Abstract of Paper for the 17th AIAA Computational Fluid Dynamics Conference
June 6-9, 2005, Toronto, Canada

Parallelization of Lower-Upper Symmetric Gauss-Seidel Method for Chemically Reacting Flow

Seokkwan Yoon*, Gabriele Jost** and Sherry Chang**

NASA Ames Research Center
Moffett Field, California 9403

Introduction

Development of technologies for exploration of the solar system has revived an interest in computational simulation of chemically reacting flows since planetary probe vehicles exhibit non-equilibrium phenomena during the atmospheric entry of a planet or a moon as well as the reentry to the Earth. Stability in combustion is essential for new propulsion systems. Numerical solution of real-gas flows often increases computational work by an order-of-magnitude compared to perfect gas flow partly because of the increased complexity of equations to solve. Recently, as part of Project Columbia, NASA has integrated a cluster of interconnected SGI Altix systems to provide a ten-fold increase in current supercomputing capacity that includes an SGI Origin system. Both the new and existing machines are based on cache coherent non-uniform memory access architecture.

Lower-Upper Symmetric Gauss-Seidel (LU-SGS) relaxation method[1] has been implemented into both perfect and real gas flow codes[2-8] including Real-Gas Aerodynamic Simulator (RGAS)[9]. However, the vectorized RGAS code runs inefficiently on cache-based shared-memory machines such as SGI systems. Parallelization of a Gauss-Seidel method is nontrivial due to its sequential nature.

The LU-SGS method has been vectorized on an oblique plane in INS3D-LU code[4] that has been one of the base codes for NAS Parallel Benchmarks[10]. The oblique plane has been called a hyperplane by computer scientists. It is straightforward to parallelize a Gauss-Seidel method by partitioning the hyperplanes once they are formed. Another way of parallelization is to schedule processors like a pipeline using software[11-12]. Both hyperplane and pipeline methods have been implemented using openMP directives. The present paper reports the performance of the parallelized RGAS code on SGI Origin and Altix systems.

Numerical Methods

Let t be time; Q the vector of conserved variables; E, F, and G the convective flux vectors; E_v, F_v, and G_v the flux vectors for the viscous terms. The source term S represents production or destruction of

*NASA Advanced Supercomputing (NAS) Division
**Computer Sciences Corporation

species due to chemical reactions. The three-dimensional Navier-Stokes and species transport equations in generalized curvilinear coordinates (ξ, η, ς) can be written as

$$\partial_t Q + \partial_\xi (E - E_v) + \partial_\eta (F - F_v) + \partial_\zeta (G - G_v) = S \tag{1}$$

The governing equations are integrated in time for both steady and unsteady flow calculations. For steady-state solutions, α is set to 1. 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 second and higher orders.

$$[I + \alpha\Delta t(D_\xi A + D_\eta B + D_\zeta C - H)]\Delta Q = RHS \tag{2}$$

where

$$RHS = -\Delta t[D_\xi (E - E_v) + D_\eta (F - F_v) + D_\zeta (G - G_v) - S] \tag{3}$$

I is the identity matrix and ΔQ denotes the correction. A, B, C, and H are the Jacobian matrices of the convective flux vectors and the source term respectively. Artificial dissipation models augment a piecewise-constant cell-centered finite-volume formulation of the right hand side.[5]

Direct inversion of a large block banded matrix becomes impractical in three dimensions because of the rapid increase of computational work and the large storage requirement. The LU-SGS scheme is one of the approximate factorization methods to alleviate the difficulties in three dimensions. Let subscripts f and s indicate fluid and species transport equations respectively. The loosely-coupled method solves the Navier-Stokes and species transport equations separately but the solutions are updated simultaneously at each time step.

$$LD^{-1}U\Delta Q = RHS \tag{5a}$$

where

$$L_f D_f^{-1} U_f \Delta Q_f = RHS_f \tag{7}$$

$$\begin{aligned} L_f &= I + \alpha\Delta t(D_\xi^- A_f^+ + D_\eta^- B_f^+ + D_\zeta^- C_f^+ \\ &\qquad - A_f^- - B_f^- - C_f^-) \\ D_f &= I + \alpha\Delta t(A_f^+ + B_f^+ + C_f^+ - A_f^- - B_f^- - C_f^-) \\ U_f &= I + \alpha\Delta t(D_\xi^+ A_f^- + D_\eta^+ B_f^- + D_\zeta^+ C_f^- \\ &\qquad + A_f^+ + B_f^+ + C_f^+) \end{aligned} \tag{7b}$$

$$L_s D_s^{-1} U_s \Delta Q_s = RHS_s \tag{8}$$

$$L_s = I + \alpha\Delta t(D_\xi^- A_s^+ + D_\eta^- B_s^+ + D_\zeta^- C_s^+ - A_s^- - B_s^- - C_s^- - H)$$

$$D_s = I + \alpha\Delta t(A_s^+ + B_s^+ + C_s^+ - A_s^- - B_s^- - C_s^-)$$

$$U_s = I + \alpha\Delta t(D_\xi^+ A_s^- + D_\eta^+ B_s^- + D_\zeta^+ C_s^- + A_s^+ + B_s^+ + C_s^+) \qquad (8b)$$

The loosely-coupled partially-implicit scheme includes the source Jacobian term H only in the L_s factor. Solving the equations in a loosely-coupled manner ignores such terms in the Jacobian matrix A, for example, as $\partial E_f / \partial Q_s$ and $\partial E_s / \partial Q_f$.

