FINAL REPORT

NASA Contract No. NAG1-1672 (Jan. 1995-Aug. 97)

Amount: $60,000

Contract Entitled:

Efficient Parallel Kernel Solvers for

Computational Fluid Dynamics Applications

Principal Investigator

Xian-He Sun, Ph.D.

Department of Computer Science

Louisiana State University

Baton Rouge, LA 70803-4020

(504) 388-3197, sun@bit.csc.lsu.edu

Summary

This project investigated the design, development and testing of new parallel numerical algorithms for solving computational intensive kernels arising in Computational Fluid Dynamics (CFD) applications. The architectures under consideration are multiple instruction multiple data (MIMD), distributed-memory machines and clusters of workstations.

List of Publications Supported by the Award

[1] "Performance Comparison of a Set of Periodic and Non-Periodic Tridiagonal Solvers on SP2 and Paragon Parallel Computers," (with S. Moitra), in *Concurrency: Practice and Experience*, pp.1-21, Vol.8(10), 1997.

[2] "Limitations of Cycle Stealing of Parallel Processing on a Network of Homogeneous Workstations," (with S. Leutenegger), in *Journal of Parallel and Distributed Computing*, Vol. 43, No. 2, pp.169-178, 1997.

[3] "Performance Prediction: A case study using a shared-virtual-memory machine," (with J. Zhu), *IEEE Parallel & Distributed Technology*, pp. 36-49, Winter 1996.

[4] "A Simple Parallel Prefix Algorithm for almost Toeplitz Tridiagonal Systems," (with D. Joslin), in *International Journal of High Speed Computing*, Vol.7, No.4, pp. 547-576, Dec. 1995.

[5] "Performance Considerations of Shared Virtual Memory Machines," (with J. Zhu), in *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, pp. 1185-1194, Nov. 1995.

[6] "Application and Accuracy of the Parallel Diagonal Dominant Algorithm," in *Parallel Computing*, Vol. 21, pp. 1241-1267, August 1995.

[7] "A Three-Level Parallelization of a Spatial Direct Numerical Simulation," (with Q. Hou), accepted to appear in *International Journal on Advances in Engineering Software.*

[8] "A Highly Accurate Fast Solver for Helmholtz Equations." (with Y. Zhuang), in *Proc. of the ACM International Conference on Supercomputing*, July, 1997. The long version of this paper is available as an ICASE Technical Report (No. 97-11).

[9] "A Simulation Study of Packed Exponential Connection Network," (with X. Liao), in *Proc. of the Int'l Conf. on Parallel and Distributed Computing Systems*, Oct. 1997.

[10] "Parallel Implementation of a Data-Transpose Technique for the Solution of Poisson's Equation in Cylindrical Coordinates," (with H. Cohl, and J. Tohline), in *Proc. of the SIAM Conference on Parallel Processing for Scientific Computing*, March, 1997.

[11] "MpPVM: A Software System for Non-Dedicated Heterogeneous Computing," (with K. Chanchio), in *Proc. of the International Conference on Parallel Processing*, August, 1996.

[12] "The Relation of Scalability and Execution Time," in *Proc. of the International Parallel Processing Symposium'96*, April, 1996.

Contributions to Education Two MS graduate students (Mr. Q. Hou and Mr. Y. Zhuang) and one Ph.D. student (Mr. K. Cameron) have been partially supported in this project and two papers based on their work have been published (see [7] [8]).

1 Introduction

Distributed-memory parallel computers dominate today's parallel computing arena. These machines, such as Intel Paragon, IBM SP2, and Cray Origin200, have successfully delivered high performance computing power for solving some of the so-called "grand-challenge" problems [1]. Despite initial success, parallel machines have not been widely accepted in production engineering environments due to the complexity of parallel programming. On a parallel computing system, a task has to be partitioned and distributed appropriately among processors to reduce communication cost and to attain load balance. More importantly, even with careful partitioning and mapping, the performance of an algorithm may still be unsatisfactory, since conventional sequential algorithms may be serial in nature and may not be implemented efficiently on parallel machines. In many cases, new algorithms have to be introduced to increase parallel performance. In order to achieve optimal performance, in addition to partitioning and mapping, a careful performance study should be conducted for a given application to find a good algorithm-machine combination. This process, however, is usually painful and elusive.

The goal of this project is to design and develop efficient parallel algorithms for highly accurate Computational Fluid Dynamics (CFD) simulations and other engineering applications. The work plan is 1) developing highly accurate parallel numerical algorithms, 2) conduct preliminary testing to verify the effectiveness and potential of these algorithms, 3) incorporate newly developed algorithms into actual simulation packages. The work plan has well achieved. Two highly accurate, efficient Poisson solvers have been developed and tested based on two different approaches: 1. Adopting a mathematical geometry which has a better capacity to describe the fluid, 2. Using compact scheme to gain high order accuracy in numerical discretization. The previously developed Parallel Diagonal Dominant (PDD) algorithm and Reduced Parallel Diagonal Dominant (RPDD) algorithm have been carefully studied on different parallel platforms for different applications, and a NASA simulation code developed by Man M. Rai and his colleagues has been parallelized and implemented based on data dependency analysis. These achievements are addressed in details in the next four sections.

2 Cylindrical Coordinate System for Rotating Fluids

Rotating fluids are often encountered in CFD applications. While Cartesian coordinate system is the "standard" geometry for numerical computation, representing a rotating object in a Cartesian coordinates results in degradation in the spatial resolution and introduces numerical in-accuracies due to attempting to resolve circular flows in an inherently non-circular geometry. A Cylindrical coordinate system is the natural coordinate system for rotating fluids.

We have developed a parallel method to solve Poisson's equation

$$\nabla^2 \Phi = f, \tag{1}$$

in Cylindrical coordinates (r, θ, z), where Φ is the potential, f is the source function, and ∇^2 denotes the Laplacian operator. All quantities are expressed in dimensionless form. We solve Eq. (1) on a uniformly zoned mesh.

This newly proposed algorithm takes full advantage of the periodic boundary condition that naturally arises in the azimuthal coordinate direction by performing a discrete Fourier transform in the azimuthal direction. Uniform zoning and Dirichlet boundary conditions in the vertical coordinate direction then is exploited by additionally performing a discrete *sine* transform in the vertical direction. This approach reduces the second-order-accurate, 7-point finite-difference equation to a large set of 3-point (tridiagonal) finite-difference equations which are then solved in parallel. A analytical comparison of this new method with other existing methods has been conducted. Analysis and preliminary experimental results show that the new algorithm is efficient on both vector and massively parallel machines. The success of the algorithm reduces the burden of using Cylindrical coordinates in an engineering environment and may make Cylindrical coordinates a popular choice in simulating rotating fluids.

