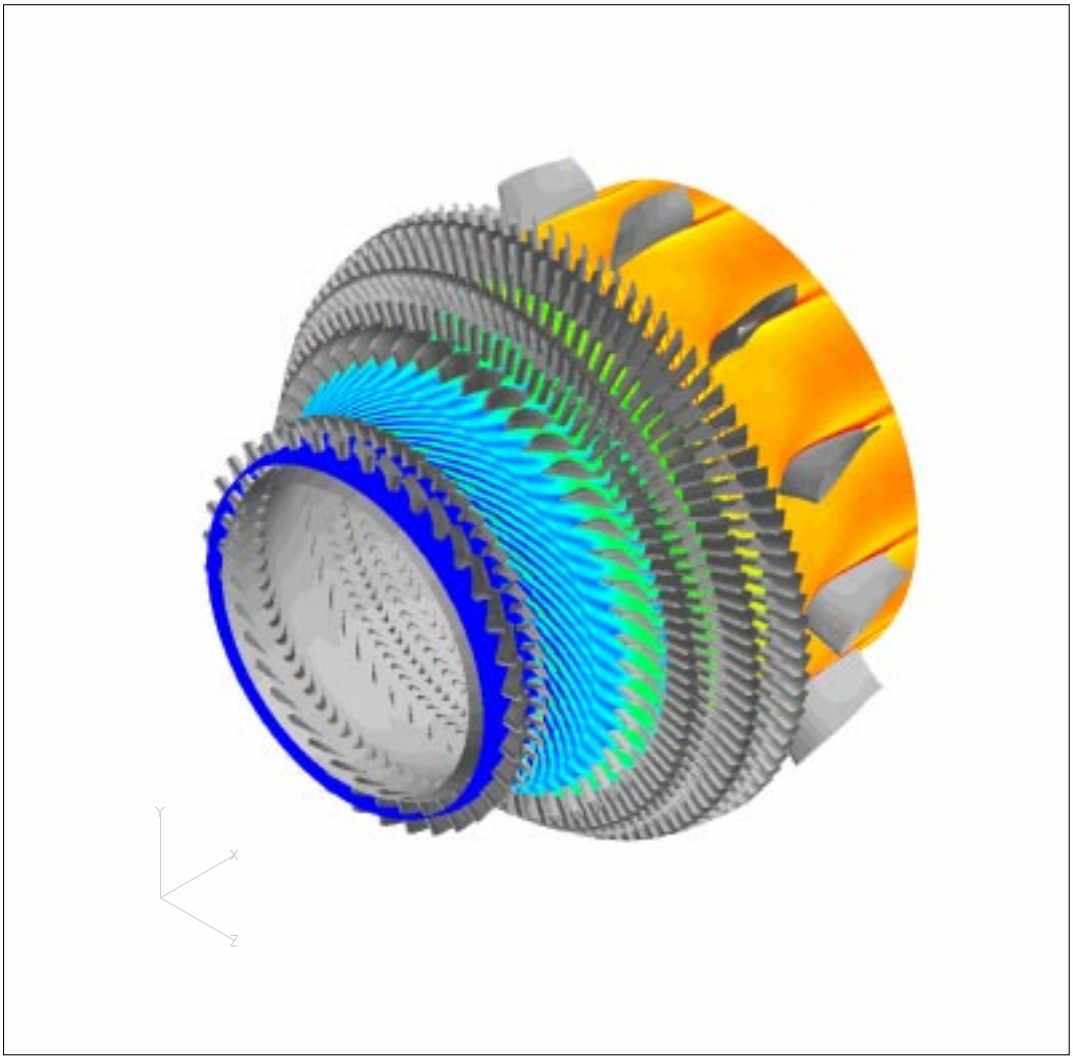
2000 Annual Technical Report



Center for Integrated Turbulence Simulations

Stanford University

Abstract

The Center for Integrated Turbulence Simulations was established at Stanford University in September, 1997, as one of five university centers in the Academic Strategic Alliances Program of the Department of Energy's Accelerated Strategic Computing Initiative. This report outlines the Center's technical work during the third year.

Contents

Abstract **i**

1 Overview **1**
1.1 ASCI and CITS . . . 1
1.2 Engine simulation plan . . . 3
1.3 RANS turbomachinery simulation . . . 5
1.4 LES combustor simulation . . . 6
1.5 Large scale multicomponent integrated simulations . . . 7
1.6 Full system integration at small scale . . . 8
1.7 Computer systems and architectures research . . . 9

2 Turbomachinery Simulations **11**
2.1 TFLO Code Development . . . 11
2.1.1 Conservative Mixing Model for Steady Flow Calculations . . . 11
2.1.2 Fully Conservative Interfacing for Three-Dimensional Turbomachinery Flows 11
2.1.3 Parallel and Scalable FILE I/O . . . 12
2.2 Turbulence Model Implementation in TFLO . . . 12
2.2.1 A Generic Algorithmic Framework for Turbulence Modeling . . . 13
2.2.2 Implementation and Validation of the v^2-f Model . . . 14
2.2.3 Wall Function Implementation . . . 16
2.2.4 Summary and Future Plans . . . 16
2.3 Large-Scale Simulations . . . 20
2.3.1 Pratt & Whitney High-Pressure Turbine Rig . . . 20
2.3.2 Wright-Patterson Stage Matching Investigation (SMI) Rig . . . 30
2.4 Turbine Wakes and Transition Simulation . . . 38
2.4.1 Flow System . . . 38
2.4.2 Simulation Approach . . . 38
2.4.3 Results . . . 39
2.4.4 Simulation Plan . . . 39

3 Combustor **47**
3.1 Large-eddy Simulation of Gas Turbine Combustors . . . 47
3.1.1 Algorithm Improvements . . . 47
3.1.2 Results . . . 49
3.1.3 Summary and Future Plans . . . 56
3.2 Combustion Models for Large-Eddy Simulations . . . 58
3.2.1 Lagrangian Flamelet Model . . . 58
3.2.2 Prediction of a Lifted Flame Using the Flamelet/Progress-Variable Approach 61

