AIAA JOURNAL
Vol. 35, No. 12, December 1997

Parallelization and Dynamic Load Balancing of NPARC Codes

N. Gopalaswamy,* H. U. Akay,† A. Ecer,† and Y. P. Chien‡
Indiana University—Purdue University at Indianapolis, Indianapolis, Indiana 46202

Parallelization and dynamic load balancing of the two-dimensional and three-dimensional NPARC flow codes are presented. A previously developed parallel database package, GPAR, and a dynamic load balancer program, DLB, are used for both the two-dimensional and three-dimensional versions of NPARC. Performance characteristics of the implemented algorithms in two-dimensional and three-dimensional internal flow configurations are explored. Dynamic load balancing studies are carried out with the two parallel codes for an engine inlet configuration. The benchmark cases consist of a two-dimensional case with 4592 grid points and 2 three-dimensional cases, one with 50,950 grid points and the other with 240,000 grid points. The grids are decomposed into solution blocks and parallel computations are carried out with varying number of processors. The pressure response to unsteady perturbations of the inlet temperature is calculated using a variable time-stepping approach specifically developed for parallel computations, which takes into account the time-step variations in blocks with selective communication between the blocks. Time accuracy is maintained with the benefits of increased speedup. Load balancing is effective in large cases in which block computation costs are more dominant than communication costs.

Nomenclature

A_k	= contravariant speed-of-sound component
B	= total number of blocks
C	= Courant number
c	= computation cost for a block
E_p	= efficiency; Eq. (26)
I	= communication cost for a block
K_k	= metrics of coordinate transformation
m	= block time-step multiplier; Eq. (7)
N	= number of grid points
P	= total number of processors
Rc	= inlet cowl-tip radius used for nondimensionalizing lengths
Re	= Reynolds number
S_p	= speedup; Eq. (25)
TC	= total cost of parallel computation
t	= time
U_k	= contravariant velocity component
W	= time spent waiting for interface data by a block
(x, y, z)	= physical coordinates
Δt	= time step
μ	= coefficient of viscosity
ρ	= nondimensionalized density
(ξ, η, ζ)	= computational coordinates

Subscripts

b	= block
bi	= block to interface communication
g	= global minimum
i	= interface
ib	= interface to block communication
j	= index for block number
k	= index for coordinate direction
p	= index for processor number

Presented as Paper 96-3302 at the AIAA/ASME/SAE/ASEE 32nd Joint Propulsion Conference, Lake Buena Vista, FL, July 1–3, 1996; received Dec. 9, 1996; revision received Aug. 18, 1997; accepted for publication Aug. 18, 1997.

*Graduate Research Assistant, Computational Fluid Dynamics Laboratory, Department of Mechanical Engineering, Purdue School of Engineering and Technology.

†Professor, Computational Fluid Dynamics Laboratory, Department of Mechanical Engineering, Purdue School of Engineering and Technology.

‡Associate Professor, Computational Fluid Dynamics Laboratory, Department of Electrical Engineering, Purdue School of Engineering and Technology.

Introduction

OUR current research efforts are aimed at achieving an efficient computing paradigm on parallel computers. Parallel computers can be of many types, including multiple instruction multiple data (MIMD) and single instruction multiple data (SIMD) computers, though our attention is focused primarily on MIMD computers. The codes chosen for parallelization are the two-dimensional and three-dimensional NPARC flow codes, which are supported through a partnership between the NASA Lewis Research Center (LeRC) and the Arnold Engineering Development Center.[1] Each code lends itself easily to parallelization by the method of domain decomposition. A software package called GPAR,[2] developed earlier in conjunction with a dynamic load balancer called DLB,[3] was used for parallelization. The conservation laws of fluid mechanics are solved by NPARC in strong conservation form after transformation to computational coordinates. Although both implicit and explicit flow-solution algorithms exist, the explicit Runge–Kutta time-integration scheme is used for solution of unsteady flows. We show parallelization of the explicit solver of the NPARC codes for unsteady flows.

The problem to be solved over a given domain is parallelized by dividing the domain into many subdomains, called blocks, and solving the governing equations over these blocks. The blocks are connected to each other through the interblock boundaries, called interfaces. The NPARC codes (two-dimensional and three-dimensional) already use a block-structured solution approach. It is only necessary to write the interface communication part, called the Interface Solver to implement parallelization. The interior point algorithm, which operates on the nodes inside the block, is termed the Block Solver and is essentially unmodified. This block-structured approach is highly suited for parallel computing because each block may represent a separate process and can be assigned to different processors. Furthermore, each block can possess its own set of parameters that more selectively describe the flowfield within the block, instead of using a global set of parameters applicable over the whole domain. For instance, blocks far from no-slip surfaces, in the absence of freestream turbulence, can be modeled adequately with the Euler equations, without the additional expense of computing the viscous terms. Such an approach has been used in the parallelization of the NPARC codes in conjunction with an improved communication strategy.

During block-structured parallel computations, often bottlenecks occur because of communication needed between the blocks over the network of processors. We expect that, over the coming years, the capability to solve larger problems with faster processors also

will increase. The communication cost may become more critical because the network speeds may not keep up with the processor speeds. Another important objective is the identification and optimization of communication cost in a heterogeneous environment, namely the existence of computers with different memory size and different processor speeds. The following sections describe the variable time-stepping approaches and load-balancing algorithm used to implement efficient execution of the NPARC codes.