2.1 The Algorithm

Our method of solution incorporates the above described general techniques in the following sequence:

1. A one-dimensional FFT is performed, in the azimuthal coordinate direction, on the MSF (Modified Source Function) at each radial and vertical location.
2. The *transpose3d* subroutine (see below) is called to align the vertical coordinate direction of the MSF with the internal memory of each PE (Processing Element).
3. A one-dimensional FST is performed in the vertical coordinate direction on the MSF at each radial and Fourier mode location.
4. The *transpose3d* subroutine is called to align the radial coordinate direction of the MSF with the internal memory of each PE.
5. Independently for each Fourier mode and each *sine* mode location, a direct solution is obtained for a set of linear-algebra equations This is accomplished via a straightforward solution of the 1D tridiagonal matrix problem using LU decomposition with forward- and back-substitution.
6. The *transpose3d* subroutine is called to align the *sine* mode direction of $\phi^i_{m,k'}(j)$ with the internal memory of each PE.
7. A one-dimensional inverse FST is performed in the *sine* mode direction on $\phi^i_{m,k'}(j)$ to obtain $\phi^i_m(j,k)$.
8. The *transpose3d* subroutine is called to align the Fourier mode direction of $\phi^i_m(j,k)$ with the internal memory of each PE.
9. A 1D inverse FFT is performed in the Fourier mode direction on $\phi^i_m(j,k)$, to obtain the coordinate-space solution $\Phi(j,k,l)$.

transpose3d is a subroutine provided by MasPar for 3-D matrix transform. Steps $1, 3, 5, 7$, and 9 are embarrassingly parallel and there is no communication needed. Steps $2, 4, 6$, and 8 contain zero computation with 100% communication, but in each of these cases the communication is partially parallelizable since communication only occurs in one of the MasPar grid directions.

2.2 Experimental Results

Our Poisson solver was developed on an 8,192-node (128×64 grid of processors) MasPar MP-1. The MasPar MP-1 is a distributed-memory, SIMD computer in which a 2D (X, Y) grid of processor elements (PEs) comprises the primary computing engine of the machine. This machine is configured such that each PE contains 64 KBytes of local dynamic RAM (DRAM); hence, the 8K-node system contains 0.5 GBytes. On MasPar systems, each of the PEs has been crafted so that it communicates most efficiently with its eight nearest neighbors via X-net hardware. Communication beyond nearest neighbors can be handled via a somewhat slower, global router network [2].

Our implementation is efficient, in part, because it exploits the particular way that the data is laid out across the distributed-memory, MasPar platform [3]. In particular, we have defined our 3D arrays so that, initially, our radial coordinate direction coincides with the X-processor grid; our vertical coordinate direction coincides with the Y-processor grid; and our azimuthal coordinate direction aligns with the internal memory of each PE. Hence, if the size of our (r, z) computational grid is at most equal to the size of the (X, Y) grid, each PE has stored in its own local DRAM only one "ring" of azimuthal data corresponding to a particular (r, z) location in our computational grid. The algorithm is not constrained to dimensions of this size, however.

The timings of the new parallel algorithm are given in Table 1. In this table we list the size of the computational grid, the corresponding timings on the MasPar MP-1 for the implementation of the newly proposed method, the corresponding timings on the MasPar MP-1 for the traditional ADI implementation, the speedup we have obtained from the proposed method over the ADI implementation and timings on a Cray C90. In examining this data, one should note that our code is written in *mpf* which automatically takes full advantage of all 8K processors available. Speedup of the proposed method over the ADI method increases with the problem size and shows that the proposed method has a better problem size scalability than that of the ADI method. Finally, the proposed method achieves a better performance on the MasPar than the ADI method on a Cray C90, which demonstrates a complete success of massively parallel computing.

As a demonstration of the overall efficiency of our algorithm, we present in Table 2 the timings for 3D array transpositions, FST, FFT, and tridiagonal inversion on the MP-1 for all the grid sizes listed in Table 1. Notice that the 3D transposition has to be performed four times during the solution process. From Table 2 we can see that the ratio of communication time to total execution time is reduced significantly when the problem size is increased. Communication cost contributes more than 30% of the total execution time when the grid size is 32^3, and less than 3% of the total execution time when the problem size is 128^3.

We are currently using a modified version of the Fortran 77 subroutine given in [4] for the FST.

grid size	MP-1, FST	MP-1, ADI	Speedup	C90, ADI
32^3	0.384	2.29	5.96	-
64^3	0.645	4.52	7.01	1.48
128^3	2.16	17.7	8.18	-

Table 1. Comparison : Execution Times in Seconds

grid size	FFT	*Sine* Trans.	Trid. Sol.	Transpose3d
32^3	1.12×10^{-2}	5.02×10^{-2}	2.19×10^{-2}	1.96×10^{-2}
64^3	2.22×10^{-2}	1.13×10^{-1}	4.56×10^{-2}	2.24×10^{-2}
128^3	1.17×10^{-1}	4.10×10^{-1}	1.50×10^{-1}	2.84×10^{-2}

Table 2. Internal Comparison of the Proposed Algorithm

This subroutine has not been optimized for the MasPar MP-1 architecture and is thus slower than the MasPar specific FFT subroutine used in the ADI method. A further speedup is expected when we implement the MasPar specific FFT subroutines into the code for the FST.

3 Compact Scheme for Highly Accurate Discretization

Since its introduction, compact scheme has been mainly used to approximate derivatives of a function when the values of the function are known. In fact, the original compact scheme can be used to discretize 1-Dimensional (1-D) operators like time-dimensional derivatives in a differential equation, but it cannot necessarily approximate spatial differential operators in Partial Differential Equations (PDEs). The reason of the limitation lies in that to approximate high dimensional differential operators, like the 2-Dimensional (2-D) Laplacian operator $\triangle = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, one cannot use 1-D compact scheme to approximate each of its 1-D component when the function values are unknown. For example, to discretize a 2-D Poisson equation in Cartesian coordinates system

$$P_{xx} + P_{yy} = R(x, y), \tag{2}$$

if we use the original one dimensional compact schemes to approximate P_{xx} and P_{yy} respectively, we will get

$$\begin{aligned} &\tfrac{1}{12}P_{xx}^{i-1,j} + \tfrac{5}{6}P_{xx}^{i,j} + \tfrac{1}{12}P_{xx}^{i+1,j} + \tfrac{1}{12}P_{yy}^{i,j-1} + \tfrac{5}{6}P_{yy}^{i,j} + \tfrac{1}{12}P_{yy}^{i,j+1} \\ &\qquad = \tfrac{1}{h^2}\left(P^{i-1,j} + P^{i+1,j} + P^{i,j+1} + P^{i,j-1} - 4P^{i,j}\right). \end{aligned} \tag{3}$$

The right-hand-side of Eq.(3) contains the unknown which we want to solve. However, the left-hand-side of Eq.(3) is not a linear combination of the Laplacians operator $\triangle P$. Equation (2) and (3) cannot be combined and manipulated to form a solvable linear equation. Current results

on compact schemes cannot be applied directly to 2-D PDEs directly. Investigation needs to be conducted to apply compact schemes on high dimensional PDE operators.

We have developed a 2 dimensional, 4-th order accurate compact scheme for the discretization of Poisson equations. Based on this discretization, an efficient algorithm is introduced, which achieves 4-th order accuracy with the same computation complexity as the best existing 2nd order accurate algorithm. What is more, the new algorithm is designed for Neumann boundary conditions which are hard to solve compared with Dirichlet boundary conditions. By analysis, this new algorithm is N times faster than the conventional 4-th order accurate method, where N is number of grid points on each dimension.

3.1 The Fourth-Order Fast Solver

After some innovative discretizations for boundary conditions, a Helmholtz equation with Neumann boundary conditions can be incorporated into the following linear system:

$$\mathbf{A}\ P = -\frac{3}{2}h^2 R - \frac{h^4}{8}\left(\lambda R + \triangle R\right) + 2hU\ + O(h^5)\ . \tag{4}$$

After tridiagonalization, equation (4) becomes

$$h^{-2}\mathbf{F}\mathbf{\Lambda}\mathbf{F}^{-1}P = B\ , \tag{5}$$

where $B = -\frac{3}{2}R - \frac{h^2}{8}\left(\lambda R + \triangle R\right) + 2h^{-1}U$, and $\mathbf{A} = \mathbf{F}\mathbf{\Lambda}\mathbf{F}^{-1}$. The fourth-order algorithm is carried out in the following steps:

1) Compute the cosine values to be used in the FCT.