3.2.3 Investigation of Scalar Dissipation Rate Fluctuations in Nonpremixed Turbulent Combustion . . . 62
3.2.4 Flamelet Modeling of Local Extinction-Reignition . . . 62
3.2.5 Large-Eddy Simulations of Premixed Turbulent Combustion . . . 63
3.3 Spray Simulation and Modeling . . . 66
3.3.1 Established Milestones . . . 66
3.3.2 Current Status . . . 66
3.3.3 Conditions of Interest . . . 67
3.3.4 Numerical Framework . . . 70
3.3.5 Baseline Simulations . . . 70
3.3.6 Future Tasks Related to LES . . . 73
3.4 Simulations of Deformable Liquid Drops . . . 75
3.4.1 Formulation/Numerical Method . . . 75
3.4.2 Validation . . . 76
3.4.3 Hexane Drops . . . 78
3.4.4 Summary and Future Plans . . . 81
3.5 Chimera Method for Heterogeneous Particle Clusters . . . 82
3.6 SCCM . . . 87
3.6.1 Collision Detection . . . 87
3.6.2 Fast Numerical Techniques . . . 89
3.6.3 Parallel Numerical Methods/Intro. to Parallel Programming . . . 93
3.6.4 DOE/HPC Seminar Series . . . 93

4 Multi-Component Integrated Simulations 101
4.1 Motivation . . . 101
4.2 Approach . . . 102
4.3 Baseline Isolated Component Simulations . . . 102
4.3.1 Pratt & Whitney Combustor . . . 103
4.3.2 Pratt & Whitney High- and Low-Pressure Turbine . . . 104
4.3.3 Scalability of TFLO Code for Parallel Computing . . . 108
4.4 Multiple-Code, Multi-Component Integrated Simulations . . . 109

5 Concurrent VLSI Architecture Group 113
5.1 High-Speed Signaling . . . 113
5.1.1 Approach . . . 113
5.1.2 Recent Accomplishments . . . 114
5.2 Router Microarchitecture . . . 116
5.2.1 Flit-Reservation Flow-Control . . . 116
5.2.2 Pipeline Model of Router Architectures . . . 117
5.3 M-Machine Prototype . . . 118

6 The SUIF Parallelizing Compiler 123
6.1 Introduction . . . 123
6.1.1 Distributive Blocking . . . 123
6.1.2 Array Contraction . . . 124
6.1.3 Blocking and Array Contraction . . . 125
6.1.4 Overview of the work . . . 125
6.2 Array Contraction . . . 126
6.2.1 Finding Independent Partitions . . . 126
6.2.2 Contractable Arrays . . . 127

6.3 Distributive Blocking . . . 131
6.3.1 Interleaving Independent Threads . . . 131
6.3.2 Distributive Blocking for Non-Perfectly Nested Loops . . . 134
6.4 Fully Permutable Loop Nest . . . 134
6.4.1 Making a Sequence of Loops Fully Permutable . . . 135
6.4.2 Making an Imperfect Loop Nest Fully Permutable . . . 135
6.5 The Data Locality Algorithm . . . 136
6.6 Experimental Results . . . 137
6.7 Related Work . . . 139

Chapter 1

Overview

1.1 ASCI and CITS

The Center for Integrated Turbulence Simulations (CITS) is one of five university research efforts established by the Department of Energy (DOE) as part of their Accelerated Strategic Computing Initiative (ASCI) under the Academic Strategic Alliances Program (ASAP). The purpose of the ASCI program is to develop advanced general-purpose technology for very large-scale multi-physics scientific computations. The DOE will use this simulation technology to maintain the safety and reliability of the U.S. nuclear weapons stockpile without resort to further physical testing. Each of the Alliances is working on an important non-weapons application that provides a framework for technical advances in this sort of computing. The powerful supercomputing capabilities in the DOE laboratories are being used by the Alliances for this work.

The CITS is building on Stanford's strengths in flow physics and computation to develop an integrated high-fidelity simulation technology for aircraft gas turbine engines. The major U.S. aircraft engine manufacturers are partners in this effort. They providing realistic geometries, data, experience, evaluations against their own flow simulation codes, and people. They are interested in seeing what this sort of envelope-pushing simulation could do for them. If the potential is significant, their interest would add further market push for the computing systems necessary for such simulations. The success of the CITS effort in engine simulation ultimately will be measured by its impact on the aircraft engine industry, and the involvement of industry with the CITS program is a good indication that there is potential for significant impact.

We believe that the future of large-scale scientific computing lies in the creative use of low-cost commodity computing components in large-scale, highly parallel computing systems. The commercial computer industry today builds server computers of up to 128 processors. While very capable at the data management and engineering simulation for which they are designed, these commercial machines are an order of magnitude slower and smaller in scale than the supercomputers required for ASCI simulations. Research in Stanford's Computer Systems Laboratory (CSL), conducted as part of the Stanford ASCI program, aims to augment this commercial technology with scalable computing technology that will enable the design of very large high-performance computers and the development of efficient programs for these machines.

The research in CSL spans the spectrum from hardware to operating systems and software tools. On the hardware side, a group in CSL is developing interconnect technology that will dramatically increase the global bandwidth of large-scale parallel computers. Operating systems research uses virtual machine technology to apply commercial operating systems to reliable, scalable operation of large-scale machines. Research on compilers, simulation tools, static program checking tools, and visualization enables more rapid development of efficient codes for large-scale machines.

