

ARMY RESEARCH LABORATORY



Performance of a Sequential and Parallel Computational Fluid Dynamic (CFD) Solver on a Missile Body Configuration

by Dixie Hisley
and Duane Frist

ARL-TR-2032

August 1999

19990921 064

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 4

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2032

August 1999

Performance of a Sequential and Parallel Computational Fluid Dynamic (CFD) Solver on a Missile Body Configuration

Dixie Hisley and Duane Frist
Corporate Information and Computing Center, ARL

Approved for public release; distribution is unlimited.

Abstract

In order to effectively port production codes originally written for vector processors to reduced instruction set (RISC)-based parallel computers, different paradigms have been tried by the parallel computing community. Among the techniques used are message-passing and loop-level parallelization using hand-inserted compiler directives or automatic-parallelizing compiler flags. The goals of this report are (1) to investigate the performance of message-passing and loop-level parallelization techniques, as they were implemented in the computational fluid dynamics (CFD) code Overflow, and (2) to validate the sequential and parallel results obtained on a demonstration problem of interest to the Army—that is, a generic missile-body configuration. The computational simulations were run, and performance data were gathered on a Silicon Graphics Incorporated (SGI) Power Challenge Array (PCA) and Origin2000.

Table of Contents

	<u>Page</u>
List of Figures	v
List of Tables	ix
1. Introduction	1
2. KTA Missile Configuration	3
3. Computational Grids	3
4. Overflow Flow Solver and Convergence Criteria	5
5. Computational Results and Discussion: KTA Case No. 3	7
5.1 Sequential Code Validation and Performance	7
5.2 Parallel PVM Code	16
5.3 Parallel Loop Optimization Code	18
5.4 Performance of Parallel Codes	20
6. Conclusions/Future Work	23
7. References	27
Appendix: Plots for KTA Case Nos. 1, 2, and 4-6	29
List of Abbreviations and Symbols	47
Distribution List	49
Report Documentation Page	51

INTENTIONALLY LEFT BLANK.

List of Figures

<u>Figure</u>	<u>Page</u>
1. KTA Geometry	4
2. Symmetry Plane of Computational Grids Used in This Study: (a) Case Nos. 1 and 2, (b) Case Nos. 3–5, and (c) Case No. 6	5
3. Convergence Criteria: (a) Right-Hand Side L-2 Norm, (b) ΔQ L-2 Norm, and (c) Minimum Pressure and Density History	6
4. Surface Pressure Plots for Case No. 3: Mach = 2.5, AOA = 14°, and Re = 1,123,000	8
5. Pitot Pressure Contours for Case No. 3: (a) x/d = 5.5, Laminar Computation; (b) x/d = 5.5, Turbulent Computation; (c) x/d = 11.5, Laminar Computation; and (d) x/d = 11.5, Turbulent Computation	10
6. Vortex Core Evaluation for Case No. 3: x/d = 11.5	11
7. Force and Moment Coefficient vs. Axial Location for Case No. 3: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment	12
8. Vortex Core Evaluation for Time-Accurate Computation: (a) x/d = 5.5, (b) x/d = 8.5, and (c) x/d = 11.5	13
9. Effect of Limiting Turbulent Region on Surface Pressure Calculations for Case No. 3	15
10. Effect of Degani-Schiff Modifications on Calculation of F_{\max} for Case No. 3: Solid Line Without Degani-Schiff, Dots With Degani-Schiff	16
11. Effect of Degani-Schiff Modifications on Surface Pressure Calculations for Case No. 3	17
12. Scalability of Loop-Level Parallel Code Implementation	22
13. Floating-Point Operations, Synchronization Cost, and False Sharing Event Counts	24
A-1. Surface Pressure Plots for Case No. 1: Mach = 1.45, AOA = 14°, and Re = 667,000.	31

<u>Figure</u>	<u>Page</u>
A-2. Surface Pressure Plots for Case No. 2: Mach = 1.8, AOA = 14°, and Re = 667,000	32
A-3. Surface Pressure Plots for Case No. 4: Mach = 3.5, AOA = 8°, and Re = 1,123,000.	33
A-4. Surface Pressure Plots for Case No. 5: Mach = 3.5, AOA = 14°, and Re = 1,123,000	34
A-5. Surface Pressure Plots for Case No. 6: Mach = 0.7, AOA = 14°, and Re = 667,000	35
A-6. Pitot Pressure Contours for Case No. 1: (a) x/d = 8.5, Laminar Computation; (b) x/d = 8.5, Turbulent Computation; (c) x/d = 11.5, Laminar Computation; and (d) x/d = 11.5, Turbulent Computation	36
A-7. Pitot Pressure Contours for Case No. 2: (a) x/d = 5.5, Laminar Computation; (b) x/d = 5.5, Turbulent Computation; (c) x/d = 8.5, Laminar Computation; (d) x/d = 8.5, Turbulent Computation; (e) x/d = 11.5, Laminar Computation; and (f) x/d = 11.5, Turbulent Computation	37
A-8. Pitot Pressure Contours for Case No. 4: (a) x/d = 11.5, Laminar Computation; and (b) x/d = 11.5, Turbulent Computation	38
A-9. Pitot Pressure Contours for Case No. 5: (a) x/d = 5.5, Laminar Computation; (b) x/d = 5.5, Turbulent Computation; (c) x/d = 11.5, Laminar Computation; and (d) x/d = 11.5, Turbulent Computation	39
A-10. Pitot Pressure Contours for Case No. 6: (a) x/d = 8.5, Laminar Computation; (b) x/d = 8.5, Turbulent Computation; (c) x/d = 11.5, Laminar Computation; and (d) x/d = 11.5, Turbulent Computation	40
A-11. Vortex Core Evaluation for Case No. 1: (a) x/d = 8.5, and (b) x/d = 11.5	41
A-12. Vortex Core Evaluation for Case No. 2: (a) x/d = 5.5, (b) x/d = 8.5, and (c) x/d = 11.5	42
A-13. Vortex Core Evaluation for Case No. 4: x/d = 11.5	43
A-14. Vortex Core Evaluation for Case No. 5: x/d = 11.5	43
A-15. Vortex Core Evaluation for Case No. 6: (a) x/d = 8.5, and (b) x/d = 11.5	44

<u>Figure</u>	<u>Page</u>
A-16. Force and Moment Coefficient vs. Axial Location for Case No. 4: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment	45
A-17. Force and Moment Coefficient vs. Axial Location for Case No. 5: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment	46

INTENTIONALLY LEFT BLANK.

List of Tables

<u>Table</u>	<u>Page</u>
1. Test Conditions	4
2. Available Experimental Data Locations	4
3. Sequential Run Summary	8
4. Pitot Pressure Data Locations	9
5. Force and Moment Coefficients	11
6. Automatic Parallelizing Compiler Performance Results	20
7. Performance Timings	21

INTENTIONALLY LEFT BLANK.

1. Introduction

At the U.S. Army Research Laboratory (ARL), scientists have typically run large-scale, computationally intensive, computational fluid dynamic (CFD) simulations on high-end, supercomputing architectures. Until about 5 years ago, this meant that scientists mainly utilized conventional vector supercomputers such as those manufactured by Cray Research, Incorporated. More recently, the comparable sustained performance-to-price ratio of scalar microprocessor-based architectures, relative to vector processors, has resulted in their purchase by the Department of Defense (DOD) community and the subsequent implementation of CFD codes on these modern reduced instruction set computer (RISC)-based parallel computers.

As part of the DOD High-Performance Computing Modernization Program, ARL is currently a Major Shared Resource Center (MSRC). Two of the modern RISC-based parallel supercomputing resources available through ARL MSRC and used extensively in this study are Silicon Graphics Incorporated's (SGI) Power Challenge Array (PCA) and Origin2000.

The ARL MSRC PCA consists of eight shared-memory multiprocessor (SMP) power challenge XL supercomputing nodes. Each PCA node is populated with 12 75-MHz R8000 RISC processors and 2 GB of shared memory and supports a multitasking, multiuser environment. In each PCA node, the 12 processors and memory boards plug into a common bus. Each processor within a PCA node has direct access to all the memory within the node. Thus, on a per-node basis, the PCA is a uniform memory access (UMA) architecture. Efficient multitasking use of a single PCA node by a single user can be achieved by using loop-level parallelization directives.

In the loop-level parallelization paradigm, the user or compiler tries to identify loops that can be run in parallel by distributing the iterations of each loop among threads of execution (usually one per processor). It is then the job of the compiler, operating system, hardware, and memory consistency protocol to ensure that the threads of execution receive the correct values for variables being shared by multiple threads.

The eight PCA nodes are interconnected through an 800 MB/s high-performance parallel interface (HIPPI) and a 155-MB/s asynchronous transfer mode (ATM) network. Thus, while each PCA node can be thought of as a shared memory processor, the entire array of nodes can be thought of as a distributed memory system and can be utilized by programming in a message-passing style. For a processor to have access to the local memory of a processor on a remote node, a copy of the desired data must be sent from the remote node to the other. This data communication is usually accomplished by using a message-passing library like message-passing interface (MPI) or parallel virtual machine (PVM) (see Gropp, Lusk, and Skjellum 1994; Snir et al. 1994; Gast et al. 1995).

While this allows a single user to access more than the 12 processors available within a single PCA node, going across nodes brings a new set of challenges; that is, rewriting code in a message-passing style and hiding the substantial latency of the communications.

The Origin2000 at ARL MSRC, is configured with 32 195-MHz processors and 12 GB of memory. Each origin node consists of two R10000 processors (4 GB of memory) and connects to a portion of the input/output (IO) subsystem. The origin nodes are connected together by a scalable interconnection network. The memory associated with each node is physically distinct; however, the directory-based cache coherence protocol adds a layer of abstraction that allows the user to see the memory across nodes as one logical memory space. The directory-based cache coherence protocol maintains the familiar shared memory programming model of a PCA node. Gone, however, are the PCA node's uniform memory access times. Thus, data placement will be an issue for parallel programs that are memory intensive and are not cache friendly on the Origin2000. Codes for use on the Origin2000 can be programmed using message-passing and/or loop-level parallelization techniques.

In order to effectively port production codes originally written for vector processors to RISC-based parallel computers, different paradigms have been tried by the parallel-computing community. Among the techniques used are message-passing and loop-level parallelization using hand-inserted compiler directives or automatic-parallelizing compiler flags. The goals of this report are (1) to investigate the performance of these two parallel paradigms as they were implemented in the CFD code Overflow (Buning et al. 1995), and (2) to validate the sequential and parallel results

obtained by Overflow on a demonstration problem of interest to the Army; that is, a generic missile body configuration. All computational simulations are run and performance data are gathered on the SGI PCA and the Origin2000.

2. KTA Missile Configuration

In order to validate and test new computational methodologies, a demonstration geometry must be selected that has been previously studied and is well documented in terms of experimental results. One geometry of interest to the aerodynamics community at ARL is an ogive-cylinder missile body configuration for transonic and supersonic velocities. The Defence Research Agency, UK, provided high-quality experimental data for this configuration that included surface pressure, flow-field pitot pressures, and force measurements. Under the auspices of The Technical Cooperation Program (TTCP) with participants from the United States, United Kingdom, and Canada, a number of Navier-Stokes solvers were applied to the missile geometry to compare predictive capabilities (Sturek et al. 1997). The experimental geometry, test conditions, and available experimental data locations as given to the participants in the study are shown in Figure 1, Table 1, and Table 2, respectively. Figure 1 shows that the configuration is a 3-cal. ogive with a 10-cal. cylindrical afterbody. Table 1 indicates that the missile body was tested at Mach conditions ranging from Mach = 0.7 to Mach = 3.5 and at angle of attacks (AOA) of either 8° or 14°.

3. Computational Grids

Three, structured, single grids were used to solve the six cases. All grids (Figure 2) were $121 \times 91 \times 89$ (axial, circumferential, normal) with a y^+ value of approximately 1. The flow about the missile was considered to be symmetric about the vertical plane. Two planes in the circumferential direction were added by reflected symmetry. These were needed by the flow solver to enforce symmetry about the vertical axis. The flow field downstream of the missile was not considered. The significant difference between the three grids is that the outer boundaries have been moved farther away from the missile for case nos. 1, 2, and 6.

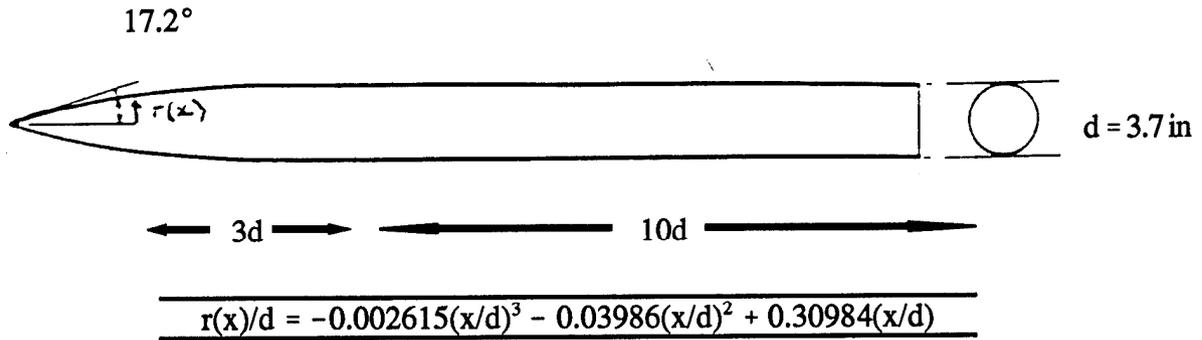


Figure 1. KTA Geometry.

Table 1. Test Conditions

Case No.	Mach No.	AOA	Re No ($10^6/d$)	Po (Hg)	To (K)
1	1.45	14	0.667	13.3	301
2	1.8	14	0.667	14	304
3	2.5	14	1.123	42	308
4	3.5	8	1.123	72	315
5	3.5	14	1.123	72	315
6	0.7	14	0.667	72	315

Table 2. Available Experimental Data Locations

Case No.	Forces	Surface Pressures	Pitot Pressures
1	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 8.5 and 11.5
2	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 5.5, 8.5, and 11.5
3	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 5.5 and 11.5
4	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 5.5
5	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 5.5 and 11.5
6	Cx,Cy,Cz,Cl,Cm,Cn	x/d = 0.2–14.5, phi = 0–180	x/d = 5.5 and 11.5

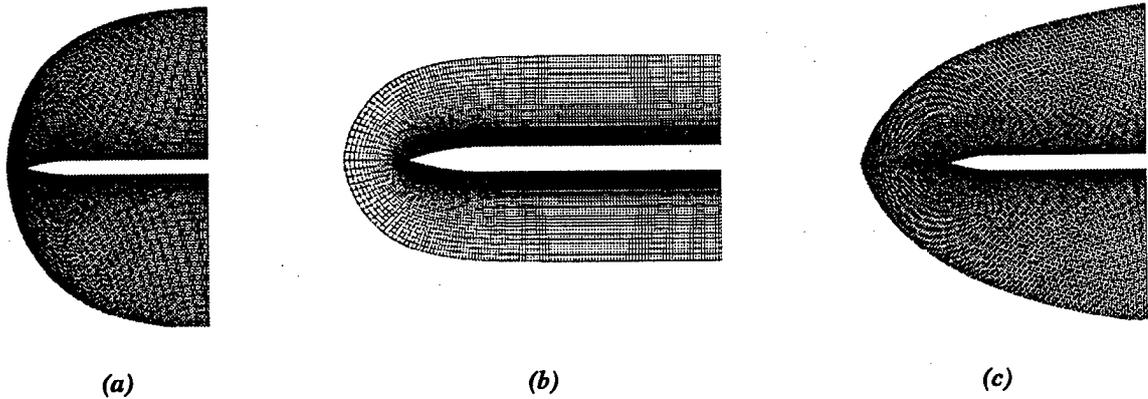


Figure 2. Symmetry Plane of Computational Grids Used in This Study: (a) Case Nos. 1 and 2, (b) Case Nos. 3–5, and (c) Case No. 6.