Variable Time-Stepping Approach

For unsteady flow cases, a variable time-stepping method can be defined in terms of the two main modules, blocks and interfaces. Figure 1 illustrates the relationship between the two in a parallel environment for an example problem with two blocks and one pair of interfaces. In general,

$$\Delta t_b(1) \neq \Delta t_b(2) \tag{1}$$

$$\Delta t_i(1,2) \neq \Delta t_i(2,1) \tag{2}$$

Blocks

For computing unsteady flows, the default NPARC explicit solver chooses the most restrictive time step among all blocks, Δt_g, and all blocks proceed with this time step:

$$\Delta t_g = \min_j \left(\frac{C}{\max_{n,k} \left[|U_k| + A_k + (2/Re)\,(\mu/\rho)|K_k|^2 \right]} \right)$$

$$n = 1, 2, \ldots, N_b, \qquad k = 1, 2, 3, \qquad j = 1, 2, \ldots, B \tag{3}$$

Communication occurs between the interfaces with a frequency of Δt_g. From Fig. 1,

$$\Delta t_i(1,2) = \Delta t_i(2,1) = \Delta t_g \tag{4}$$

$$\Delta t_{\mathrm{ib}}(1) = \Delta t_{\mathrm{bi}}(1) = \Delta t_{\mathrm{ib}}(2) = \Delta t_{\mathrm{bi}}(2) = \Delta t_g \tag{5}$$

In the variable time-stepping approach applied to blocks,[4,5] for each block j, a time-step $\Delta t_b(j)$ is picked as a multiple of Δt_g:

$$\Delta t_b(j) = m_j \Delta t_g, \qquad j = 1, 2, \ldots, B \tag{6}$$

$$m_j = \min \left(5, \frac{C}{\max_{n,k} \left[|U_k| + A_k + (2/Re)(\mu/\rho)|K_k|^2 \right]} \right)$$

$$n = 1, 2, \ldots, N_b, \qquad k = 1, 2, 3 \tag{7}$$

Communication still occurs according to Eqs. (4) and (5). Hence, each interface needs to interpolate the solution variables at each Δt_g before sending them to its neighbor. The number 5 in Eq. (7) serves as an upper limit to minimize the interpolation errors for the time-accurate solution of unsteady flows.

Interfaces

The block-based variable time-stepping approach outlined above reduces computation time for the blocks. However, reducing communication is also very important since it can contribute significantly to the overall computation time. In block-based variable time-stepping each block marches with its own time-step $\Delta t_b(j)$ and stores the boundary information into its interface. However, it is not necessary for each interface to communicate with the neighbor and update its parent block every Δt_g. With reference to Figure 1, the following three communication schemes were investigated with a decreasing level of communication frequency.

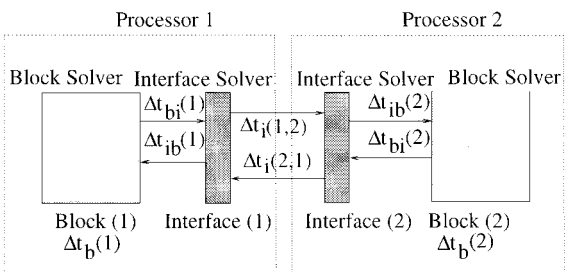

Fig. 1 Example of parallel computation with blocks and interfaces.

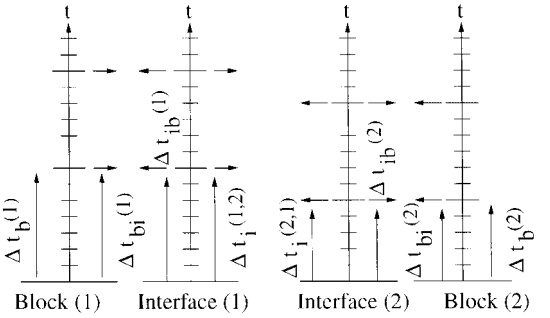

Fig. 2 Communication according to scheme 1 (minor tick marks denote Δt_g).

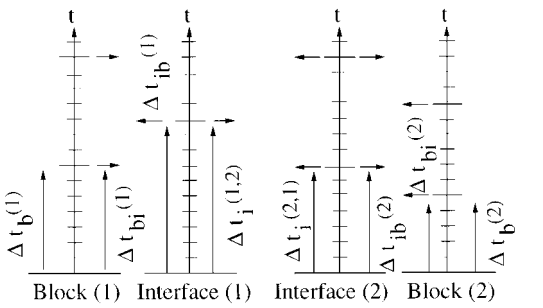

Fig. 3 Communication according to scheme 2 (minor tick marks denote Δt_g).