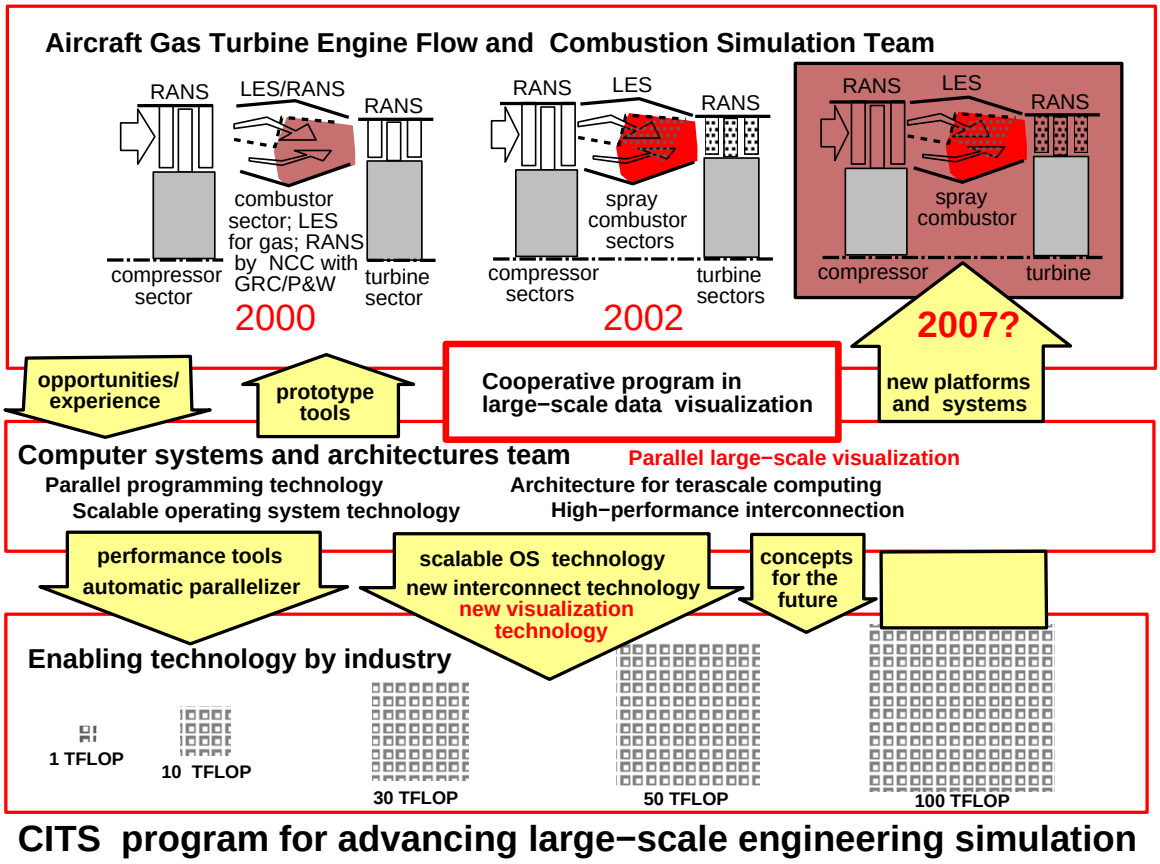


Figure 1.1: Overall CITS roadmap

The coupling between the CSL team and the engine simulation team has increased as the work progressed. Some of the programming tools developed in CSL are now being used on an exploratory basis by the engine simulation team. Needs identified in the engine simulation are beginning to influence research in the CSL. And a promising new cooperative program in visualization of large data bases in taking shape.

However, the primary impact of the CSL research for ASCI will be on the industry that will produce the next generation of scalable computing components, which in due course will be assembled to build the 100 teraflop and faster supercomputers to be used by the DOE and the Alliances. Hence the success of the CSL effort should be measured in part by the speed of transition of the new technologies to industry. The entrepreneurial environment of the Silicon Valley lends itself to this rapid transition, which often involves the faculty who made the research advance.

The roadmap for the overall CITS program is shown in Fig. 1.1. This roadmap suggests that, with the advances enabled by CSL and other computer systems research, it should be possible to do a full integrated engine simulation on hardware expected to become available in the 2007 timeframe. With such capability, the engine designers could use simulation to look at complex off-design situations that are crucial for engine performance and reliability, reducing the time and money that must be spent on experimental development, thereby increasing competitiveness of the U.S engine industry. The importance of this to the national economy cannot be overstated.

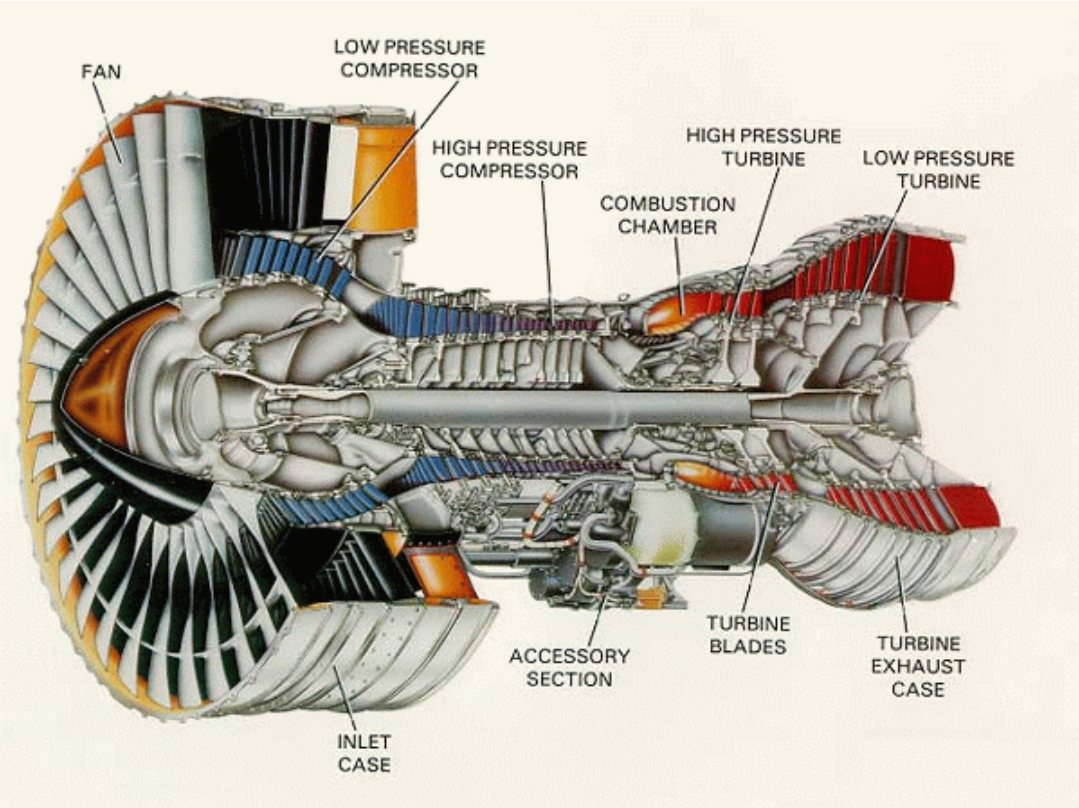


Figure 1.2: Gas turbine engine.

