Distributed-Memory Computing With the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA)

Christopher J. Riley* and F. McNeil Cheatwood†
NASA Langley Research Center, Hampton, VA 23681

Paper Presented at the
4th NASA National Symposium on Large-Scale Analysis and Design on
High-Performance Computers and Workstations
Oct. 15–17, 1997/Williamsburg, VA

The Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA), a Navier-Stokes solver, has been modified for use in a parallel, distributed-memory environment using the Message-Passing Interface (MPI) standard. A standard domain decomposition strategy is used in which the computational domain is divided into subdomains with each subdomain assigned to a processor. Performance is examined on dedicated parallel machines and a network of desktop workstations. The effect of domain decomposition and frequency of boundary updates on performance and convergence is also examined for several realistic configurations and conditions typical of large-scale computational fluid dynamic analysis.

Introduction

The design of an aerospace vehicle for space transportation and exploration requires knowledge of the aerodynamic forces and heating along its trajectory. Experiments (both ground-test and flight) and computational fluid dynamic (CFD) solutions are currently used to provide this information. At high-altitude, high-velocity conditions that are characteristic of atmospheric reentry, CFD contributes significantly to the design because of the ability to duplicate flight conditions and to model high temperature effects. Unfortunately, CFD solutions of the hypersonic, viscous, reacting-gas flow over a complete vehicle are both CPU time and memory intensive even on the most powerful supercomputers; hence, the design role of CFD is generally limited to a few solutions along a vehicle's trajectory.

One CFD code that has been used extensively for the computation of hypersonic, viscous, reacting-gas flows over reentry vehicles is the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA).[1,2] LAURA has been used in the past to provide aerothermodynamic characteristics for a number of aerospace vehicles (e.g. AFE,[3] HL-20,[4] Shuttle Orbiter,[5] Mars Pathfinder,[6] SSTO Access to Space[7]) and is currently being used in the design and evaluation of blunt aerobraking configurations used in planetary exploration missions[8,9] and Reusable Launch Vehicle (RLV) concepts (e.g. the X-33[10,11] and X-34[12] programs). Although the LAURA computer code is continually being updated with new capabilities, it is a mature piece of software with numerous options and utilities that allow the user to tailor the code to a particular application.[13]

LAURA was originally developed and tuned for multiprocessor, vector computers with shared memory such as the CRAY C-90. Parallelism using LAURA is achieved through the use of macrotasking where large sections of code are executed in parallel on multiple processors. Because LAURA employs a point-implicit relaxation strategy that is free to use the latest available data from neighboring cells, the solution may evolve without the need to synchronize tasks. This results in a very efficient use of the multitasking capabilities of the supercomputer.[14] But future supercomputing may be performed on clusters of less powerful machines that offer a better price per performance than current large-scale vector systems. Parallel computers such as the IBM SP2 consist of large numbers of workstation-class processors with memory distributed among the processors instead of being shared. In addition, improvements in workstation processor and network speed and the availability of message-passing libraries allow networks of desktop workstations (that may sit idle during non-work hours) to be used for practical parallel computations.[15] As a result, many CFD codes are making the transition from serial to parallel computing.[16–20] The current shared-memory, macrotasking version of LAURA requires modification before exploiting these distributed-memory parallel computers and workstation clusters.

Several issues need to be addressed in creating a

*Research Engineer, Aerothermodynamics Branch, Aero- and Gas-Dynamics Division.

†Research Engineer, Vehicle Analysis Branch, Space Systems and Concepts Division.

distributed-memory version of LAURA: 1) There is the choice of programming paradigm to use. A domain decomposition strategy[17] (which involves dividing the computational domain into subdomains and assigning each to a processor) is a popular approach to massively parallel processing and is chosen due to its similarity to the current macrotasking version. 2) To minimize memory requirements, the current data structure of the macrotasking, shared-memory version is changed since each processor requires storage only for its own subdomain. 3) The choice of message-passing library (which processors use to explicitly exchange information) may impact portability and performance. 4) The frequency of boundary data exchanges between computational subdomains can influence (and may impede) convergence of a solution although the point-implicit nature of LAURA already allows asynchronous relaxation.[14] 5) There are also portability and performance concerns involved in designing a version of LAURA to run on different (cache-based and vector) architectures. 6) Finally, a distributed-memory, message-passing version of LAURA should retain all of the functionality, capabilities, utilities, and ease of use of the current shared-memory version.

This paper describes the modifications to LAURA that permit its use in a parallel, distributed-memory environment using the Message-Passing Interface (MPI) standard.[21] An earlier, elementary version of LAURA for perfect gas flows using the Parallel Virtual Machine (PVM) library[22] provides a guide for the current modifications.[23] Performance of the modified version of LAURA is examined on dedicated parallel machines (e.g. IBM SP2, SGI Origin 2000, SGI multiprocessor) as well as on a network of workstations (e.g. SGI R10000). Also, the effect of domain decomposition and frequency of boundary updates on performance and convergence is examined for several realistic configurations and conditions typical of large-scale CFD analysis.

LAURA

LAURA is a finite-volume, shock-capturing algorithm for the steady-state solution of inviscid or viscous, hypersonic flows on rectangularly ordered, structured grids. The upwind-biased inviscid flux is constructed using Roe's flux-difference splitting[24] and Harten's entropy fix[25] with second-order corrections based on Yee's symmetric total-variation-diminishing scheme (TVD).[26] Gas chemistry options include perfect gas, equilibrium air, and air in chemical and thermal nonequilibrium. More details of the algorithm can be found in Refs. 1, 2 and 13.