Scheme 1: Same Time Steps for Blocks and Interfaces

The block time step and the communication and update frequency of the interfaces belonging to the parent block are chosen to be identical. For instance, in Fig. 2,

$$\Delta t_i(1,2) = \Delta t_{\mathrm{ib}}(1) = \Delta t_{\mathrm{bi}}(1) = \Delta t_b(1) = m_1 \Delta t_g \tag{8}$$

$$\Delta t_i(2,1) = \Delta t_{\mathrm{ib}}(2) = \Delta t_{\mathrm{bi}}(2) = \Delta t_b(2) = m_2 \Delta t_g \tag{9}$$

Let both blocks start at time-level t. After one solution step, block 1 advances to time-level $t + m_1 \Delta t_g$ and block 2 advances to time-level $t + m_2 \Delta t_g$. If $m_1 > m_2$, then, for block 2 to advance from time-level $t + m_2 \Delta t_g$, it needs the boundary conditions at that time level. Interface 2 supplies the boundary conditions by linearly interpolating the solution received from interface 1 at time levels t and $t + m_1 \Delta t_g$. Similarly, block 1 waits for block 2 to advance to a time level greater than or equal to $t + m_1 \Delta t_g$ and send interface solution values to interface 1 before proceeding.

Scheme 2: Different Time Steps for Blocks and Interfaces

The partial differential equations of fluid mechanics are usually very stiff. The time step used is a function of the maximum eigenvalue of the linearized system of equations and is quite small to satisfy the numerical stability requirements. Because this time step also allows the integration of the complete eigenvalue spectrum of the linearized equations, accuracy requirements also can usually be satisfied. This observation, coupled with the frequently encountered scenario that the block time step is decided by a relatively small region of the block, allows us to propose a scheme in which the interfaces need to be updated less frequently, for example, for a time interval corresponding to the maximum stable time step for the interface nodes. For interface 1, in Fig. 3,

$$\Delta t_i(1,2) = \frac{C}{\max_{n,k} \left[|U_k| + A_k + (2/Re)\,(\mu/\rho)|K_k|^2 \right]}$$

$$n = 1, 2, \ldots, N_i(1), \qquad k = 1, 2, 3 \tag{10}$$

Similarly, in interface 2, a time-step $\Delta t_i(2, 1)$ is calculated. Also, the following relationships hold:

$$\Delta t_b(1) = m_1 \Delta t_g \tag{11}$$

$$\Delta t_{\text{bi}}(1) = \Delta t_{\text{ib}}(1) = \Delta t_i(1, 2) \tag{12}$$

$$\Delta t_b(2) = m_2 \Delta t_g \tag{13}$$

$$\Delta t_{\text{bi}}(2) = \Delta t_{\text{ib}}(2) = \Delta t_i(2, 1) \tag{14}$$

In general, even for matching and overlapping interfaces, $\Delta t_i(1, 2) \neq \Delta t_i(2, 1)$ because the metrics K_k calculated for one interface may not be the same as the other. As before, linear interpolation of the solution values in the interface is carried out by the receiving interface to yield the solution values at the interface time level. This solution becomes the boundary condition of the parent block of the receiving interface, which enables it to advance in time. Thus, this approach leads to reduced communication compared to scheme 1. For a particular block j, all interfaces belonging to that block obey the following relation:

$$\Delta t_i(j, n) \geq \Delta t_b(j) \tag{15}$$

where $\Delta t_i(j, n)$ is the interface communication time step from an interface belonging to block j to a neighboring interface n.

Scheme 3: Interface Time Steps Based on Interface Characteristic Speeds

A communication frequency based on the local characteristic speeds in the interface region also is proposed. This scheme is primarily of use in Euler/inviscid flows. Figure 4 shows a simple one-dimensional example of the above discussion. As can be seen from Fig. 4, if a block is completely supersonic, then from the direction of the characteristics or eigenvalues, it is not necessary for the downstream block to communicate with the upstream block. However, the upstream block must necessarily send information downstream. With reference to Fig. 4, if the blocks are subsonic, then the communication frequency between the blocks would be obtained as follows:

$$\Delta t_i(1, 2) = \frac{C}{\max_n[|U_1| + A_1]}, \qquad n = 1, 2, \ldots, N_i(1) \tag{16}$$

$$\Delta t_i(2, 1) = \frac{C}{\max_n[-|U_1| + A_1]}, \qquad n = 1, 2, \ldots, N_i(2) \tag{17}$$

Hence, the following relationships between the time steps hold for this scheme:

$$\Delta t_b(1) = m_1 \Delta t_g \tag{18}$$

$$\Delta t_{\text{bi}}(1) = \Delta t_i(1, 2) \tag{19}$$

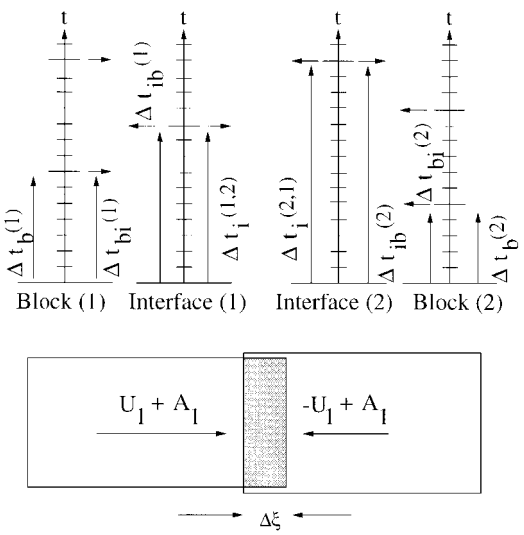

Fig. 4 Communication according to scheme 3 (minor tick marks denote Δt_g).