1.2 Engine simulation plan

An aircraft gas turbine engine is a very complex system, an example of which is shown in Fig. 1.2. Flow enters at the left through the fan, is compressed by a series of rotating blades and stationary vanes in the low- and high-pressure compressors, and then is heated by burning fuel in the combustor. The flow exiting the combustor is discharged through the high-pressure turbine, which drives the high-pressure compressor, then passes through the low pressure turbine, which drives the fan and low-pressure compressor, and finally exits through the jet engine nozzle. The goal of the CITS program is to produce high-fidelity integrated simulations of the complete flow path for a modern engine, and thereby to enable study important component interactions.

The turbomachines (compressor and turbines) may have several stages (blade rows) and many more blade passages per blade row, for a total of several hundred blade passages. Current industry simulations typically involve only one blade passage per blade row, and are made using steady-flow codes that assume complete mixing of the discharge of each blade row before it is ingested

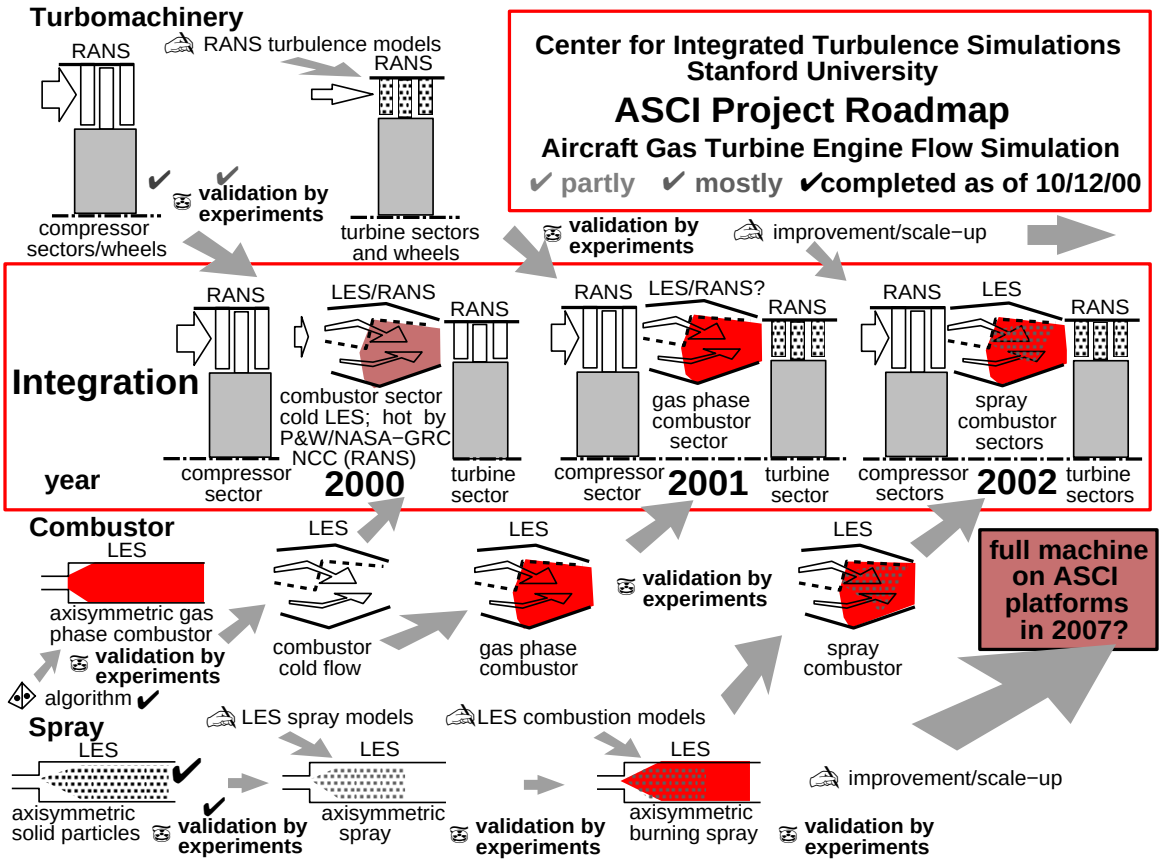


Figure 1.3: Simulation roadmap

into the next. Detailed unsteady calculations show that the shock waves and moving wakes of upstream blades have a pronounced effect on the flow in the subsequent blade passages. Hence unsteady calculations that can handle shocks and wakes are needed to address many important design problems. Moreover, some problems, such as rotating stall in the compressor, can only be addressed by considering every blade passage in the machine. Problems caused by localized hot streaks from the combustor entering the turbine can be addressed only with integrated combustor-turbine simulations. The CITS program provides the first opportunity to carry out calculations of this scope and scale, which is why the industry is so interested.

The roadmap for the engine simulation is shown in Fig. 1.3. The effort is organized in two teams, one focusing on the turbomachinery and the other on the combustor. The approach is to handle the turbomachines using the unsteady compressible Reynolds-Averaged Navier-Stokes (RANS) equations with advanced models for the wall and wake turbulence. Because the combustor flow is very complex, and RANS models of combustion have not been very adequate, the CITS approach is to use Large Eddy Simulation (LES) for the combustor. The combustor is a critical element in the system. The designs are geometrically very complex, with passages for dilution air, recirculating flow, fuel spray insertion and evaporation, and highly turbulent combustion. The Mach numbers in the combustor are typically low, but the density changes due to heat release are very substantial. Hence the combustor LES code is currently based on low Mach number compressible flow equations. The RANS calculations can use various methods (such as upwinding, fourth-order dissipation, *etc.*) to achieve adequate flow predictions. However, LES must use non-dissipative conservative methods in order to capture the turbulence dynamics correctly. Hence research on algorithms and sub-grid modeling have been required to develop LES for the combustor.