The point-implicit relaxation strategy is obtained by treating the variables at the local cell center L at the advanced iteration level and using the latest available data from neighboring cells. Thus, the governing relaxation equation is

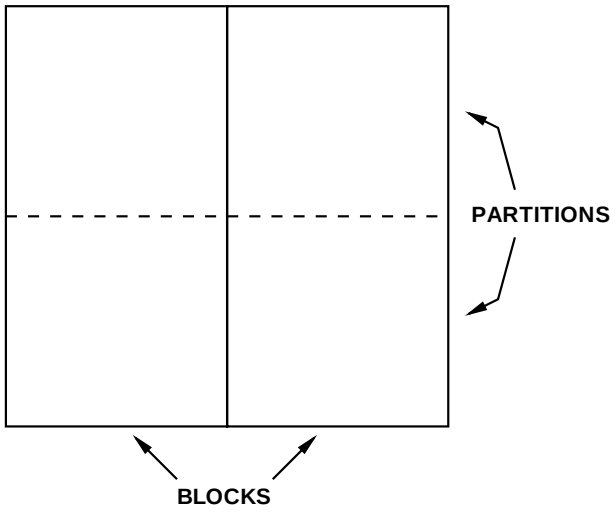


Fig. 1 Domain decomposition of macrotasking version.

$$\mathbf{M}_L \delta \mathbf{q}_L = \mathbf{r}_L \tag{1}$$

where $\mathbf{M}_L$ is the n x n point-implicit Jacobian, $\mathbf{q}_L$ is the vector of conserved variables, r_L is the residual vector, and n is the number of unknown variables. For a perfect gas and equilibrium air, n is equal to 5. For nonequilibrium chemistry, n is equal to 4 plus the number of constituent species. The residual vector $\mathbf{r}_L$ and the Jacobian $\mathbf{M}_L$ are evaluated using the latest available data. The change in conserved variables, $\mathbf{q}_L$, may be calculated using Gaussian elimination. An LU factorization of the Jacobian can be saved (frozen) over large blocks of iterations ($\approx$ 10 to 50) to reduce computational costs as the solution converges. However, the Jacobian will need to be updated every iteration early in the computation when the solution is changing rapidly.

Macrotasking

LAURA utilizes macrotasking by assigning pieces of the computational domain to individual tasks. First, the computational domain is divided into blocks, where a block is defined as a rectangularly ordered array of cells containing all or part of the solution domain. Then each block may be subdivided in the computational sweep direction into one or more partitions. Partitions are then separately assigned to a task (processor). Figure 1 shows a two-dimensional (2D) domain divided into 2 blocks with each block divided into 2 partitions. Thus a task may work on one or more partitions which may be contained in a single block or may overlap several blocks. Each task then gathers and distributes its data to a master copy of the solution which resides in shared memory. With the point-implicit relaxation, there is no need to synchronize tasks which results in a very efficient parallel

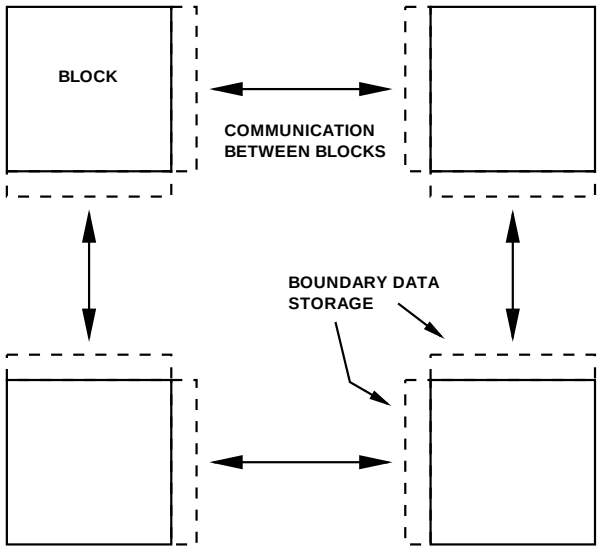


Fig. 2 Domain decomposition of message-passing version.

implementation.

Message-passing

In the new message-passing version of LAURA, the computational domain is again subdivided into blocks along any of the three (i, j, k) coordinate directions with each block assigned to a processor. As compared to the macrotasking version, this is analogous to defining each block to contain only one partition and assigning each partition to a separate task. The number of blocks is therefore equal to the total number of processors. Due to the distributed memory of the processors, each task only requires storage only for its own block plus storage for boundary data from as many as six neighboring blocks (i.e. one for each of the six block faces). Figure 2 shows a 2D domain divided equally into 4 separate blocks. Each processor works only on its own block and pauses at user-specified intervals to exchange boundary data with its neighbors. The boundary data exchange is explicitly handled with send and receive calls from the MPI message-passing library.[21] The MPI library was chosen because it is a standard and because there are multiple implementations that run on workstations as well as dedicated parallel machines.[27,28] Synchronization of tasks occurs when messages are exchanged, but this exchange is not required for any particular iteration due to the point-implicit relaxation scheme. As in the macrotasking version, tasks (or blocks) of various sizes may accumulate differing numbers of iterations during a run. For blocks of equal size, it may be convenient to synchronize the message exchange at specified iteration intervals.