4. Overflow Flow Solver and Convergence Criteria

Overflow originated at the National Aeronautics and Space Administration (NASA) Ames Research Center in the sequential F3D/Chimera CFD code developed by Joseph Steger. Over the years, a number of people have made significant contributions that have taken the Chimera-overlapped grid capability from a research stage to a production-code status. The Overflow documentation written by Buning (1995) identifies these individuals and their contributions.

Overflow has been tuned for optimizing memory accesses on a single-processor, RISC-based system. In addition, Overflow has both loop-level parallelization coding and message-passing coding to take advantage of computer systems that utilize multiple processors. These two parallel paradigms, as implemented in Overflow, are discussed in section 5.

Overflow is a thin-layer, Reynolds-averaged, Navier-Stokes solver that utilizes a finite-volume, implicit, factored diagonal scheme. In this study, code options were selected that produce second-order spatial accuracy, first-order temporal accuracy, local time-stepping, central differencing in all directions, and Baldwin-Lomax turbulence modeling (Baldwin and Lomax 1978) plus Degani-Schiff

cutoff criteria. The turbulence model has been modified to limit the value of F_{\max} and y_{\max} . This is done by comparing the value of F_{\max} for the current circumferential location with that from the windward side. If there is a significant difference between the two, the value of F_{\max} and y_{\max} from the current circumferential location is replaced by F_{\max} and y_{\max} from the previous circumferential location (Degani and Schiff 1983). The modifications were optimized using case no. 3 and then applied to the remaining cases.

A combination of criteria was used to determine when a solution had converged. The right-hand side L-2 norm dropped two orders of magnitude, while the L-2 norm of ΔQ dropped by four orders of magnitude after 4,000 iterations (Figure 3). Also, the minimum pressure and density values had become converged, as well as the integrated forces. Finally, surface pressure plots from successive runs were compared, with no difference indicating a converged solution.

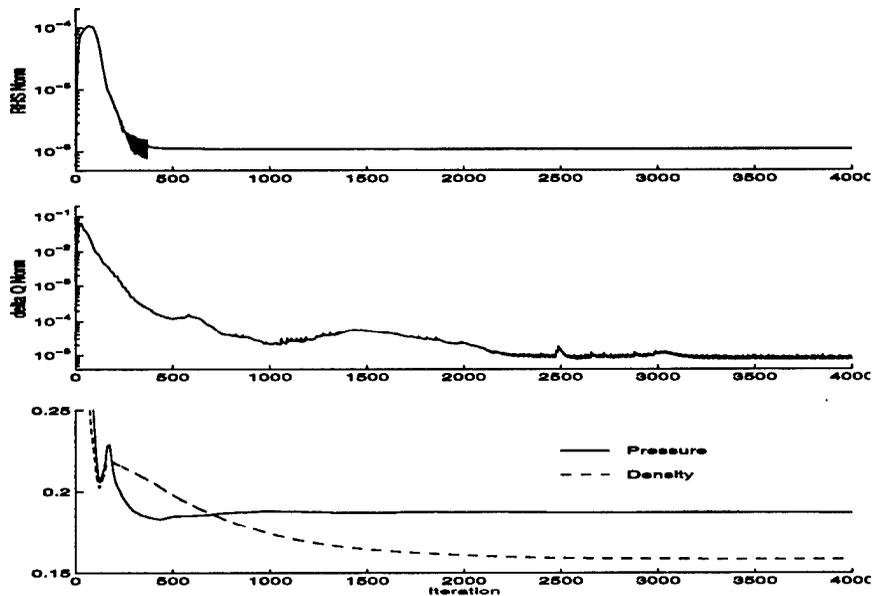


Figure 3. Convergence Criteria: (a) Right-Hand Side L-2 Norm, (b) ΔQ L-2 Norm, and (c) Minimum Pressure and Density History.

5. Computational Results and Discussion: KTA Case No. 3

Plots for case no. 3 are presented in the following section. Plots for case nos. 1, 2, and 4–6 can be found in the appendix.

5.1 Sequential Code Validation and Performance. The sequential flow solver was run to convergence as described in section 4 on an SGIPCA utilizing one processor. Memory requirements were approximately 16 MB, which could be broken into the following categories:

- flow field array ~ 3 Mb,
- grid arrays (X, Y, Z, metrics, etc.) ~ 8 MB, and
- temporary arrays ~ 5 MB.

The SGI PCA does not utilize a queuing system; therefore, run time was greatly dependent upon system utilization. When the load factor of the system was low to moderate, the code ran at approximately 3.4×10^{-5} s per grid point per iteration. During times of heavier usage, performance dropped to 5.2×10^{-5} s per grid point per iteration. On the Origin2000, the code ran at 1.83×10^{-5} s per grid point per iteration during periods of low to moderate load factors. The factor of 2 decrease on the Origin2000 can be attributed to its 195-MHz (R8000) processor in contrast to the PCA's 75-MHz (R10000) processor. Table 3 summarizes the various sequential runs made.

Surface pressure plots are provided in Figure 4 and Figures A-1–A-5. The plots show C_p vs. ϕ for X/D locations of 2.4, 3.5, 4.5, 5.5, 6.5, 7.5, 9.5, and 11.5 for laminar and turbulent computations, as well as experimental data. As can be expected, the laminar computations did not adequately predict flow separation.

Table 3. Sequential Run Summary

Code	Computer	Case No.	Grid	Turbulence	Remarks
Overflow	SGI PCA	1	121 × 91 × 89	BL + DS	Unstable laminar run after 2,000 iterations
		2	121 × 91 × 89	BL + DS	Unstable laminar run after 2,000 iterations
		2	121 × 91 × 89	BL + DS	Time-accurate computation
		3	121 × 91 × 89	BL	—
		3	121 × 91 × 89	BL + DS	—
		4	121 × 91 × 89	BL + DS	—
		5	121 × 91 × 89	BL + DS	—
		6	121 × 91 × 89	BL + DS	—

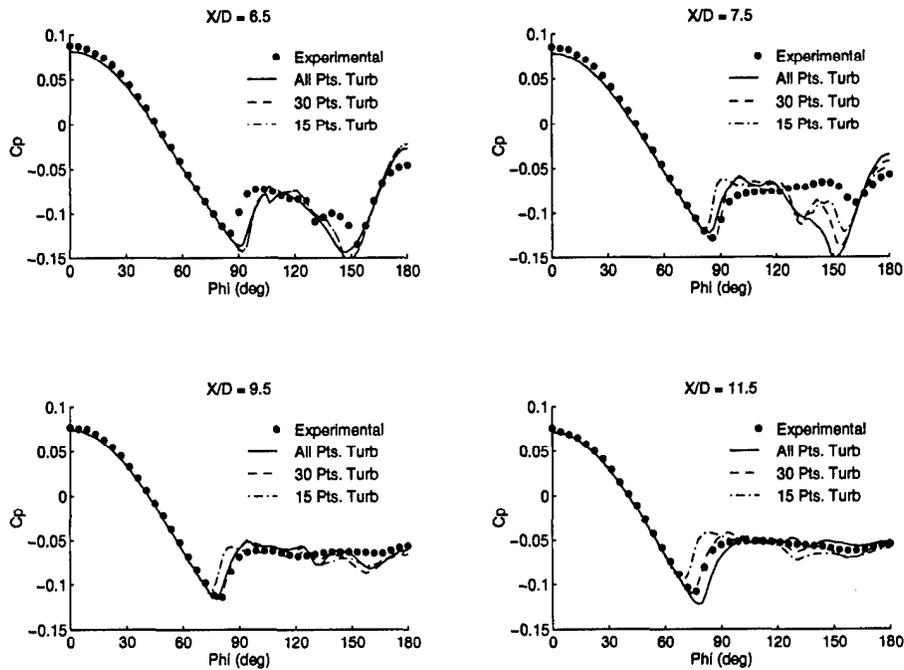


Figure 4. Surface Pressure Plots for Case No. 3: Mach = 2.5, AOA = 14°, and Re = 1,123,000.

Pitot pressure calculations are determined for the locations as shown in Table 4. The results of pitot pressure calculations are shown in Figure 5 and Figures A-6–A-9, where computational results are shown on the right and experimental data are on the left. In Figure 5 and Figures A-6–A-8, there are 15 contours evenly spaced between 0 and 0.925, while, in Figure A-9, there are 30 contours spaced evenly between 0.1 and 0.915.

Table 4. Pitot Pressure Data Locations

Case No.	Locations (x/d)
1	8.5 and 11.5
2	5.5, 8.5, and 11.5
3	5.5 and 11.5
4	11.5
5	5.5 and 11.5
6	8.5 and 11.5

The vortex core is evaluated by determining the pitot pressure in a horizontal line from the vortex core to the symmetry plane. The center of the vortex core for the experimental data is used as the starting point. Figure 6 and Figures A-11–A-15 show the comparison between experimental and computational predictions of the strength of the vortex core. Excellent agreement can be seen in case nos. 3–5, while in case nos. 1, 2, and 6, the strength of the vortex core becomes washed out.

A force and balance is used to measure the experimental, axial, and normal force coefficients, as well as the pitching-moment coefficient. The pressure and viscous contributions from the computational results are integrated to calculate the same coefficients. The results are shown in Table 5.

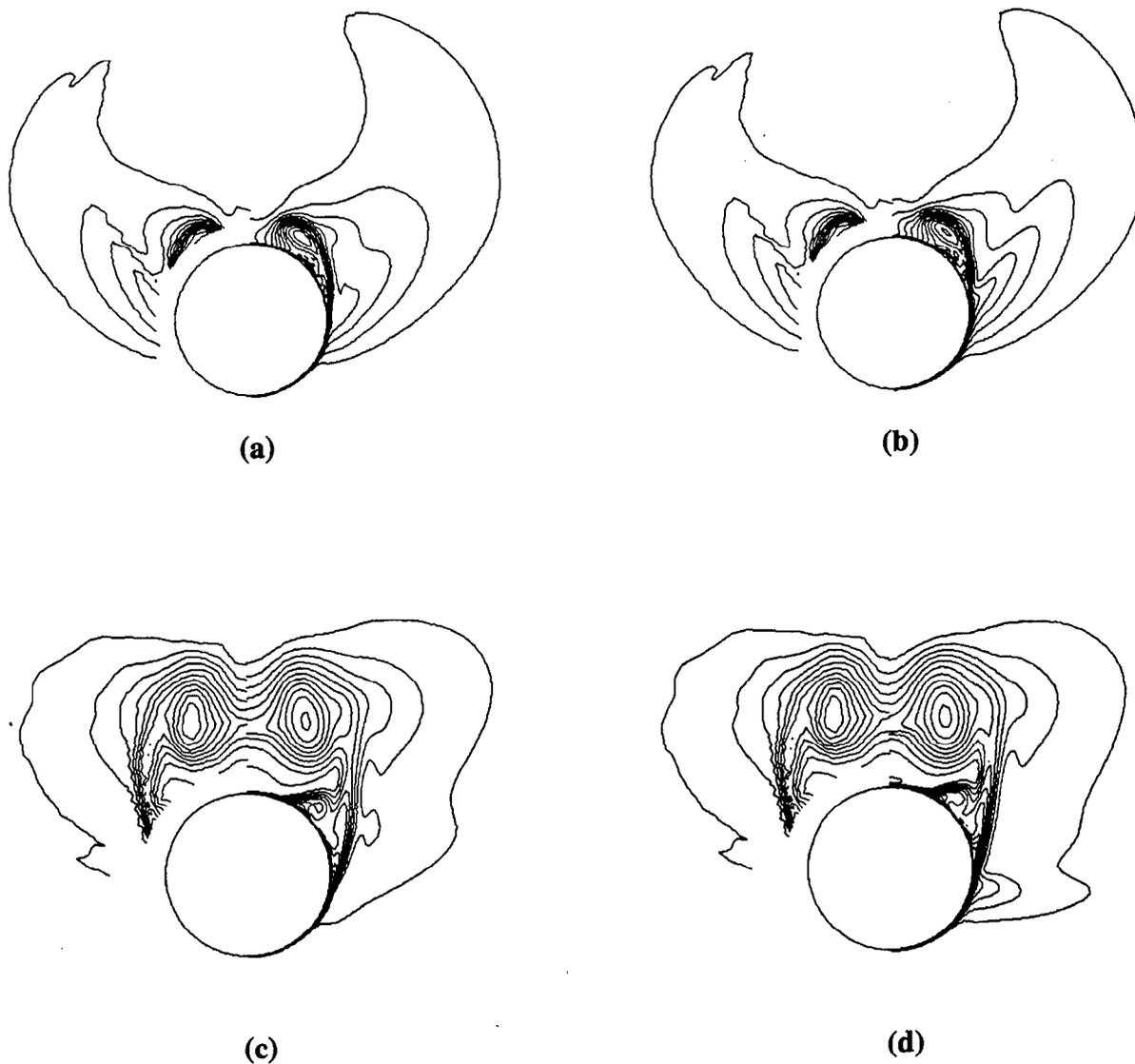


Figure 5. Pitot Pressure Contours for Case No. 3: (a) $x/d = 5.5$, Laminar Computation; (b) $x/d = 5.5$, Turbulent Computation; (c) $x/d = 11.5$, Laminar Computation; and (d) $x/d = 11.5$, Turbulent Computation.

Additionally, experimental and computational surface pressures are integrated to compare the pressure contribution of the axial, normal, and pitching coefficients as a function of axial location. The results of this integration can be seen in Figure 7 and Figures A-16–A-17.

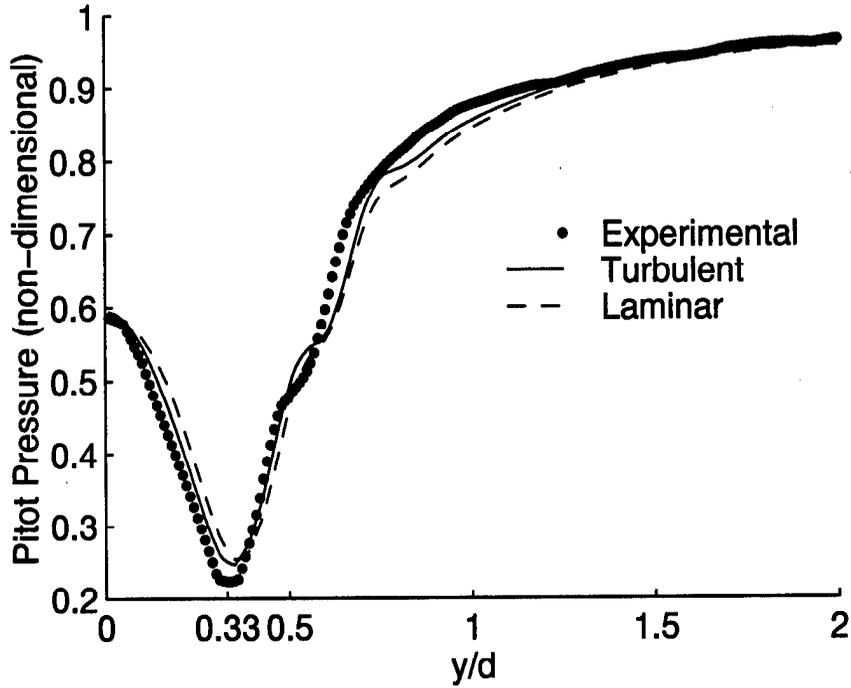


Figure 6. Vortex Core Evaluation for Case No. 3: $x/d = 11.5$.

