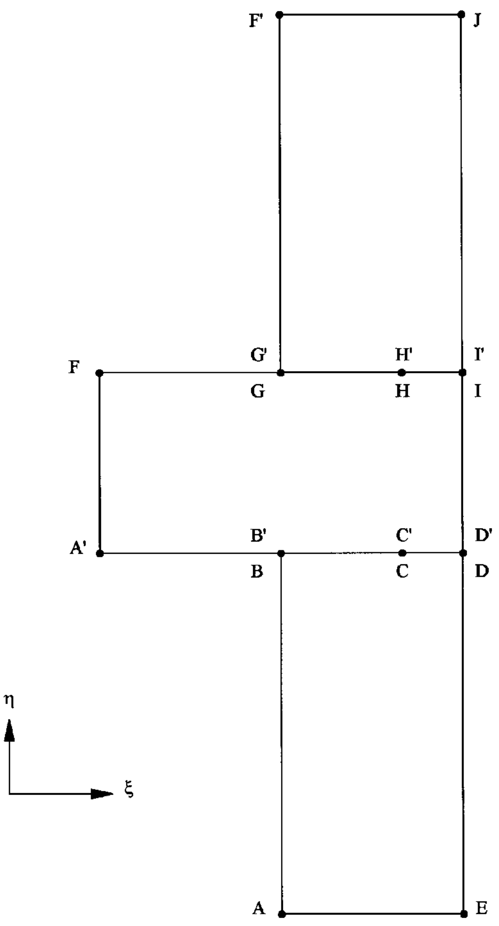

Fig. 3 Computational domain for the staggered biplane configuration.

Two grids were generated for this problem. For the coarse grid, the lower and upper blocks contained 51 × 51 nodes each, whereas the middle block conatined 101 × 26 nodes. Similarly, for the fine grid, the lower and upper blocks each had 101 × 101 nodes, whereas the middle block had 201 × 51 nodes. The sizes of these blocks were chosen in such a way so as to achieve a good load balancing when the problem is solved in parallel. Assuming the computational load of any block to be proportional to the total number of cells within that block, an effort has been made to keep the number of cells per block as close as possible.

Parallelization Approach

Before discussing the parallel implementation of the LUCI and the SSOR schemes, let us define the following: version I refers to the case when the two solution steps needed to advance the solution to the next time step are done serially using one processor, whereas version II denotes the case when these two steps are done in parallel using two processors. Because of the cycle dependency of these steps in the SSOR case, only version I is used. However, the LUCI scheme was parallelized using both versions because these two steps are independent of each other and thus can be implemented concurrently.

A. Version I

The steps used to parallelize the SSOR and the LUCI schemes using version I can be outlined as follows:

Setup of the Parallel Environment

The first step is to initialize the MPI environment and to establish communicators that describe the communication contexts and the associated groups of processors. The MPI_COMM_WORLD default communicator that defines one communication context and uses the set of all processors as its group is the one we used in this version.

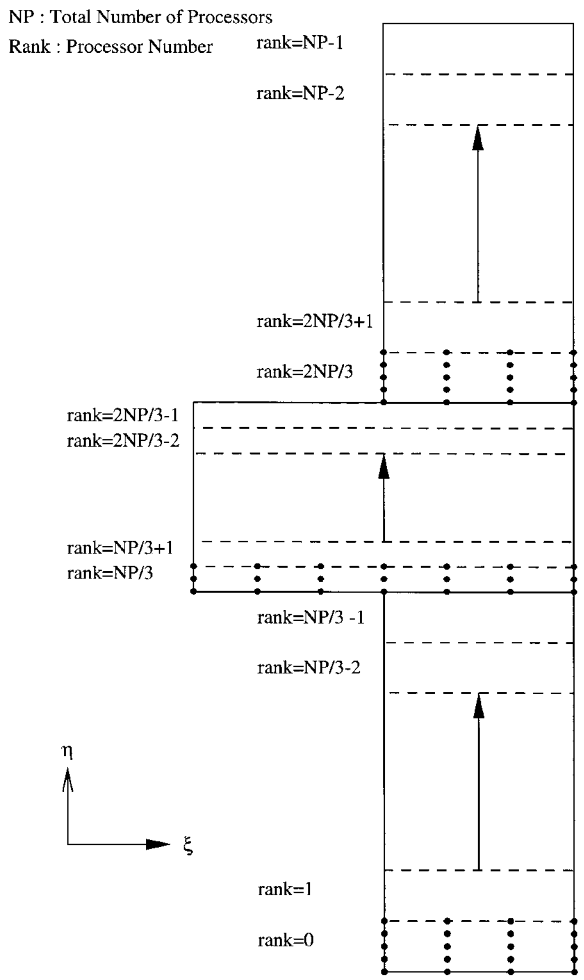

Fig. 4 One-dimensional domain decomposition on LACE.

