Multitasking the INS3D-LU Code on the Cray Y-MP

Rod Fatoohi* Seokkwan Yoon†

Report RNR-91-015, April 1991

Abstract

This paper presents the results of multitasking the INS3D-LU code on eight processors of the Cray Y-MP. The code is a full Navier-Stokes solver for incompressible fluid in three dimensional generalized coordinates using a lower-upper symmetric-Gauss-Seidel implicit scheme. This code has been fully vectorized on oblique planes of sweep and parallelized using autotasking with some directives and minor modifications. The timing results for five grid sizes are presented and analyzed. The code has achieved a processing rate of over one Gflops and needs less than one μsec per grid point per iteration on the Cray Y-MP for several grid sizes.

A revised version of this report appeared as paper No. AIAA-91-1581 in the Proceedings of the AIAA 10th Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 24-27, 1991.

*Computer Scientist. NAS Applied Research Branch, NASA Ames, MS T045-1, Moffett Field, CA 94035. The author is an employee of Computer Sciences Co. This work was funded in part through NASA Contract NAS 2-12961.

†Senior Scientist. The author is an employee of MCAT, Moffett Field, California. This work was funded in part through NASA Contract NCC 2-505.

1 Introduction

During the past decade, a number of numerical algorithms have been developed for computational fluid dynamics (CFD). Explicit methods which have been successful for inviscid flows appears to be inefficient for viscous flows because of the well-known Courant-Friedrichs-Lewy (CFL) stability condition. Stability of the explicit methods are further restricted when solving a coupled set of differential equations for high speed reacting flows past the National Aerospace Plane (NASP) or transatmospheric vehicles for Earth and Mars. These inherent disadvantages of explicit methods let CFD researchers put their emphasis on implicit methods. One of the fastest implicit methods is a line Symmetric-Gauss-Seidel (SGS) relaxation method, which is not vectorizable because of its recursive nature. Recently, a new SGS based implicit algorithm (LU-SGS) was derived for the incompressible Navier-Stokes equations.[1] This algorithm is not only unconditionally stable but completely vectorizable in three dimensions. Based on this scheme, a code named INS3D-LU was developed which uses two-dimensional arrays in three dimensions for vectorization on oblique planes of sweep. However, unlike the explicit methods, implementation of implicit methods on parallel computers is not straightforward.

Despite the fact that the current generation of supercomputers can perform at peak rates of several billion floating point operations per second, very few codes can achieve these rates. While multitasking has been available on the Cray series of multiprocessor supercomputers since 1984, its use has been very limited. Some experiments have shown that reasonable performance can be achieved on these machines.[2] The work reported here was just such an experiment. The INS3D-LU code has been multitasked on eight processors of the Cray Y-MP using a technique called autotasking. This technique as well as the numerical algorithm and the measured results are described in the following sections.

2 Autotasking

Autotasking, or automatic multitasking, is a technique where the compiling system attempts to detect and exploit parallelism in a Fortran program and generates code to be executed on multiple processors.[3] This process is automatic but not all types of parallelism can be detected and programmer intervention is frequently required. Autotasking works on do-loop boundaries. If dependence analysis allows it, autotasking turns a nested do loop to a vector inner loop and a parallel outer loop. In determining how to optimize a program, autotasking favors vectorization over parallelism. With user direction, autotasking can stripmine a single vectorized do loop with a large iteration count. Stripmining effectively turns the single vectorized loop into a nested do loop with a vector inner loop and a parallel outer loop.

Autotasking accepts compiler directives which appear as comment lines. These directives fall into two categories: CFPP$ and CMIC$. The CFPP$ directives tell the dependence analyzer where to look for parallelism in a program or make assertions that may allow it to recognize parallelism. The CMIC$ directives are used to enforce

parallelism when it is safe to do it.

3 Numerical Algorithm

Let t be time; ρ and p the density and pressure; u, v, and w the velocity components in Cartesian coordinates (x, y, z); $\hat{Q}$ the vector of conserved variables; $\hat{E}$, $\hat{F}$, and $\hat{G}$ the convective flux vectors; and $\hat{E}_v$, $\hat{F}_v$, and $\hat{G}_v$ the flux vectors for the viscous terms. Then the three-dimensional Navier-Stokes equations in generalized curvilinear coordinates $(\xi,\ \eta,\ \zeta)$ can be written as

$$\partial_t \hat{Q} + \partial_\xi(\hat{E} - \hat{E}_v) + \partial_\eta(\hat{F} - \hat{F}_v) + \partial_\zeta(\hat{G} - \hat{G}_v) = 0 \tag{1}$$