Table 5. Force and Moment Coefficients

Case No.	Computation	Axial Force Coefficient (C_x)	Normal Force Coefficient (C_n)	Pitching-Moment Coefficient (C_m)
3	Experimental	0.1957	1.9100	10.2417
	Laminar	0.1160	2.0006	11.3061
	Turbulent	0.1486	1.8779	10.1600
4	Experimental	0.1694	0.7606	3.4735
	Laminar	0.0972	0.8228	4.2570
	Turbulent	0.1196	0.7507	3.5854
5	Experimental	0.1833	1.9195	11.1132
	Laminar	0.1129	1.9554	11.4540
	Turbulent	0.1407	1.9214	11.2106

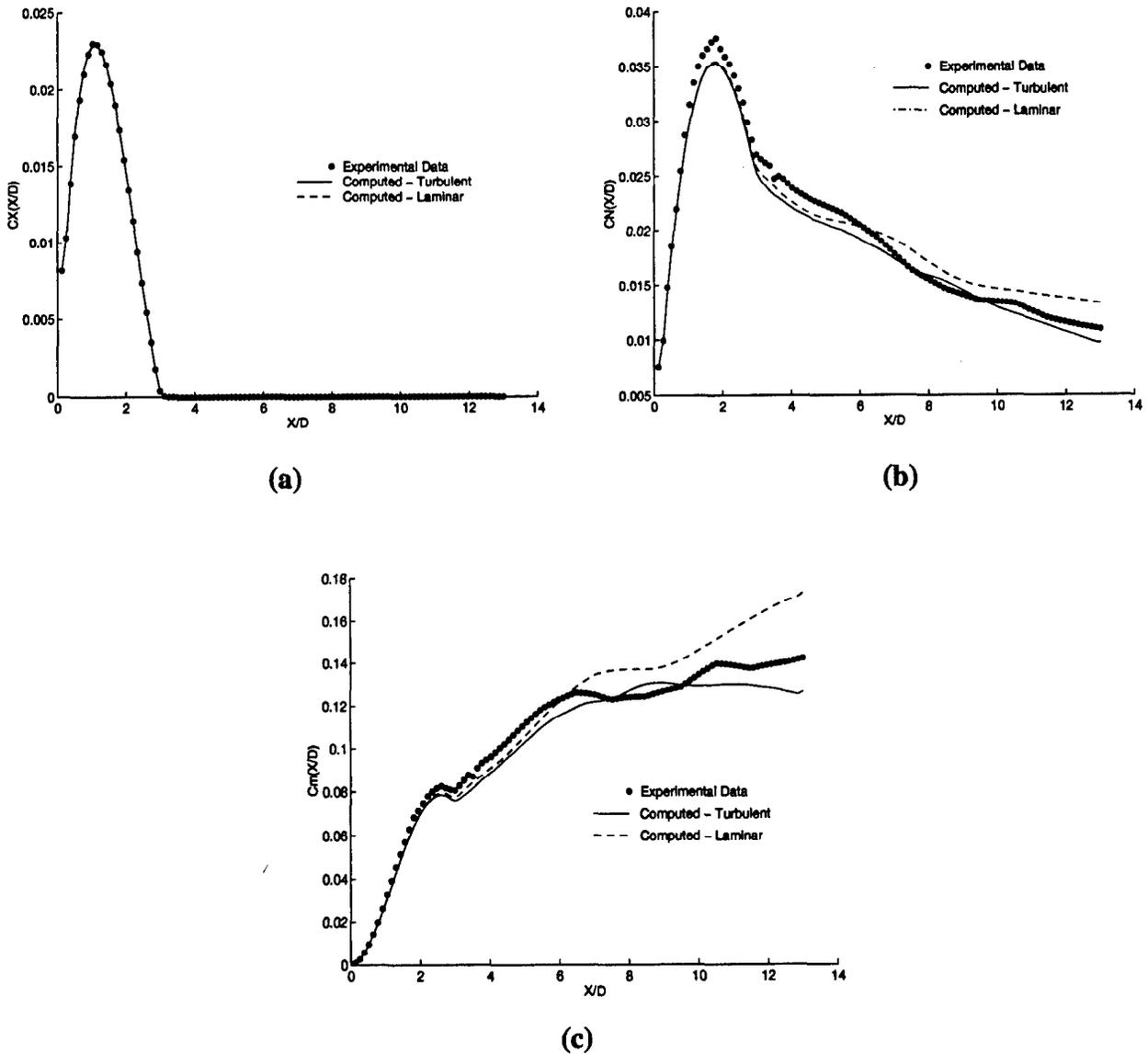


Figure 7. Force and Moment Coefficient vs. Axial Location for Case No. 3: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment.

In case no. 2, computational results underpredict the strength of the vortex core. A time-accurate run was made with lowering amounts of dissipation with the hope that this would prevent the vortex from being washed out. As could be expected, the time-accurate solution was computationally much more expensive. Additionally, it required a lower initial time step (0.0001 as opposed to 0.01) and became unstable after 5,650 iterations. At the point it became unstable, the solution was nearly converged. Unfortunately, the time-accurate computation did not improve the prediction of the

strength of the vortex core. As the solution converged, the strength of the vortex core became washed out, as happened with the local time-stepping calculations (Figure 8).

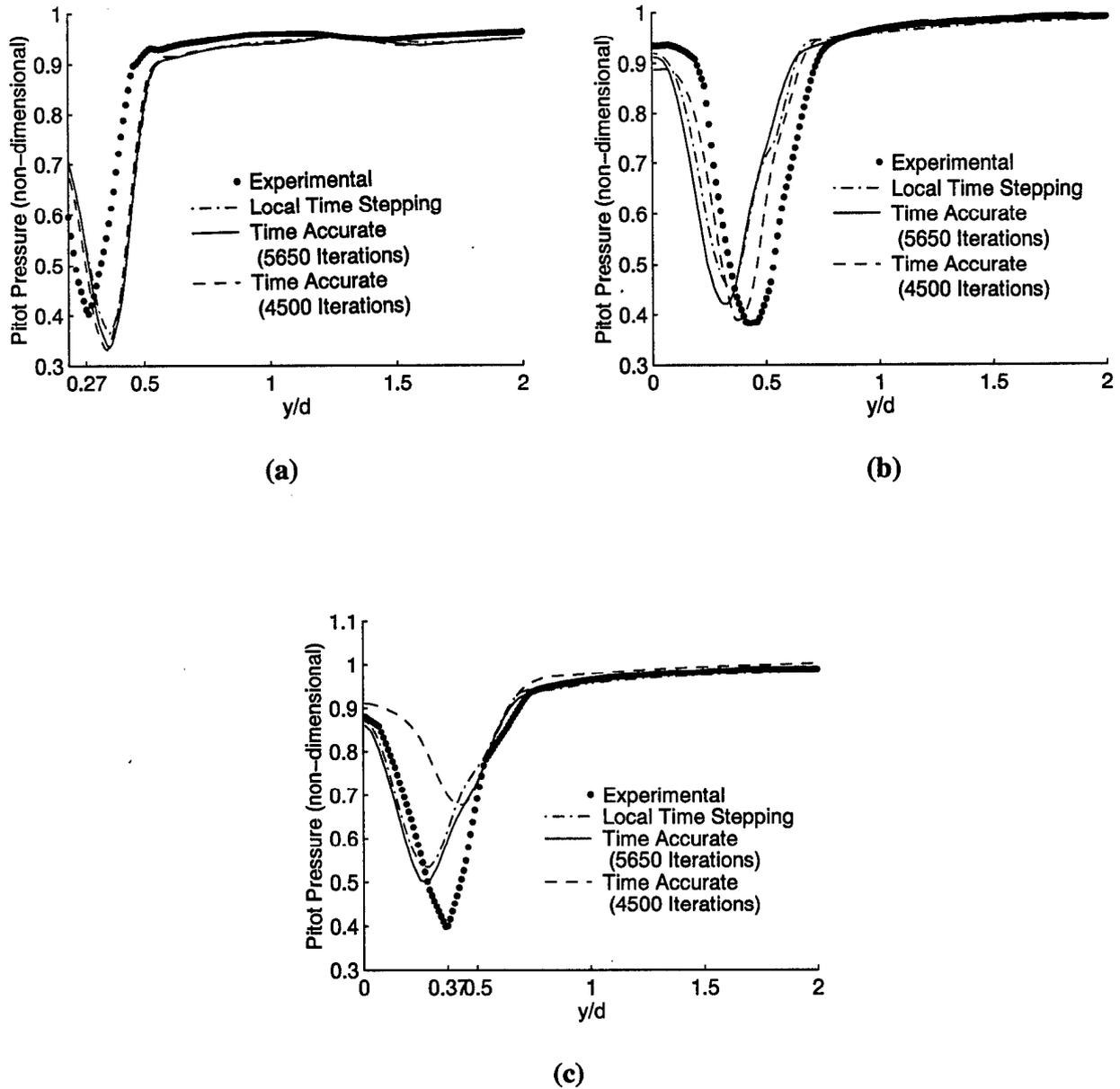


Figure 8. Vortex Core Evaluation for Time-Accurate Computation: (a) $x/d = 5.5$, (b) $x/d = 8.5$, and (c) $x/d = 11.5$.

Computations were begun using the standard Baldwin-Lomax turbulence model that uses the F_{\max} and y_{\max} length scales in determining the turbulent eddy viscosity, μ_t . F_{\max} is found to be searching in the direction normal to the missile surface for the maximum value of $F(y)$, and y_{\max} is the location of F_{\max} . In the vicinity of flow separation, a distinct F_{\max} and y_{\max} cannot be determined until the far field is reached. Attempts were made at limiting the region in which the search for F_{\max} took place. Runs were made in which only the first 30 and 15 points in the normal direction were computed as turbulent flow. Figure 9 shows the results of these attempts. Clearly, limiting the number of turbulent points normal to the missile surface had a significant effect, but the number of points to limit the search region changes as a function of the axial location.

A full implementation of Degani-Schiff modifications to the Baldwin-Lomax turbulence model was then added. The original Degani-Schiff modifications restricted the search for F_{\max} based upon the value of y_{\max} of the previous ray using

$$y_{\text{cutoff}}(\phi) = 1.5 y_{\max}(\phi - \Delta\phi).$$

If a maximum $F(y)$ is not found before y_{cutoff} is reached, the values of F_{\max} and y_{\max} from the previous ray are used. For case no. 3, this method did not limit F_{\max} as desired.

The Degani-Schiff criteria is simply a method of limiting F_{\max} ; therefore, the turbulence model was altered so that F_{\max} was used as the cutoff criteria rather than y_{\max} . In this method, the current value of F_{\max} is compared with that of the windward side. If there is a significant difference between the two, the current values of F_{\max} and y_{\max} are replaced with those from the previous ray. After some testing, the following criteria was settled upon.

$$\phi < 110^\circ$$

$$F_{\max} < 2.75 F_{\max 0},$$

$$\phi > 110^\circ$$

$$F_{\max} < 1.4 F_{\max 0},$$

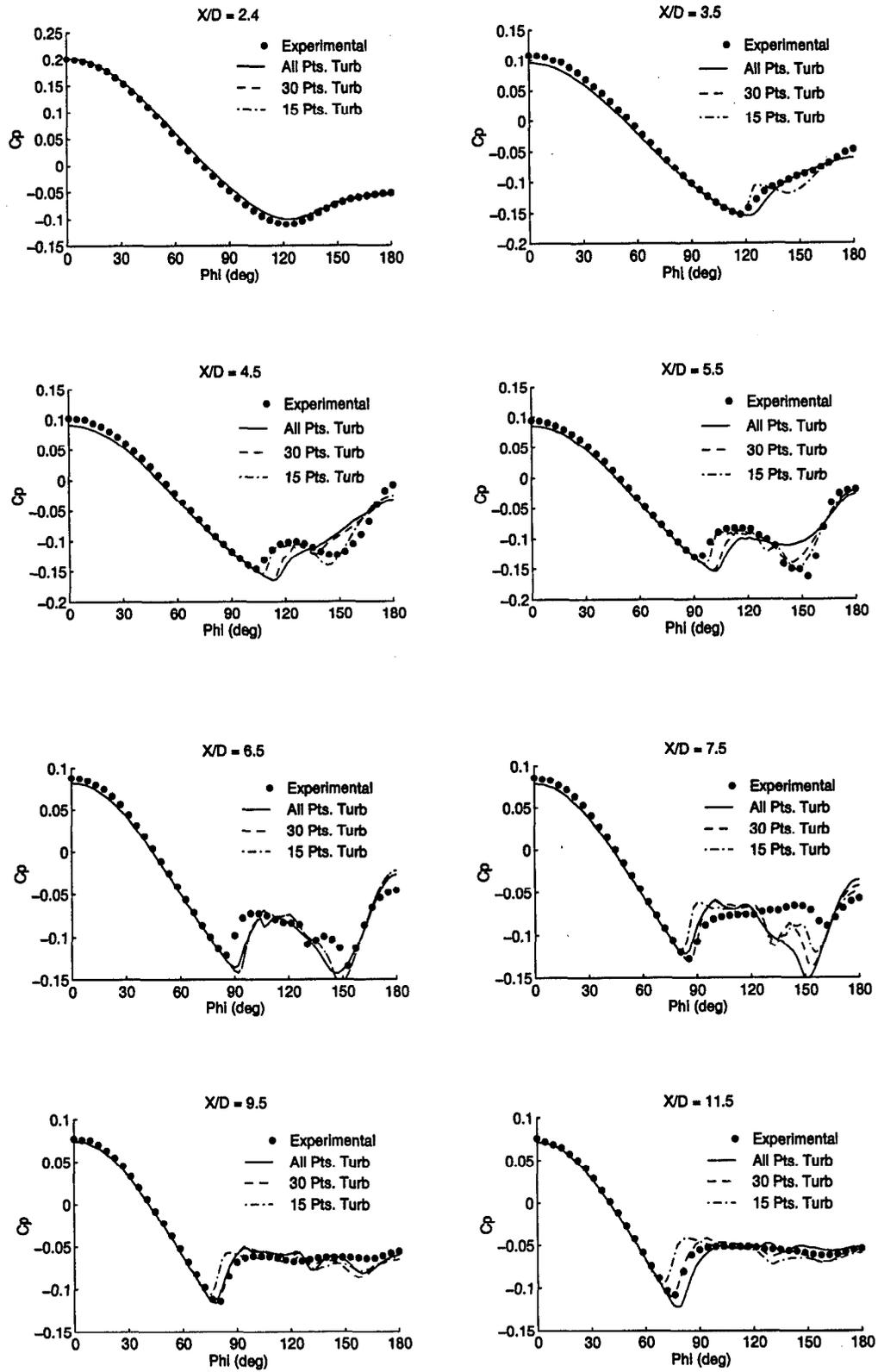


Figure 9. Effect of Limiting Turbulent Region on Surface Pressure Calculations for Case No. 3.

where $F_{\max 0}$ is the value of F_{\max} from the windward side. For ϕ less than 110° , F_{\max} must be sufficiently large to prevent separation from occurring too soon. Above 110° , a smaller cutoff is used to prevent the flow solver from adding excessive turbulence that does not really exist. As seen in Figure 10, this criteria significantly alters the calculation of F_{\max} . As seen in Figure 11, the modified turbulence model more accurately predicts flow separation.