$$\Delta t_{\text{ib}}(1) = \Delta t_i(2, 1) \tag{20}$$

$$\Delta t_b(2) = m_2 \Delta t_g \tag{21}$$

$$\Delta t_{\text{bi}}(2) = \frac{C}{\max_{n,k}[|U_k| + A_k]}$$

$$n = 1, 2, \ldots, N_i(2), \qquad k = 1, 2, 3 \tag{22}$$

$$\Delta t_{\text{ib}}(2) = \Delta t_i(1, 2) \tag{23}$$

If the interfaces are supersonic, then $\Delta t_i(2, 1) \to \infty$ and there would be a significant reduction in the communication. As before, if the time levels in the two interfaces do not match during communication, a linear interpolation is carried out by the receiving interface from the solution variables received.

For multidimensional flows, we can extend the above reasoning by considering each direction separately. The physical coordinates (x, y, z) map to the computational coordinates (ξ, η, ζ). Because the interfaces usually are aligned along a constant index or computational coordinate, the contravariant velocities (U_1, U_2, U_3) along each computational coordinate direction should be considered. Along each interface, only one contravariant velocity will exist in a direction crossing the interface; the other two are parallel to the interface direction because they are mutually orthogonal to each other. For example, if there is an interface along a constant ξ direction, only U_1 exists in a direction crossing the interface; the other two contravariant velocities U_2 and U_3 are parallel to this interface. The same approach is used for interfaces aligned along constant η and constant ζ directions.

Load Balancing

Following this discussion, the objective is to reduce both the computation and the communication costs by making parallel computing optimally suited to local conditions. Apart from the algorithmic considerations, one also needs to consider the performance of the overall computation itself in terms of the processor speeds and communication speeds. Bottlenecks also can arise because of the computational load of the processors and communication times between the processors. Obtaining maximum efficiency leads to the necessity of load balancing, or balancing the computational load on each processor during the execution. In many cases, it is advantageous to decompose the problem into more blocks than the number of machines available because the load balancing can be used to alleviate bottlenecks due to a portion of the domain, or the processor itself. As an example, if only 10 processors are available to solve a three-dimensional wing-section problem, it may be advantageous to decompose it into 30 blocks or so.

The load-balancing program needs statistics about the execution of the application code in terms of the computational and communication costs for each block on every processor and also the number of extraneous processes on those processors. This then is factored into calculating a cost function. For example, the cost of computation can vary because of a change in the solution algorithm or because of an increase in extraneous processes started or stopped by other users. The response to the two causes is different. A program called Ptrack (process tracker) executes concurrently on each processor on which the blocks are executing, and monitors the extraneous process load on the respective processors.[3] The execution scenario is illustrated in Fig. 5.

First, the blocks and interfaces are initially allocated to the machines (or processors) on the basis of block and interface sizes, proximity of blocks and interfaces, speed of the individual machines, and the communication speed and bandwidth of the network. In dynamic load balancing, this initial distribution can change according to external factors such as extraneous processes added to or removed from the machines during computation and also because of changes in computational speed of the blocks and interfaces themselves on the processors due to changes in the solution algorithm or the transient solution behavior.

The load-balancing scheme developed by our team is based on the greedy algorithm,[3] which tries to minimize the total cost of executing all blocks. Formulation of the cost function can be described as follows.

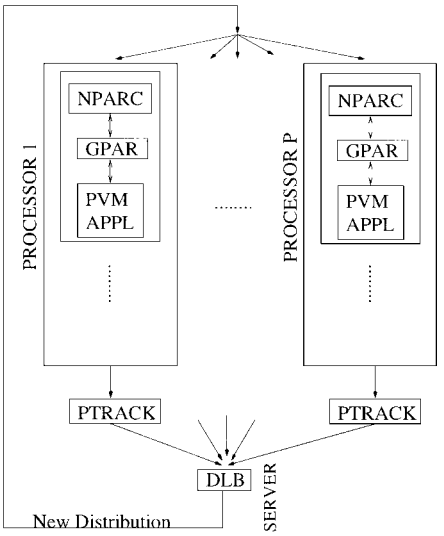

Fig. 5 Balancing of the NPARC code.

1) Let the computation cost of processing block j on a processor p be c_j^p. Here, j can take values from 1 to B, and p can take values from 1 to p.

2) Let the communication cost of sending interface data from an interface on processor p to its neighboring interface be I^{pq}, per block j on processor p, which may be on a different processor q.

3) The computation cost of executing blocks on a computer p also is influenced by the waiting time W_j for each block j because it has to wait to receive the interface information. The total cost of computation on a processor p is

$$TC^p = \sum_{j=1}^{B} \left(c_j^p + I_j^{pq} + W_j\right) \tag{24}$$

The load distribution problem then reduces to minimizing the maximum of these computation costs among all of the processors because it is the slowest processor that is the bottleneck. Hence, if $TC = \max_p(TC^p)$, then TC should be minimized to achieve load balancing. The greedy algorithm is used to minimize TC, the computational work for that being equal to $\mathcal{O}(BP^2)$.