Results

The performance of the distributed-memory, message-passing version of LAURA is examined in terms of computational speed and convergence. Measuring the elapsed wall clock time of the code on different machines estimates the communication overhead and message-passing efficiency of the code. The communication overhead associated with exchanging boundary data information between nodes depends on the parallel machine, the size of the problem, and the frequency of exchanges. The frequency of data exchanges may be decreased if necessary to reduce the communication penalty, but this may adversely affect convergence. Therefore, the impact of boundary data exchange frequency on convergence is determined for several realistic vehicles and flow conditions.

Computational Speed

Timing estimates using the message-passing version of LAURA are presented for an IBM SP2, an SGI Origin 2000, an SGI multiprocessor machine, and a network of SGI R10000 workstations. The single-node performance of LAURA on a cache-based (as opposed to vector) architecture is not addressed. Viscous, perfect gas computations are performed on the forebody of an X-33[10,11] configuration with a grid size of 64 x 56 x 64. The computational domain is split along each of the coordinate directions (depending on the number of nodes) into blocks of equal size. The individual block sizes are shown below. Because the blocks are equal in size, boundary data exchanges are synchronized at a specified iteration interval for convenience. Each run begins with a partially converged solution and is run for 200 iterations with second order accuracy. Two values (1 and 20) are used for *nexch*, the number of iterations between boundary data exchanges, to estimate the communication overhead on each machine. The number of iterations that the Jacobian is held fixed, *njcobian*, is equal to 20 and represents a typical value for solutions that are partially converged.

Table 1 Block sizes for timing study.

Nodes	Block
2	32 x 56 x 64
4	32 x 28 x 64
8	32 x 28 x 32
16	16 x 28 x 32
32	16 x 14 x 32
64	16 x 14 x 16
128	8 x 14 x 16

Four different architectures are used to obtain timing estimates. The first is a 160-node IBM SP2 located at the Numerical Aerospace Simulation (NAS) Facility at NASA Ames using IBM's implementation of MPI. The second is a 64 processor (R10000) SGI Origin 2000 also located at NAS using SGI's version of MPI. The third is a 12 processor (R10000) SGI machine operating in a multiuser environment, and the fourth is a network of SGI R10000 workstations connected by

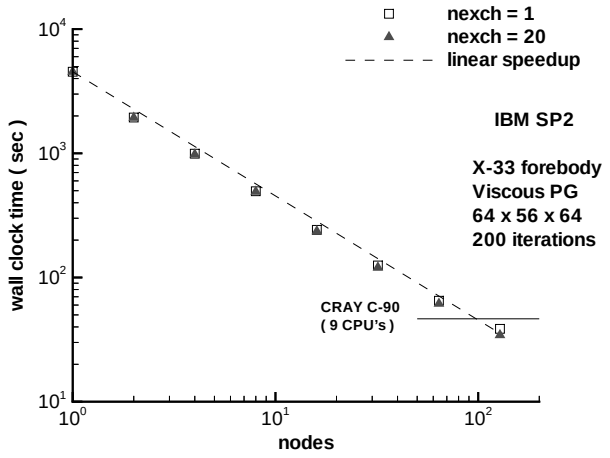


Fig. 3 Elapsed wall clock time on IBM SP2.

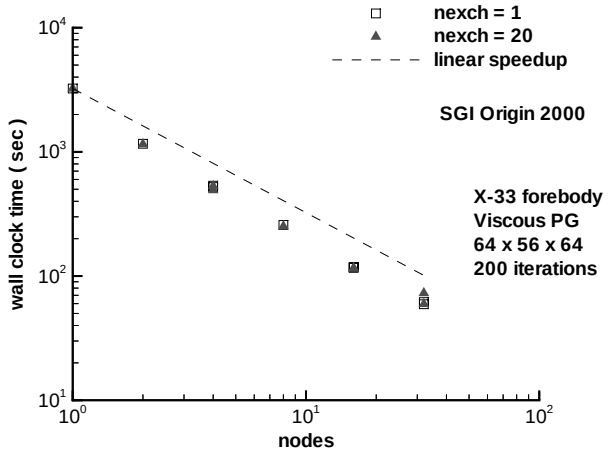


Fig. 4 Elapsed wall clock time on SGI Origin 2000.

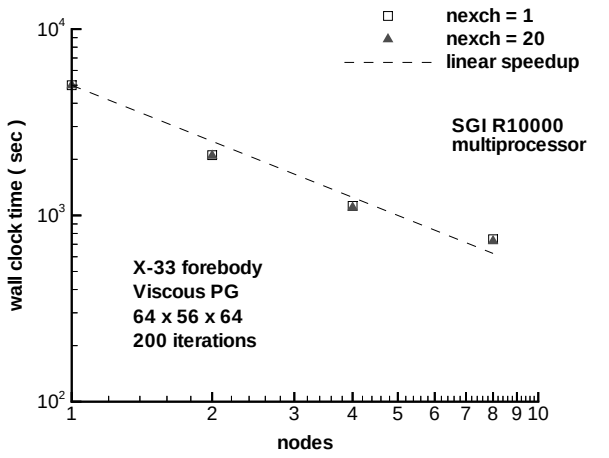


Fig. 5 Elapsed wall clock time on SGI multiprocessor.