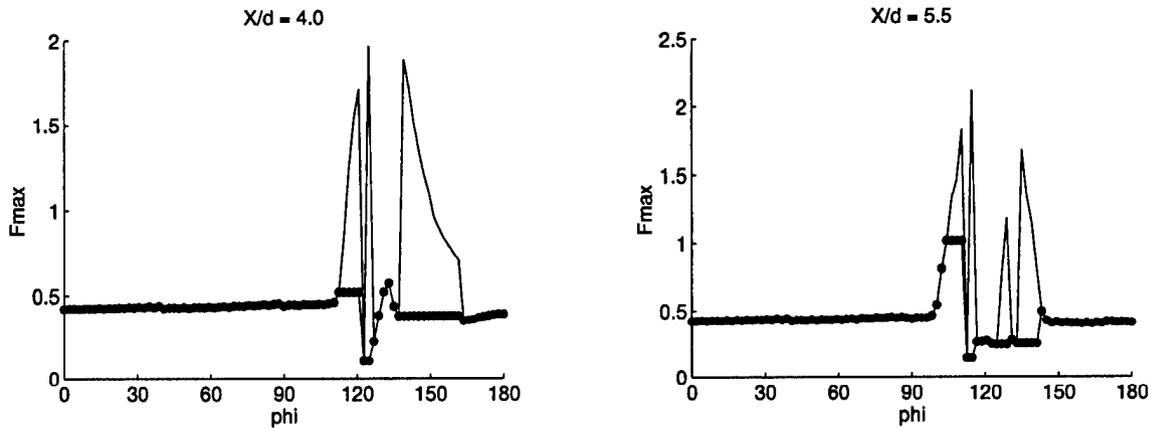


Figure 10. Effect of Degani-Schiff Modifications on Calculation of F_{\max} for Case No. 3: Solid Line Without Degani-Schiff, Dots With Degani-Schiff.

5.2 Parallel PVM Code. One approach to achieving parallel speedup in Overflow is by splitting the computational work into coarse-grained tasks that communicate using message passing. There are two options for accomplishing task parallelism within Overflow using PVM. One option invokes a manager-to-worker paradigm, wherein one processor takes on the role of manager and all communication between the other processes (the workers) must go through the manager. The other option uses worker-to-worker communication. In this option, the manager becomes a worker and, instead of having all data communication (i.e., message passing) go via the manager, communication occurs from peer to peer. The worker-to-worker paradigm reduces the total amount of communication but involves the use of PvmDataInPlace encoding of the messages being passed. Unfortunately, this encoding is not supported in SGI's implementation of PVM on either the PCA or the Origin2000. Thus, only the manager-to-worker paradigm could be tested in this study.

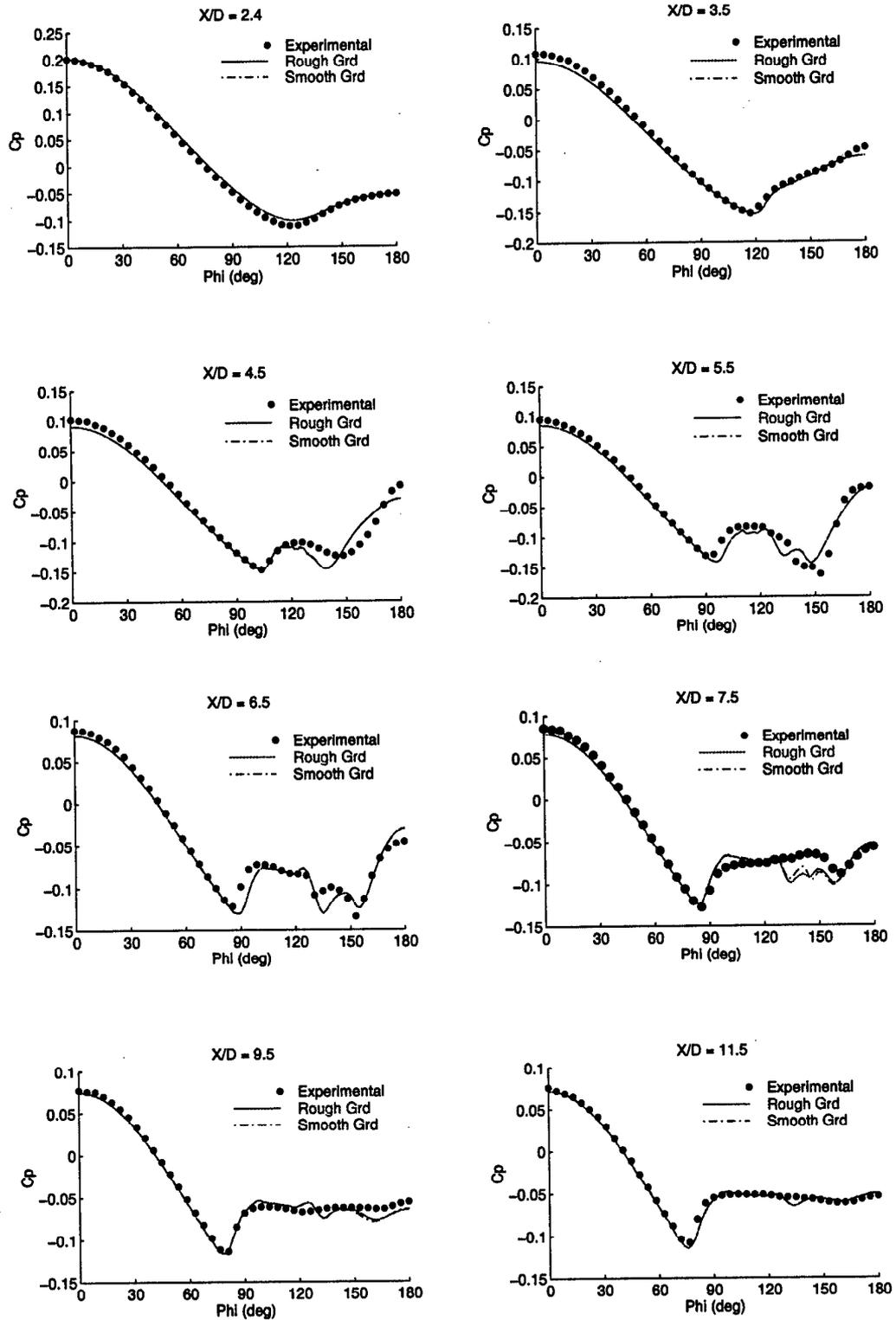


Figure 11. Effect of Degani-Schiff Modifications on Surface Pressure Calculations for Case No. 3.

The strategy of the manager-worker paradigm requires a spatial domain that has been decomposed into multiple subdomains or zones, which are then distributed over the specified number of central processing units (CPU). The decomposition of the spatial domain into multiple subdomains for the purpose of parallel computing is a topic worthy of discussion itself. However, for the purpose of this report, it is assumed that the decomposition has already been done and the required subdomains are available. It is up to the user to distribute the zones such that static load balancing is achieved.

The tasks of the manager code are to (1) enroll in PVM; (2) read the parallel name list that contains the number of machines, machine names, and executable program names and locations; (3) read the global name list that contains parameters normally initialized in the sequential run; (e.g., number of time steps, Mach no., boundary conditions, etc.); (4) allocate storage for the largest of the zones and for the Chimera interpolation data; (5) start worker processes; arrange to be notified if any worker quits; read input parameters, grid, and restart information for each zone; and send all of a zone's information to a worker; (6) monitor workers, saving checkpoint files, Chimera information, and dependent variables; if any worker dies, stop everything; (7) save final dependent state; and (8) exit PVM, halt.

The tasks of the worker code are to (1) enroll in PVM, (2) get task id of the manager and arrange to be notified if the manager dies, (3) receive global parameters and grid dimensions from the manager and allocate storage for the grid and dependent variables, (4) receive from manager other input parameters and restart information needed to perform computation, and (5) start working, (i.e., advance the solution of the Navier-Stokes equations, abort run if metrics, density, or pressure goes negative and tell manager; otherwise, send dependent variables and residual norms to manager as needed).

5.3 Parallel Loop Optimization Code. Another approach to obtaining parallel speedup is by exploiting loop-level parallelism in the code. Loop-level parallelism can take the form of data parallelism and/or task parallelism, wherein the iterations of a loop are distributed among the user-specified number of processors. Data parallelism involves loops with only computational manipulations, while task parallelism involves loops with subroutine calls. Either is accomplished by

by the addition of doacross SGI compiler directives before loops with significant work and no data dependencies.

An example of task parallelism from the Overflow code itself is shown as follows.

```
C$DOACROSS  SHARE (JPER, KS,KE,LS,LE,Q,VGAMMA,S,XX,XY,XZ,XT,  
C$&          JD,KD,LD,TMP2,NTMP2)  
C$&          LOCAL (L,ITMP2)  
  
      DO 100 L = LS,LE  
      CALL GETARX (NTMP2, TMP2, ITMP2)  
      CALL RECJ2  (JPER, KS,LE,L,Q,VGAMMA,S,XX,XY,XZ,XT,  
&                TMP2 (ITMP2,1,1), JD,KD,LD)  
      CALL FREARX (NTMP2, TMP2, ITMP2)  
100  CONTINUE
```

In Overflow, a select set of subroutines that should benefit from parallel execution is compiled with the -pfa option. The -pfa option to the FORTRAN compiler attempts to find and implement data parallelism automatically. It does not attempt to find task parallelism automatically. To assist the compiler, Overflow contains some user-inserted doacross directives within the select set of subroutines. The doacross directives essentially let the compiler know that the loop iterations do not carry data dependencies and it is "safe" to distribute the iterations of the loop among the processors. One advantage of loop-level parallelization is that it can be done incrementally, with checking for correctness and performance increases.

Experimentation with the automatic parallelizing -pfa option and the user-inserted compiler directives led to the results in Table 6. Recall that the sequential version of the code completes 50 iterations in 15:07. The second column shows what happens when -pfa is used on all of the subroutines in Overflow; that is, the executable generated gets into an infinite loop. The third column shows the timings generated when -pfa is used on the select subset of files thought to benefit

Table 6. Automatic Parallelizing Compiler Performance Results

No. of Processes	-pfa, All Subroutines, Compiler Directives On	-pfa, Select Subset of Subroutines, Compiler Directives Off	-mp, Select Subset of Subroutines, Compiler Directives On	-pfa, Select Subset of Subroutines, Compiler Directives On
1	—	15:07	15:08	15:20
4	—	13:48	6:41	4:42
8	—	13:42	4:48	2:47
16	126:20+	13:43	4:12	2:01
31	—	13:45	3:58	1:35

most from parallel execution, but with the user-inserted compiler directives commented out. Parallel speedup is practically nil, and there is no noticeable scalability. The fourth column uses an option to the FORTRAN compiler (-mp) that only produces parallel code for the loops preceded by user-inserted compiler directives. No automatic parallelism is attempted by the compiler. The timings show some speedup and scalability. The final column presents the timings that are obtained when the -pfa option is used on the select subset of subroutines and the user-inserted compiler directives are left on. This column produces a reasonable level of speedup and scalability. The table clearly indicates that the indiscriminate use of the -pfa option on a large scientific code like Overflow can lead to incorrect executable code and that even more thoughtful use of -pfa must be supplemented by user-inserted compiler directives for reasonable results to be obtained.

5.4 Performance of Parallel Codes. Table 7 presents Overflow performance timings from the sequential version of the code followed by timings using loop-level parallelization and, finally, using PVM message passing. All timings are based on 50 iteration runs that use -mips4 and -O3 options to the ftn77 compiler. Also, the PVM runs on the PCA were executed using CPUs within a single node. All timings were obtained while in dedicated mode. During times of normal usage, the system load fluctuates and it is very hard to get repeatable run times. In general, the run times for a heavily loaded node can increase by as much as 30–40%.

Table 7. Performance Timings

Code Version	No. of Processors	PCA (wall clock time) ^a	Origin2000 (wall clock time) ^a
Sequential	1	33:01	15:07
Loop-level parallelization, compiler directives, and -pfa	1	33:09	15:20
	4	8:24	4:42
	8	5:29	2:47
	16	—	2:01
	31	—	1:35
PVM message passing	4	9:31	5:35
	8	7:58	3:11
	16	—	1:57

^a Timing obtained while in time-sharing mode.

While the loop-level parallelization paradigm appears to produce better scalability in the range of 1–8 processors, scalability falls off in the range of 9–31 processors, and, in fact, the message-passing paradigm appears to be producing a slightly better timing (1:57) than the timing (2:01) produced by the current loop-level parallel implementation of the code for the case of 16 processors on the Origin2000.

Figure 12 plots the speedup vs. the number of processors on the Origin2000 for the loop-level parallel code. Above six processors, the scalability of the code falls significantly short of ideal. However, given the code is not embarrassing parallel, a better comparison to make is to Amdahl's law: $\text{Speedup}(n) = 1/((p/n) + (1-p))$, where n equals number of processors and p equals fraction of the program's code that has been made parallel.

The speedup predicted by Amdahl's law and plotted in Figure 12 is obtained by measuring the execution times $T(1)$ and $T(2)$ for a one-processor and two-processor case, respectively. Then, $\text{Speedup}(2) = T(1)/T(2)$. Further, Amdahl's law is rearranged to yield the fraction (p) of the program's code that can be made parallel: $p = 2 * (\text{Speedup}(2) - 1)/(\text{Speedup}(2))$. This fraction can

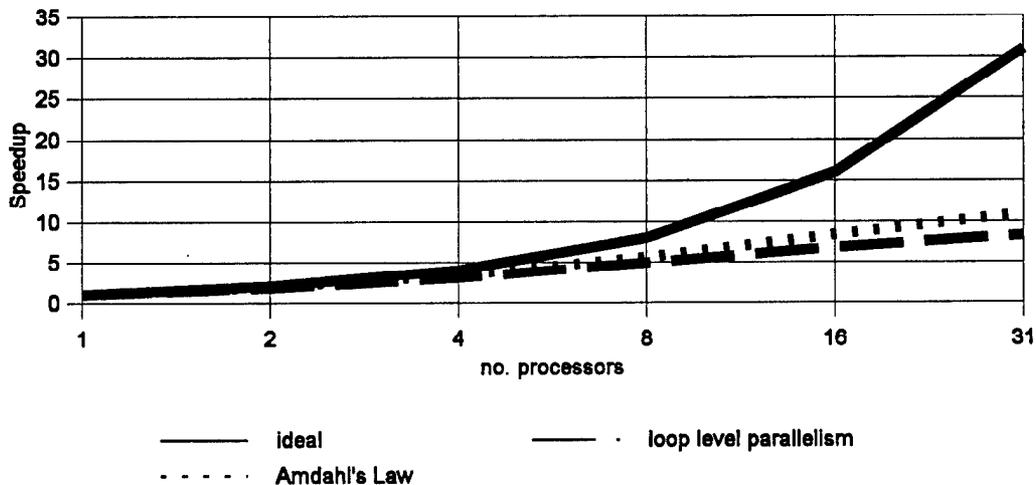


Figure 12. Scalability of Loop-Level Parallel Code Implementation.

then be used to extrapolate the speedup for cases where the number of processors is greater than two. However, Amdahl's law assumes that all CPUs are equal and that the units of work assigned to each CPU are also equal. Thus, it is important to check the two-processor case to make sure that indeed each processor is doing the same amount of work. In this study, 50 iterations did not produce good load balancing, while 1,000 iterations did for the two-processor case. The timings for the 1,000-iteration case produced a $\text{Speedup}(2) = 1.89$ and a parallel fraction $p = 0.939$.

In Figure 12, the actual speedup is slightly lower than Amdahl's law predicts. There can be many reasons why the actual speedup is less than predicted. Contributors can be discovered by looking at performance data. Performance data were collected on the Origin2000 using the profiling tools *perfex* and *speedshop*.

The R10000 design in the Origin2000 provides hardware support for counting various types of events, such as cache misses, memory coherence operations, and graduated (completed) floating-point operations. These counters are useful for gaining insight into where and how time is spent in an application and for discovering performance bottlenecks. The counters were designed by SGI to extract information without affecting the behavior of the program being monitored and to not degrade the performance of other hardware.

