89-1940-CP

A MASSIVELY-PARALLEL NAVIER-STOKES IMPLEMENTATION

Robert Wesley*

Eng-Shien Wu**

D. A. Calahan+

The University of Michigan
Ann Arbor, Michigan

Abstract

The results of implementing a 3-D production MacCormack explicit Navier-Stokes code with damping on a 1024-node NCUBE hypercube is presented. Among the issues and results presented are (1) grid partitioning, (2) performance analysis and comparison with CRAY vector processors, and (3) detailed timing model.

I. Introduction

The regularity of the uniform grids traditionally used in flow modeling has been responsible for the success of vector multiprocessors - such as CRAY-class machines - in solving fluid flow problems. Parallelization of serial codes on such shared-memory machines by multitasking is usually straightforward and achieves a high processor utilization for the common 4- to 8-processor configuration. [1,2]

Indeed, success on such architectures has inspired other investigators to study the distribution of data as well as computation on so-called massively-parallel machines such as the hypercube. [3,4]. Although such implementations are typically far more difficult to code - since no common memory exists - the payoff for systematically organizing inter-processor data flow is the reduction of total data flow and the consequent conflicts over data paths. In contrast, access conflicts in shared-memory multiprocessors are assumed to occur randomly and presently limit the number of processors to eight. Fortunately, the same distributed CFD algorithms can be the basis of many production codes and are applicable to several current MIMD commercial architectures, so that significant programming effort can be justified.

This paper extends the distributed CFD results of Catherasoo and others to include a full Navier-Stokes (N-S) code with damping and, more importantly, shows the efficiency of distributed N-S algorithms up to 1024 nodes (processors). [3,5] Detailed timing analysis identifies sources of the modest parallelization overhead.

II. Grid Partitioning For Distributed Solution

Problem Description

The code on which this is based is a MacCormack explicit solution of the Navier-Stokes equations with damping. [6,7]

In physical coordinates, data for a right circular cylinder of radius 1.0 was used (see Figure 1); this was transformed to a rectangular uniform coordinate system.

A simple grid generator was written which would develop grid x,y,z coordinates for a grid of specified

* Student, Aerospace Engineering

** Student, Elec. Engring. & Comp. Science

+ Professor, Elec. Engring. & Comp. Science size. The common characteristics among all grids used in timing cases was that grid points were evenly spaced at increments of .2 radially (outward from the cylinders surface) and .2 axially (along the length of the cylinder). See Figure 1 for two views of a typical grid.

The same data set was used for all timing cases. The data set had the following characteristics:

1) Freestream Mach Number = 1.1
2) Freestream Reynolds Number = 10,000
3) Freestream Temperature = 461.7 R
4) Wall Temperature = 550.25 R
5) Characteristic Length = 1.0 (cylinder of radius 1.0)
6) Damping coefficients =3

Grid Partitioning

Grid partitioning is a major aspect of parallelizing a computational fluid dynamics problem. The first consideration is to reduce the interprocessor data movement by processing as many grid points as possible in a node - within local memory limits.

To achieve this, grid points are blocked in three-dimensional cubes [3]. Grid points near the block periphery must be shared with another node so that 2-nd order differencing (due to damping) and grid transformations can be performed in both nodes. This creates a 3-wide shell of shared points, illustrated in Figure 2 in two dimensions.

This partitioning results in the following nomenclature.

(I_g, J_g, K_g) - The dimensions of the overall rectangular grid, exclusive of boundary grid points.

(I_n, J_n, K_n) - The dimensions of the rectangular grid processed by a node, including grid points shared with other nodes.

(I_c, J_c, K_c) - The dimensions of the rectangular grid processed by a node, excluding grid points shared with other nodes. These will be termed computation grid points.

III. Memory Requirements

Study of the resultant code shows the memory requirements in the nodes are (in bytes)

$$152300 + 4\ [6100 + J_n + 133K_n + 11J_nK_n + 27I_nJ_nK_n] \qquad (1)$$

Based on these formulae, it is possible to calculate the largest grid partition that can be accommodated by each processor. Table 1 shows the results, including the 512 kbyte/node case used for measured data. With recent increases in memory density, the 2048-kbyte case should be easily achievable by the next generation of 1024-node machines.

Table 1. Memory requirements, grid sizes, and efficiencies

Node memory kbytes	(I_n, J_n, K_n)	Efficiency E_c
256	(8,8,8)	.67
512	(14,14,14)	.94
1024	(19,19,19)	.97
2048	(25,25,25)	.98
4096	(33,33,33)	.99

IV. Performance

The initial release of hypercube hardware suffered the liabilities of limited individual node processor capabilities and too few processors to approach state-of-the-art vector processor performance, even for efficient parallel algorithms.