Parallelization methods

The original vector code ran inefficiently on cache-based systems. First, manual optimization that included array changes enhanced the performance of the serial code greatly. The LU-SGS scheme in the code was vectorized on a hyperplane where $i+j+k=const$. The key element was the conversion of three-dimensional indices (i,j,k) to two-dimensional ones $(ipoint, iplane)$[4].

Once the hyperplane is formed, it is straightforward to parallelize the algorithm by partitioning the plane. The method has the limitation that parallelism is restricted to points within one hyperplane. In order to improve memory access and to alleviate communication related problems, the code has been converted manually to use a canonical ordering. The restructured code improves the serial performance by a factor of two, already a significant speed-up on its own. Then the processors are scheduled like a pipeline on the outermost loop level. Sequential operations in each processor are performed in a cache. This approach exploits partial parallelism in loops that carry dependencies.

Both hyperplane and pipeline codes are parallelized using Computer-Aided Parallelizer and Optimizer (CAPO) parallelization tool for the OpenMP parallelization. This task would have been very time consuming when performed manually, particularly in view of the fact that the code requires sophisticated parallelization techniques such as pipelined thread execution, which is not available via automatic parallelization of the vendor commercial compiler. The rapid tool based parallelization allows for the comparison of different strategies and to choose the most efficient implementation.

The parallelization is non-trivial, since the implementation gives rise to a number of conservative and actual data dependencies. CAPO uses the extensive dependency analysis module of the ParaWise system, and, based on the information resulting from the analysis, inserts OpenMP directives into the source code. The following features of CAPO, which are not available via automatic compiler parallelization, are essential for the efficiency of the parallel code. CAPO provides an extensive set of browsers to allow user interaction for improvements of the generated code. This makes it possible to interactively declare the scope of certain as either shared or private and thereby removing conservatively assumed dependencies, which would inhibit parallelization for the compiler. CAPO optimizes the parallel code by merging the parallelized loops within a routine into a large parallel region. This reduces time spent in overhead to fork and join at the beginning and end of parallel loops.

Preliminary Results

The SGI Origin and Altix shared-memory systems are based on 0.6 GHz RISC and 1.5 GHz Intel Itanium-2 processors respectively. Timings for the serial and the parallel executions were obtained using the -O2 optimization compiler flag during compilation.

In order to investigate the performance of parallel Gauss-Seidel methods for reacting flow, a scramjet problem has been calculated as a test case. While the weight of the oxygen tank exceeds thirty percent of

the total weight of the Space Shuttle at launch, only one percent of the total weight is for the payload. The air-breathing rocket propulsion systems, which consume oxygen in the air, offer clear advantages by making vehicles lighter and more efficient. Fuel-air mixing and rapid combustion are of crucial importance for the success of scramjet engines since the spreading rate of the supersonic mixing layer decreases as the Mach number increases. In our test case, hydrogen fuel is injected transversely to incoming supersonic flow of air. The incoming air speed, pressure and temperature are assumed to be Mach 2, 1 atm and 1,000° K. Gaseous hydrogen is injected at the sonic speed through a hole at the bottom whose non-catalytic wall is cooled at 600° K. The length of combustion chamber is 40 times the diameter of injector. The Reynolds number based on the length is approximately 10^5. A 257 x 257 x 257 structured grid (approximately 17 million points) has been used with symmetric boundary conditions at the top and side walls. Supersonic flow boundary conditions are imposed at the inlet and outlet planes.

Figure 1 compares the parallel efficiency of the pipeline code on SGI Altix and Origin systems. Both systems show a comparable performance up to 16 processors while the efficiency of the Origin are better than the Altix on 32 and 64 processors. However, the Altix appears to outperform the Origin on 128 processors since the speedup of the Origin reaches a plateau at 64 processors. Figure 2 shows the relative speedup of the Altix over the Origin. It is not surprising that the Altix is two to three times faster than the Origin considering the speed of Altix chip is 2.5 times faster than the Origin's. What is interesting is that the best performance of the Altix seems to be at 128 processors. Final manuscript will include a detailed analysis of different parallelization methods.

Summary

Parallelization methods have been implemented for a symmetric Gauss-Seidel relaxation algorithm in conjunction with a loosely-coupled scheme for chemically reacting non-equilibrium flow. Both hyperplane and pipeline methods have been implemented into Real-Gas Aerodynamic Simulator code using openMP directives on cache coherent non-uniform memory access architecture. Performance of the parallelization methods have been demonstrated on SGI Altix and Origin shared memory systems.