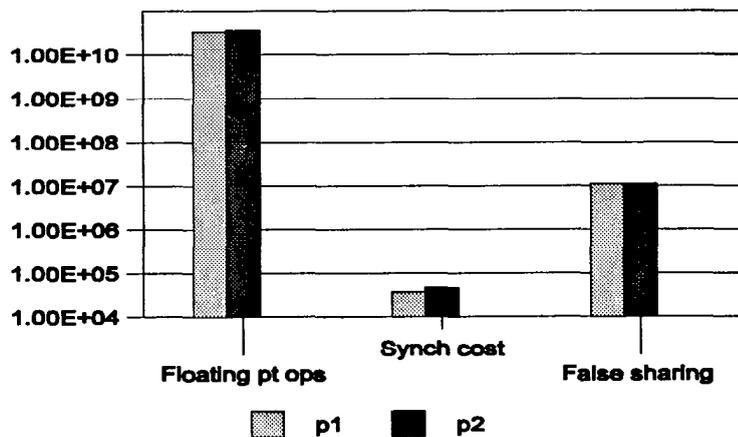
Several sources of poor scaling that can be examined by looking at event counts are load imbalance, excessive synchronization costs, and false sharing. Load imbalance can be checked by determining whether all threads issue a similar number of graduated floating-point operations. Excessive synchronization costs are determined by examining whether the counts of store conditionals are high. Finally, if false sharing is a problem, then the counts of store exclusive to shared blocks should be high. False sharing is a potential problem in any cache-based, SMP system. False sharing occurs when two or more processors access different variables that happen to be colocated on the same cache block, with at least one of the accesses being a write. Once the write occurs, the entire cache line is invalidated to other processors. Thus, any attempt by the other processors to use another data item in the cache line will require the entire cache line to be updated first.

Figure 13 presents the event counts obtained for floating-point operations, synchronization, and false sharing. It is evident from the bar graph that the majority of the run-time cost is being spent on floating-point operations followed by false sharing and, finally, synchronization. The countervalues multiplied by the typical event times did not change the relative magnitude and ordering of the counters. In addition, note that the last processor is always performing more work than the others, signifying load imbalance. To increase the overall scalability of the code past eight processors requires more effort in the parallelization effort of the remaining sequential code. Then, the next step to improving the comparison to Amdahl's law should probably involve trying to improve load balance.

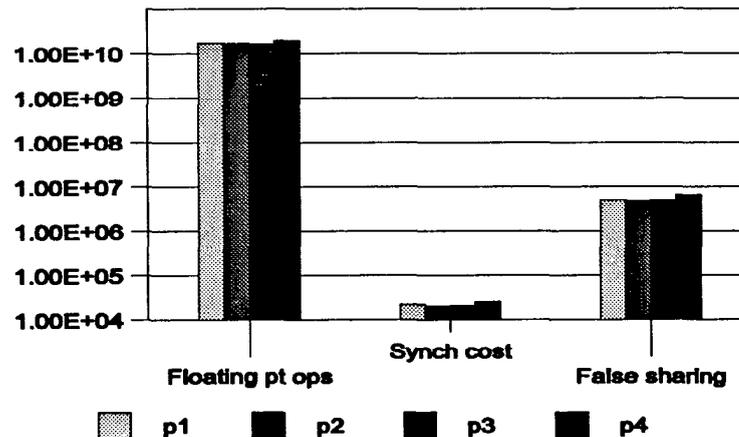
6. Conclusions/Future Work

The validation part of this study clearly illustrates that a laminar solution is not adequate for flows with moderate amounts of separation. By making simple modifications to the Baldwin-Lomax turbulence model, significantly more accurate surface pressure predictions can be made. In some cases (case nos. 1, 2, and 6) the strength of the vortex core becomes washed out.

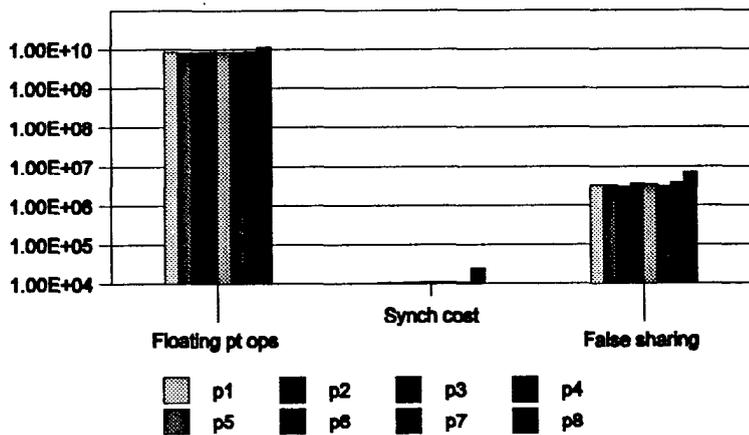
Event Counts for 2 CPUs



Event Counts for 4 CPUs



Event Counts for 8 CPUs



Event Counts for 16 CPUs

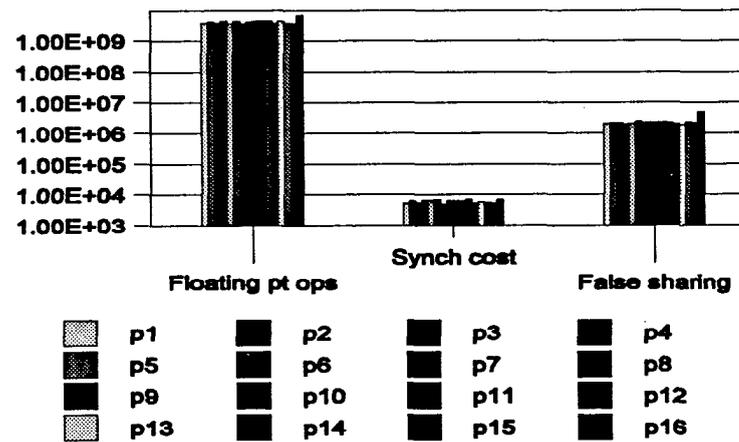


Figure 13. Floating-Point Operations, Synchronization Cost, and False Sharing Event Counts.

The flow solver was found to be straightforward and easy to use. The manual (Buning 1995), though, could be significantly improved. As stated in the manual, it is intended for users familiar with the flow solver F3D. The manual mostly describes differences between Overflow and F3D, input and output files, and name list specifications. There is a small section on experiences with Overflow. Many little helpful hints, such as the need to add two planes by reflected symmetry to enforce symmetry about the vertical axis, are not mentioned in the manual.

On the SGI PCA, performance utilizing a single processor was just adequate, as 2,000 iterations could be completed in less than a 24-hr wall clock time. On the Origin2000, the same run could be completed in less than a 12-hr wall clock time. Significant improvements in wall clock time could be obtained by using a parallel version of the code. The loop-level parallel version of Overflow performed the best on both the Origin2000 and the PCA in the range of two to eight processors. However, ideal scalability significantly fell off above eight processors on the Origin2000. As confirmed by an analysis using Amdahl's law, the scalability above eight processors is severely limited by the fraction of the code being run sequentially. The actual speedup is even less than Amdahl's law predicts due mainly to load imbalance. Even with the scalability limitations just described, the loop-level parallel version of the code achieves a speedup of approximately 7.5 and 10 using 16 and 31 processors of the Origin2000, respectively. The message-passing version of the code performed similarly. Thus, both loop-level parallelization and message passing are viable paradigms for a shared memory or distributed shared memory architecture. The shared memory paradigm has the advantage of being easier to program, but has the distinct disadvantage of being less portable in an efficient manner than a message-passing implementation for distributed memory architectures. However, research into compile-time and run-time software distributed shared memory systems (Dwarkadas, Cox, and Zwaenepoel 1996) may eventually make shared memory as efficient as message passing, even on distributed memory architectures.

Future work will focus on additional loop-level parallelization of the remaining sequential fraction of Overflow's program. Also, a larger grid size will be used to determine if the sequential code increases, decreases, or stays the same with increasing grid size. If the sequential fraction of code decreases, the scalability should improve. Additional comparisons to the message-passing

paradigm will be continued for a larger numbers of processors. However, setting up runs for message-passing runs becomes very tedious above 16 subdomains; therefore, automatic-domain decomposition tools will be studied as well. Finally, preliminary performance profiling and analyses are presented in this report. Future work will further investigate the costs associated with sequential vs. parallel code and the costs due to memory hierarchy accesses.

7. References

- Baldwin, B. S., and H. Lomax. "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flow." AIAA Paper 78-257, January 1978.
- Buning, P., et al. *OVERFLOW User's Manual*. NASA-Ames Research Center, June 1995.
- Degani, D., and L. B. Schiff. "Computation of Supersonic Viscous Flows Around Pointed Bodies at Large Incidence." AIAA Paper 83-0034, January 1983.
- Dwarkadas, S., A. Cox, and W. Zwaenepoel. "An Integrated Compile-Time/Run-Time Software Distributed Shared Memory System." White Paper, Department of Computer Science, Rice University, 1996.
- Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PUM3 User's Guide and Reference Manual*. ORNL-TM-12187, May 1993.
- Gropp, W., E. Lusk, and A. Skjellum. "Using MPI Portable Parallel Programming With the Message-Passing Interface." MIT Press, Cambridge, MA, 1994.
- Snir, M., S. Otto, S. Huss-Lederman, and D. Walker. "MPI: the Complete Reference." MIT Press, Cambridge, MA, 1994.
- Sturek, W., et al. "The Application of CFD to the Prediction of Missile Body Vortices." AIAA Paper 97-0637, January 1997.

INTENTIONALLY LEFT BLANK.

Appendix:

Plots for KTA Case Nos. 1, 2, and 4-6

INTENTIONALLY LEFT BLANK.

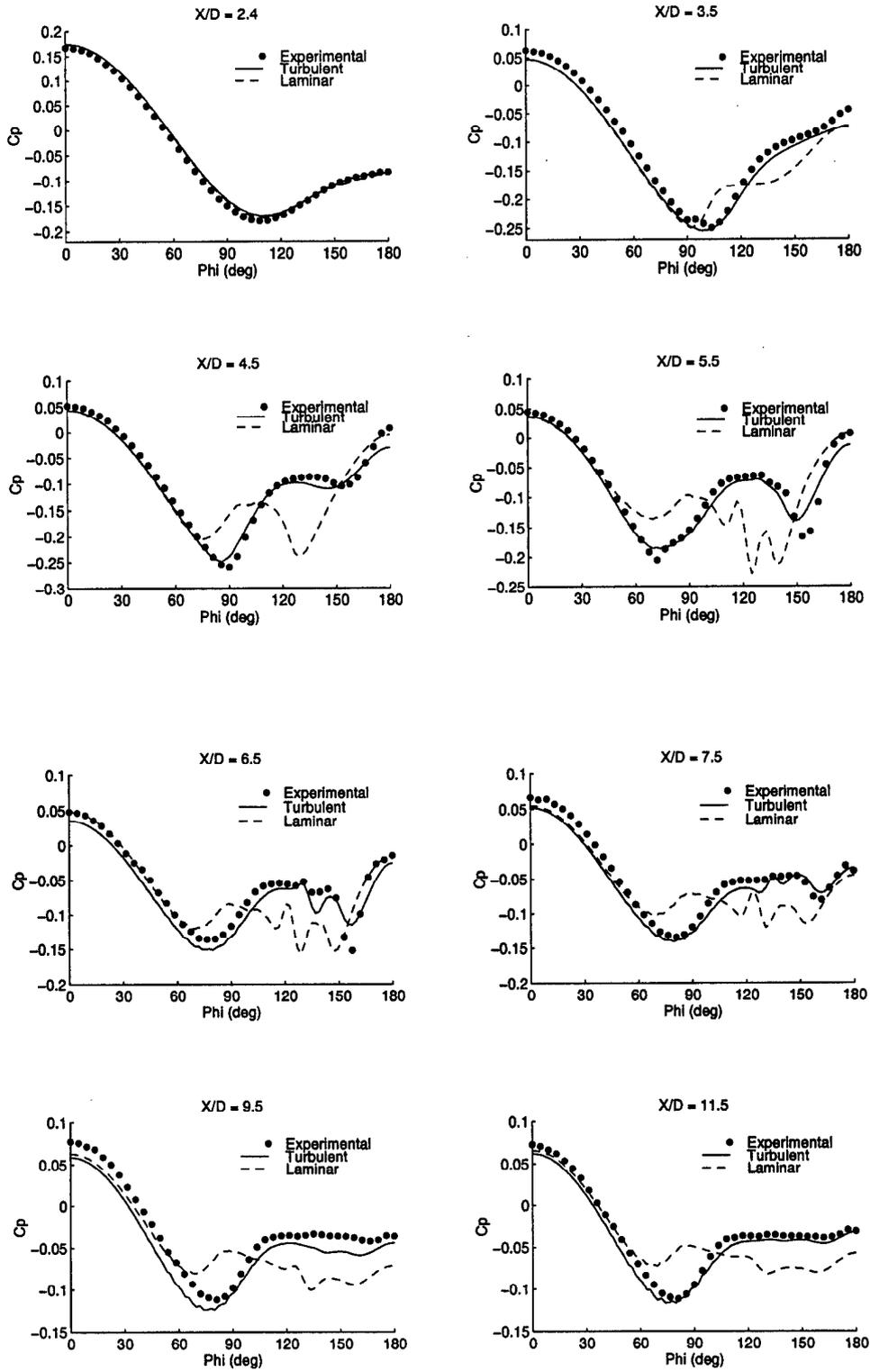


Figure A-1. Surface Pressure Plots for Case No. 1: Mach = 1.45, AOA = 14°, and Re = 667,000.

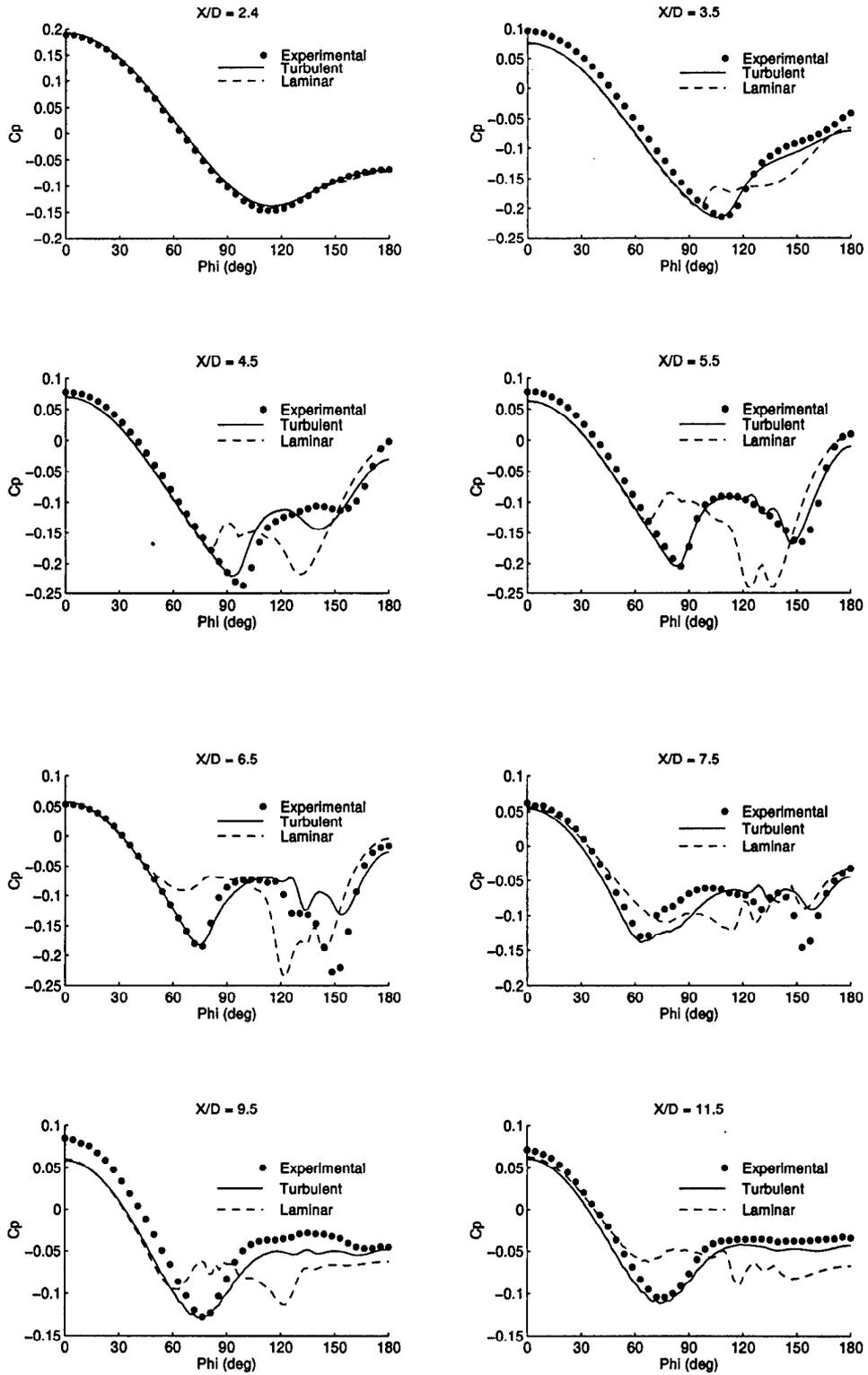


Figure A-2. Surface Pressure Plots for Case No. 2: Mach = 1.8, AOA = 14°, and Re = 667,000.