2) Compute the value of each entry in matrix $\mathbf{\Lambda}$.

3) Compute the vector U in equation (4).

4) Compute the right-hand side of (5).
This is done by adding the results from step 1) to $-\frac{3}{2}R - \frac{h^2}{8}\left(\lambda R + \triangle R\right)$ which is to be computed in this step.

5) Apply inverse FCT to the right-hand side of equation (5).
This is done by multiplying matrix $\mathbf{F}_1^{-1}$ to each block B_i in the matrix B, for $i = 0, 1, ..., M$, where $B_i = (b_{i0},\ b_{i1},\ \ldots,\ b_{iN})^T$ is the i-th N-component-long block of vector B.

6) Solve the tridiagonal system $\mathbf{\Lambda}Y = \mathbf{F}^{-1}B$ for Y, where $Y = \mathbf{F}^{-1}P$.
Because of the particular structure of matrix $\mathbf{\Lambda}$, we get N independent tridiagonal systems, each of size M. Reassemble the right-hand side $\mathbf{F}^{-1}B$ according to the structure of $\mathbf{\Lambda}$, and then solve the N tridiagonal systems.

7) Apply FCT to vector Y computed in step 6) to recover P.
This is done by multiplying matrix $\mathbf{F}_1$ to each of vector Y's N-component-long block Y_i for $i = 0, 1, ..., M$.

The operation count of each step of the fourth-order algorithm on a square domain of $N \times N$, from steps 1 to 7, is:

1. N multiplications and $0.5N$ additions;
2. $6N$ multiplications and $2N$ additions;
3. $36N$ multiplications and $48N$ additions;
4. $N^2 + 10N$ multiplications and $4N^2 + 12N$ additions;
5. $N^2 \log_2 N + N(\log_2 N - 4)$ multiplications and $N^2(1.5 \log_2 N + 1.5) - N \log_2 N$ additions;
6. $5N^2 + 6N$ multiplications and $3N^2 + 4N$ additions;
7. $N^2 \log_2 N + N(\log_2 N - 3)$ multiplications and $N^2(1.5 \log_2 N + 2.5) - N \log_2 N$ additions.

The total operation count of the algorithm is the sum of the work of the seven steps, which is

$$\begin{aligned} &N^2(6 + 2\log_2 N) + N(2\log_2 N + 52) \text{ multiplications and} \\ &N^2(3\log_2 N + 11) + N(-2\log_2 N + 67) \text{ additions} \\ &= N^2(5\log_2 N + 17) + 119N. \end{aligned} \tag{6}$$

It is $N^2 + 45N$ multiplications and $4N^2 + 53N$ additions more than the operation count of the conventional second-order fast Poisson solver [5].

The above computation analysis is based on N, the reciprocal of mesh size. Second order algorithms need smaller mesh sizes than that of 4-th order algorithms to achieve the same accuracy. Therefore, they are slow in high performance computing practice compared with 4-th order algorithms.

Let T_4 and T_2 denote the time needed by our 4-th order and the conventional order 2 methods respectively to solve a problem within a given error tolerance. The following equation shows the execution time difference of these two methods [6].

$$T_2 : T_4 = \frac{C^2N^4(5\log_2 CN^2 + 12) + 21C^2N}{N^2(5\log_2 N + 17) + 119N}. \tag{7}$$

For "well-behaved" equations (e.g. the derivatives of different orders differ within a factor of 5 and λ is moderately small), the parameter C assumes a range of $[\frac{1}{4}, 1]$. For these "well behaved" equations, the time ratio $T_2 : T_1$ will fall between $\frac{1}{10}N^2$ and $2N^2$ by formula (7). In other words, the conventional second order solver will take roughly $\frac{1}{10}N^2$ to $2N^2$ times more computation than that of the newly-proposed fourth order solver to reach the same accuracy. The difference is huge. The

newly proposed algorithm is unparalleled faster than other existing direct solvers when a fourth order accurate solution is wanted.

A companion algorithm is also derived in [6] for Helmholtz equations with Neumann-Dirichlet conditions.

3.2 Experiment Results

By definition, a numerical method is fourth order accurate if and only if when the number of grid points doubles the discretization error will decrease at a rate of $\left(\frac{1}{2}\right)^4$. Five Helmholtz equations with known exact solutions have been chosen as test problems to verify the analytical results and to illustrate the performance gain of the high order method [6]. Among the five test problems, two have polynomial solutions, with one having a fractional degree. The other three are with solutions of sine-exponential function, polynomial-cosine function, and two dimensional cosine function, respectively. They represent a large class of practical Helmholtz equations. Experimental tests have been conducted on a DEC Station 5000 to measure the numerical results and execution time. As listed in Tables 3 to 8, experimental results match analytical results closely. Measured performance confirms that the newly proposed algorithm is highly accurate and efficient.

Table 3. Solving $P_{xx} + P_{yy} - 11P = R$ on Unit Square with $P = e^x sin(y)$

Fourth Order							
N =	8	16	32	64	128	256	512
Maximal error	1.10E-04	7.30E-06	4.67E-07	2.95E-08	1.85E-09	1.16E-10	9.40E-12
Relative error	4.22E-05	2.57E-06	1.58E-07	9.81E-09	6.11E-10	3.80E-11	3.22E-12
Order		3.9	4.0	4.0	4.0	4.0	3.6
Time(seconds)	0.004	0.016	0.066	0.27	1.20	5.3	24
Second Order							
N =	8	16	32	64	128	256	512
Maximal error	1.94E-03	4.94E-04	1.24E-04	3.10E-05	7.76E-06	1.94E-06	4.85E-07
Relative error	7.22E-04	1.69E-04	4.06E-05	9.93E-06	2.46E-06	6.11E-07	1.52E-07
Order		2.0	2.0	2.0	2.0	2.0	2.0
Time(seconds)	0.004	0.012	0.066	0.26	1.13	5.2	24

Two performance metrics are used in the measurement. The metric **Order**, defined by

$$\text{Order(n, n+1)} = \log_2 \frac{MaxE(n)}{MaxE(n+1)},$$

measures the order of accuracy of numerical solutions. The term **Relative error** is defined as

$$Relative\ error = \frac{\|P - \widehat{P}\|_1}{\|P\|_1},$$

where P and $\widehat{P}$ denote the exact solution and computed solution respectively, and $\|\cdot\|_1$ is the l_1

Table 4. Solving $P_{xx} + P_{yy} = R$ on Unit Square with $P = cos(xy)$

Fourth Order							
N =	8	16	32	64	128	256	512
Maximal error	5.26E-05	2.90E-06	1.66E-07	9.88E-09	5.99E-10	3.70E-11	1.49E-12
Relative error	9.32E-06	5.00E-07	2.95E-08	1.79E-09	1.11E-10	6.91E-12	1.71E-13
Order		4.2	4.1	4.1	4.0	4.0	4.7
Time(seconds)	0.004	0.012	0.062	0.28	1.20	5.18	23
Second Order							
N =	8	16	32	64	128	256	512
Maximal error	8.15E-04	2.01E-04	5.00E-05	1.25E-05	3.12E-06	7.80E-07	1.95E-07
Relative error	1.11E-04	2.53E-05	6.05E-06	1.48E-06	3.67E-07	9.12E-08	2.27E-08
Order		2.0	2.0	2.0	2.0	2.0	2.0
Time(seconds)	0.004	0.016	0.062	0.28	1.13	5.04	23

norm. All the testing is conducted on the unit square domain $[0,1] \times [0,1]$ with the same uniform mesh size h on each dimension. $N = 1/h$ is the number of grid points on each x- and y-dimension.