The computation and communication costs are computed by placing some time stamps inside the NPARC codes. On the basis of this information, the cost function TC^p for each processor is calculated, and the data blocks j are redistributed among the processors if necessary to balance the computational load. This process is done periodically during the execution of the code for every specified interval, called the balance cycle, to monitor the progress of the computation. Typical balance cycles are in the order of 100–500 time steps.

Test Cases Considered

All the grids for the following three test cases were supplied by NASA LeRC.[6] The Euler/inviscid version of the NPARC code was used to compute the test cases with the three-stage version of the pseudo-Runge–Kutta time-stepping scheme. An axisymmetric engine inlet is subjected to an incoming Mach number of 2.5 and a subsonic compressor-face Mach number of 0.3. The compressor-face boundary condition is based on a scheme developed previously for NPARC.[7] The reference inlet pressure is 117.8 lb/ft^2, and the reference inlet temperature is 395°R. The cowl-tip radius of the inlet, $Rc = 18.61$ in., is used to nondimensionalize the lengths. The three test cases correspond to different grids for the same inlet and flow conditions.

First, a steady-state solution is obtained for each of the three cases for the above conditions, which yields a normal shock in the diverging section of the inlet. Then, a sinusoidal temperature perturbation is introduced upstream of the inlet for all three cases. The amplitude of the sinusoidal temperature perturbation is 5% of the mean value (395°R), with a frequency of 225 Hz. The unsteady pressure response is measured at two locations, $x/Rc = 4.08$ and 5.01, downstream of the normal shock. These test cases were used previously to validate the accuracy of the parallel NPARC codes for NASA LeRC.

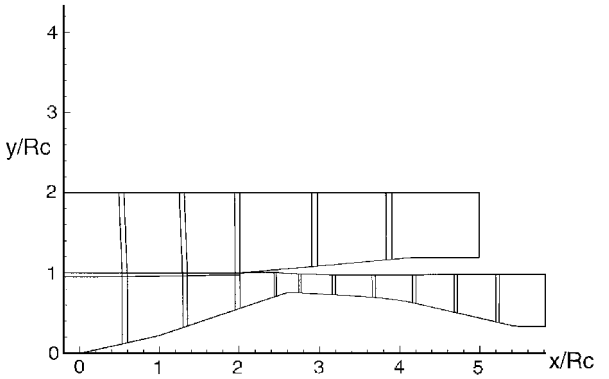

Fig. 6 Grid division into 17 blocks for two-dimensional test case.

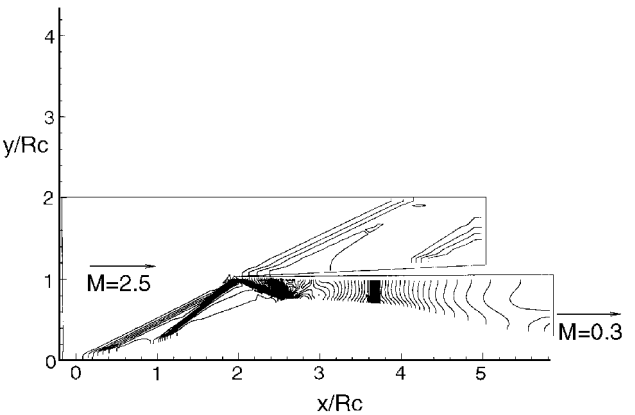

Fig. 7 Density contours of steady-state solution for two-dimensional test case.

Case 1

A two-dimensional inviscid case with 4592 grid points is divided into 17 blocks, and the number of machines or processors is varied from 1 to 8. The blocks contain approximately the same number of grid points, and each block has an evenly spaced grid. The grid is shown in Fig. 6 and the steady-state solution in the form of density contours is shown in Fig. 7. The axisymmetric version of the NPARC code is used to solve this test case.

Case 2

A three-dimensional inviscid case with 50,950 grid points corresponding to a 60-deg sector of the axisymmetric inlet is divided into 16 blocks of approximately the same size, and the three-dimensional version of the NPARC code is used to solve this test case.

Case 3

A three-dimensional inviscid case with 240,000 grid points corresponding to a 360-deg O-grid of the axisymmetric inlet was divided into 20 approximately equal-sized blocks. The three-dimensional version of the NPARC code is used to solve this test case.

For each of the preceding test cases, the following strategies were considered to investigate the performance of the new algorithms.

Default Scheme

The base case is chosen to be global time stepping with the same time step for all of the blocks. A time step of 6 μs is chosen as the global time step and computations are performed for approximately 5000 steps until a periodic response is achieved. This corresponds to an interval of approximately 0.03 s. Then, the same case is run for all three grids and, this time, dynamic load balancing is enabled. The block distribution after load balancing and the resultant elapsed time and CPU time are recorded.

Scheme 1

This time, the variable time-stepping option is enabled, in which each block picks a certain multiple of the global time step, depending on the critical Courant number inside the block. The initial distribution of the blocks is the same as obtained from the preceding step with constant global time stepping. The interfaces communicate with their neighbors according to scheme 1. The elapsed time and the CPU time are recorded for this case with and without load balancing enabled. The time variation of the pressure response is plotted for the two monitoring stations and compared with the base case.