The research plan includes work on turbulence modeling for rotating systems, spray modeling and simulation, drop breakup and evaporation, and other flow physics that needs to be improved for a high-fidelity engine simulation. It also involves the development of new numerical methods for flow computation. At Stanford there is a long tradition and record of contributions to numerical analysis from flow physics faculty, who teach the principal courses in numerical analysis, have written books and many papers on the subject, and are journal editors in the field. Hence in the ASCI program at Stanford this fundamental numerical analysis research is carried out by the turbomachinery and combustor simulation teams as an integrated part of their programs. The relatively small group in the Computer Science Department that also does numerical analysis assists in this effort.

1.3 RANS turbomachinery simulation

The CITS turbomachinery code TFLO was developed starting with well-established aircraft flow codes built previously by Prof. Antony Jameson, the leader of the TFLO team. This big head-start enabled TFLO to be developed quickly. It has since undergone an extensive set of verification (against other codes used by industry and NASA) and validation (against data) tests by the extended TFLO team, which includes several industrial participants.

TFLO is a scalable code that is used for large-scale steady and unsteady compressible RANS simulations. Coupled to a combustor code, it can be used to study complex component interaction problems. A goal of the program is to develop numerical simulation techniques that make this type of calculation computationally affordable, and to demonstrate this capability by simulating typical interactions of engineering interest.

TFLO uses a multiblock structured multi-grid mesh with double halos at interfaces. The mesh is body-fitted to each stage, with a mixing-plane interface (steady flow) or sliding-mesh interface (unsteady flow) between stator and rotors. Unsteady simulations use a dual time step algorithm. Jameson-Schmidt-Turkel (JST) or Convective Upwind Split Pressure (CUSP) algorithms are used to control numerical stability. A variety of turbulence models have been implemented, including Durbin's V2F method developed at Stanford. A special preprocessor provides domain decomposition and load balancing in parallel processing systems. The parallel implementation uses a standard Message Passing Interface (MPI), and has been shown to be quite scalable and efficient with over 1000 processors.

Key TFLO developments made during the past year include new conservative sliding mesh interface treatments, the implementation of parallel file input/output (Sect. 2.1.2), and a generic algorithmic framework for dealing with various turbulence models (Sect. 2.2) including V2F. Section 2.3 provides an overview of some of the important validation and verification work involving tens of millions of meshpoints. Additional information on large-scale TFLO performance is provided in the section on large-scale integrated simulations (Sect. 4.3.2 and 4.3.3). Prof. Juan Alonso and Dr. Roger Davis, who is on assignment to the Stanford TFLO team from the United Technologies Research Center (UTRC), and Dr. Jixan Yao, are key contributors to these efforts.

Transition from laminar to turbulent flow is an important phenomena in the low-pressure turbine. This process is very complicated because of the unsteady wakes from upstream blades that are injected into each blade passage (rotor or stator). In order to develop understanding of the physics, and in the hope that this would enable modifications of V2F to account for transition, landmark Direct Numerical Simulations (DNS) for a low-pressure turbine blade passage has been carried out under the direction of Prof. Paul Durbin as part of this program. This work (Sect. 2.4) has revealed new phenomena of lasting scientific value and immediate engineering importance.

Other fundamental studies are ongoing as part of the turbomachinery program. Professors Joel Ferziger and Sanjiva Lele are working with a student on LES of a single turbine blade passage. The goal here is to produce a simulation that can be used to assess and improve turbulence models. Prof. W.C. Reynolds, NASA's Drs. Alan Wray and Karim Shariff, and Dr. Stavros Kassinos are conducting large-scale DNS of homogeneous turbulent shear and straining flows in rotating frames on the ASCI Red machine in order to develop a database for assessing and improving turbulence models. This work will be reported next year.

1.4 LES combustor simulation

Large Eddy Simulation was pioneered at Stanford starting in the mid 1970s. It has been a focal activity of the NASA/Stanford Center for Turbulence Research (CTR) since its inception in the late 1980s. The dynamic sub-grid scale modeling approach, which does not require empirical constants and has become the workhorse of modern LES, was developed largely through work of the CTR. CTR work has also included considerable numerical simulations of turbulent reacting flows. Thus, the Stanford team is well qualified to take on the development of an LES framework for complex combustors such as found in modern aircraft gas turbine engines.

LES requires conservative, non-dissipative algorithms. This presents a challenge for the complex geometries of typical combustors, where unstructured meshes are preferred. Once it had been decided that LES would be necessary for high-fidelity combustor simulation, the combustor group under Prof. Parviz Moin set out to develop a conservative, non-dissipative algorithm for complex unstructured meshes. This algorithm, by Dr. Krishnan Mahesh, was a major contribution from the first year of the CITS program. It is patterned after the staggered approach of Harlow and Welch, which had become the workhorse for LES on Cartesian structured meshes.

The algorithm, which was first developed for incompressible flows, has now been extended to low-Mach number variable density flows and validated against other simulations for a variety of steady and unsteady flows. The dynamic Smagorinsky model has been extended to unstructured grids and implemented in the code. Section 3.1.1 describes recent algorithm improvements. Section 3.1.2 reports some of the validations against laminar and turbulent flows, and outlines our current work in setting the code up to run cold flow LES in a modern engine combustor. The code is written in Fortran 90 and designed for effective parallel implementation. The work has progressed to the point where the algorithm is beginning to be tested on large numbers of processors.

In parallel with the combustor flow code development, we are developing new methods for handling the chemistry in LES. The first is an unsteady Lagrangian flamelet approach of Dr. Heinz Pitsch. A somewhat different flamlet-progress variable approach by Mr. Charles Pierce and Prof. Moin also shows promise for use in LES. These are described in Sect. 3.2.

Another parallel effort is developing technology for high-fidelity spray simulation. This must describe the spray injection, sheet and droplet breakup, and evaporation that precede the gas-phase combustion process. Dr. Joseph Oefelein, a graduate of Penn State University with considerable practical aircraft engine experience, came to us after his PhD on LES for spray simulation. He has used both the Pierce/Moin gas phase combustor LES code and his own code developed at Penn State to study various aspects of spray modeling in simple axisymmetric and planar geometries. Both codes use structured grids and staggered data. The Pierce/Moin code is based on the low Mach number equations and uses a Cartesian axisymmetric mesh, whereas Oefelein's code can handle fully compressible flow in generalized curvilinear coordinates. The first work, reported last year, showed that standard drag models for solid particles produce excellent replications of the particle distributions (compared to data) when tracked individually in a LES of a gas that is fairly

lightly loaded with particles. Current work using his own code is extending these evaluations to more highly loaded channel flow, for which there is experimental data from Stanford, and to a more complex axisymmetric swirling combustor geometry in which spray and combustion experiments are being conducted now at Stanford. This work is described in Sect. 3.3.