Tables 3 to 6 present the time-accurate comparison between the new fourth-order fast solver and the traditional second-order fast solver, for the four testing problems with Neumann-Neumann boundary conditions. Measured experimental results show our new method is indeed fourth order accurate and achieves the high order accurate solution without increasing execution time, as compared with the conventional second-order fast solver.

Table 7 lists measured experimental results for a special Poisson equation whose solution is a polynomial of degree four. The measured performance shows that the fourth-order direct Neumann solver gives the exact solution while the second order method cannot reach high accuracy even with enlarged problem size and with extended execution time.

Table 8 compares the measured execution time of the conventional second-order faster Poisson solver and the newly-proposed fourth-order solver. The testing problems are solved by the fourth-order algorithm. Then, the same problems are solved with the conventional second order method to match the achieved accuracy with increased number of grid points and execution time. The execution times of the fourth order algorithm and the second order algorithm are listed side-by-side in Table 8 for each of the testing problems. Table 8 shows that the new method is 300 to 1500 times faster, as indicated by the column of time ratio for the two solvers. Notice that the performance gain increases largely when the problem size increase with the problem domain. This time ratio increase is no surprise. It is around $N^2/4$ and is well predicted by the range $[\frac{1}{10}N^2, 2N^2]$ given in Section 3.1. The new algorithm is well suitable for scalable computing where problem size increases with the computational power.

Experimental results also provided in [6] for corresponding Neumann-Dirichlet boundary conditions. As confirmed by the measured results, solutions of Neumann-Dirichlet problems are also of fourth order accurate. In addition, they even have a smaller error than that of the Neumann-

Table 5. Solving $P_{xx} + P_{yy} - P = R$ on Unit Square with $P = (x^3 - x^2)cosy$

Fourth Order							
N =	8	16	32	64	128	256	512
Maximal error	7.73E-05	4.90E-06	3.07E-07	1.92E-08	1.20E-09	7.50E-11	4.64E-12
Relative error	5.72E-04	3.12E-05	1.83E-06	1.10E-07	6.79E-09	4.21E-10	2.63E-11
Order		4.0	4.0	4.0	4.0	4.0	4.0
Time(seconds)	0.008	0.016	0.086	0.269	1.24	5.30	24
Second Order							
N =	8	16	32	64	128	256	512
Maximal error	6.86E-03	1.72E-03	4.31E-04	1.08E-04	2.69E-05	6.73E-06	1.68E-06
Relative error	5.46E-02	1.21E-02	2.83E-03	6.85E-04	1.68E-04	4.18E-05	1.04E-05
Order		2.0	2.0	2.0	2.0	2.0	2.0
Time(seconds)	0.004	0.012	0.059	0.262	1.14	5.02	24

Neumann boundary problems, since there is no discretization error arising along x-dimensional Dirichlet boundary conditions. This feature is also well matched by our experiment results.

4 Testing of Tridiagonal Solvers

Solving tridiagonal systems is a basic computational kernel of many scientific applications. Tridiagonal systems appear in multigrid methods, the Alternating Direction Implicit (ADI) method, the wavelet collocation method, and in line-SOR preconditioners for conjugate gradient methods [7]. In addition to solving PDE's, tridiagonal systems also arise in digital signal processing, image processing, stationary time series analysis, and in spline curve fitting. Because of its importance, intensive research has been carried out on the development of efficient parallel tridiagonal solvers. One direct motivation in developing efficient kernels for solving tridiagonal systems at NASA is that the implicit systems of compact schemes [8], which are relatively new finite difference schemes widely used in production codes at NASA Langley and Ames research center, are tridiagonal systems.

Three parallel tridiagonal solvers, namely the PDD, the Reduced PDD, and the PPT, have been carefully tested and compared during the funding period. The PDD and Reduced PDD algorithm are two newly proposed algorithms [7]. They are developed based on a new approach, introducing bounded perturbation if necessary, to reduce communication and computation for optimal performance. The PDD algorithm, designed for strictly diagonally dominant problems, is very efficient in communication. The Reduced PDD algorithm maintains the minimum communication provided by the PDD algorithm and has a reduced operation count. It has a smaller operation count than the conventional sequential algorithm for many applications. The PPT algorithm's [9] computation and communication complexity is similar to the widely used Wang's algorithm [10]. It also has a substructure similar to the algorithm of Lawrie and Sameh [11]. Its performance can be used as a good representation of existing algorithms. While the sequential algorithm requires more opera-

Table 6. Solving $P_{xx} + P_{yy} - 24.75P = R$ on Unit Square with $P = x^{5.5} + y^{5.5}$

Fourth Order							
N =	16	32	64	128	256	512	1024
Maximal error	4.75E-04	3.46E-05	2.32E-06	1.50E-07	9.55E-09	6.01E-10	3.84E-11
Relative error	3.45E-04	2.50E-05	1.67E-06	1.07E-07	6.81E-09	4.28E-10	2.67E-11
Order		3.8	3.9	4.0	4.0	4.0	4.0
Time(seconds)	0.016	0.059	0.285	1.24	5.30	23	102
Second Order							
N =	16	32	64	128	256	512	1024
Maximal error	1.82E-02	4.61E-03	1.15E-03	2.89E-04	7.22E-05	1.81E-05	4.51E-06
Relative error	8.34E-03	2.09E-03	5.20E-04	1.30E-04	3.24E-05	8.09E-06	2.02E-06
Order		2.0	2.0	2.0	2.0	2.0	2.0
Time(seconds)	0.016	0.055	0.281	1.13	5.05	25	97

Table 7. Solving $P_{xx} + P_{yy} = R$ on Unit Square with $P = x^4 + y^4$

Method	Fourth Order	Second Order	Second Order
N =	8	256	1024
Maximal error	6.66E-16	3.05E-05	1.91E-06
Relative error	1.09E-15	2.53E-05	1.59E-06
Time(seconds)	0.004	5.04	100

tions for solving periodic systems, the three parallel algorithms basically have the same operation count for solving periodic and non-periodic systems. An outline of the testing results of these three algorithms is given below.

4.1 Operation Comparison

Table 9 gives the computation and communication count of the tridiagonal solvers under consideration for solving non-periodic systems. Tridiagonal systems arising in many applications are multiple right-hand-side (RHS) systems. They are usually "kernels" in much larger codes. The computation and communication counts for solving multiple RHS systems are listed in Table 9, where factorization of the matrix is not considered. Parameter $n1$ is the number of RHS. Note that for multiple RHS systems, the communication cost increases with the number of RHS. For the PPT algorithm, the communication cost also increases with the ensemble size. The best sequential algorithm is the conventional Thomas algorithm [12], the LU decomposition method for tridiagonal systems.