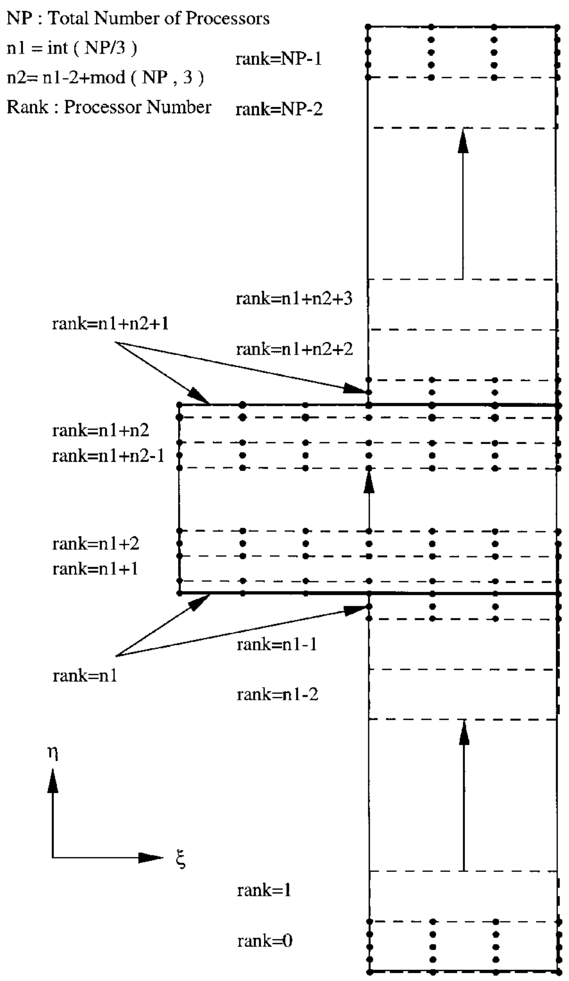

Fig. 5 One-dimensional domain decomposition on the Cray T3D.

Domain Decomposition

This is a very critical step in the design of any parallel program. A poor domain decomposition choice can cause disastrous parallel performance. Geometric decomposition achieved by splitting the problem into pieces along the main coordinate directions is a very popular method for domain decomposition.

Geometric decomposition can be easily done in one, two, or three dimensions. For the staggered biplane configuration we are solving, employing a two-dimensional domain decomposition approach or a one-dimensional approach along the ξ direction will complicate the communication routine and increase the number of messages (and hence latency costs) needed to exchange the boundary information. Therefore, we decided to use the one-dimensional domain decomposition approach in the η direction (see Figs. 4 and 5). As shown in Fig. 4, the computational domain for our test problem consists of three major sections. Each of these sections is split separately into smaller blocks. This is done to ensure that the interfaces for these sections would always be block boundaries and that they would not be interior to any block. This approach simplifies the boundary conditions application process as well as the solution procedure.

Mapping

This is one of the services that MPI offers to the programmer because the best or a good decomposition depends on the details of the underlying hardware. Mapping on LACE is straightforward. We just assign each block to a separate processor and the calculation proceeds. However, on the Cray T3D, there is a system restriction that the chosen number of processors has to always be a power of 2. This problem was solved by allowing an arbitrary number of blocks per processor (see Fig. 5). To get a good load balancing, the combined sum of the sizes of the blocks assigned to processors $n1$ or $n1 + n2 + 1$ has to be almost equal to that of any other block assigned to any other processor.

Grid Communication

The input/output (I/O) operations are not a part of the MPI standard yet, and therefore no standard (portable) MPI routines are available to distribute the data among the processors. To ensure maximum portability, we decided to do the grid communication using only standard MPI routines. An easy way to perform this step is to let the master processor read the whole grid file, extract the appropriate piece of data required by each processor, buffer it, and then send it in a direct message to that processor.

Boundary Information Exchange

To be able to solve each partitioned block separately, we need to update the solution at the boundaries of each block from the appropriate processors. This is done by exchanging the solution in a one-cell-deep layer of ghost cells. This exchange process is done after every iteration.

There are several modes of communication in MPI.[13] In this work, the data exchange at the block boundaries is implemented using the MPI_SENDRECV routine. This routine is a locally blocking one, which means that a send or a receive would not return until it is complete, and therefore a tight synchronization is achieved among the processors.

System Setup and Solution

The system of equations that needs to be solved is set up and solved separately for each block. The required two solution steps for both the SSOR and the LUCI schemes are done sequentially, and hence no communication is needed.

Convergence Check

To check for convergence, the value of the L_1 norm is monitored.

Output Collection

As mentioned previously in the grid communication step, the I/O operations are not a part of the MPI standard yet, and therefore standard MPI routines were used to perform this step in an exactly opposite way to the grid communication.