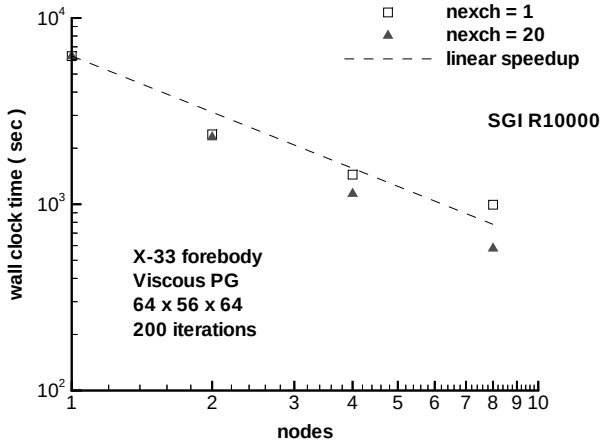
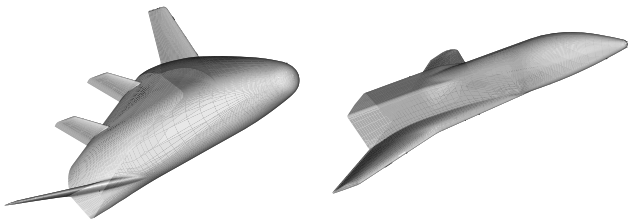


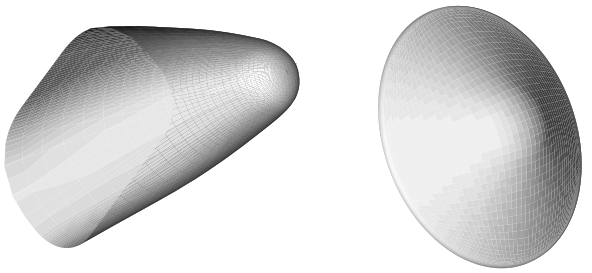
Fig. 6 Elapsed wall clock time on network of SGI R10000 workstations.

Ethernet. Both of these SGI machines use the MPICH implementation[27] of MPI. On all architectures, the MPI defined timer, MPI_WTIME, is used to measure elapsed wall clock time for the main algorithm only. The time to read and write restart files and to perform pre- and post-processing is not measured although it may account for a significant fraction of the total time. Compiler options include '-O3 -qarch=pwr2' on the IBM SP2 and '-O2 -n32 -mips4' on the SGI machines. No effort is made to optimize the single-node performance of LAURA on these cache-based architectures.

Figures 3 - 6 display the elapsed wall clock times on the various machines. A time based on the single-node time and assuming a linear speedup equal to the number of nodes is shown for comparison. The measured times are less than the comparison time for most of the cases as a result of the smaller blocks on each node making better use of the cache. This increase in cache performance offsets the communication penalty. Improving the single-node performance of LAURA on these cache-based architectures would reduce the single-node times and give a more accurate measure of the communication overhead. Nevertheless, the speedup of the code on all machines is good. As anticipated, the relative message-passing performance on the dedicated machines (IBM SP2, SGI Origin 2000, SGI multiprocessor) is better than on the network of SGI workstations. Also, the performance with data exchanged every 20 iterations is noticeably better on the network of workstations than with data exchanged every iteration. However, there is little influence of *nexch* on elapsed time on the dedicated machines which indicates that the communication overhead is very low. The degradation in performance of the 8 processor runs on the SGI multiprocessor is due to the load on the machine from other users and is not a result of the communication overhead. Of course, the times (and message-passing efficiency) measured will vary depending on machine and problem size. Also shown in Fig. 3 is the elapsed time from a multitasking run with the original version of LAURA on a CRAY C-90 using 9 CPU's. This



a) X-33 b) X-34



c) X-33 forebody d) Stardust capsule

Fig. 7 Vehicle geometries.

shows that performance comparable to current vector supercomputers may be obtained on dedicated parallel machines (albeit with more processors) using this distributed-memory version of LAURA.

Convergence

The effect of problem size, gas chemistry, and boundary data exchange frequency on convergence is examined for four realistic geometries: the X-33[10,11] and X-34[12] RLV concepts, the X-33 forebody, and the Stardust sample return capsule forebody.[8] All four geometries are shown in Fig. 7. A viscous (thin-layer Navier-Stokes), perfect gas solution is computed over the X-33 and X-33 forebody configurations. The convergence of an inviscid, perfect gas solution is examined using the X-34 vehicle. Nonequilibrium air chemistry effects on the convergence and performance of the distributed-memory version of LAURA are determined from a viscous, 11-species air calculation over the Stardust capsule. For all geometries, the vehicle is defined by the $k = 1$ surface, and the outer boundary of the volume grid is defined by $k = kmax$.