Communication cost has a great impact on overall performance. For most distributed-memory computers, communication time with nearest neighbors is found to vary linearly with the problem size. Let S be the number of bytes to be transferred. Then the transfer time to communicate with

Table 8. Computation time of the two methods for the same accuracy

Problem	Method	N	Maximal error	Time(seconds)	Time ratio
1	Order 4	32	4.67E-07	0.066	
	Order 2	512	4.85E-07	24	364
1	Order 4	64	2.95E-08	0.27	
	Order 2	2048	3.03E-08	423	1570
2	Order 4	32	1.66E-07	0.062	
	Order 2	512	1.95E-07	24	389
2	Order 4	64	9.88E-09	0.28	
	Order 2	2048	1.22E-08	423	1510
3	Order 4	16	4.90E-06	0.016	
	Order 2	256	6.73E-06	5.02	314
3	Order 4	32	3.07E-07	0.062	
	Order 2	1024	4.21E-07	97	1560
4	Order 4	32	3.46E-05	0.059	
	Order 2	512	1.81E-05	25	424
4	Order 4	64	2.32E-06	0.285	
	Order 2	2048	1.13E-06	422	1480

a neighbor can be expressed as $\alpha + S\beta$, where α is a fixed startup time and β is the incremental transmission time per byte. Assuming 4 bytes are used for each real number, the PDD and Reduced PDD algorithm take $\alpha+8\beta$ and $\alpha+4\beta$ time respectively on any architecture which supports single array topology. The PPT algorithm requires a total-data-exchange communication. Communication cost of the total-data-exchange depends greatly on the architecture present. The listed communication cost of the PPT algorithm, in Table 9 and 10, is based on a square 2-D torus with p processors (i.e. 2-D mesh, wrap-around, square) [3]. With a hypercube or multi-stage Omega network connection, the communication cost would be $\log(p)\alpha+12(p-1)\beta$ and $\log(p)\alpha+8(p-1)n1\cdot\beta$ for single systems and systems with multiple RHS, respectively [9, 13].

If boundary conditions are periodic, tridiagonal systems arising in scientific applications are periodic tridiagonal systems. Computation and communication counts for solving periodic systems are listed in Table 10. The conventional sequential algorithm used is the periodic Thomas algorithm [12]. Compared with Table 9, we can see that while the best sequential algorithm has an increased operation count, the parallel algorithms have the same operation and communication counts for both periodic and non-periodic systems. The only exception to this observation is the PPT algorithm which has a slightly increased operation count. For the PDD and Reduced PDD algorithm, however, the communication is given for any architecture which supports Ring communication instead of 1-D array. Notice that when $j < n/2$, the Reduced PDD algorithm has a smaller operation count than that of Thomas algorithm for periodic systems with multiple RHS.

System	Algorithm	Computation	Communication
Single system	best sequential	$8n-7$	0
	the PPT	$17\frac{n}{p}+16p-23$	$(2\alpha+8p\beta)(\sqrt{p}-1)$
	the PDD	$17\frac{n}{p}-4$	$2\alpha+12\beta$
	the Reduced PDD	$11\frac{n}{p}+6j-4$	$2\alpha+12\beta$
Multiple right sides	best sequential	$(5n-3)\cdot n1$	0
	the PPT	$(9\frac{n}{p}+10p-11)\cdot n1$	$(2\alpha+8p\cdot n1\cdot\beta)(\sqrt{p}-1)$
	the PDD	$(9\frac{n}{p}+1)\cdot n1$	$(2\alpha+8n1\cdot\beta)$
	the Reduce PDD	$(5\frac{n}{p}+4j+1)\cdot n1$	$(2\alpha+8n1\cdot\beta)$

Table 9. Comparison of Computation and Communication (Non-Periodic)

System	Algorithm	Computation	Communication
Single system	best sequential	$14n-16$	0
	the PPT	$17\frac{n}{p}+16p-7$	$(2\alpha+8p\beta)(\sqrt{p}-1)$
	the PDD	$17\frac{n}{p}-4$	$2\alpha+12\beta$
	the Reduced PDD	$11\frac{n}{p}+6j-4$	$2\alpha+12\beta$
Multiple right sides	best sequential	$(7n-1)\cdot n1$	0
	the PPT	$(9\frac{n}{p}+10p-3)\cdot n1$	$(2\alpha+8p\cdot n1\cdot\beta)(\sqrt{p}-1)$
	the PDD	$(9\frac{n}{p}+1)\cdot n1$	$(2\alpha+8n1\cdot\beta)$
	the Reduce PDD	$(5\frac{n}{p}+4j+1)\cdot n1$	$(2\alpha+8n1\cdot\beta)$

Table 10. Comparison of Computation and Communication (Periodic)

4.2 Experimental Results

The PDD and the Reduced PDD algorithms were implemented on the 48-node IBM SP2 and 72-node Intel Paragon available at NASA Langley Research Center. Both the SP2 and Paragon machines are distributed-memory parallel computers which adopt message-passing communication paradigms and support virtual memory. Each processor (node) of the SP2 is either functionally equivalent to a RISC System/6000 desktop system (thin node) or a RISC System/6000 deskside system (wide node). The Paragon XP/S supercomputer uses the i860 XP microprocessor which includes a RISC integer core processing unit and three separate on-chip caches for page translation, data, and instructions. The Langley SP2 has 48 wide nodes with 128 Mbytes local memory and peak performance of 266 MFLOPS each. In contrast, the Langley Paragon has 72 nodes with 32 Mbytes of local memory and peak performance of 75 MFLOPS each. The heart of all distributed memory parallel computers is the interconnection network that links the processors together. The SP2 High-Performance Switch is a multi-stage packet switched Omega network that provides a minimum of four paths between any pair of nodes in the system. The processors of Intel Paragon are connected in a two-dimensional rectangular mesh topology. The diameter of the 2-D mesh topology will increase with the number of processors. For the SP2, the measured latency (start time), α, is 45 microseconds and the measured transmission time per byte, β, is 2 microseconds. For Paragon, the measured latency and transmission time per byte is 50 microseconds and 6 microseconds, respectively.

Speedup is one of the most frequently used performance metrics in parallel processing. From the problem size point of view, speedup can be divided into the *fixed-size speedup* and the *scaled speedup*. Depending on the scaling restrictions of the problem size, the scaled speedup can be classified as the *fixed-time speedup* [14] and the *memory-bounded speedup* [15]. As the number of processors increases, memory-bounded speedup scales problem size to utilize the associated memory increase. In general, operation count increases much faster than memory requirement. Therefore, in general, the work load on each processor will not decrease with increase in the number of processors in memory-bounded scale up. Thus, scaled speedup is more likely to get a higher speedup than that of fixed-size speedup. Figure 1 demonstrates the speedups of the PDD algorithm on the SP2 machine. Since the cost of one-to-one communication does not increase with the number of processors on the SP2 multi-stage Omega network, for number of processors from 2 to 32, the PDD algorithm reaches a linear speedup on memory-bounded speedup. The measured speedup is below ideal speedup because there is no communication in uniprocessor processing. In accordance with the isospeed metric [2], the PDD algorithm is perfectly scalable in the multi-stage SP2 machine from ensemble size 2 to 32.