As part of the development of improved spray simulation technology, we have two efforts in DNS related to droplets and droplet interactions. The first (Sect. 3.4), being conducted by Dr. Brian Helenbrook, studies flow in and around deformable axisymmetric drops using a multigrid spectral element method. The second (Sect. 3.5), conducted by PhD student Tristan Burton, investigates the interaction of droplets in clusters using Chimera grids. The expectation is that these studies will lead to improvements in the modeling of spray droplet behavior, and that this will improve the fidelity of the LES spray combustor simulations.

In addition to these developments in numerical analysis being carried out by the flow physics group, the Scientific Computing and Computational Mathematics (SCCM) group, led by Prof. Eugene Golub, has been working with Prof. Joel Ferziger to assist the spray modeling effort. SCCM Prof. Andrew Stuart initiated work with a student on particle collision detection before leaving Stanford to return to the U.K. This work has continued in collaboration with Prof. Ferziger and SCCM's Dr. Wing-Loc Wan. Dr. Wan has also worked on fast multigrid methods for flows with discrete interfaces, and on other problems of interest to our ASCI program. This SCCM work is summarized in Sect. 3.6.

All of the work outlined above is intended to give us the capability for high-fidelity simulation of the very complex physics of turbulent combustion in the very complex geometries of modern aircraft engine combustors. The contributions from this work should have lasting impact on both the science and simulation of turbulent spray combustion.

Several Stanford experimentalists are involved to be sure that we use the most current and best experimental data to guide and validate the combustor model. Prof. John Eaton continues his experiments on particle-laden turbulent flow under other support in parallel with his guidance of Mr. Burton's ASCI droplet/turbulence interaction research. Prof. Chris Edwards, a spray expert, is conducting spray injection and spray combustion experiments designed to complement the ASCI simulation under NASA suppor, and is guiding the ASCI work of Dr. Helenbrook. Profs. Tom Bowman, a combustion and kinetics expert, and Prof. Godfrey Mungal, a turbulent mixing expert, consult with the combustion team. Other combustion experts (*e.g.* Norbert Peters, George Kosaly) visiting the CTR also consult with the team.

1.5 Large scale multicomponent integrated simulations

The ultimate objective of our program is the integrated coupled simulation of the compressor, combustor, turbine, and secondary air flows in a realistic aircraft gas turbine engine. The computing power to do this simulation does not presently exist, but it should by the end of this decade. Because TFLO had a head start and is now operational, but the LES combustor code is still under development, we entered into a partnership with the NASA/Glenn Research Center to use their National Combustor Code (NCC). The goal is to couple the NCC to TFLO to study various integration issues. Prof. Juan Alonso is leading this effort in collaboration with Dr. Roger Davis, our partner from UTRC in residence at Stanford.

The NCC is a RANS code, so its fidelity in combustion simulation is not expected to be great, but it has enough in common with our LES combustor code to be useful in gaining some early integration experience. It uses an unstructured mesh that can deal with complex geometries, and

can run in an unsteady-RANS mode similar to LES. The NCC is export-controlled, hence because we have foreign nationals involved (faculty, RAs, and students) we can not obtain the source code. We have not provided TFLO source code to NASA/Glenn, so that the two parties must find a way to couple their codes without one knowing the details of the other. Each party has written its side of the interface, which exchanges information between the NCC and TFLO. Stanford has tested the interface by coupling two TFLO calculations through the interface. NASA/Glenn is making the same sort of tests of its side of the interface. TFLO and the NCC have been coupled, but some technical problems need to be solved and so it is too soon to report any details.

At Stanford, Dr. Roger Davis is learning to run the NCC without access to its source. As a key member of the TFLO team, he is also running very large scale turbine calculations involving some of the important secondary air flows using TFLO. These very large scale simulations have been done in dedicated time on the DOE Blue Pacific computer at LLNL, using over 1000 processors. These have the potential to become landmark simulations for the engine industry, and if successful they should generate considerable enthusiasm for this sort of large-scale simulation. This work is reported in Sect. 4.

The attempts to begin integrating the NCC with TFLO have been seriously hampered by the lack of time that knowledgeable NASA/Glenn personnel have been able to devote to this collaborative effort. We are hopeful that this will improve in the coming fall. NCC-TFLO coupling would provide a valuable RANS-RANS baseline with which future LES-RANS coupling could be compared. However, if the NCC-TFLO effort proves to be unsuccessful or slow in maturation, the ASCI effort will be redirected to accelerate development and integration of the LES combustor code.

Coupled combustor-turbine simulations will provide new insight into the origins of "hot streaks" observed in the first turbine stages. These probably arise as a result of non-uniformities in the combustor, and are a critical factor in determining turbine blade lifetime. The results could make an important near-term impact on engine design, and ultimately such calculations could become an important step in the engine design process.

1.6 Full system integration at small scale

The DOE platforms will not be sufficient for full-scale simulation of large aircraft gas turbine engines until the 2007 timeframe. The best we are able to do now is a sector of an engine. Therefore, we are considering starting a parallel project on a much smaller gas turbine engine manufactured for educational purposes by Turbine Technologies, Ltd. Their little SR-30 engine (Fig. 1.4) is only about 7 inches in diameter. It has a single-stage radial compressor, an annular combustor with six injectors, and a single-stage axial flow turbine. We have just obtained two of these engines at Stanford. One is sectioned for instructional viewing, and the other is fully instrumented and running in an undergraduate teaching laboratory. The objective would be to make fully-integrated simulations at this scale to predict the overall engine performance, which could then be compared with experiments as part of the laboratory course.