Each viscous solution is computed with the same sequence of parameters for consistency and is started with all flow-field variables initially set to their freestream values. Slightly different values are used for the inviscid solutions due to low densities on the leeside of the vehicle causing some instability when switching from first to second order accuracy. Methods to speed convergence such as computing on a coarse grid before proceeding to the fine grid and converging blocks sequentially beginning at the nose (i.e. block marching)[5] are not used. The relevant LAURA parameters are shown below. Two values of *nexch* are used (except for the run involving the complete X-33 configuration). A baseline solution is generated with *nexch* equal to 1. Updating the boundary data every iteration should mimic the communication between blocks in the shared-memory version of LAURA. A second computation is made with *nexch* equal to *njcobian* since acceptable values for both parameters depend on transients in the flow. Solutions that are changing rapidly should update the Jacobian and exchange boundary data frequently, while partially converged solutions may be able to freeze the Jacobian and lag the boundary data for a number of iterations. A simple strategy is to link the two parameters. Convergence is measured by the L_2 norm defined by

$$L_2 = \frac{1}{C_N^2} \sum_{L=1}^{N} \frac{(\mathbf{r}_L \cdot \mathbf{r}_L)}{\rho_L^2} \qquad (2)$$

where C_N is the Courant number, N is the total number of cells, $\mathbf{r}_L$ is the residual vector, and ρ_L is the local density. All solutions are generated on the IBM SP2.

Table 2 LAURA parameters - viscous.

Iterations	Order	*njcobian*
0-100	1	1
101-300	1	2
301-500	2	10
> 500	2	20

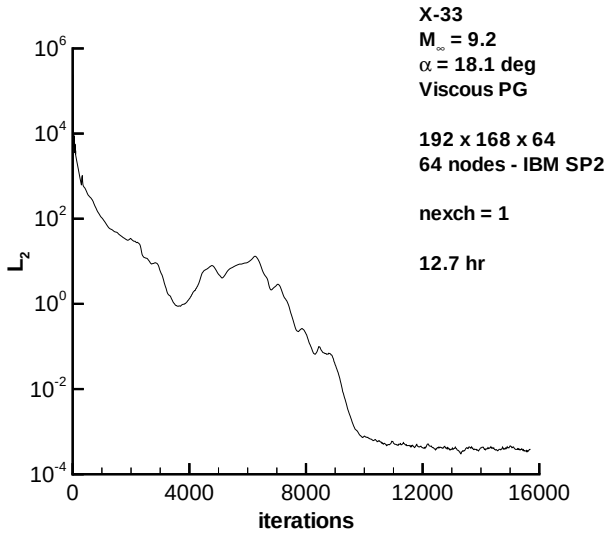
Table 3 LAURA parameters - inviscid.

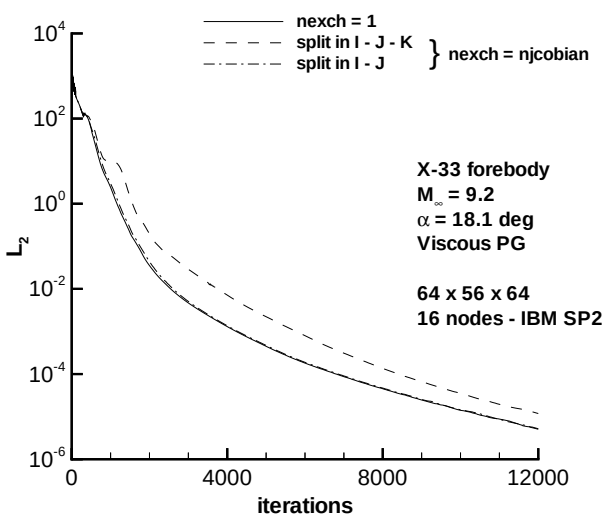
Iterations	Order	*njcobian*
0-100	1	1
101-300	1	2
301-900	1	10
901-1100	2	10
> 1100	2	20

X-33

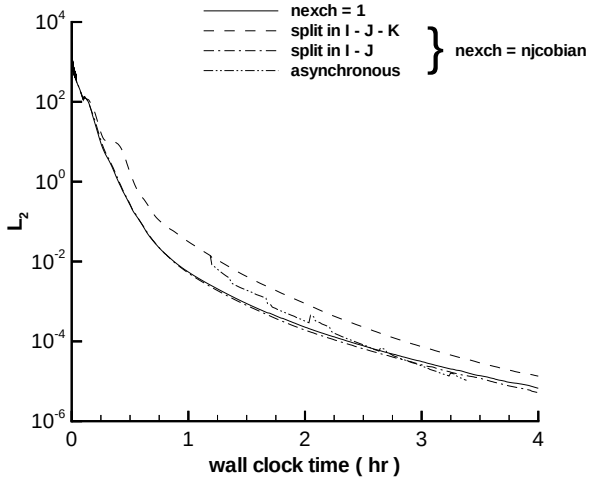
The viscous, perfect gas flow field is computed over the X-33 RLV configuration (without the wake) to demonstrate the ability of the new message-passing version of LAURA to handle large-scale problems in a reasonable amount of time. The freestream Mach number is 9.2, the angle of attack is 18.1 deg, and the altitude is 48.3 km. The grid size is 192 x 168 x 64 and is divided into 64 blocks of 48 x 42 x 16.

Figure 8 shows the L_2 convergence as a function of number of iterations. The elapsed wall clock time on the SP2 is 12.7 hr. Only the baseline case($nexch = 1$) was computed due to resource limitations. The effect of *nexch* on convergence will be examined in greater detail on the nose region of this vehicle. The stall in convergence after 10000 iterations is due to a limit cycle in the convergence at the trailing edge of the tip of the canted fin. Iterations are continued past





a) Convergence as function of number of iterations



b) Convergence as function of time

Fig. 8 Convergence history of viscous, perfect gas flow field over X-33 vehicle.