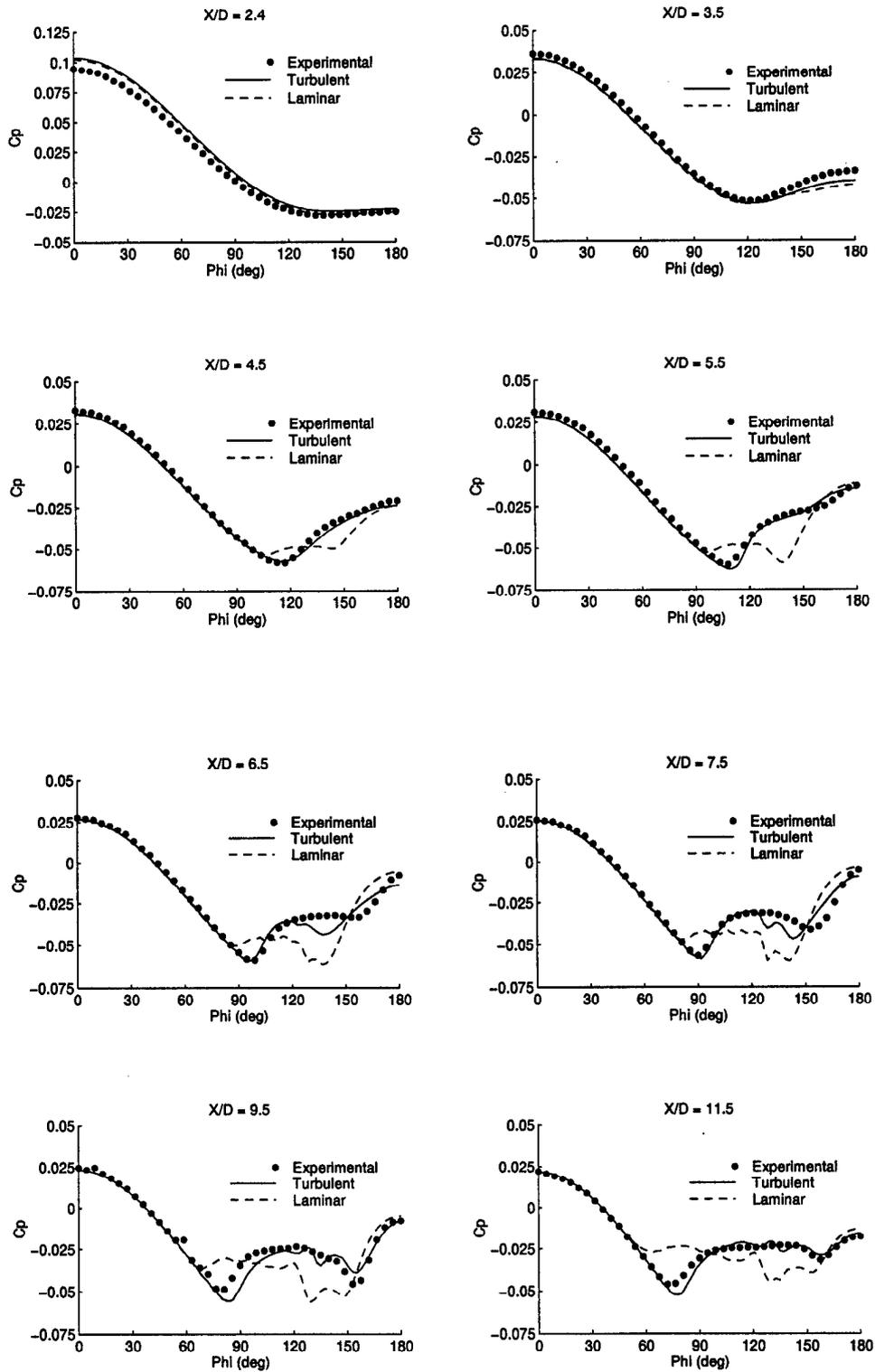


Figure A-3. Surface Pressure Plots for Case No. 4: Mach = 3.5, AOA = 8°, and Re = 1,123,000.

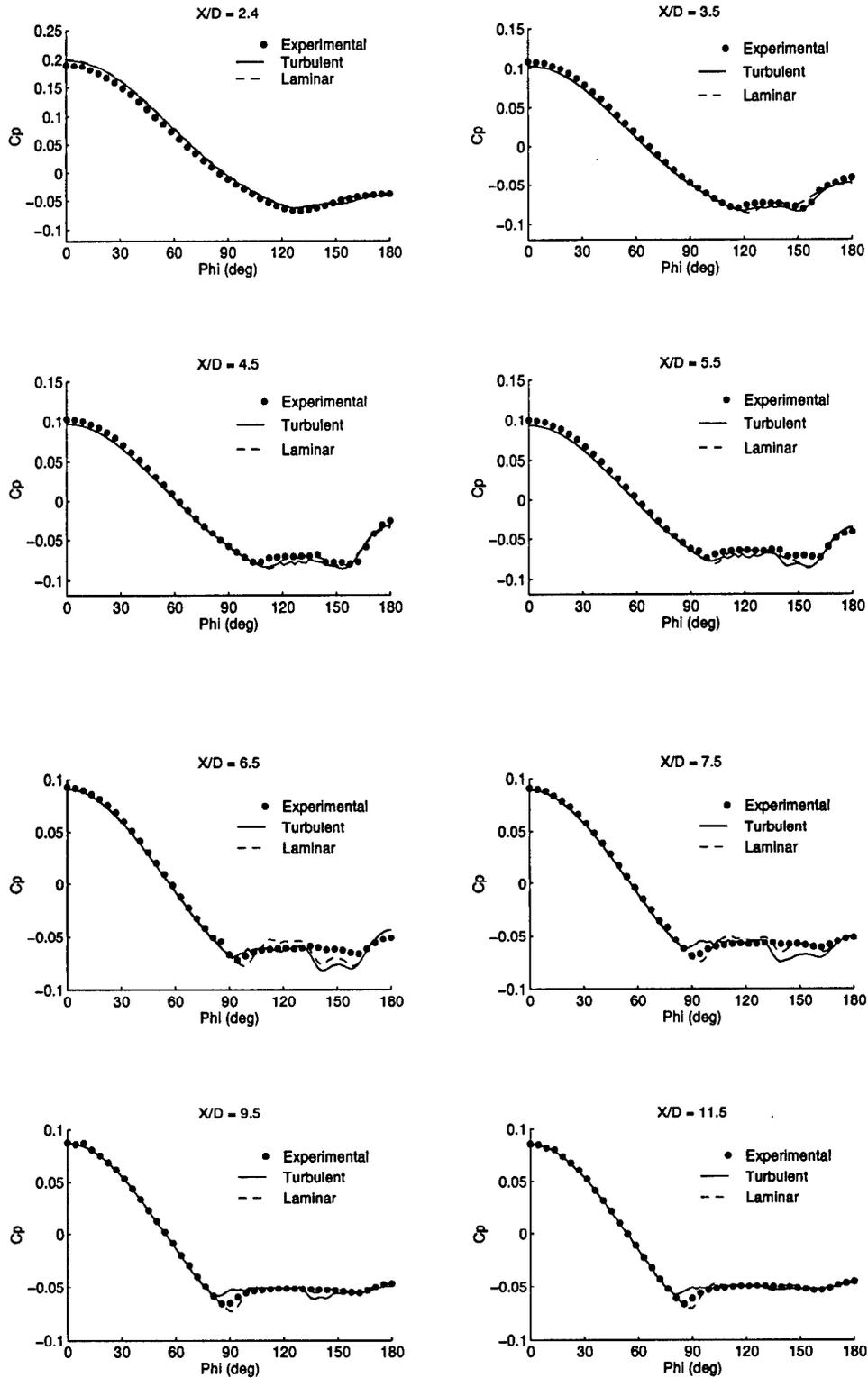


Figure A-4. Surface Pressure Plots for Case No. 5: Mach = 3.5, AOA = 14°, and Re = 1,123,000.

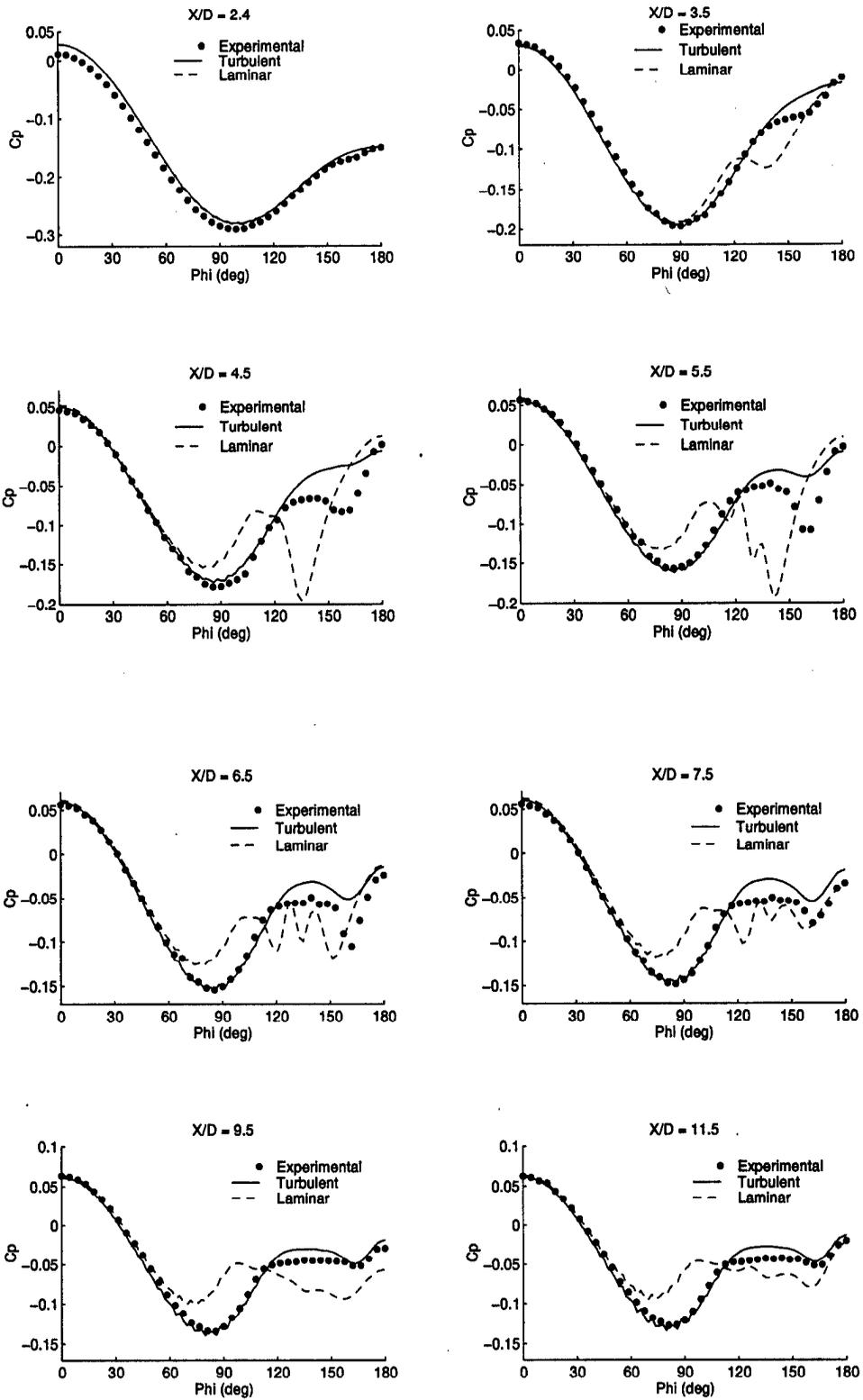


Figure A-5. Surface Pressure Plots for Case No. 6: Mach = 0.7, AOA = 14°, and Re = 667,000.

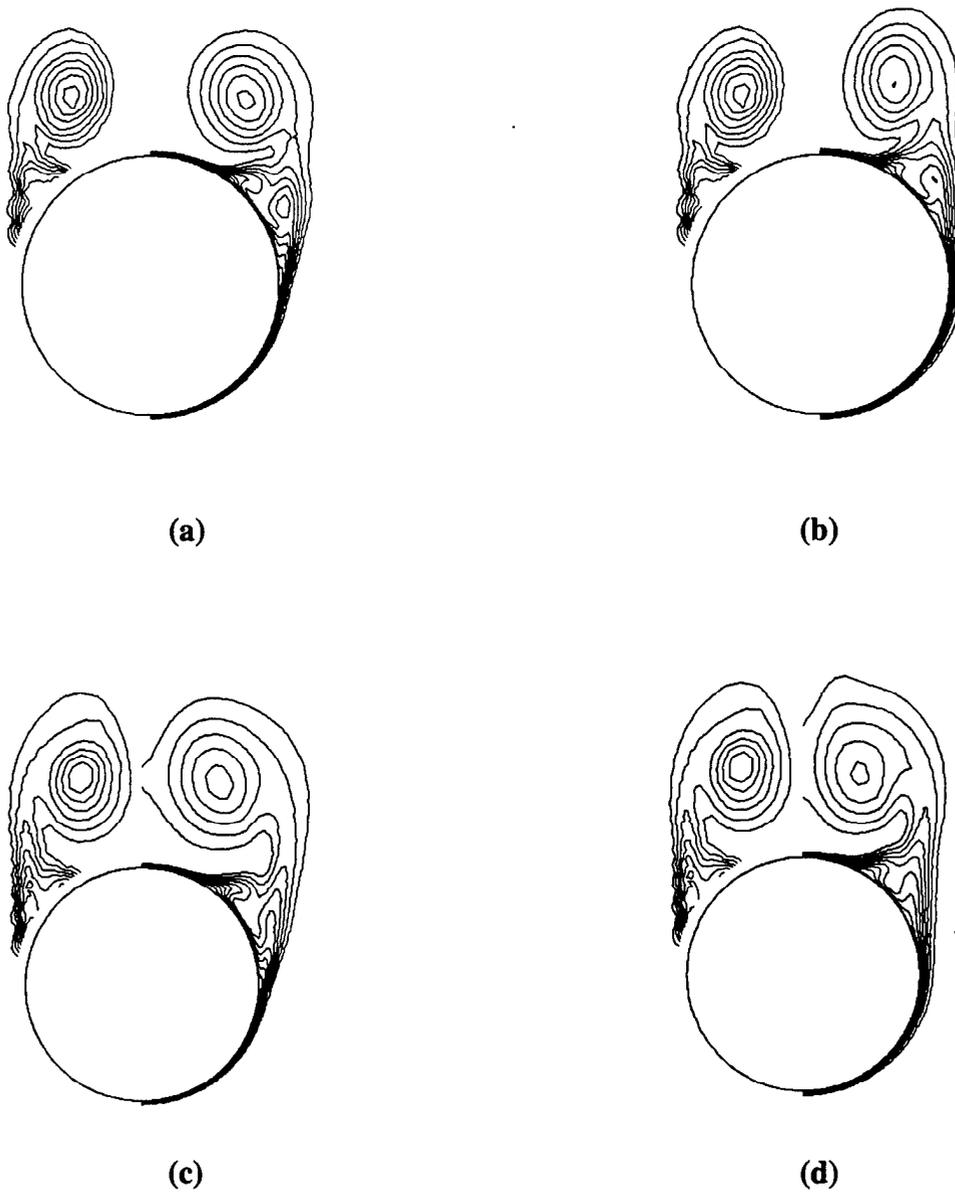
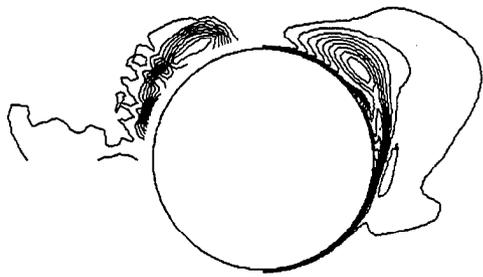
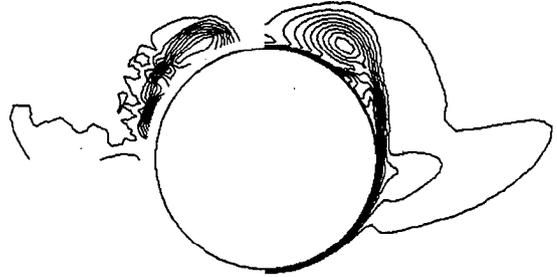


Figure A-6. Pitot Pressure Contours for Case No. 1: (a) $x/d = 8.5$, Laminar Computation; (b) $x/d = 8.5$, Turbulent Computation; (c) $x/d = 11.5$, Laminar Computation; and (d) $x/d = 11.5$, Turbulent Computation.