where

$$\hat{Q} = h \begin{bmatrix} p \\ u \\ v \\ w \end{bmatrix}, \quad \hat{E} = h \begin{bmatrix} \beta U \\ Uu + \xi_x p \\ Uv + \xi_y p \\ Uw + \xi_z p \end{bmatrix},$$

$$\hat{F} = h \begin{bmatrix} \beta V \\ Vu + \eta_x p \\ Vv + \eta_y p \\ Vw + \eta_z p \end{bmatrix}, \quad \hat{G} = h \begin{bmatrix} \beta W \\ Wu + \zeta_x p \\ Wv + \zeta_y p \\ Ww + \zeta_z p \end{bmatrix} \tag{2}$$

The contravariant velocity components U, V, and W are defined as

$$\begin{aligned} U &= \xi_x u + \xi_y v + \xi_z w \\ V &= \eta_x u + \eta_y v + \eta_z w \\ W &= \zeta_x u + \zeta_y v + \zeta_z w \end{aligned} \tag{3}$$

where β is the pseudocompressibility parameter. h denotes the cell volume.

An unfactored implicit scheme can be obtained from a nonlinear implicit scheme by linearizing the flux vectors about the previous time step and dropping terms of the second and higher order.

$$[I + \alpha \triangle t(D_\xi \hat{A} + D_\eta \hat{B} + D_\zeta \hat{C})]\delta\hat{Q} = -\triangle t \hat{R} \tag{4}$$

where $\hat{R}$ is the residual

$$\hat{R} = D_\xi(\hat{E} - \hat{E}_v) + D_\eta(\hat{F} - \hat{F}_v) + D_\zeta(\hat{G} - \hat{G}_v) \tag{5}$$

and I is the identity matrix. $\delta\hat{Q}$ is the correction $\hat{Q}^{n+1}$ -$\hat{Q}^n$, where n denotes the time level. D_ξ, D_η, and D_ζ are difference operators that approximate ∂_ξ, ∂_η, and ∂_ζ. $\hat{A}$, $\hat{B}$, and $\hat{C}$ are the Jacobian matrices of the convective flux vectors.

$$\hat{A} = \frac{\partial \hat{E}}{\partial \hat{Q}}, \quad \hat{B} = \frac{\partial \hat{F}}{\partial \hat{Q}}, \quad \hat{C} = \frac{\partial \hat{G}}{\partial \hat{Q}} \tag{6}$$

For $\alpha = \frac{1}{2}$, the scheme is second order accurate in time. For other values of α, the time accuracy drops to first order.

The LU-SGS scheme can be written as

$$LD^{-1}U\delta\widehat{Q} = -\triangle tR \tag{7}$$

where

$$\begin{aligned} L &= I + \alpha\triangle t(D_\xi^- \widehat{A}^+ + D_\eta^- \widehat{B}^+ + D_\zeta^- \widehat{C}^+ - \widehat{A}^- - \widehat{B}^- - \widehat{C}^-) \\ D &= I + \alpha\triangle t(\widehat{A}^+ - \widehat{A}^- + \widehat{B}^+ - \widehat{B}^- + \widehat{C}^+ - \widehat{C}^-) \\ U &= I + \alpha\triangle t(D_\xi^+ \widehat{A}^- + D_\eta^+ \widehat{B}^- + D_\zeta^+ \widehat{C}^- + \widehat{A}^+ + \widehat{B}^+ + \widehat{C}^+) \end{aligned} \tag{8}$$

In the framework of the LU-SGS algorithm, a variety of schemes can be developed by different choices of numerical dissipation models and Jacobian matrices of the flux vectors. It is desirable that the matrix should be diagonally dominant to assure the convergence. Jacobian matrices leading to diagonal dominance are constructed so that '$+$' matrices have nonnegative eigenvalues while '$-$' matrices have nonpositive eigenvalues. For example,

$$\widehat{A}^\pm = \frac{1}{2}[\widehat{A} \pm \rho(\widehat{A})I] \tag{9}$$

and

$$\rho(\widehat{A}) = \kappa\ max[\mid \lambda(\widehat{A}) \mid] \tag{10}$$

where $\lambda(\widehat{A})$ represent eigenvalues of Jacobian matrix $\widehat{A}$ and κ is a constant that is greater than or equal to 1. The diagonal matrix of eigenvalues is

$$\widehat{\Lambda}(\widehat{A}) = \begin{bmatrix} U & 0 & 0 & 0 \\ 0 & U & 0 & 0 \\ 0 & 0 & U + C_\xi & 0 \\ 0 & 0 & 0 & U - C_\xi \end{bmatrix} \tag{11}$$