B. Version II

The basic steps undertaken to parallelize this version are very similar to those of version I. The details of some of these steps are somewhat different.

Setup of the Parallel Environment

To perform the two solution steps concurrently, the default MPI communicator is split into two new communicators, called LCOMM and UCOMM, each having half the total number of processors as its associated group.

System Setup and Solution

Processors within LCOMM perform the L step [solutions to Eqs. (9a) and (10a) of Ref. 8], whereas processors within UCOMM perform the U step [solutions to Eqs. (9b) and (10b) of Ref. 8] simultaneously. Afterward, a communication between LCOMM and UCOMM is needed where LCOMM sends the solution of the L step and receives that of the U step, whereas UCOMM sends the solution of the U step and receives that of the L step. Again, the communication routine used here is the MPI local blocking routine MPI_SENDRECV.

C. Time Measurement

To deal with time discrepancies resulting from other network traffic, we ran each case a few times and used an average of these times as our run time. The MPI_WTIME routine was used to actually perform the timings, and the values it returned were wall clock times.

Results

Figure 6 shows the pressure contours picture obtained using the LUCI scheme and the fine grid. Excellent shock capturing can be seen in this figure, which is yet one more evidence of the accuracy of the LUCI scheme.

To show the discrepancies in the execution time measurements on LACE, we show time measurements for the LUCI scheme in Fig. 7. In this figure, the deviations of the individual execution time measurements from the mean value are presented. It can be seen from this figure that these deviations are very insignificant (maximum around 0.004 s) and that we can consider our measured execution time values to be very reliable. On the Cray T3D, however, the system is set up in such a way to give exclusive access to the requested number of processors once the job starts running on them. Therefore, no discrepancies showed up in the time measurement there.

Figure 8 shows the variation of the execution time per iteration with the number of processors for both the LUCI and the SSOR

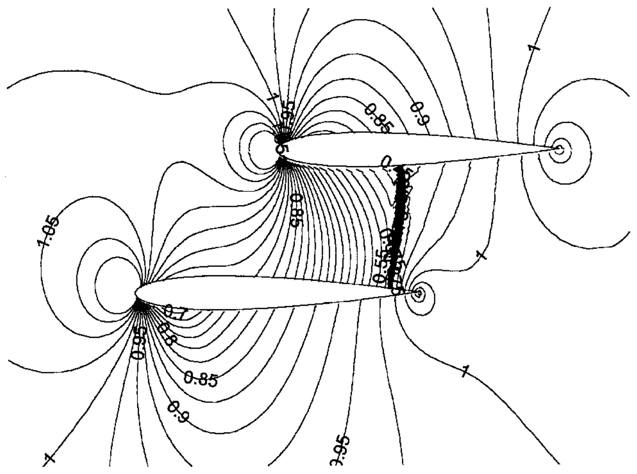

Fig. 6 Pressure contours for the staggered biplane configuration (LUCI scheme).

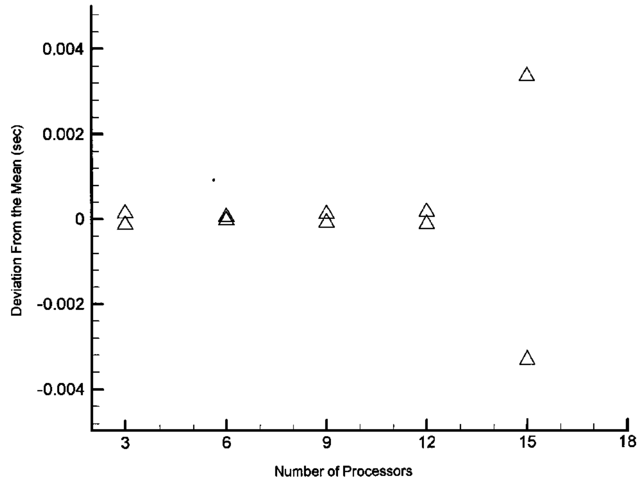

Fig. 7 Uncertainties in time measurements for the LUCI scheme (version I).

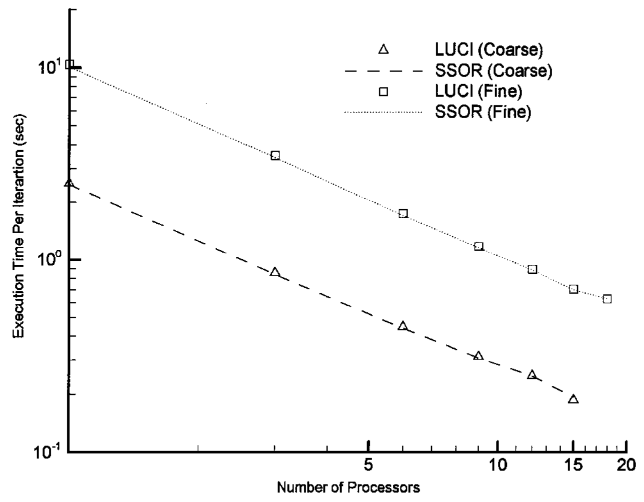

Fig. 8 Execution time for version I for different grid sizes (LACE).