(a)



(b)



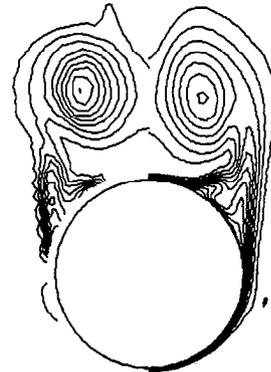
(c)



(d)

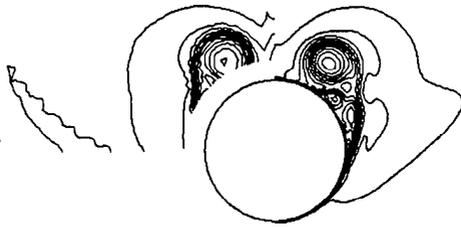


(e)

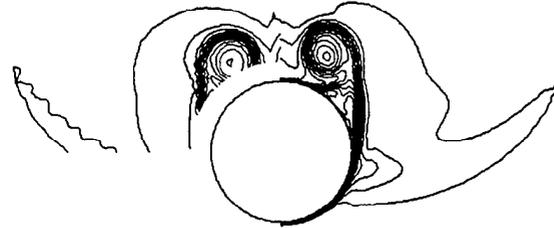


(f)

Figure A-7. Pitot Pressure Contours for Case No. 2: (a) $x/d = 5.5$, Laminar Computation; (b) $x/d = 5.5$, Turbulent Computation; (c) $x/d = 8.5$, Laminar Computation; (d) $x/d = 8.5$, Turbulent Computation; (e) $x/d = 11.5$, Laminar Computation; and (f) $x/d = 11.5$, Turbulent Computation.

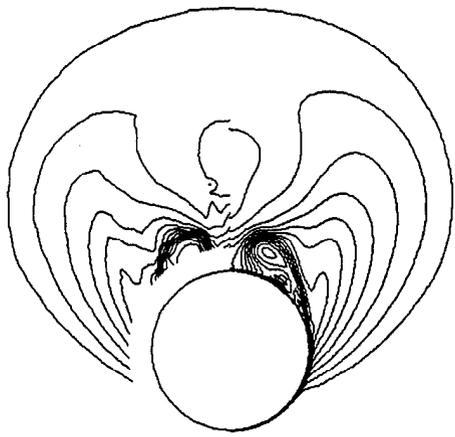


(a)

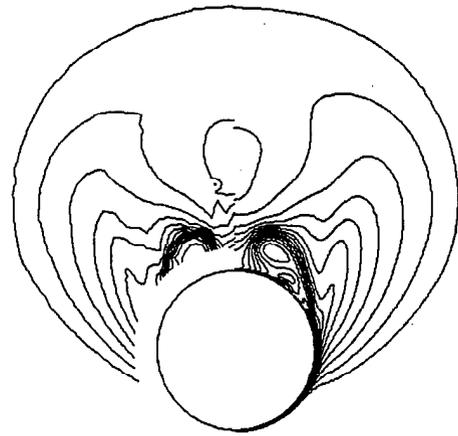


(b)

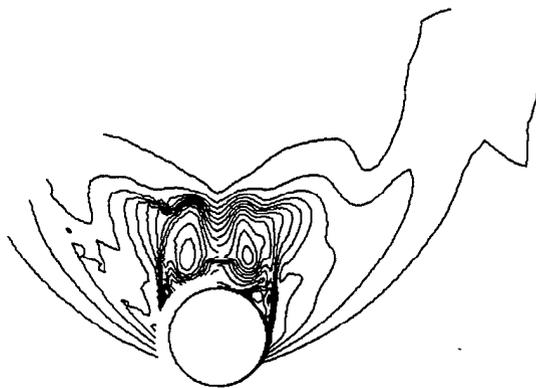
Figure A-8. Pitot Pressure Contours for Case No. 4: (a) $x/d = 11.5$, Laminar Computation; and (b) $x/d = 11.5$, Turbulent Computation.



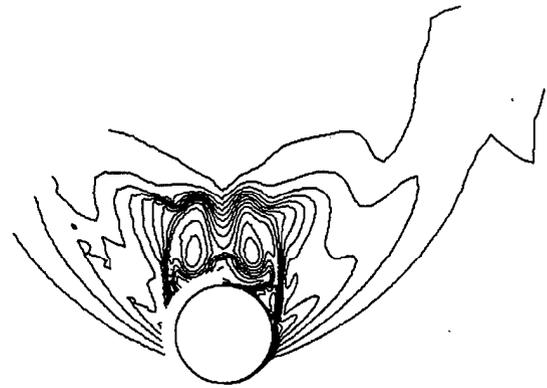
(a)



(b)



(c)



(d)

Figure A-9. Pitot Pressure Contours for Case No. 5: (a) $x/d = 5.5$, Laminar Computation; (b) $x/d = 5.5$, Turbulent Computation; (c) $x/d = 11.5$, Laminar Computation; and (d) $x/d = 11.5$, Turbulent Computation.



(a)



(b)



(c)



(d)

Figure A-10. Pitot Pressure Contours for Case No. 6: (a) $x/d = 8.5$, Laminar Computation; (b) $x/d = 8.5$, Turbulent Computation; (c) $x/d = 11.5$, Laminar Computation; and (d) $x/d = 11.5$, Turbulent Computation.

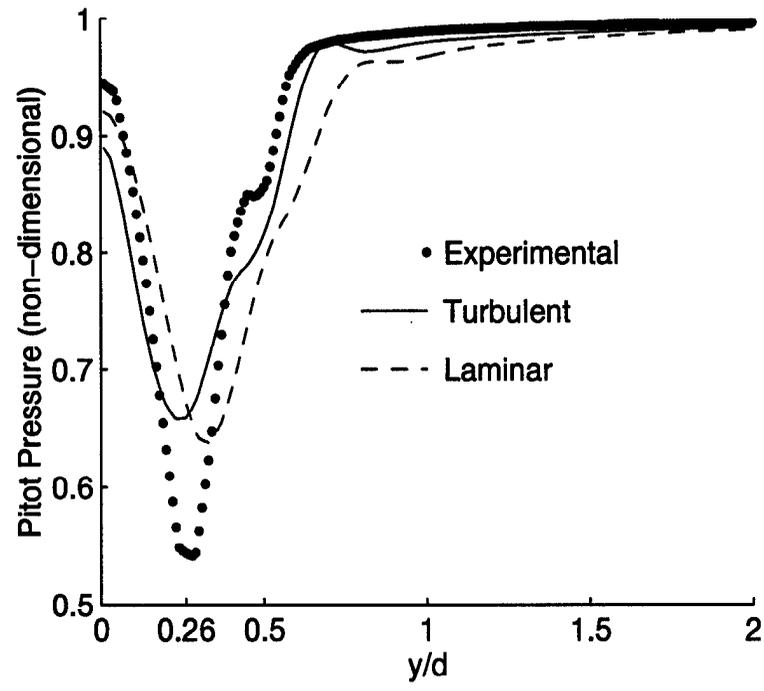
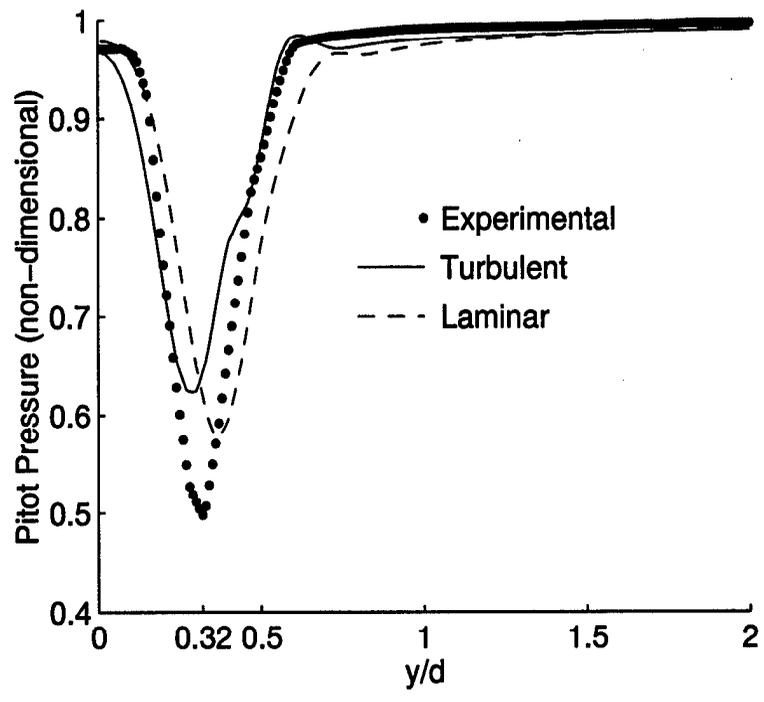


Figure A-11. Vortex Core Evaluation for Case No. 1: (a) $x/d = 8.5$, and (b) $x/d = 11.5$.

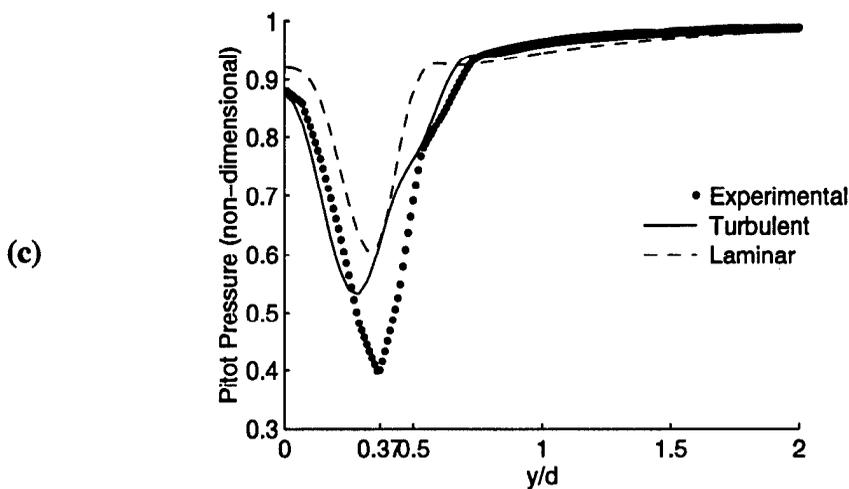
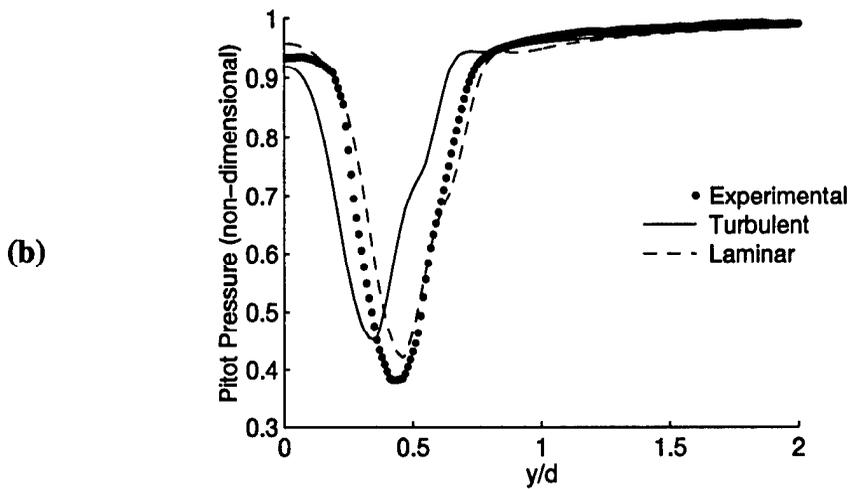
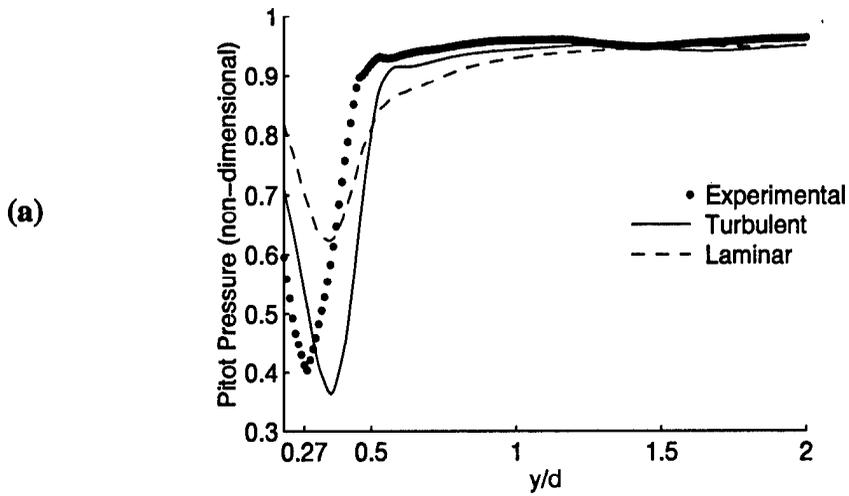


Figure A-12. Vortex Core Evaluation for Case No. 2: (a) $x/d = 5.5$, (b) $x/d = 8.5$, and (c) $x/d = 11.5$.