Table 2 shows that 1024 processors allows the parallel code to compete with (uniprocessor) CRAY-class machines. Two caveats should be noted: (1) the NCUBE results are in 32-bit arithmetic and CRAY results in 64-bit arithmetic, the hardware wordlengths of each machine, and (2) in the grids studied, each node processed the largest cubic grid partition permitted[++] ; a total grid was then constructed from these cubes. Both of these caveats favor the parallel performance, although the next hardware generation of nodes will likely perform 64-bit arithmetic.

Table 2. Measured timings of MacCormack code

# of nodes (n)	grid size (I_c,J_c,K_c)	time (sec)	time/gp/iter.
1	(7,7,7)	7.69	2.24×10^{-2}
64	(28,28,28)	8.71	3.97×10^{-4}
128	(56,28,28)	8.72	1.94×10^{-4}
256	(56,56,28)	8.54	9.81×10^{-5}
512	(56,56,56)	8.61	4.90×10^{-5}
CRAY-2	(64,64,32)	--	3.50×10^{-5}
X-MP	(64,64,32)	--	3.30×10^{-5}
1024	(112,56,56)	8.55	2.43×10^{-5}
Y-MP	(64,64,32)	--	2.00×10^{-5}

V. Algorithm Timing Analysis

Speedup and Efficiency

The common standard of parallel processing performance is speedup, defined as

$$S = T_1/T_n$$

where T_1 is the uniprocessor execution time and T_n the execution time on the same problem with n processors. With limited local memory, however, T_1 cannot be determined for representative grid sizes. Therefore, a reference problem of size $(I_c,J_c,K_c) = (7,7,7)$ and devoid of message-passing or synchronization is run on a single node. The choice of grid size of this reference problem is arguable and affects efficiency calculations, since a node processes a larger partition more efficiently (i.e., less time per grid point).

If the execution time of this reference problem is T_r for one iteration with a total of G_r computation grid points, then with n processors and G_n total computation grid points, the speedup with per-iteration time T_n is defined as

$$S = (T_r/G_r)/(T_n/G_n) \qquad (2)$$

and the measured efficiency is

$$E_m = S/n \leq 1 \qquad (3)$$

The speedups and efficiencies for grids chosen to optimize the parallel performance are given in Table 3.

Interprocessor Communication

Communication occurs for two reasons within each iteration.

(1) Plane swap. Data from the shell of shared grid points in adjacent cells (Figure 3) are exchanged; this occurs twice within each iteration - after both the predictor and corrector steps - in simultaneous exchanges between all nodes. A corresponding timing diagram is shown in Figure 4. All nodes participate in this exchange, although nodes processing grid points on the periphery of the grid with some missing neighbors exchange less data. Nodes without missing neightbors are termed *internal nodes.*

(2) Time-step calculation. Stability limitations are calculated

[++] It should be possible to process a (8x8x8) partition on a 512-kbyte NCUBE node. However, only a (7x7x7) partition would complete a sufficient number of iterations to obtain repeatable timings on successive iterations.

for the cube of grid points private to each node; these are communicated to node 0, where an overall time step is determined and communicated back to all processors.

(It should be noted that nodes on the k-edge wraparound are processed as internal nodes; i.e., with a full set of neighbors.)

Since all internal nodes carry out the same computation, their efficiencies are equal and can be calculated from

$$E_c = \text{Numerical computation time} / \text{Total time}$$

for each iteration. These are shown in Table 3; they are found to be in excellent agreement with measured efficiencies (E_m).

Noting that more local memory permits less data flow per floating-point computation, it is possible to extrapolate the timing model of Figure 4 to hardware with more local memory. The resultant calculated efficiency E_c is shown in Table 1, showing only modest gains in efficiency for more local memory. Thus, future machines with similar numerical computation and data transfer rates could tradeoff local memory size for more nodes (parallelism) [8].

VI. Conclusions

The above results show the viability of both massive parallelism and small (512-kbyte) local memory for 3-D production explicit Navier-Stokes codes. Performance of this code is also being studied as a function of processor characteristics other than memory size, such as MFLOP rate, interprocessor data-transfer rate, and message startup time (latency). Such a generic model will be able to predict performance for other message-passing commercial multiprocessors expected in the near-term.

Acknowledgement

The authors are indebted to Sandia Corporation (Albuquerque) for access to their 1024-node hypercube. John Holm's contribution in development of an early version of the code is acknowledged; Paul Kominsky's assistance is also noted. The code was provided by Dr. J. Shang of WRDC, WPAFB.

The research was supported by the Air Force Office of Scientific Research under Grant 88-0212.