schemes. Both the coarse and the fine grid results are shown in this figure. It can be seen from this figure that the execution time falls almost linearly in all of these cases with some sublinearity starting to show up at about 9 processors for the coarse grid and 12 processors for the fine grid. Furthermore, we can see that the execution times for the LUCI and the SSOR schemes are very comparable over the whole range of processors. This implies that the overhead in the LUCI scheme, due to the extra algebraic step needed to compute the overall solution, is of no practical importance, and this overhead diminishes even further as the number of processors increases.

An important measure of parallel performance is speedup[14] (execution time on a single processor divided by parallel execution time). Figure 9 shows the speedup for different grid sizes vs the number of processors for both the LUCI and the SSOR schemes on the LACE platform. A comparison with the ideal speedup is shown as well. This figure shows that the speedup degradation as the number of processors increases is worse for the coarse grid than for the fine grid because of the higher computation to communication ratio for the fine grid relative to the coarse grid. Another conclusion that can be drawn from Fig. 9 is that the speedup achieved with the LUCI scheme is slightly better than that achieved with the SSOR scheme. This again can be explained on the basis of the higher computation to communication ratio for the LUCI scheme relative to the SSOR scheme. The extra step needed in the LUCI scheme to get the overall solution adds a little bit more computation relative to the SSOR scheme, which improves the computation to communication ratio accordingly.

Another important measure of parallel performance is the parallelization efficiency[14] (speedup divided by the number of processors). Figure 10 shows the variation of this parallelization efficiency

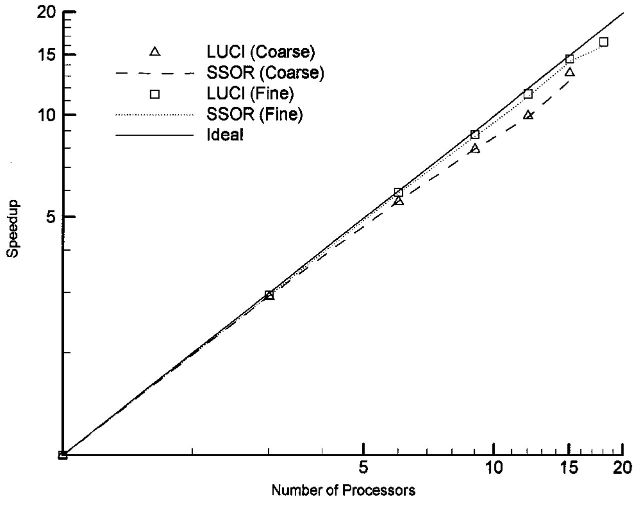

Fig. 9 Speedup for version I for different grid sizes (LACE).

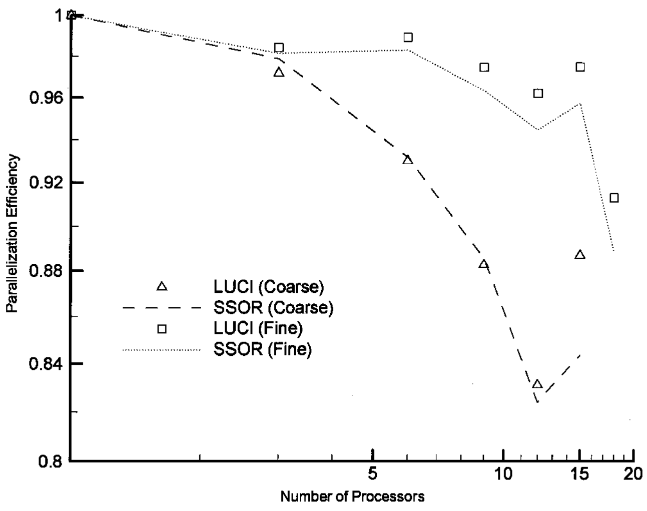

Fig. 10 Parallelization efficiency for version I for different grid sizes (LACE).