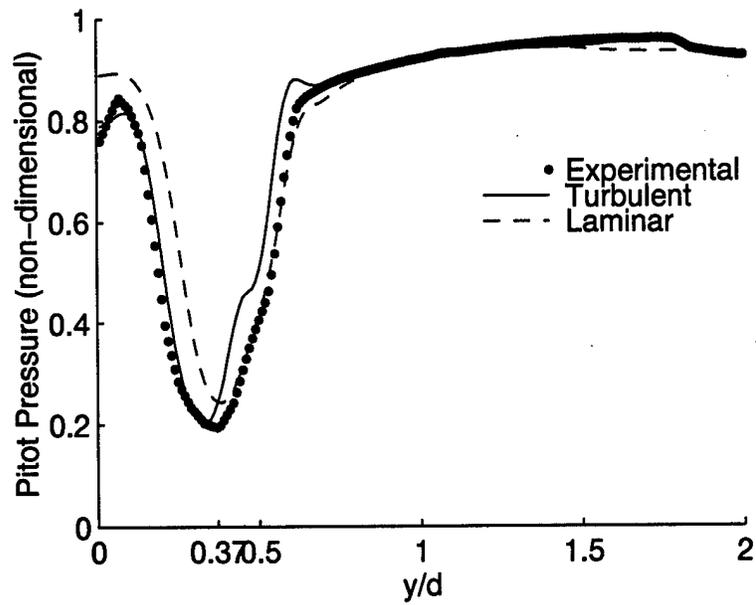


Figure A-13. Vortex Core Evaluation for Case No. 4: $x/d = 11.5$.

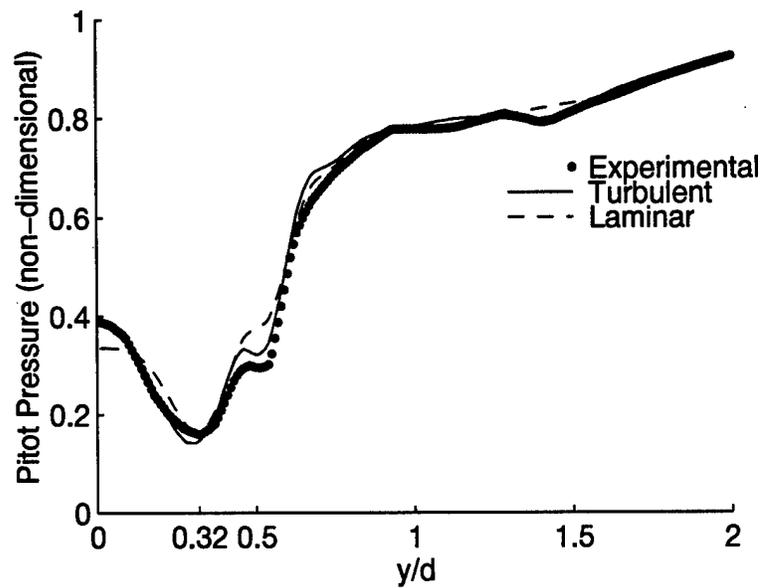
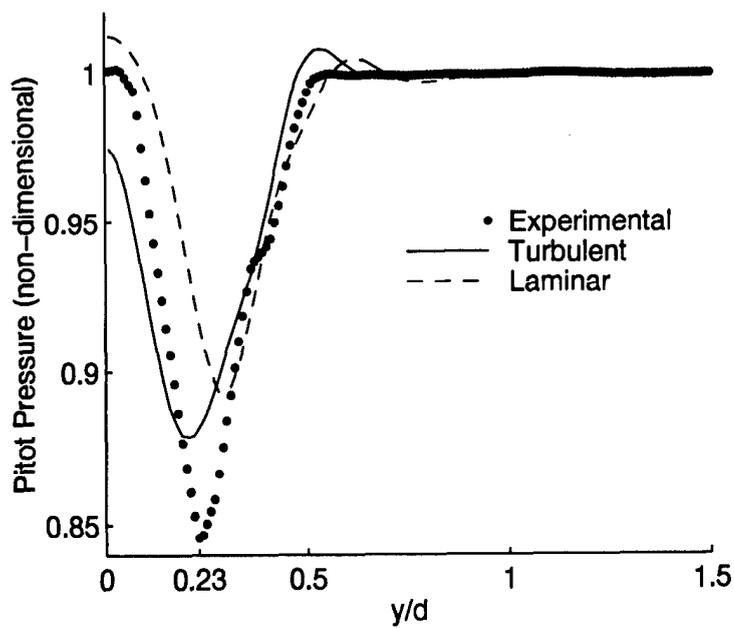
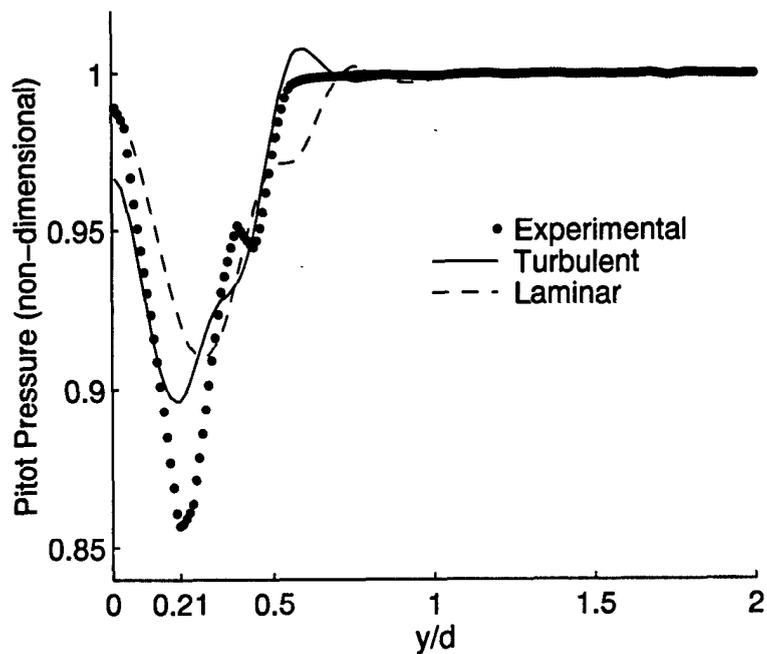


Figure A-14. Vortex Core Evaluation for Case No. 5: $x/d = 11.5$.

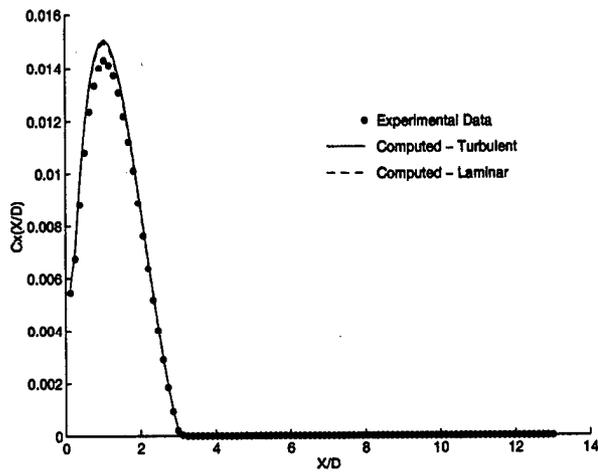


(a)

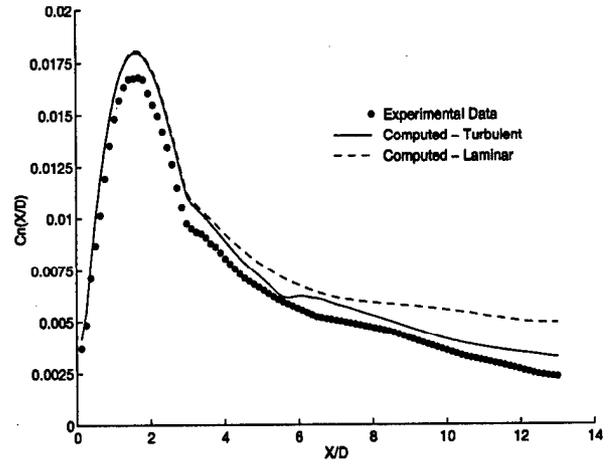


(b)

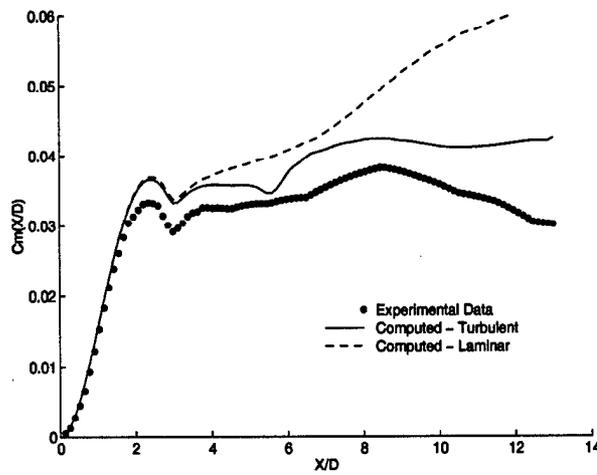
Figure A-15. Vortex Core Evaluation for Case No. 6: (a) $x/d = 8.5$, and (b) $x/d = 11.5$.



(a)

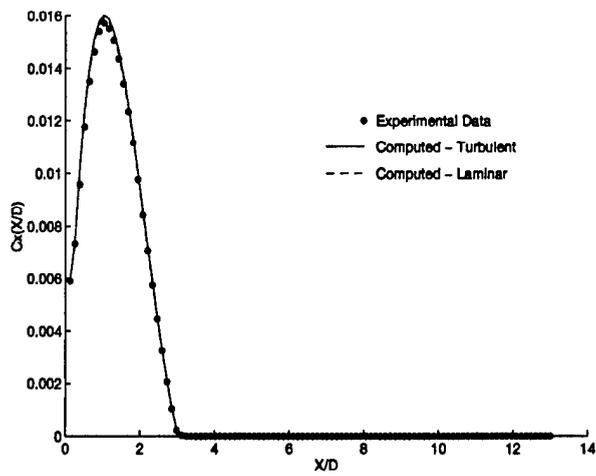


(b)

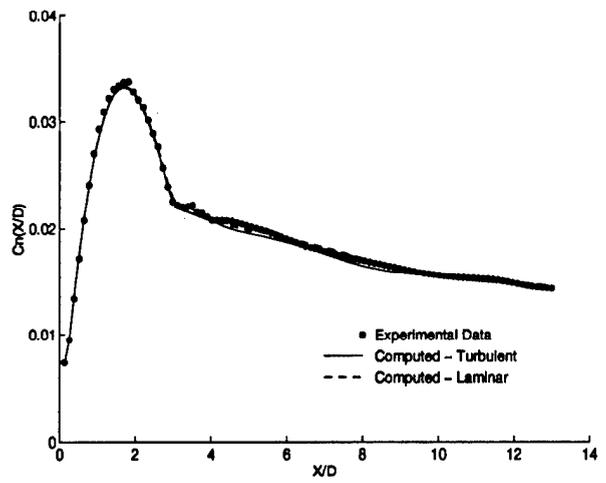


(c)

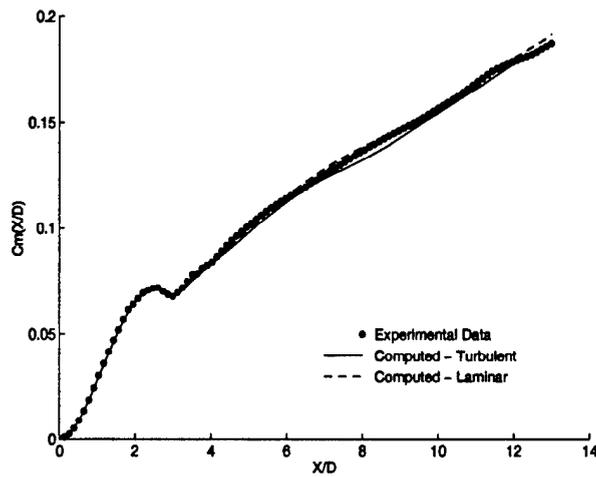
Figure A-16. Force and Moment Coefficient vs. Axial Location for Case No. 4: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment.



(a)



(b)



(c)

Figure A-17. Force and Moment Coefficient vs. Axial Location for Case No. 5: (a) Axial Force, (b) Normal Force, and (c) Pitching Moment.

List of Abbreviations and Symbols

AOA	angle of attack
ARL	U.S. Army Research Laboratory
BL	Baldwin-Lomax
CFD	computational fluid dynamics
C_m	pitching-moment coefficient
C_n	normal force coefficient
C_p	coefficient of pressure
C_x	axial force coefficient
DOD	Department of Defense
DS	Degani-Schiff
F_{\max}	maximum of function $F(y)$
GB	gigabyte(s)
Hg	mercury
K	Kelvin
KTA	key technical area: the application of computational fluid dynamics (CFD) to the prediction of missile-body vortices
MB	megabyte(s)
MHz	megahertz
MPI	message-passing interface, a message-passing library
MSRC	major shared resource center
ρ	density
PCA	Power Challenge Array
P_o	stagnation pressure
PVM	parallel virtual machine, a message-passing library
Q	vector of dependent variables
Re	Reynolds number
RISC	reduced instruction set computer
SGI	Silicon Graphics Incorporated
SMP	shared-memory multiprocessor
T_o	stagnation temperature
UMA	uniform memory access
u_t	friction velocity, $\sqrt{\tau_w/\rho_w}$
y^+	law-of-the-wall coordinate, $\rho_w u_t y/\mu_w$
y_{\max}	value of y at which $F(y)$ is maximum
μ	first coefficient or molecular coefficient of viscosity
μ_t	eddy viscosity coefficient
τ	viscous stress

INTENTIONALLY LEFT BLANK.

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ D SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDE HQ US ARMY MATERIEL CMD AMCRD MG CALDWELL 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MAJ M D PHILLIPS THAYER HALL WEST POINT NY 10996-1786

<u>NO. OF COPIES</u>	<u>ORGANIZATION</u>
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD J J ROCCHIO 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AS (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	<u>ABERDEEN PROVING GROUND</u>
4	DIR USARL AMSRL CI LP (305)

NO. OF
COPIES ORGANIZATION

14 AMSRL CI C NIETUBICZ
 AMSRL CI HA W STUREK
 AMSRL CI HC
 D PRESSEL
 D HISLEY
 C ZOLTANI
 J GROSH
 A PRESSLEY
 P DYKSTRA
 AMSRL SC S A MARK
 AMSRL WM B
 J SAHU
 K HEAVEY
 H EDGE
 P WEINACHT
 AMSRL WM BE M NUSCA

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE August 1999	3. REPORT TYPE AND DATES COVERED Final, May 96 - Jun 97	
4. TITLE AND SUBTITLE Performance of a Sequential and Parallel Computational Fluid Dynamic (CFD) Solver on a Missile Body Configuration		5. FUNDING NUMBERS 423612.000 MS4B	
6. AUTHOR(S) Dixie Hisley and Duane Frist			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: AMSRL-CI-HC Aberdeen Proving Ground, MD 21005-5067		8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-2032	
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) In order to effectively port production codes originally written for vector processors to reduced instruction set (RISC)-based parallel computers, different paradigms have been tried by the parallel computing community. Among the techniques used are message-passing and loop-level parallelization using hand-inserted compiler directives or automatic-parallelizing compiler flags. The goals of this report are (1) to investigate the performance of message-passing and loop-level parallelization techniques, as they were implemented in the computational fluid dynamics (CFD) code overflow, and (2) to validate the sequential and parallel results obtained on a demonstration problem of interest to the Army—that is, a generic missile-body configuration. The computational simulations were run, and performance data were gathered on a Silicon Graphics Incorporated (SGI) Power Challenge Array (PCA) and Origin2000.			
14. SUBJECT TERMS loop-level parallelization, message-passing parallelization, computational fluid dynamics, computer simulations, SGI Power Challenge Array, SGI Origin2000		15. NUMBER OF PAGES 54	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2032 (Hisley) Date of Report August 1999

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS	_____
	Organization
	Name E-mail Name
	Street or P.O. Box No.
_____	City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS	_____
	Organization
	Name
	Street or P.O. Box No.
_____	City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)