Scheme 2

Variable time stepping in addition to interface communication, which takes place at an interval corresponding to the critical Courant number for the interface nodes, is studied. The communication scheme as in scheme 2 is used. The elapsed time and the CPU time are recorded, and the pressure response is plotted with respect to time. Load balancing is enabled, and the same case is rerun with all parameters recorded.

Scheme 3

Finally, variable time stepping in addition to interface communication, which takes place according to the characteristic speeds of the solution variables in the interface nodes, is investigated. As before, all parameters are recorded for cases with and without load balancing.

Results

The three test cases were computed on an IBM SP2 computer with 7–10 processors available for dedicated runs. The network configuration used was regular 10-Mbps Ethernet between the processors. This simulates a setup of a cluster of workstations connected to each other through Ethernet. The timing information for the cases considered is presented in the form of speedup and efficiency, which are defined in the next two equations:

$$S_p = \frac{\text{elapsed time with 1 processor (default scheme)}}{\text{elapsed time with } p \text{ processors}} \tag{25}$$

$$E_p = S_p/p \tag{26}$$

The total elapsed time for solving the test case using the default communication scheme with one processor is used as a basis for comparison when speedup and efficiency are calculated. Because the default scheme is used as the basis for computing efficiencies, it is possible to obtain efficiencies greater than one for other schemes.

Case 1

The speedup and efficiency for case 1 are shown in Figs. 8 and 9, respectively. It is usually observed that if the parallel computation involves a lot of communication, as the number of processors increases there is a decrease in the speedup because any decrease in the computation time is eclipsed by the increase in communication. Clearly, it can be seen that this case is highly communication bound for the default scheme. For schemes 1 and 2, which involve a reduction in both the computation and the communication because

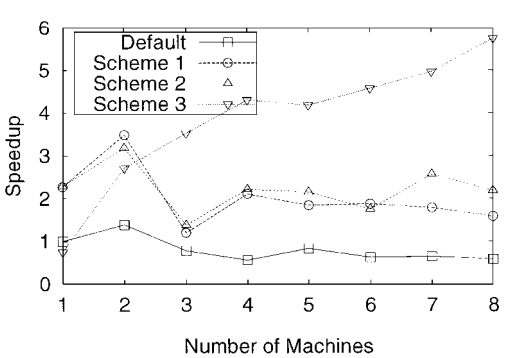

Fig. 8 Speedup for two-dimensional test case; speedup of various communication schemes.

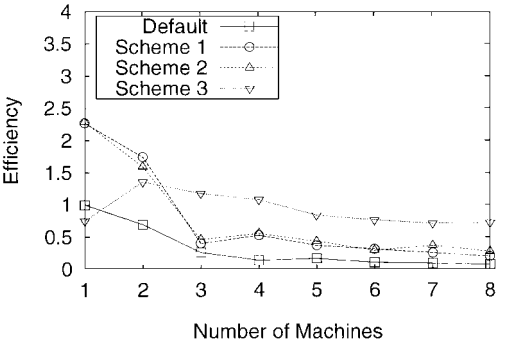

Fig. 9 Efficiency for two-dimensional test case; efficiencies of various communication schemes.

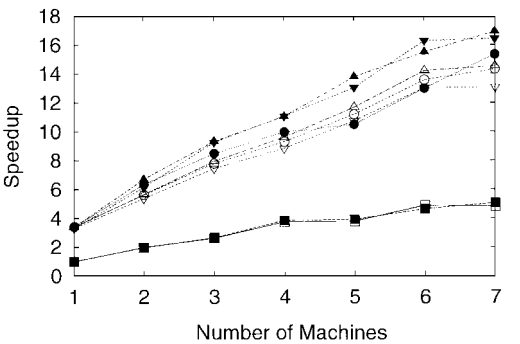

Fig. 10 Speedup for three-dimensional test case with 50,950 nodes; speedup for various communication schemes: □, default; ○, scheme 1; △, scheme 2; ▽, scheme 3; ■, default (DLB); ●, scheme 1 (DLB); ▲, scheme 2 (DLB); and ▼, scheme 3 (DLB).

of the variable time-stepping approaches, any gains in the speedup and efficiency are offset quickly by rising communication costs for more than two processors. For scheme 3, communication is reduced by almost 50% compared with scheme 2 because most of the interfaces are supersonic. Hence, because of this large reduction in communication, an almost linear increase in speedup is observed, and high efficiency is maintained. However, scheme 3 with one processor is found to perform worse than the default scheme. This is because of the overhead imposed by the necessity of calculating the time step in the interfaces according to Eqs. (16) and (17). Hence, there is actually a decrease in the speedup when computation occurs with one processor, but this overhead is more than compensated for by the resultant decrease in the communication cost with greater number of processors. In Fig. 9, one can see that efficiencies are greater than one for schemes 1 and 2 with one and two processors. This is because of the variable time-stepping approach used, which leads to a decrease in the computation time compared with the default scheme. Thus, with one processor, it can be seen that there is a decrease of more than two times the computation when compared with the default scheme, leading to efficiencies of more than two being obtained.

According to Ref. 3, the load balancer performs poorly when the communication costs are higher than the computation cost, and hence this case was not load balanced.