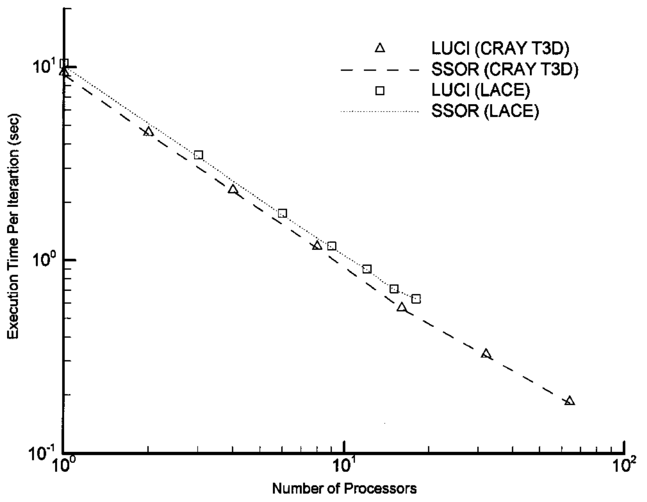

Fig. 11 Execution time for version I (fine grid).

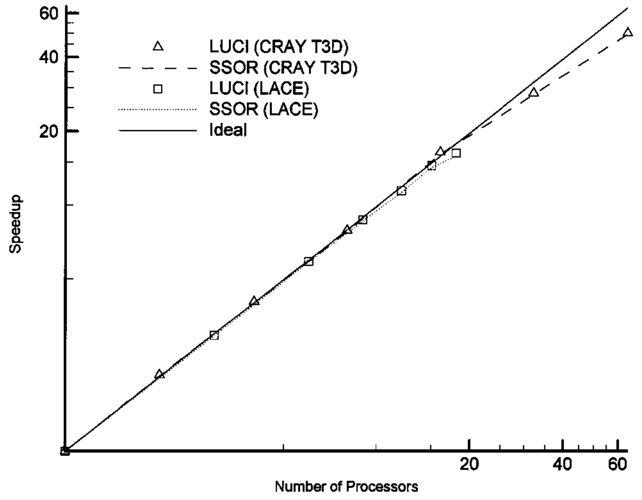

Fig. 12 Speedup for version I (fine grid).

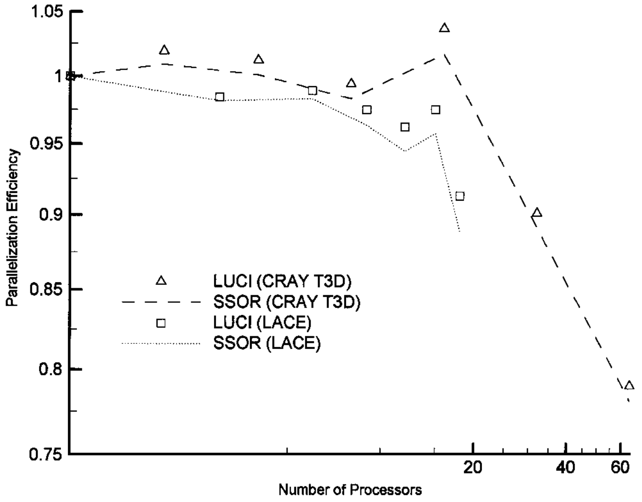

Fig. 13 Parallelization efficiency for version I (fine grid).

with the number of processors for the cases discussed in Fig. 9. The better speedup for the fine grid relative to the coarse grid and for the LUCI scheme relative to the SSOR scheme translates into better parallelization efficiencies, which can be seen in Fig. 10.

Figures 11–13 compare the execution time, speedup, and parallelization efficiency of both the LUCI and the SSOR schemes on LACE and the Cray T3D, respectively. On the Cray T3D, the slightly superlinear speedups in Fig. 12 and the slightly higher than one (maximum around 1.05) parallelization efficiencies in Fig. 13 are caused by heterogeneity resulting from operating within different levels in the memory hierarchy. The Cray T3D processors have only primary caches and have no secondary caches, contrary to LACE processors that are equipped with primary and large secondary caches. When the time measurements for the single processor were taken, the test problem did not fit within the primary cache of the T3D processor, and therefore most of the data accesses were to the main (slower) memory. However, as more processors are used, the effective cache size becomes larger, and more data accesses will be to the primary (faster) cache rather than the main memory. We can also see from Figs. 11–13 that the parallel performance of the LUCI and the SSOR schemes on the Cray T3D is better than that on LACE. The execution time is less due to the fast alpha chips on the T3D processors, the achieved speedup is higher, and so is the parallelization efficiency.

It is clear from the preceding discussion (Fig. 8) that the execution times per iteration for both schemes are comparable; however, the convergence deterioration characteristics of both schemes as the number of processors increases are different. Therefore, it is also important to compare the time to achieve a converged solution to a certain tolerance using the LUCI and the SSOR schemes. For the fine grid calculations on the Cray T3D, the execution times to convergence are given in Table 1.

It is obvious from this table that for a small to moderate number of processors on distributed memory machines the difference between the two execution times becomes smaller and smaller. This is consistent with the LUCI's small convergence deterioration as the number of processors increases compared with the SSOR scheme that is shown in Tables 2 and 3 of Part 1 of this paper.[8] For large numbers of processors, however, we do not have any data to compare the performance of the two schemes, but it is important to stress here the desirable stability advantages of the LUCI scheme.