Fig. 9 Convergence histories of viscous, perfect gas flow field over X-33 forebody.

this point to converge the boundary layer and surface heating.

X-33 forebody

The effects of boundary data exchange frequency and block splitting on convergence are evaluated for the nose section of the X-33. This is the same configuration used to obtain the timing estimates, and freestream conditions correspond to the complete X-33 vehicle case. The 64 x 56 x 64 grid is first divided in the i-, j-, and k- directions into 16 blocks comprised of 32 x 28 x 16 cells each. Two cases, $nexch = 1$ and $nexch = njcobian$, are run using this blocking. Another case is computed with the grid divided in the i- and j-directions only resulting in blocks of 16 x 14 x 64 cells. Next, the asynchronous relaxation capabilities of LAURA are tested by reblocking a partially converged restart file in the k-direction to cluster work (and iterations) in the boundary layer. Each block has i x j dimensions of 32 x 28, but the k dimension is split into 8, 8, 16, and 32 cells. Blocks near the wall contain 32 x 28 x 8 cells, while blocks near the outer boundary have 32 x 28 x 32 cells. Thus, the smaller blocks accumulate more iterations than the larger outer blocks in a given amount of time and should converge faster.

Figure 9 shows the convergence history for this flow field. For viscous solutions, convergence is typically divided into two stages. First, the inviscid shock layer develops and then the majority of the iterations are spent converging the boundary layer (and surface heating). Lagging the boundary data appears to have more of an impact on the early convergence of the inviscid features of the flow and less of an impact on the boundary layer convergence. This effect is much larger when the blocks are split in the k-direction across the shock layer. The communication delay affects the developing shock wave as it crosses the block boundaries in the k-direction.

Figure 9(b) shows the convergence as a function of wall clock time. Because of the low communication overhead on the IBM SP2, the time saved by making fewer boundary data exchanges is small. As seen from the timing data, this would not necessarily be true on a network of workstations where the decrease in communication overhead might offset any increase in number of iterations. Also shown are LAURA's asynchronous relaxation capabilities. After 1 hr (and 3500 iterations), the outer inviscid layer is partially converged. Restructuring the block structure at this point by splitting the k dimension into 8, 8, 16, and 32 cells allows the boundary layer to accumulate more iterations and accelerates convergence. The result is a 15 percent decrease in wall clock time compared to the

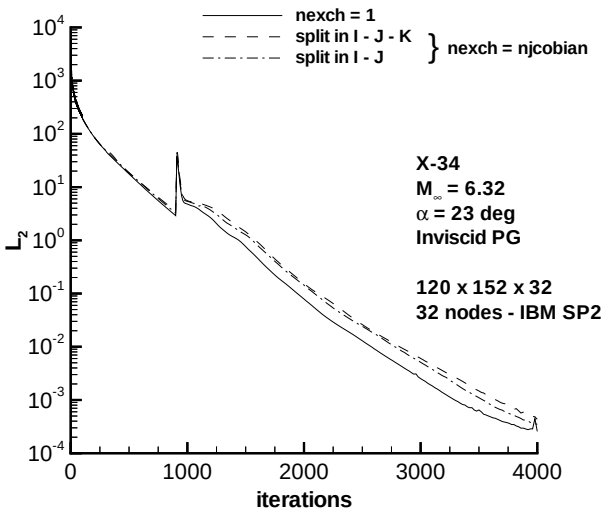
baseline ($nexch = 1$) case. A similar strategy would also have accelerated the convergence of the baseline case.

X-34

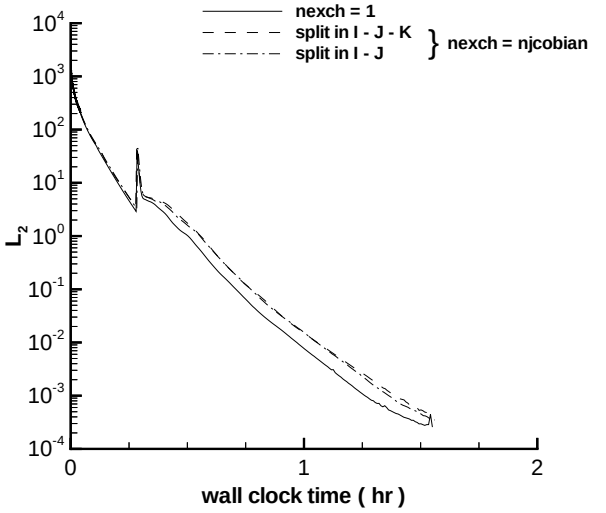
The effect of boundary data exchange frequency and block splitting on convergence of inviscid, perfect gas flows is investigated for the X-34 configuration (minus the body flap and vertical tail). Inviscid solutions are useful in predicting aerodynamic characteristics for vehicle design and may be coupled with a boundary-layer technique to predict surface heat transfer as well. The freestream Mach number is 6.32, the angle of attack is 23 deg, and the altitude is 36 km. The grid is 120 x 152 x 32 and is first divided into 32 blocks of 30 x 38 x 16 cells. The grid is also split in the i- and j-directions into blocks of 30 x 19 x 32 cells to check the effect of block structure on convergence. The convergence histories are shown in Figure 10. The aerodynamics (not shown) of the vehicle are converged at 4000 iterations. The spike in convergence at 900 iterations is caused by the switch from first to second order accuracy. With the grid split in all directions, the baseline solution ($nexch = 1$) reaches an L_2 norm of 10^{-3} at 3300 iterations while the solution with boundary data lagged takes 3640 iterations. The solution with the grid split in the i- and j-directions requires 3530 iterations. As shown in Fig. 10(b), there is a corresponding difference in run times to reach that convergence level because the savings from fewer boundary data exchanges are small on the SP2. Nevertheless, the effect of lagging the boundary data on convergence is minimal.