There would be several advantages to using this engine for a first integration step. It is of a scale that it could be handled completely with high fidelity using TFLO and the LES combustor code on current ASCI platforms, and possibly even on our in-house clusters. We would encounter most if not all of the RANS/LES/RANS coupling problems needed to be solved to do large-scale full-engine simulations. We would not have the problems of data display that we have encountered when using proprietary commercial geometries from industry. And if this simulation can be done on our in-house cluster, the use of this simulation tool could become an integral part of our teaching

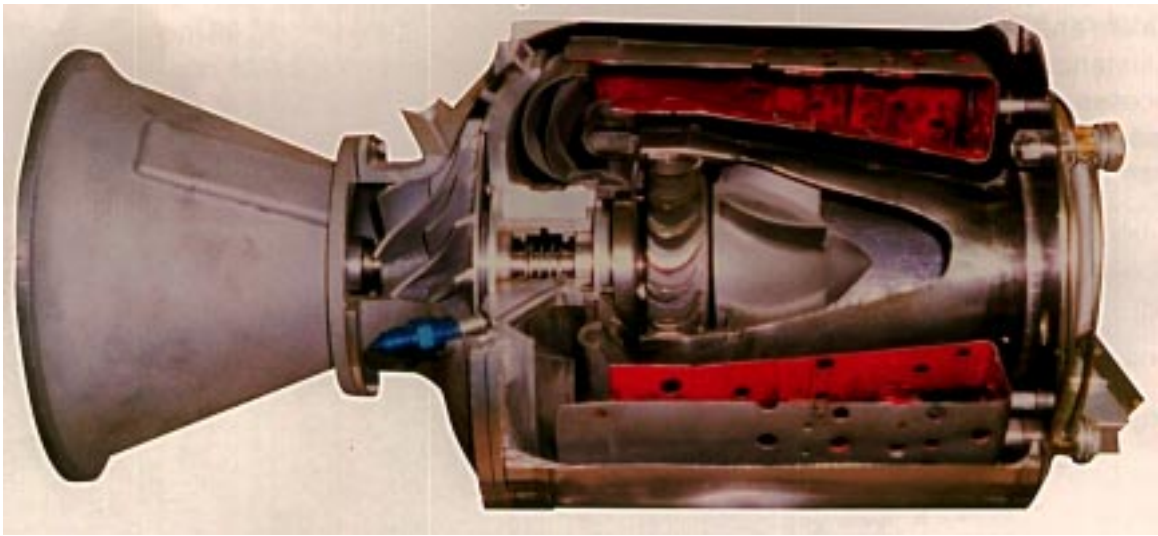


Figure 1.4: Turbine Technologies SR-30 Mini-Lab Engine. Reproduced by permission of Turbine Technologies, Ltd.

program in Mechanical Engineering, thereby stimulating student interest in simulation, which is another objective of the ASCI Alliance program.

We have discusses this new idea with Turbine Technologies, which expressed interest in collaborating with us on this project by providing the geometrical information that we need to construct the flow grid. Further discussions among our simulation group and with the DOE are needed before a final decision on this plan is made.

1.7 Computer systems and architectures research

Faculty and students in the Computer Systems Laboratory (CSL) have been an integral part of the ASCI program since its inception. The CSL vision that made its participation in ASCI attractive to the DOE has not changed, although the rapid elevation of the team leader, then Dean John Hennessy, to Provost and then to President of the University, has had some impact. His vision was recently reaffirmed by the current CSL faculty leaders, who see the future of large-scale scientific computation as involving many thousands of interconnected processors assembled from commodity components, sharing their memory at high bandwidth, and operating in a fully-integrated manner. The CSL ASCI work has been, and continues to be, in support of this vision.

Prof. Hennessy's work in ASCI has included Flash, a special machine with a programmable node controller (MAGIC) that handles cache coherence and all other node traffic. This machine provides a very flexible platform for study of different cache control strategies, for simulation of alternative machine architectures, and for instrumentation of application codes. FlashPoint is a software tool that uses Flash to assess performance of Distributed Shared Memory (DSM) codes. The engine simulation group has begun to use it to study performance of its OpenMP codes running on SGI machines. Flash was described previously and an additional report is not included this year.

The current faculty lead for the CSL ASCI program is Prof. William Dally. His work on interconnection networks and high-speed signaling is described in Sect. 5. High bandwidth interconnections will simplify the programming of large parallel computers, and Dally has come up with new router architecture and signaling technology that offers a large potential increase in bandwidth. He is

already taking this technology to industry, a first step in getting this into commodity components for supercomputing.

Simulation of very large supercomputers is an important aspect of scalable operating system design. This requires big computers to simulate much bigger computers. Prof. Mendel Rosenblum's group completed the development of this technology during the first year of the ASCI program. His work on SimOS, a large-scale operating system simulation tool, was reported previously. He has been on leave for the past two years bringing this technology to market. The work has been continued by his students and faculty colleagues, and is not reported on this year.

A concept for virtual clusters has been explored by Prof. Mendel Rosenblum and his team. The nodes of a large shared-memory machine are partitioned into groups that use virtual machine technology with each running a separate operating system image. The groups are then clustered using the memory system of the underlying system for communication. This approach offers scalable performance, dynamic resource partitioning, fault containment, and low implementation cost. Hence it addresses several needs of large-scale shared memory multiprocessors. This work is ongoing and hence will be reported next year.

Prof. Dawson Engler and his students are developing tools for automatic error checking. They have developed a *meta compiler* that has detected hundreds of errors in contemporary operating systems including Linux. The system works by statically checking a number of simple rules, *e.g.* "every lock that is acquired must be released", against software. While these tools have been primarily applied to system software, they hold the potential to check for synchronization and consistency errors in parallel applications. This is new work to be reported next year.

Prof. Monica Lam's work on automatic parallel compilers is building a firm mathematical foundation for automatic parallel compilation. She has demonstrated her Stanford University Intermediate Format (SUIF) compiler on a number of ASCI codes, including some provided by the engine simulation team. Section 6 gives a detailed report on her work.

As a result of a suggestion provided to Prof. Reynolds at the annual DOE Conference on High Speed Computing this February, we initiated a collaboration between the engine simulation team and visualization research of Profs. Pat Hanrahan and Ron Fedkiw. Prof. Hanrahan has been developing technology for parallel visualization on clusters under support from the ASCI Views program. His team has extended the Visualization Tool Kit to parallel clusters, and he will make his cluster system available for use by the simulation team. Dr. Massimiliano Fatica of the engine simulation group is now writing parallel VTK scripts to be run on Hanrahan's cluster, and the CTR/CITS is upgrading its cluster for compatibility with Hanrahan's so that these scripts can be tested at CITS. The visualization researchers and engine simulators have defined some specific objectives for the collaborative effort and identified research needed to advance visualization capabilities. We hope to grow this cooperative visualization effort over the coming year, and look forward to including a report on this in next year's Annual Technical Report.