References

1. Yoon, S. and Jameson, A., "Lower-Upper Symmetric Gauss-Seidel Method for the Euler and Navier-Stokes Equations," AIAA Journal, Vol. 26, Sept. 1988, pp. 1025-1026.
2. Shuen, J.S. and Yoon, S., "Numerical Study of Chemically Reacting Flows Using an LU-SSOR Scheme," AIAA Journal, Vol. 27, Dec. 1989, pp. 1752-1760.
3. Park, C. and Yoon, S., "Calculation of Real-Gas Effects on Blunt-Body Trim Angles," AIAA Journal, Vol. 30, Apr. 1992, pp. 999-1007.
4. Yoon, S. and Kwak, D., "Three-Dimensional Incompressible Navier-Stokes Solver using Lower-Upper Symmetric Gauss-Seidel Algorithm," AIAA Journal, Vol. 29, June 1991, pp. 874-875.
5. Yoon, S. and Kwak, D., "Multigrid Convergence of an Implicit Symmetric Relaxation Scheme," AIAA Journal, Vol. 32, May 1994, pp. 950-955.
6. Chen, C.L., McCroskey, W.J., and Obayashi, S., "Numerical Solutions of Forward-Flight Rotor Flow using an Upwind Method," AIAA Paper 89-1846, June 1989.
7. Soetrisno, M., Imlay, S.T., and Roberts, D.W., and Taflin, D.E., "Development of a 3-D Zonal Implicit Procedure for Hybrid Structured-Unstructured Grids," AIAA Paper 96-0167, Jan. 1996.
8. Sharov, D. and Nakahashi, K., "Reordering of 3-D Hybrid Unstructured Grids for Vectorized LU-SGS Navier-Stokes Computations," AIAA Paper 97-2102, June 1997.
9. Yoon, S., "Calculation of Supersonic Combustion using Implicit Schemes," AIAA Paper 2003-3546, June 2003, AIAA Journal (to appear).
10. Bailey, D., Barton, J., Lasinski, T., and Simon, H., "The NAS Parallel Benchmarks," NAS TR-91-002, Jan. 1991.
11. Van der Wijngaart, R., Private communications
12. Jin, H., Private communications

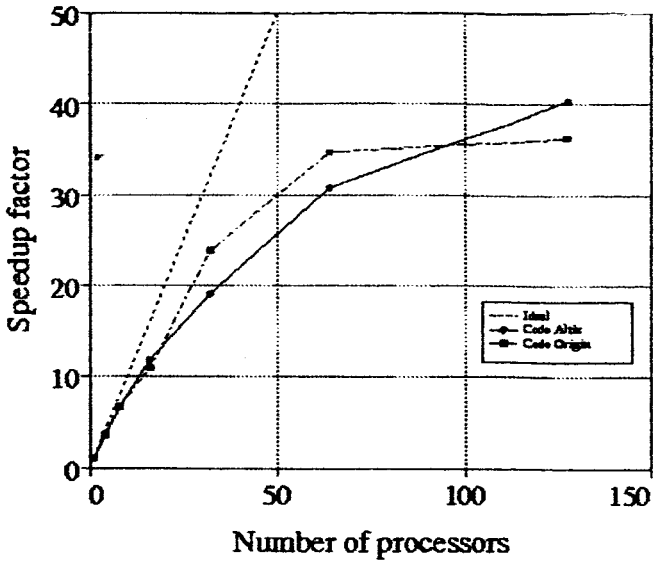

Fig. 1. Parallel Efficiency of Altix and Origin

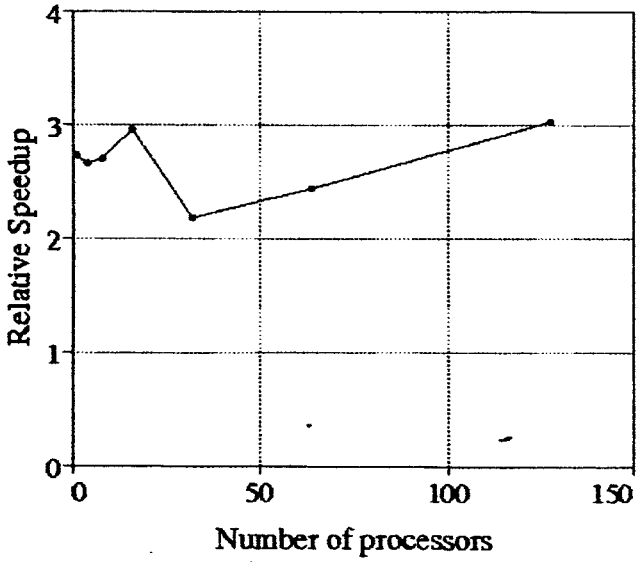

Fig. 2. Relative Speedup of Altix over Origin