Stardust

The convergence of a nonequilibrium air (11 species, two temperature), viscous computation is examined for the forebody of the Stardust capsule. The freestream Mach number is 17 and the angle of attack is 10 deg. The grid is 56 x 32 x 60 and is divided into 32 blocks of 7 x 8 x 60 cells each. There are no splits in the k-direction. Figure 11 shows the convergence as a function of iterations and elapsed wall clock time. Because of the larger number of flow-field variables, considerably more data must be exchanged between blocks for nonequilibrium flows. Even on a dedicated parallel machine such as the IBM SP2, the communication penalty for this particular case has a significant impact on the elapsed time. The baseline case reaches an L_2 norm of 10^{-4} at 6900 iterations compared to 7500 iterations for the $nexch = njcobian$ solution. However, the savings in communication time allows the $nexch = njcobian$ solution to converge 1 hr faster than the baseline case.



a) Convergence as function of number of iterations

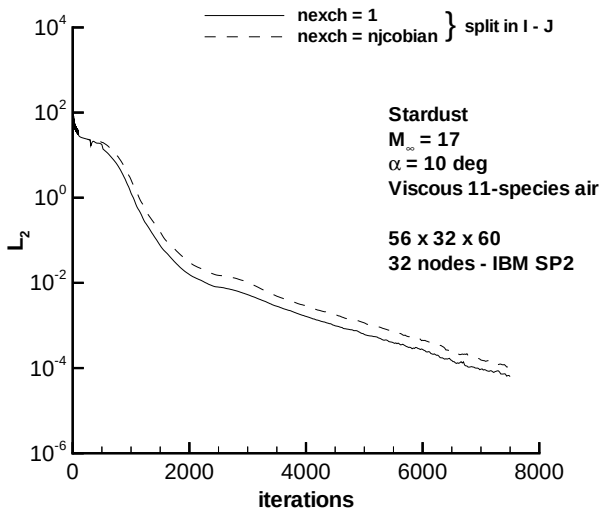


b) Convergence as function of time

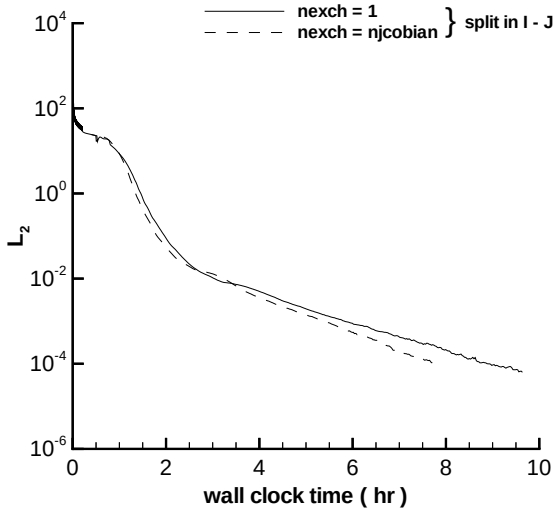
Fig. 10 Convergence histories of inviscid, perfect gas flow field over X-34 vehicle.

Conclusions

The shared-memory, multitasking version of the CFD code LAURA has been successfully modified to take advantage of distributed-memory parallel machines. A standard domain decomposition strategy yields good speedup on dedicated parallel systems, but the single-node performance of LAURA on cache-based architectures requires further study. The point-implicit relaxation strategy in LAURA is well-suited for parallel computing and allows the communication overhead to be minimized (if necessary) by reducing the frequency of boundary data exchanges. The communication overhead is greatest on the network of workstations and for nonequilibrium flows due to more data passing between nodes. Lagging the boundary data between blocks appears to affect the development of the inviscid shock layer more than the convergence of the boundary layer. Its largest effect occurs when



a) Convergence as function of number of iterations



b) Convergence as function of time

Fig. 11 Convergence histories of viscous, nonequilibrium flow field over Stardust capsule.

the grid is split in the direction normal to the vehicle surface. However, restructuring the blocks to cluster work and iterations in the boundary layer improves overall convergence once the inviscid features of the flow have developed. These results demonstrate the ability of the new message-passing version of LAURA to effectively use distributed-memory parallel systems for realistic configurations. As a result, the effectiveness of LAURA as an aerospace design tool is enhanced by its new parallel computing capabilities. In fact, this new version of LAURA is currently being applied to the evaluation of vehicles used in planetary exploration missions and the X-33 program.

Acknowledgements

The authors wish to acknowledge Peter Gnoffo of the Aerothermodynamics Branch at NASA LaRC for his assistance with the inner workings of LAURA and Jerry Mall of Computer Sciences Corporation for his help in profiling LAURA on the IBM SP2.

References

[1]Gnoffo, P. A., "An Upwind-Biased, Point-Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flows," NASA TP–2953, Feb. 1990.