Though the PDD and the Reduced PDD have similar relative speedup patterns, the execution times of the two algorithms are very different. The Reduced PDD algorithm has a smaller execution time than that of the PDD algorithm. For periodic systems, for the sample matrix, the Reduced PDD algorithm even has a smaller execution time than the conventional sequential algorithm. The timings of Thomas algorithm, the PDD algorithm, and the Reduced PDD algorithm on a single

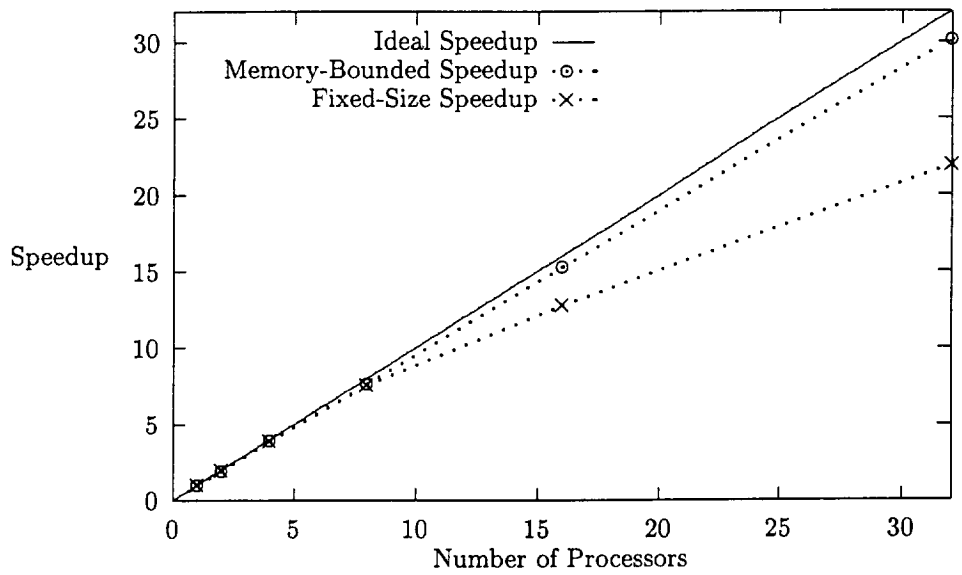

Figure 1. Measured Speedup of the PDD Algorithm on a SP2 Machine
1024 System of Order 6400, periodic

	Size	Thomas Alg.	PDD Alg.	Reduced PDD Alg.
Paragon	1600	0.8265	0.9026	0.6432
SP2	6400	0.7387	0.856	0.5545

Table 11. Sequential Timing (in seconds) on Paragon and SP2 machines

node of the SP2 and Paragon machine are listed in Table 11. The problem size for all algorithms on SP2 is $n = 6400$ and $n1 = 1024$, and on Paragon is $n = 1600$ and $n1 = 1024$. The measured results confirm the analytical results given in Tables 9 and 10.

Figure 2 shows the speedup of the Reduced PDD algorithm over the conventional sequential algorithm, Thomas algorithm. The PDD algorithm increases computation count for high parallelism. The Reduced PDD reduces computation count by taking advantage of diagonal dominance. Compared to Thomas algorithm, the Reduced PDD algorithm has a smaller execution time for some applications and achieved a superlinear speedup for our testing. Experimental results confirm that the Reduced PDD algorithm maintains the good scalability of the PDD algorithm and delivers an efficient performance in terms of execution time as well.

The PDD and the Reduced PDD algorithms are perfectly scalable, in the sense that their communication costs do not increase with the order of the matrix and ensemble size, and the workload is balanced. The PPT algorithm, however, has a serial processing part and a communication cost which increases with the ensemble size. While the PDD and the Reduced PDD algorithms have similar speedup curves on both the Paragon and the SP2 machines, the PPT has quite different

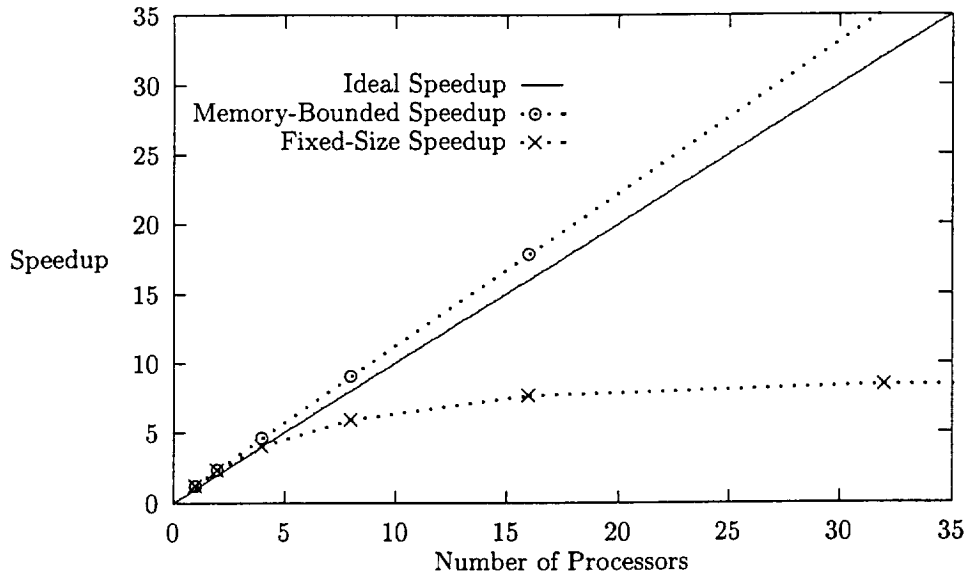

Figure 2. Speedup of the Reduced PDD Algorithm Over Thomas Algorithm.
1024 Systems of Order 1600, periodic

speedup curves on the Paragon and the SP2 machines. Figure 3 shows the scaled and the fixed-size speedup of the PPT algorithm on the SP2 machine. The measured speedup is considerably lower than that of the PDD and the Reduced PDD algorithms. Parallel efficiency is usually defined as speedup divided by the number of processors. Unlike the PDD and the Reduced PDD algorithm, the efficiency of the PPT algorithm decays with the ensemble size. From Figure 3 we can see that the PPT algorithm cannot reach linear speedup on the SP2 machine.

Our theoretical and experimental results show that the newly developed PDD and Reduced PDD algorithm are scalable and out perform traditionally used parallel solvers, in terms of execution time and speedup. A technical paper has been written to report the experimental measurement and analytical comparison of the PDD, Reduced PDD, and PPT algorithm [16]. The interested reader may refer to [16] for more detailed informations.

5 Parallelization of a CFD Simulation Package

One CFD simulation code which is of interest to many researchers is the code developed by Man Mohan Rai and his coworkers [17, 18]. This code (we call it Rai's code) solves 3-D Navier-Stokes equations with Newton-Rapson-type techniques. During the past few years, different variations of Rai's code have been developed by researchers for different CFD simulations. The PDD algorithm developed by the PI [7] is perfectly scalable and has been used successfully in parallelizing the CDNS (Compressible Direct Navier-Stokes Simulation) code [19]. Unlike CDNS, the systems arising from Rai's code are neither periodic, symmetric, nor Toeplitz. The applicability of the PDD