where C_ξ is the speed of sound, or the pseudospeed of sound in the case of incompressible flow

$$C_\xi = \sqrt{U^2 + \beta(\xi_x^2 + \xi_y^2 + \xi_z^2)} \tag{12}$$

It is interesting to note that the need for block inversions can be eliminated if we use approximate Jacobian matrices of equation (9). Setting $\alpha = 1$ and $\triangle t = \infty$ gives a Newton-like iteration. Then, equation (7) reduces to

$$\begin{aligned} L &= \rho I - \widehat{A}^+_{i-1,j,k} - \widehat{B}^+_{i,j-1,k} - \widehat{C}^+_{i,j,k-1} \\ D &= \rho I \\ U &= \rho I + \widehat{A}^-_{i+1,j,k} + \widehat{B}^-_{i,j+1,k} + \widehat{C}^-_{i,j,k+1} \end{aligned} \tag{13}$$

where

$$\rho = \rho(\widehat{A}) + \rho(\widehat{B}) + \rho(\widehat{C}) \tag{14}$$

The algorithm permits scalar diagonal inversions since

$$Diagonal(L \text{ or } U) = \begin{bmatrix} \rho & 0 & 0 & 0 \\ 0 & \rho & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & \rho \end{bmatrix} \tag{15}$$

Another interesting feature of the present algorithm is that the scheme is completely vectorizable on $i + j + k = constant$ oblique planes of sweep. An oblique plane is illustrated in Figure 1. This potential is achieved by reordering the three-dimensional arrays into two-dimensional arrays, that is,

$$\widehat{Q}(ipoint, iplane) = \widehat{Q}(i, j, k) \tag{16}$$

where *iplane* is the serial number of plane of sweep, and *ipoint* is the address on that plane.

4 Implementation and Results

The flow solver code, INS3D-LU, was developed based on the numerical algorithm described above. The code was used to compute the viscous incompressible flow through a straight square duct. The Reynolds number is 790. In order to study the impact of task granularity on the performance of the code, five grid sizes were used to solve the same problem. These grids are: $41 \times 21 \times 21$ (0.018 million grid points), $63 \times 63 \times 63$ (0.25 million grid points), $127 \times 63 \times 63$ (0.5 million grid points), $101 \times 101 \times 101$ (1 million grid points), and $127 \times 127 \times 127$ (2 million grid points).

The flow solver has two parts: an explicit part, the right hand side (RHS) of equation (7), and an implicit part, the left hand side (LHS) of equation (7). These two parts have different degrees of parallelism. The RHS is highly parallel with no data dependency in the three dimensions. In computing this part, the inner loops were vectorized and the outer loops were multitasked. The dependence analyzer was able to detect parallelism for most loops of the RHS. Other loops were multitasked either by inserting directives to enforce parallelism or by reordering the computations to exploit parallelism. The LHS has a limited degree of parallelism. This part uses two-dimensional arrays of the form given in equation (16) where the first dimension, *ipoint*, is the parallel dimension and the second dimension, *iplane*, is the serial dimension. The value of *ipoint* ranges from 1 to the multiplication of the smallest two dimensions of the domain. For a relatively large problem, this means that there is enough work to combine parallelism with vectorization in the parallel dimension. An autotasking directive was used to stripmine the do loops involved in computing the L and U terms of the LHS.

We multitasked the INS3D-LU code on the Cray Y-MP at NASA Ames Research Center. This machine has eight processors, 128 Mwords of main memory, and 6 nsec clock cycle. Table 1 contains the required memory, the measured execution time per grid point per iteration, the speedup, the parallel efficiency, and the processing rate for the five grid sizes on p processors of the Cray Y-MP, where p is ranging from one to

eight. All timings were measured in a dedicated environment. Speedup was computed by taking the ratio of the time to solve the problem using one processor to the time to solve the same problem using p processors. The parallel efficiency was determined by taking the ratio of the speedup using p processors to p. The processing rate for the single processor cases was measured using the hardware performance monitor of the machine. The single processor rate was multiplied by the speedup to obtain the multitasked rate. As shown in Table 1, the multitasked cases require slightly more memory than the single processor cases. The two million grid point case uses about 90% of the total memory of the machine. The parallel efficiency and processing rate are also shown in Figures 2 and 3 for the five grid sizes.