[2]Gnoffo, P. A., "Upwind-Biased, Point-Implicit Relaxation Strategies for Viscous, Hypersonic Flows," AIAA Paper 89–1972, Jun. 1989.

[3]Gnoffo, P. A., "Code Calibration Program in Support of the Aeroassist Flight Experiment," *Journal of Spacecraft and Rockets*, Vol. 27, No. 2, 1990, pp. 131–142.

[4]Weilmuenster, K. J. and Greene, F. A., "HL-20 Computational Fluid Dynamics Analysis," *Journal of Spacecraft and Rockets*, Vol. 30, No. 5, 1993, pp. 558–566.

[5]Gnoffo, P. A., Weilmuenster, K. J., and Alter, S. J., "Multiblock Analysis for Shuttle Orbiter Re-Entry Heating From Mach 24 to Mach 12," *Journal of Spacecraft and Rockets*, Vol. 31, No. 3, 1994, pp. 367–377.

[6]Mitcheltree, R. A. and Gnoffo, P. A., "Wake Flow About the Mars Pathfinder Entry Vehicle," *Journal of Spacecraft and Rockets*, Vol. 32, No. 5, 1994, pp. 771–776.

[7]Weilmuenster, K. J., Gnoffo, P. A., Greene, F. A., Riley, C. J., Hamilton, H. H., and Alter, S. J., "Hypersonic Aerodynamic Characteristics of a Proposed Single-Stage-to-Orbit Vehicle," *Journal of Spacecraft and Rockets*, Vol. 33, No. 4, 1995, pp. 463–469.

[8]Mitcheltree, R. A., Wilmoth, R. G., Cheatwood, F. M., Brauckmann, G. J., and Greene, F. A., "Aerodynamics of Stardust Sample Return Capsule," AIAA Paper 97–2304, Jun. 1997.

[9]Mitcheltree, R. A., Moss, J. N., Cheatwood, F. M., Greene, F. A., and Braun, R. D., "Aerodynamics of the Mars Microprobe Entry Vehicles," AIAA Paper 97–3658, Aug. 1997.

[10]Cook, S. A., "X-33 Reusable Launch Vehicle Structural Technologies," AIAA Paper 96–4573, Nov. 1996.

[11]Gnoffo, P. A., Weilmuenster, K. J., Hamilton, H. H., Olynick, D. R., and Venkatapathy, E., "Computational Aerothermodynamic Design Issues for Hypersonic Vehicles," AIAA Paper 97–2473, Jun. 1997.

[12]Levine, J., "NASA X-34 Program," Meeting Papers on Disc A9710806, AIAA, Nov. 1996.

[13]Cheatwood, F. M. and Gnoffo, P. A., "User's Manual for the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA)," NASA TM–4674, Apr. 1996.

[14]Gnoffo, P. A., "Asynchronous, Macrotasked Relaxation Strategies for the Solution of Viscous, Hypersonic Flows," AIAA Paper 91–1579, Jun. 1991.

[15]Jayasimha, D. N., Hayder, M. E., and Pillay, S. K., "An Evaluation of Architectural Platforms for Parallel Navier-Stokes Computations," NASA CR–198308, Mar. 1996.

[16]Venkatakrishnan, V., "Parallel Implicit Unstructured Grid Euler Solvers," AIAA Paper 94–0759, Jan. 1994.

[17]Wong, C. C., Blottner, F. G., Payne, J. L., and Soetrisno, M., "A Domain Decomposition Study of Massively Parallel Computing in Compressible Gas Dynamics," AIAA Paper 95–0572, Jan. 1995.

[18]Borrelli, S., Schettino, A., and Schiano, P., "Hypersonic Nonequilibrium Parallel Multiblock Navier-Stokes Solver," *Journal of Spacecraft and Rockets*, Vol. 33, No. 5, 1996, pp. 748–750.

[19]Domel, N. D., "Research in Parallel Algorithms and Software for Computational Aerosciences," NAS 96-004, Apr. 1996.

[20]Van der Wijngaart, R. F. and Yarrow, M., "RANS-MP: A Portable Parallel Navier-Stokes Solver," NAS 97-004, Feb. 1997.

[21]Forum, M. P. I., "MPI: A message-passing interface standard," Computer Science Dept. Technical Report CS–94–230, University of Tennessee, Knoxville, TN, 1994.

[22]Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., "PVM 3.0 User's Guide and Reference Manual," Tech. rep., Feb. 1993.

[23]Balasubramanian, R., "Modification of Program LAURA to Execute in PVM Environment," Spectrex Report 95.10.01, Oct. 1995.

[24]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.

[25]Harten, A., "High Resolution Schemes for Hyperbolic Conservation Laws," *Journal of Computational Physics*, Vol. 49, No. 3, 1983, pp. 357–393.

[26]Yee, H. C., "On Symmetric and Upwind TVD Schemes," NASA TM–86842, Sep. 1985.

[27]Gropp, W. and Lusk, E., "User's Guide for `mpich`, a Portable Implementation of MPI," Tech. Rep. ANL/MCS–TM–ANL–96/6, Argonne National Laboratory, 1996.

[28]Burns, G., Daoud, R., and Vaigl, J., "LAM: An Open Cluster Environment for MPI," Tech. rep., Ohio Supercomputing Center, May 1994.