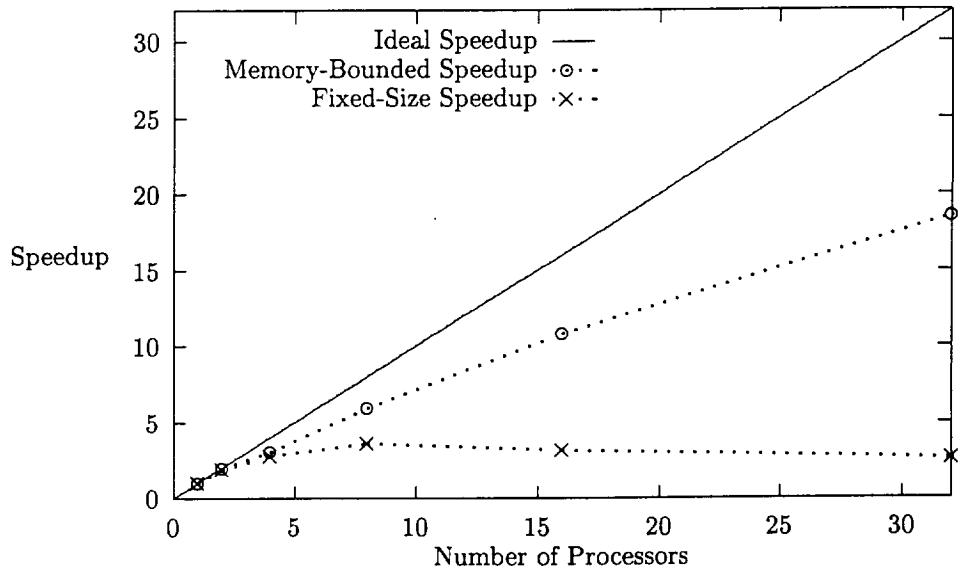

Figure 3. Speedup of the PPT algorithm on SP2 Machine.
1024 Systems of Order 6400, non-periodic

algorithm on Rai's code was studied in 1995. Analytical and experimental results show that the PDD algorithm is applicable for Rai's code. In 1996, we first applied the PDD algorithm to parallelizing Rai's code. The parallelization went smoothly, however, the speedup of the parallelization was unsatisfactory. Puzzled by the low performance, we conducted a detailed study on the data structure and dependency of the code. Performance profiling showed that the tridiagonal solver subroutine *lsh*1, which was parallelized by the PDD algorithm, only contributed less than 7% of the total execution time. Time distributions showed that the module *control* was the critical section of performance. The *control* module and its children subroutines contributed more than 90% of the total execution time. Data dependence analysis was then performed on *control* for parallelization. High level loop independence was identified and tested, and parallel partition was carried out based on the identified independence. Special care was also given to the loop boundaries. Finally, parallel PVM implementations of Rai's code were conducted on the SP2 machine available at the Cornell National Supercomputer Facility and on a cluster of IBM RS/6000 workstations available at LSU. A speedup of 11 was obtained with 16 processors on the SP2 machine for solving the *control* module. Limited by our access on the Cornell machine, only small size problems were tested. A better speedup is expected for large data sets.

5.1 Data Dependence and Partition

Functional decomposition is the first step toward analyzing the dependence structure of the parallelization of a sequential program. It breaks a main module into sub modules and reveals the parent-children relationship within a module. Rai's code has 4K lines of code. It consists of 36

subroutines which can be grouped into several modules. The high level modules are *initia*, *data*, *control* and *output*. Routines *initia* and *control* are the main modules. They have 12 and 20 children subroutines respectively. While subroutine *initia* is only executed once for each run. The *control* module is called at each of the iteration steps. Execution of the *control* module dominates the overall execution time. The function dependence structure of modules *control* is shown in Figure 4. The decomposition is obtained through analyzing and utilizing Unix utilities such as *grep*, *egrep*, *sort*, *awk*, *head*, *tail* and *pipes*.

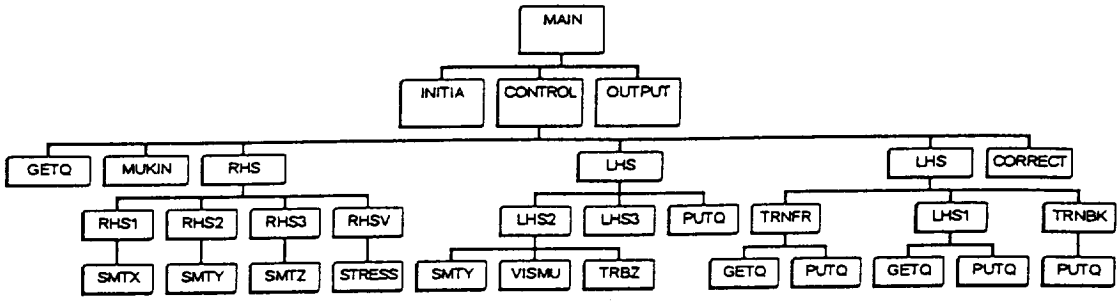

Figure 4. Function Decomposition of the *control* module

Data Dependence Concept

If the iterations of a loop can be executed in random order and still produce the correct result, it is an independent loop [20]. Independent loops are perfect for parallel execution. But loops are rarely independent. Dependent loops, in which the dependency involves all the statements in the loop, must be executed serially on any machine due to their dependency. When the dependency does not involve all the statements in the loop, partial overlapping, or pipelining of successive iterations may be possible during the execution.

Data dependence is a consequence of the flow of data in a program. A task that uses a variable in an expression is data dependence on the task which computes the value of that variable. If task T_w is data-dependent on task T_v, then execution of task T_v must precede execution of task T_w. Data dependence can be further classified into five categories:

1. Flow dependence: A statement S2 is flow-dependent on statement S1 if an execution path exists from S1 to S2 and if at least one output of S1 feeds in as input to S2.

2. Antidependence: Statements S2 is antidependent on statement S1 if S2 follows S1 in program order and if the output of S2 overlaps the input of S1.

3. Output dependence: Two statements are output-dependent if they write the same output variable.

4. I/O dependence: Read and write refer to the same file.

5. Unknown dependence: the dependence can not be explicitly determined.

Bernstein's conditions imply that two processes can be executed in parallel if they are flow-independent, anti-independent, and output-independent [21].

The above concepts of data dependence are used to analyze the CFD program for parallelism. Loops perfect for parallelization are discovered at a very high level for both subroutines *control* and $lhs1$. Some careful manipulation of data initialization in *control* is done to achieve this goal. Since *control* takes more than 90% execution time, its parallelization is very significant to the overall performance.

Data Dependence Analysis

The module *control* does the calculation at each iteration step. It has quite a few nested loops. To get maximum parallelization, a natural approach is to go to the outermost loop. Observing Rai's code, the loop 10 is to get the iterative solution at each iteration step. Since the next iteration always uses the result of the current one, loop dependence is internally determined. Actually, it goes through all the X direction planes to do the solutions for Y and Z directions in loop 160 and then swaps to all the Y planes for solutions in X direction (this is done by $lhs1$).

The analysis gives the same result. First, there is a flow dependence from one iteration to the next. While the current iteration writes q using *putq* in loop 300, the next iteration reads vector q in loop 150 and loop 160. Second, there is an anti-flow dependence within each iteration because the input q in loop 150 is also the output in loop 300. These two types of data dependence suffice that it is very unlikely to do parallelization at the loop 10 level. The next highest loop level is loop 160. It has 13 direct subroutine calls and the out loop goes over all the X direction planes. It involves solving for Y and Z direction and calculating of all the rhs which includes $rhs1$, $rhs2$, $rhs3$ and $rhsv$. It should be noticed that $rhsv$ calls the time consuming subroutine *stress* directly.

There are several *getq* calls and one *putq* call within loop 160. This however, does not introduce a flow or anti-flow dependence because they operate on different units of the $q's$. The call $getq(ifetch, 1, qeq1)$ seems to be an anti-flow dependence because the $i-th$ iteration uses the $qeq1$ from iteration $i+4$ in X direction. The boundary elements are exceptions. Two things should be noticed here. First, in sequential calculation, the iteration $i+4$ is done after the $i-th$ iteration and the values of $qeq1$ used are from the last time step iteration. Second, if the i-th and the $i+4th$ iterations are grouped into the same process for parallelization, the dependence goes away. This uses the exceptions of boundaries because their elements are not affected by the $i+4th$ iteration. This is exactly what comes from the data dependence analysis: loop 160 can be parallelized perfectly.