References

1. Johnson, G.M., Swisshelm, J.M., Pryor, D.V., and Ziebarth, J.P., "Multitasked Embedded Multigrid for Three-dimensional Flow Simulation," Lecture Notes in Physics, vol. 264, Springer, Berlin, 1986, pp 350-356.

2. Mulac, R.A., M.L. Celestina, J.J. Adamczyk, K.P. Misegades, and J.M. Dawson, "The Utilization of Parallel Processing in Solving the Inviscid Form of the Average-Passage Equation System

Table 3. Results of timing analysis

Nodes n	Total grid size (I_g, J_g, K_g)	Time (sec)	Speedup (S)	Efficiency (E_m)	Efficiency (E_c)
1	(7,7,7)	7.69	1.	1.	1.
64	(28,28,28)	8.71	56.4	.882	.905
128	(56,28,28)	8.72	113.	.882	.905
256	(56,56,28)	8.54	231.	.901	.905
512	(56,56,56)	8.61	457.	.893	.905
1024	(112,56,56)	8.55	922.	.900	.905

for Multistage Turbomachinery," AIAA 8th Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 9-11, 1987, pp 70-80.

3. Cathersoo, C.J., "Separated Flow Simulations Using the Vortex Method on a Hypercube," Proc. 8th Computational Fluid Dynamics Conference, AIAA Paper 87-1109, Honolulu, Hawaii, June 9-11, 1987, pp 81-86.

4. Bassett, M.E., and Catherasoo, C.J., "Simulation of Transonic Flow on a Hypercube," (reprint from AMETEK Corporation)

5. Gustafson, J.L., Montry, G.R., and Benner, R.E., "Development of Parallel Methods for a 1024-Processor Hypercube," SIAM J. on Scientific and Statistical Computing,. vol.9, no.4, July, 1988.

6. MacCormack, R.W., "The Effect of Viscosity in Hypervelocity Impact Cratering," AIAA Paper 69-534, (Cinncinnati, Ohio),1969.

7. Shang, J.S., Buning, P.G., Hankey, W.L., and Wirth, M.C., "Performance of a Vectorized Three-dimensional Navier-Stokes Code on the CRAY-1 Computer," Paper AIAA 79-1448R, AIAA Journal, vol. 18, no. 9, September 1980, pp 1073-1079.

8. Wu, Eng-Shien, D. A. Calahan, and Robert Wesley, "Performance Analysis and Projections for a Massively-Parallel Navier-Stokes Implementation," Fourth Conference on Concurrent Hypercube Computers and Applications, Monterey, CA, March, 1989.

Figure 1. Cylindrical physical grid

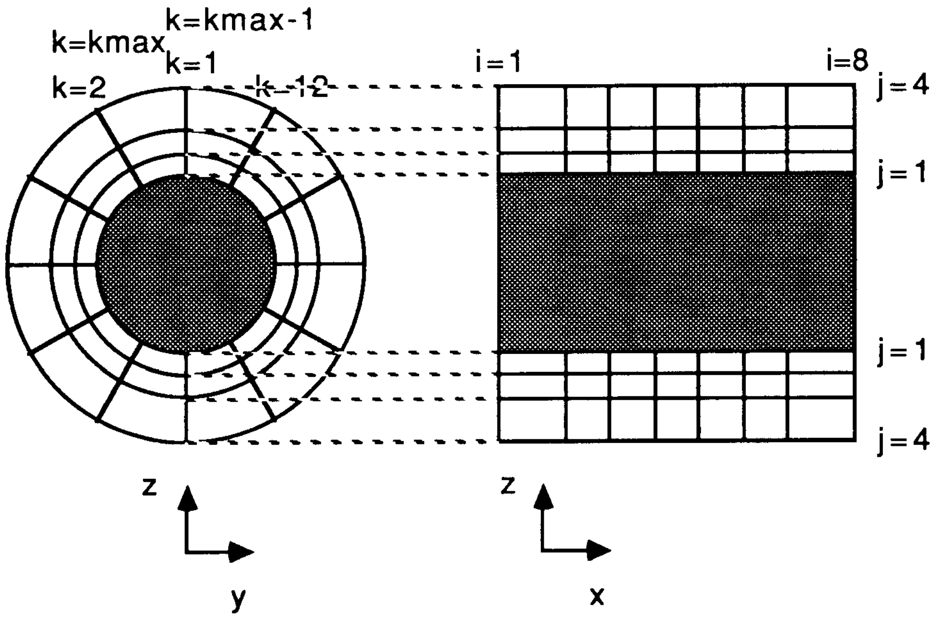
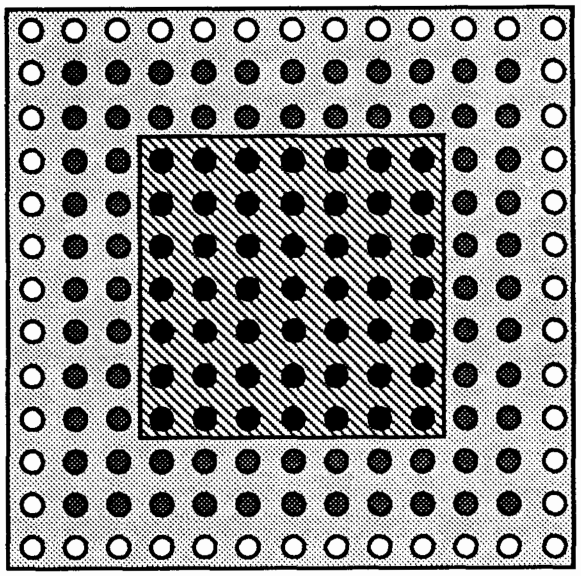