Before we start talking about the results for the parallel implementation of version II, a few observations are in order. First, we did time our code on a single processor and found out that, out of the total CPU run time for the code, 60% of this time was used to compute the right-hand side (residual) of the system of equations that is to be solved, whereas 40% of the time was taken to actually solve this system. The LUCI scheme, in its basic form, introduces savings in the system solution step, which means savings in the 40% portion of the total run time. It is savings in this portion that we are interested in examining in this work. Of course, savings in the 60% portion are very significant and should be sought, but for the purposes of our current work only savings from decoupling the solution steps are considered.

Figures 14 and 15 compare the execution time and speedup of both versions I and II of the LUCI scheme on the Cray T3D, respectively.

Table 1 Total execution time (in hours) for the LUCI and the SSOR schemes

	Number of processors						
Scheme	1	2	4	8	16	32	64
LUCI	6.67	3.32	1.72	0.93	0.49	0.33	0.22
SSOR	4.91	2.51	1.33	0.75	0.42	0.29	0.19

All of these measures are plotted against the number of processors. These figures show that the parallel performance of version I is superior to that of version II. Version II needs more execution time and delivers less speedup. The reason for this behavior is that if we specify the same number of processors for both versions I and II, then version I will have twice as many blocks as version II has. This means that each processor in version II is working on a block that has double the size of a block that a processor in version I is working on. Even though we achieve some gains due to the decoupling of the solution steps, the increase in the block size offsets that which results in poorer performance. If savings in the set up phase mentioned in the previous paragraph were realized, then still there will be no way for the parallel performance of version II to match that of version I if the number of processors is used as the basis for comparison.

To be able to see the savings achieved by decoupling the solution steps, we replotted the modified parallel performance parameters (execution time, speedup, and parallelization efficiency) in Figs. 16–18, respectively, but this time we used the effective number of processors as the basis for comparison. The use of the word "modified" to describe the parallel performance parameters in these figures is merely to distinguish them from the traditional parallel performance parameters presented before. The effective number of processors is simply equal to the actual number of processors when version I is considered; however, it is equal to half that number for version II. When using this effective number of processors as the basis for comparison, the number of blocks for both versions I and II will be the same and so will the block size. Therefore, only effects

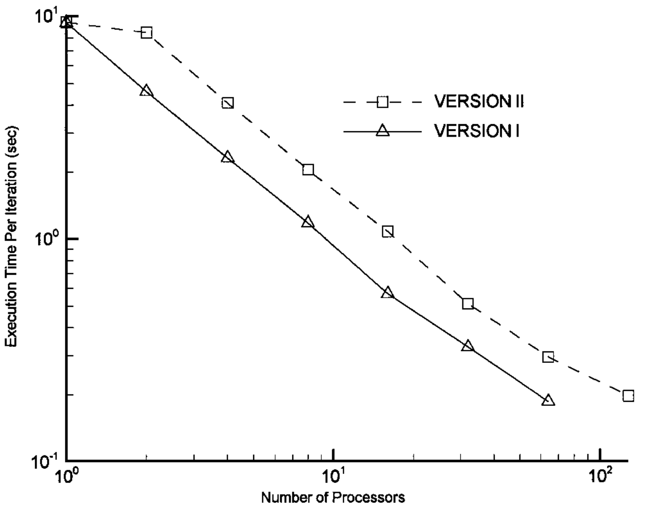

Fig. 14 Execution times for versions I and II of the LUCI scheme on the Cray T3D (fine grid).

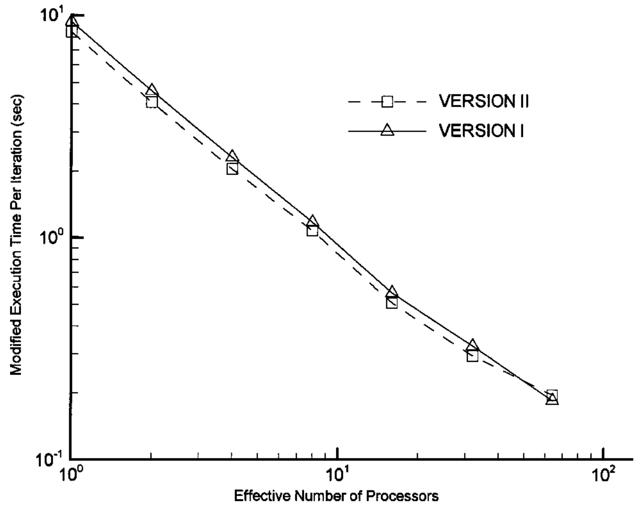

Fig. 16 Modified execution times for versions I and II of the LUCI scheme on the Cray T3D (fine grid).