Case 2

The timing results for the three-dimensional inviscid test case shown in Figs. 10 and 11 show that this case is not highly communication bound like the two-dimensional test case. Because of the variable time-stepping approaches employed, many blocks have time steps ranging from three to four times the most restrictive time step Δt_g, which occurs in the block containing the normal shock. Thus, as can be seen in Fig. 10, it is possible to reduce computation by nearly three times for the single-processor case for all three variable time-stepping schemes when compared with the default scheme. Because this is an inviscid test case, no grid stretching is employed to cluster nodes near the walls, etc., and hence the block and interface time steps are not very different. Hence, schemes 2 and 3 yield an improvement of only a few percent compared with

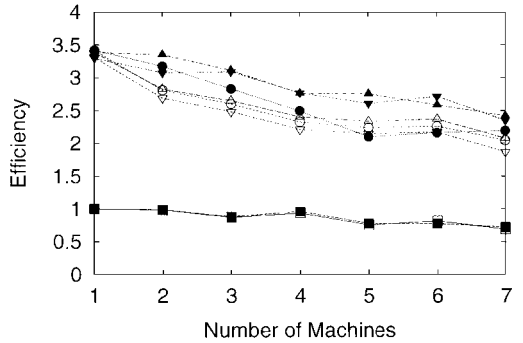

Fig. 11 Efficiency for three-dimensional test case with 50,950 nodes; efficiency of various communication schemes: □, default; ○, scheme 1; △, scheme 2; ▽, scheme 3; ■, default (DLB); ●, scheme 1 (DLB); ▲, scheme 2 (DLB); and ▼, scheme 3 (DLB).

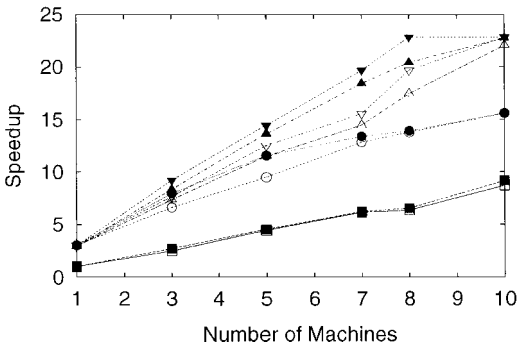

Fig. 12 Speedup for three-dimensional test case with 240,000 nodes; speedup for various communication schemes: □, default; ○, scheme 1; △, scheme 2; ▽, scheme 3; ■, default (DLB); ●, scheme 1 (DLB); ▲, scheme 2 (DLB); and ▼, scheme 3 (DLB).

scheme 1. Because the blocks are fairly evenly divided, the computation cost per block also is not much different. Therefore, load balancing eliminates a few bottlenecks caused by a change in the communication costs as different schemes are employed, and yields an additional improvement of approximately 15% for each scheme. In general, as the number of processors increases, load balancing yields smaller benefits because there are fewer blocks to balance on each processor. For the default scheme, load balancing produces negligible improvement because the blocks are approximately the same size and constant time stepping is used in all blocks.

Case 3

The timing results for the three-dimensional inviscid test case with 240,000 grid nodes in Figs. 12 and 13 show a behavior similar to those in Figs. 10 and 11. In this case, most of the blocks proceed with a time step that is approximately three times the most restrictive time step Δt_g. Hence, with a single processor, computation costs are reduced by three times for the variable time-stepping schemes compared with the default scheme, leading to efficiencies of three for the variable time-stepping schemes. For this test case, two blocks contain 3200 nodes; the rest of the blocks contain 3000 nodes. Load balancing alleviates the bottleneck due to these larger blocks and yields improvements of almost 25% for each of the three schemes as seen in Figs. 12 and 13. For the blocks in the subsonic region, a greater grid stretching exists when compared with the three-dimensional grid in case 2, resulting in a larger difference between the block and the interface time step in those blocks. Hence, when compared with scheme 1, schemes 2 and 3 perform better and yield higher speedup and efficiency. Also, for cases 2 and 3, the number of grid points in the interior of the domain increases faster than the number of grid points in the interfaces. Hence, for case 3 with a total of 240,000 grid points, approximately 30,000 grid points are present in all of the interfaces, whereas for case 2 with 50,950 grid points, approximately 7000 grid points are present in the interfaces. Hence, case 3 is less dominated by communication than case 2. The load-balancing algorithm performs better for cases in which communication is less dominant than computation.

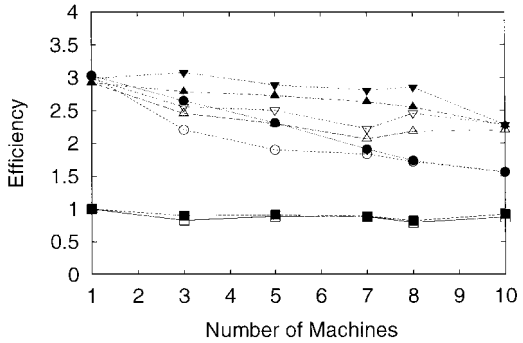

Fig. 13 Efficiency for the three-dimensional test case with 240,000 nodes; efficiency of various communication schemes: □, default; ○, scheme 1; △, scheme 2; ▽, scheme 3; ■, default (DLB); ●, scheme 1 (DLB); ▲, scheme 2 (DLB); and ▼, scheme 3 (DLB).