Figure 2 shows that for a fixed grid size, as the number of processors in use was increased, the amount of work per processor decreased thereby reducing the parallel efficiency of the code. However, for a fixed number of processors, increasing the number of grid points caused an increase in the amount of work per processor, especially in the LHS, thereby improving the performance of the code. Except for the smallest grid, the code has achieved a processing rate of over one Gflops and requires less than one μsec per grid point per iteration on eight processors of the Y-MP. These goals were also achieved on seven processors of the machine for couple cases. Moreover, the parallel efficiency stayed within 80% for all grid sizes except the smallest one. The smallest grid, $41 \times 21 \times 21$, achieved a speedup of only four because of lack of parallelism in the LHS; the largest oblique plane for this grid has only 400 points. The results also show that the $127 \times 63 \times 63$ grid has outperformed the $101 \times 101 \times 101$ grid because the former uses vectors of length 64 or 128, which matches the length of the Cray vector registers, in the RHS. The highest obtained rate was 1.218 Gflops. This is about 46% of the peak performance rate of the machine (2.667 Gflops).

5 Conclusions

A full Navier-Stokes solver for incompressible fluid in three dimensional generalized coordinates was multitasked on the Cray Y-MP. This solver is based on LU factorization and SGS implicit relaxation scheme. It requires less than one μsec per grid point per iteration on eight processors of the Cray Y-MP for all grid sizes except the smallest one. Despite the fact the LUSGS scheme is an implicit scheme with a limited degree of parallelism, a speedup of over seven and a processing rate of over 1.2 Gflops were achieved on the Cray Y-MP using two million grid points. This study also shows that the Cray Y-MP is a well balanced architecture with fast memory and processing units and adequate memory bandwidth. Although this scheme requires scatter and gather operations and variable vector lengths, a reasonable percentage of the peak performance of the machine was achieved for several cases. The next step will be adapting this scheme to massively parallel machines such as the Connection Machine and Intel iPSC/860.

References

1. S. Yoon, D. Kwak, and L. Chang, *LU-SGS implicit algorithm for three-dimensional incompressible Navier-Stokes equations with source term*, AIAA Paper 89-1964-CP, 1989.

2. R. Fatoohi, *Multitasking on the Cray Y-MP: an experiment with a 2-D Navier-Stokes code*, International Journal of High Speed Computing, Vol. 1, No. 3, 1989, pp. 433 – 447.

3. Cray Research, Inc., *UNICOS autotasking user's guide*, SN-2088, 1989.

TABLE 1
INS3DLU on the Cray Y-MP

Grid points (Million)	Proc	Memory (Mwords)	Time/grd-pt/itr (μsec)	speedup	Efficiency (%)	Performance (Mflops)
0.018	1	1.1	6.525	-	-	155
	2	1.2	3.744	1.74	87.1	270
	3	1.2	2.749	2.37	79.1	367
	4	1.2	2.207	2.96	73.9	457
	5	1.2	1.941	3.36	67.2	520
	6	1.2	1.825	3.58	59.6	553
	7	1.2	1.532	4.26	60.9	659
	8	1.2	1.615	4.04	50.5	625
0.25	1	15.1	6.319	-	-	167
	2	15.3	3.331	1.90	94.8	318
	3	15.3	2.256	2.80	93.4	469
	4	15.3	1.736	3.64	91.0	609
	5	15.3	1.432	4.41	88.3	739
	6	15.3	1.236	5.11	85.2	856
	7	15.3	1.060	5.96	85.2	998
	8	15.3	0.968	6.53	81.6	1093
0.5	1	28.1	6.289	-	-	169
	2	28.4	3.283	1.92	95.8	324
	3	28.4	2.210	2.85	94.9	481
	4	28.4	1.688	3.73	93.1	630
	5	28.4	1.385	4.54	90.8	768
	6	28.4	1.184	5.31	88.5	898
	7	28.4	1.024	6.14	87.8	1039
	8	28.4	0.924	6.80	85.0	1151
1.0	1	56.3	6.727	-	-	161
	2	56.7	3.509	1.92	95.9	309
	3	56.7	2.359	2.85	95.1	459
	4	56.7	1.786	3.77	94.2	607
	5	56.7	1.447	4.65	93.0	749
	6	56.7	1.228	5.48	91.3	883
	7	56.7	1.098	6.13	87.5	987
	8	56.7	0.983	6.84	85.5	1102
2.0	1	115.6	6.347	-	-	170
	2	116.2	3.286	1.93	96.6	328
	3	116.2	2.191	2.90	96.6	491
	4	116.2	1.657	3.83	95.7	650
	5	116.2	1.342	4.73	94.6	802
	6	116.2	1.130	5.62	93.6	953
	7	116.2	0.996	6.37	91.0	1081
	8	116.2	0.884	7.18	89.7	1218