What about I/O dependence, output dependence and unknown dependence? Since there is no file writing or reading involved within loop 160, the I/O dependence is eliminated. Although each iteration updates or writes the *putq*, there is no overlapping because they use different i values. This takes care of the output dependence. Finally, the unknown dependence is eliminated through tracing and analyzing all the called subroutines. The fact that the arrays do not involve implicit or nested subscripts makes the analyzing a little easier.

The above description for data dependence analysis seems to be simple, but the actual work is very complicated.

Data Partition Analysis

Since the loops for perfect parallelization are found in modules *control* and *lhs1*, the data partition is straightforward. From the partition point of view, more sub-tasks correspond to the less execution time. However, communication is necessary for parallel processing of the loop. The communication overhead is highly determined by the network conditions and by the whole virtual machine system in the case of PVM. Massive parallelization may not necessary lead to a good performance. To reduce the communication cost, multicasting mode is used for data communication in the parallel PVM implementation.

5.2 Experimental Results on SP2

PVM implementation of Rai's code has been tested on an SP2 system and on a cluster of workstations. We only present the experimental results on SP2 here. Results for a cluster of workstations can be found in [22]. The parallel implementation on SP2 is carried out with EASY-LL batch system using architecture RS6K and data encoding PvmDataRaw.

The execution time of the parallelized *control* module is measured for performance study. Five iterations are conducted in the code to reach a satisfactory solution. The execution time of each iteration of *control* and the corresponding speedup are presented in Table 12. The parallelized *control* module achieves a pretty good speedup. Its measured speedup is a linear function of the number of processors. Notice that the measured time of each iteration does not include the communication cost between each iteration. Even without considering the communication overhead, due to sequential/parallel portion increases, the measured speedup gets farther away from the ideal speedup when number of processors increase. The power of parallel computing can be utilized more efficiently if scaled computing is adopted, in which problem size increases with system size [15]. Figure 5 shows the input data used in our testing. The test problem is small due to our limited access to the Cornell SP2 machine. While our current experiment results do not show the best speedup possible, they clearly demonstrate the correctness and effectiveness of our dependence analysis and partition approach. They have shown the high potential of parallel processing gain in solving Rai's code. Also, our results show that using data encoding PvmDatRaw provides a better performance than using PvmDataDefault in solving Rai's code.

Run	Seria	2 nodes	4 nodes	8 nodes	16 nodes
Time (Sec.)	64.18	34.83	18.45	10.49	5.79
Speedup	1	1.84	3.48	6.12	11.08

Table 12. Execution Time and Speedup of the *Control* Module

Input Format
read(5,*) iread, iwrite
read(5,*) niter, nprint
read(5,*) imax, jmax, kmax, iplt, ilcc
read(5,*) datu, cour, amach, reperin

Input Data:
0 0
1 1
768 48 7 728 182
0.002 0.25 2.25 635000.0

Figure 5. Input Data Used in Testing

6 Stimulus Assessment

Analytical and experimental results show that our newly developed algorithms are significant faster than the traditionally used method when accurate solutions are required. By their practical importance, these algorithms, especially the compact scheme based algorithms, should be well noticed by the high performance computing community. They will make a contribution in the development of highly accurate simulations. One direction of future research is to extend the compact discretization scheme to full reacting Navier-Stokes equations and other PDE systems for high-order accurate algorithms. Another direction of future research is to combine the parallelization and discretization of compact schemes with the recently developed adaptive Wavelet collocation method [23] for portability (automatic mult-level resolution) and effectiveness (non-uniform stencil spacing). Study how to incorporate the newly developed algorithms into actual engineering applications is also a possible direction under consideration. The success of these algorithm development and their impact on the simulation of CFD application certainly will directly contribute to NASA missions.

Partially supported by this grant, the PI has established his research group, Scalable Computing Software (SCS) group, at LSU. He has been successfully receiving fundings from NFS and Louisiana State to conduct research in high performance computing. Two master students have graduated through the support of this research. Before graduation. these students had already received multiple offers and with starting pay more than $50,000s. It demonstrates the success of the research project toward education. Several other students have expressed their interests to join the PI's research group recently.

References

[1] Committee on Physical and Mathematical and Engineering Sciences, "Grand challenges: High performance computing and communications," National Science Foundation, 1992.

[2] X.-H. Sun and D. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Transactions on Parallel and Distributed Systems*, pp. 599–613, June 1994.

[3] V. Kumar and et al., *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Commings, 1994.

[4] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in FORTRAN, The Art of Scientific Computing.* Cambridge University Press, 1992. Second Edition.

[5] R. Hockney, "A fast direct solution of Poisson's equation using Fourier analysis," *J. ACM*, vol. 12, pp. 95–113, 1965.

[6] X.-H. Sun and Y. Zhuang, "A high order fast solver for the helmhotz equation with neumann boundary conditions." ICASE Technical Report No. 97-11, 1997.

[7] X.-H. Sun, "Application and accuracy of the parallel diagonal dominant algorithm," *Parallel Computing*, pp. 1241–1267, Aug. 1995.

[8] S. Lele, "Compact finite difference schemes with spectral-like resolution," *J. Comp. Phys.*, vol. 103, no. 1, pp. 16–42, 1992.

[9] X.-H. Sun, H. Zhang, and L. Ni, "Efficient tridiagonal solvers on multicomputers," *IEEE Transactions on Computers*, vol. 41, no. 3, pp. 286–296, 1992.

[10] H. Wang, "A parallel method for tridiagonal equations," *ACM Trans. Math. Software*, vol. 7, pp. 170–183, June 1981.

[11] D. Lawrie and A. Sameh, "The computation and communication complexity of a parallel banded system solver," *ACM Trans. Math. Soft.*, vol. 10, pp. 185–195, June 1984.

[12] C. Hirsch, *Numerical Computation of Internal and External Flows.* John Wiley & Sons, 1988.

[13] V. Bala and et al., "Ccl: A portable and tunable collective communication library for scalable parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 154–164, Feb. 1995.

[14] J. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, pp. 532–533, May 1988.

[15] X.-H. Sun and L. Ni, "Scalable problems and memory-bounded speedup," *J. of Parallel and Distributed Computing*, vol. 19, pp. 27–37, Sept. 1993.

[16] X.-H. Sun and S. Moitra, "A fast parallel tridiagonal algorithm for a class of cfd applications." NASA Techonical Paper 3585, Aug. 1996.

[17] M. M. Rai and P. Moin, "Direct simulations of turbulent flow using finite-difference schemes," *Journal of Computational Physics*, vol. 96, pp. 15–53, 1991.

[18] M. M. Rai and P. Moin, "Direct numerical simulation of transition and turbulence in a spatially evolving boundary layer," in *AIAA Paper 91-1607*, Nov. 1991.

[19] G. Erlebacher, M. Hussaini, H. Kreiss, and S. Sarkar, "The analysis and simulation of compressible turbulence," *Theoret. Comput. Fluid Dynamics*, vol. 73, no. 2, 1990.

[20] T. Lewis and H. El-Rewini, *Introduction to Parallel Computing.* Prentice Hall, 1992.

[21] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, 1993.

[22] Q. Hou, "A parallel implementation of a 3D computational fluid dynamics application." Master of Science Project Report, Department of Computer Science, Louisiana State University, Jan. 1997.

[23] W. Cai and J. Wang, "Adaptive wavelet collocation methods for initial value boundary problems of nonlinear PDE's." ICASE Technical Report, 93-48, ICASE, NASA Langley Research Center, 1993.