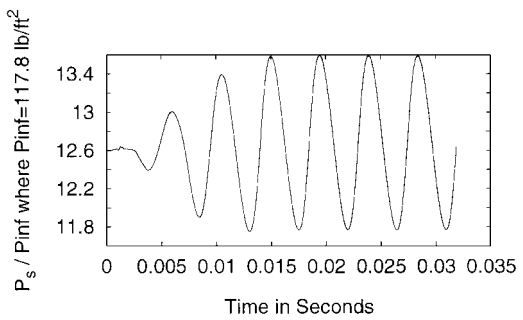

Fig. 14 Pressure response at $x/Rc = 4.08$ to a 5% sinusoidal inlet temperature perturbation: ——, default; – – –, scheme 1; - - - -, scheme 2; and ·····, scheme 3.

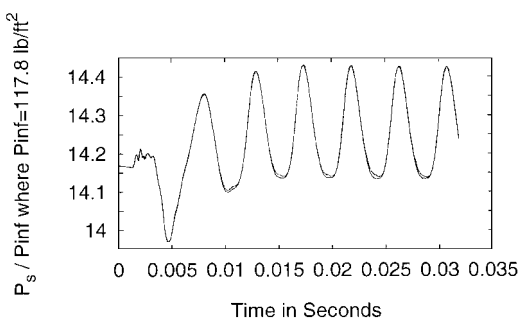

Fig. 15 Pressure response at $x/Rc = 5.01$ to a 5% sinusoidal inlet temperature perturbation: ——, default; – – –, scheme 1; - - - -, scheme 2; and ·····, scheme 3.

Next, the pressure response to the sinusoidal temperature perturbation is plotted for the two monitoring stations at $x/Rc = 4.08$ and 5.01 in Figs. 14 and 15, respectively. As can be seen, all three schemes preserve good time accuracy with respect to the globally uniform time-stepping case.

Conclusions

The two-dimensional/axisymmetric and three-dimensional versions of NPARC have been parallelized and enabled for dynamic load balancing. A variable time-stepping block-solution algorithm is implemented in addition to various communication schemes, and their efficiency is explored with the help of three test cases. The combination of the variable time-stepping approach and the communication schemes are shown to be time accurate for unsteady computations. These schemes also can be applied to other parallel solvers to improve their efficiency. Scheme 3 is primarily useful for inviscid flows where distinct characteristics exist. Scheme 2 is applicable to more general flow regimes, in viscous flow situations. Scheme 1 may be needed for complex flows where the time variation of the solution is very rapid. Significant savings in total elapsed time can be achieved with the developed variable time-stepping schemes

when the interface time steps and characteristic speeds are considered. The dynamic load balancing provides additional efficiency when the problem size increases. The variable time-stepping tools introduced here can significantly reduce the cost of solving unsteady perturbation problems with NPARC codes. The reduction in total elapsed times is four to five times that of a constant time-stepping algorithm when large size problems are solved. Further work is being done in increasing the Courant number inside the block and using an extended version of scheme 2 to reduce both computation and communication to yield high parallel efficiencies when compared to the default solution algorithm.

Acknowledgments

This research was funded by the NASA Lewis Research Center (LeRC) under NAG3-1577. The authors appreciate the support provided by Rich Blech, Gary Cole, and Joongkee Chung of the Computational Technologies Branch of NASA LeRC.

References

[1]Cooper, G. K., and Sirbaugh, J. R., "The PARC Code: Theory and Usage," Arnold Engineering Development Center, TR-89-15, Arnold AFB, TN, 1989.

[2]Akay, H. U., Blech, R. A., Ecer, A., Ercoskun, D., Kemle, B., Quealy, A., and Williams, A., "A Database Management System for Parallel Processing of CFD Algorithms," *Parallel CFD '92*, edited by R. B. Pelz, A. Ecer, and J. Haüser, Elsevier, Amsterdam, 1993, pp. 9–23.

[3]Chien, Y. P., Ecer, A., Akay, H. U., Carpenter, F., and Blech, R. A., "Dynamic Load Balancing on a Network of Workstations for Solving Computational Fluid Dynamics Problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 199, No. 1, 1994, pp. 17–33.

[4]Gopalaswamy, N., Chien, Y. P., Ecer, A., Akay, H. U., Blech, R. A., and Cole, G. L., "An Investigation of Load Balancing Strategies for CFD Applications on Parallel Computers," *Parallel CFD '95*, edited by A. Ecer, J. Périaux, N. Satofuka, and S. Taylor, 1995, pp. 703–710.

[5]Akay, H. U., and Ecer, A., "Efficiency Considerations for Explicit CFD Solvers on Parallel Computers," *Solution Techniques for Large-Scale CFD Problems*, edited by W. G. Habashi, Wiley, New York, 1995, pp. 3–25.

[6]Chung, J., and Cole, G. L., "Comparison of Compressor Face Boundary Conditions for Unsteady CFD Simulations of Supersonic Inlets," NASA TM 107194, 1996.

[7]Chung, J., "Numerical Solution of a Mixed Compression Supersonic Inlet Flow," AIAA Paper 94-0583, Jan. 1994.

P. R. Bandyopadhyay
Associate Editor