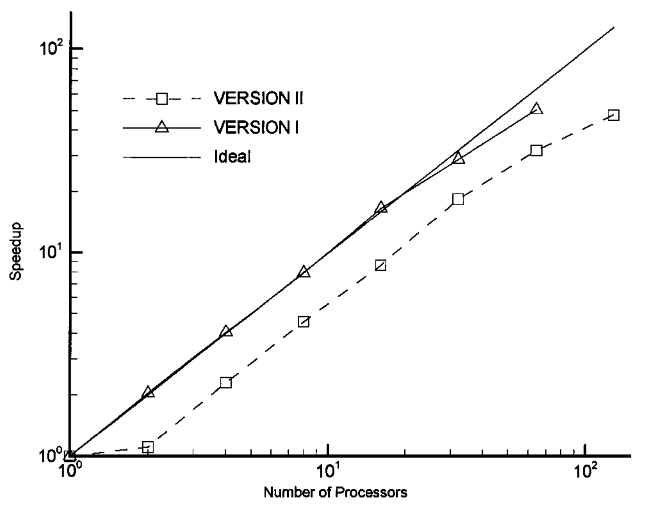

Fig. 15 Speedup for versions I and II of the LUCI scheme on the Cray T3D (fine grid).

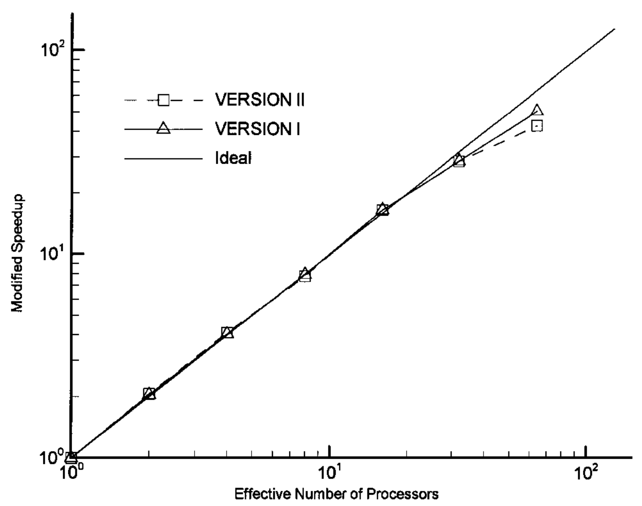

Fig. 17 Modified speedup for versions I and II of the LUCI scheme on the Cray T3D (fine grid).

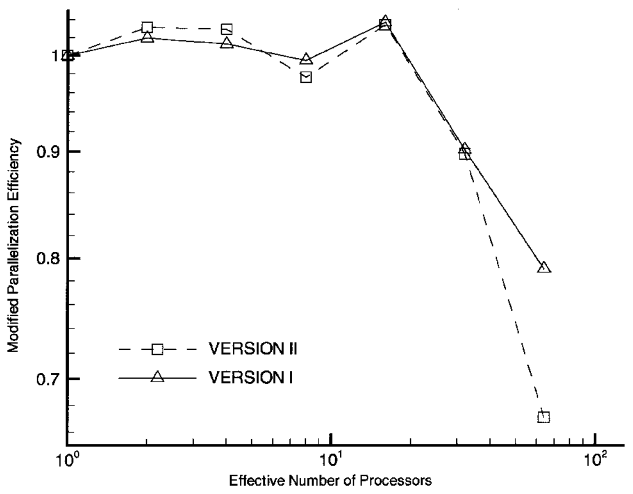

Fig. 18 Modified parallelization efficiency for versions I and II of the LUCI scheme on the Cray T3D (fine grid).

due to decoupling the solution steps will show up. We can now see the lower execution time and higher speedup of version II relative to version I. However, these improvements seem to become less and less significant as the number of processors increases. This is when communication will outweigh the savings achieved by decoupling the solution steps and the point of diminishing returns is reached.

Conclusions

A staggered biplane configuration employing a hybrid C-H-C multiblock grid was used to benchmark the LUCI and SSOR schemes. Parallel benchmarks using two grids (coarse and fine) were performed. The fine grid produced better parallel performance due to its higher computation to communication ratio relative to the coarse grid (maximum gains in speedup and parallelization efficiency are about 13% and 15 percentage points, respectively).

The LUCI scheme was implemented in parallel using two versions: version I, when both solution steps were solved on the same processor, and version II, when these two steps were done in parallel. However, only version I for the SSOR scheme was possible. In version I, the LUCI scheme slightly outperformed the SSOR scheme in terms of the speedup and the parallelization efficiency achieved (about 2 percentage points maximum gain in parallelization efficiency), whereas it required slightly more execution time (maximum slowdown around 1%) due to the algebraic step needed to compute the overall solution. When the parallel performances of the LUCI and the SSOR schemes on LACE and on the Cray T3D were compared, a relatively better performance on the Cray T3D was achieved (maximum gains in speedup and parallelization efficiency are about 7% and 8 percentage points, respectively).