Figure 2. 2-D slice of grid block for a node N
$(I_n, J_n, K_n) = (13,13,13)$



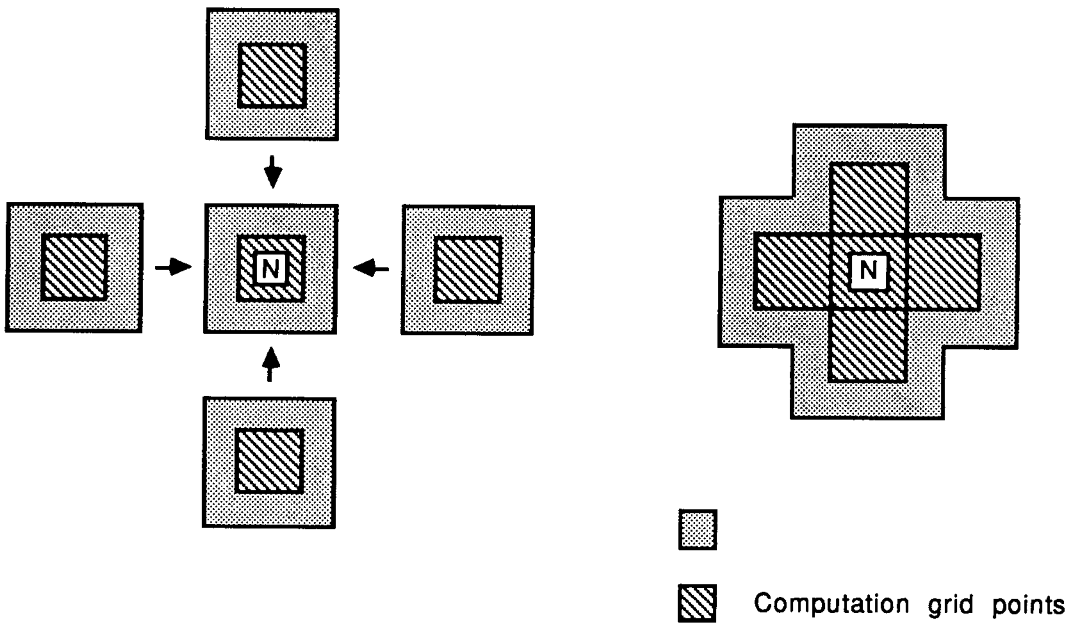
◍ Shared grid points for differencing in node N

○ Shared grid points for coordinate transformation in node N

● Computation grid points

▨ Grid points within node N

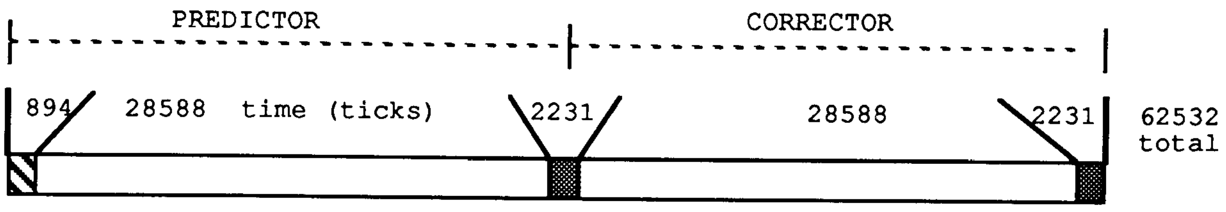
Figure 3. 2-D slice of grid cover of node N



(a) Node N and neighbors (exploded view)

(b) Composite view of (a)

Figure 4. Timing model with $(I_C,J_C,K_C) = (7,7,7)$; 1 tick = 1024 cp = 136.5 µsec



NUMERICAL COMPUTATION

LOCAL TIME STEP CALCULATION

PLANE SWAP