When the parallel performances of versions I and II were compared, two interpretations were presented. If the number of processors was used as the basis for comparison, then version I would outperform version II. However, using the effective number of processors as the basis for comparison changed the story and showed a relatively better performance for version II. In the future, as the number of processors increases and the communication cost decreases, version II might become feasible to use. The reason for this will probably be that the stability and convergence characteristics of the employed implicit scheme in the massive range of number of processors will suffer so that no further use of extra processors can be made. Version II, which can use twice as many processors as version I can (for the same number of blocks), will be the target then.

Another interesting feature about the LUCI scheme is that it appears to us that it may be highly effective on any type of parallel computer where, at a minimum, groups of two processors share memory. On such machines the residual calculation reported earlier in the results section to take 60% of the computation time could be easily split between the L and U processors used in version II of the code because of the shared memory. In addition, the time needed to transfer the solution between the L and U processors could be reduced or even eliminated. Even on distributed memory machines, the residual calculation should be easy to parallelize because of the lack of dependencies; however, the time needed for data transfer between the L and U processors could be prohibitive. Machines where a small number of processors have shared memory are likely to become more and more prominent as multihead computers begin to appear on the mass market.

Acknowledgments

This work was supported in part by Ohio Supercomputer Center Grant PES532-1, NASA Grant NAG3-1472, and David Taylor Research Center Contract P873030B5651037.

References

[1]Henley, G. J., and Janus, J. M., "Parallelization and Convergence of a 3D Implicit Unsteady Turbomachinery Flow Code," *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992, pp. 238–245.

[2]Naik, V., Naik, N., and Nicoules, M., "Implicit CFD Applications on Message Passing Multiprocessor System," *Parallel Computational Fluid Dynamics Implementations and Results*, MIT Press, Cambridge, MA, 1992, pp. 97–125.

[3]Jespersen, D., and Levit, C., "A Multiple-Grid Navier–Stokes Code for the Connection Machine CM-2," *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1993, pp. 3–7.

[4]Ajmani, K., Liou, M. S., and Dyson, R. W., "Preconditioned Implicit Solvers for the Navier–Stokes Equations on Distributed-Memory Machines," AIAA Paper 94-0408, Jan. 1994.

[5]Warming, R. F., and Beam, R. M., "On the Construction and Application of Implicit Factored Schemes for Conservation Laws," *Proceedings of the Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics*, edited by H. B. Keller, American Mathematical Society, Providence, RI, 1978, pp. 85–129.

[6]Beam, R. M., and Warming, R. F., "An Implicit Finite-Difference Algorithms for Hyperbolic Systems in Conservation-Law Form," *Journal of Computational Physics*, Vol. 22, No. 1, 1976, pp. 87–110.

[7]Jameson, A., and Yoon, S., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," *AIAA Journal*, Vol. 25, No. 7, 1987, pp. 929–935.

[8]Ghizawi, N., and Abdallah, S., "Parallel Processing Scheme for the Navier–Stokes Equations, Part 1: Scheme Development," *AIAA Journal*, Vol. 36, No. 11, 1998, pp. 2013–2019.

[9]Sotiropoulos, F., and Abdallah, S., "A Primitive-Variable Method for the Solution of the Three-Dimensional Incompressible Viscous Flows," *Journal of Computational Physics*, Vol. 103, No. 2, 1992, pp. 336–349.

[10]Larson, L., Patel, V. C., and Gilbert, D. (eds.), *Ship Viscous Flow, Proceedings of 1990 SSPA Maritime Consulting AB-Chalmers University of Technology-Iowa Institute of Hydraulic Research Workshop*, FlowTECH International AB, Gothenburg, Sweden, 1990, pp. 8–16.

[11]Yaprak, E., Sanz, J., Poilen, G., and Holbert, R., "Lewis Advanced Cluster Environment Performance Study," *Proceedings of the SIMTEC '93—1993, International Simulation Technology Conference*, Society for Computer Simulation, San Diego, CA, 1993, pp. 23–28.

[12]Anderson, T. A., Culler, D. E., Patterson, D. A., and NOW Team, "A Case for NOW (Networks of Work Stations)," *IEEE MICRO*, Vol. 15, No. 1, 1995, pp. 54–64.

[13]Dongarra, J., Walker, D., and Lusk, E., "Message Passing Interface Forum, MPI: A Message Passing Interface Standard," *International Journal of Supercomputer Applications*, Vol. 8, Nos. 3, 4, 1994.

[14]Baker, L., and Smith, B. J., *Parallel Programming*, McGraw–Hill, New York, 1996, Chap. 6.

K. Kailasanath
Associate Editor