Chapter 2

Turbomachinery Simulations

The focus of this year's effort in the turbomachinery simulation group has been on three main areas that are described below. The majority of the effort has been directed towards the validation of large-scale computations (both steady and unsteady) using an increasing number of processors. In addition to this main effort, our group has implemented a number of new turbulence models whose efficiency and accuracy will be tested during the coming year. Finally, several modifications have been made to the baseline TFLO code including parallel I/O, wall-functions, and a conservative sliding interface model, so that our calculations can be performed more efficiently and accurately. The following sections focus on each of these efforts.

2.1 TFLO Code Development

2.1.1 Conservative Mixing Model for Steady Flow Calculations

A non-conservative mixing model and sliding interface were originally implemented in TFLO for both steady and unsteady flows in multi-stage turbomachinery. From the results of preliminary large-scale calculations with increasing number of blade rows and strong interaction between adjacent blade rows, it was found that a significant loss of mass flow (around 1%) across the interface between two blade rows could exist. This realization motivated a series of updates to the mixing model and sliding interface implementations such that mass, momentum, and energy fluxes were conserved both locally and globally. The data structures in TFLO provide the necessary flexibility for these revisions, since the future need for a conservative interface was contemplated when the initial non-conservative models were implemented.

Multistage turbomachinery calculations in TFLO are performed with meshes that are attached to both rotors and stators. Since there rotors move relative to the stators, the meshes slide past each other at the interface between adjacent blade rows. In all TFLO meshes, we require that the mesh lines in the circumferential direction have matching radii, while the circumferential distribution of grid points may vary widely. In the following, conservation is defined independently for mesh strips at constant radii.

In the current implementation, the transfer of information across the interface is globally conservative. Additional work is being carried out so that the interfaces can also be considered locally conservative.

2.1.2 Fully Conservative Interfacing for Three-Dimensional Turbomachinery Flows

We are adopting a more sophisticated approach proposed by Philip M. Cali (AFRL) and Vincent Couailier (ONERA) for the conservation of fluxes across the blade row interface. This approach

defines a unique patch boundary on which fluxes are evaluated directly. In order to maintain accuracy across the interface, a limited least-squares polynomial reconstruction is conducted using information from both sides of the boundary. The implementation of this approach is in progress and will be reported at a later time.

2.1.3 Parallel and Scalable FILE I/O

Our experience with large-scale calculations on the ASCI platforms has pointed out that the serial I/O procedures initially implemented in TFLO had become a significant bottleneck when dumping solution and restart files. Clearly, the use of parallel I/O techniques is the proper way to tackle this problem.

During the last decade, the computational and communication performance of large-scale parallel computers has increased dramatically. Regrettably, parallel I/O has lagged behind with the consequence that implementing parallel I/O procedures is a tedious task which can involve programmer's input about some or all of the following factors:

- Multiprocessors with parallel access to multiple disks .
- Hardware architecture.
- Operating system support and limitations.
- Parallel databases.
- Networking.
- Algorithms.
- Software and language support, programmer interfaces (ex: MPI-I/O).

MPI-2 has outlined the standard for its parallel I/O interface, but the existing implementations are not fully compatible with those specifications. TFLO follows the MPI-2 standard for its parallel I/O implementation. On the IBM SP platform, using MPI-IO with IBM's GPFS file system, TFLO achieves a significant I/O performance improvement. The parallel I/O wall clock time can be reduced to less than 25% of that required by serial I/O for calculations using more than 512 processors. A similar improvement has been found on the SGI Origin2000.

Although TFLO's parallel I/O subsystem is currently based on the MPI-2 standard, we are in the process of considering other procedures such as HDF together with the industry-emerging standard CGNS for CFD computations. These development tasks will be tackled in the early part of this coming year.

2.2 Turbulence Model Implementation in TFLO

Most industrial RANS solvers offer a variety of different turbulence models that may be selected at run-time to suit the physical complexity and the cost/time constraints of a given simulation. TFLO follows the same philosophy with a number of turbulence models now in place or undergoing development.

When a new turbulence model is to be implemented, aside from considering whether a candidate model is suited to the type of flows of interest, a primary issue is selection of an appropriate numerical method. Indeed algorithmic concerns are of comparable importance to the physical concerns as ultimately the numerics will influence the efficiency of the flow solver from both algorithmic (convergence) and computational (CPU time) standpoints. Furthermore, the chosen algorithm will

determine the overall robustness/reliability of the solver and hence its usability. In the case of massively parallel computation, the numerical approach adopted also raises the question of algorithmic scalability.

2.2.1 A Generic Algorithmic Framework for Turbulence Modeling

In earlier work on a one-equation turbulence model, it was discovered that when the entire system of RANS equations (mean flow and turbulence) were updated in a fully-coupled manner, an issue of robustness arose, with convergence not always attainable. This finding indicated the need to develop a more robust numerical implementation for the turbulence model in question. The model implementation that subsequently was chosen solves the turbulence equation implicitly using an Alternation Direction Implicit (ADI) scheme based on an approximate factorization. In this context, the mean flow solver is retained as an explicit Jameson scheme with multigrid used to accelerate convergence.

The segregated nature of the new algorithm was found to reduce a de-stabilizing, nonlinear coupling between the mean flow equations and the turbulence equation, while the implicit scheme could better handle the stiffness arising from the turbulence model. With this approach it has been possible to achieve convergence to machine zero, even in the case of realistic 3-D transonic test cases involving shock/boundary-layer interaction.

The same algorithmic approach was followed in order to implement a two-equation turbulence model –the Wilcox k-ω model– which had previously been implemented in TFLO using a fully-coupled explicit scheme. With the resultant implicit k-ω implementation, convergence to machine zero was also achieved, For the implementation of turbulence models within TFLO, a generic algorithmic framework has been introduced that may be used in subsequent modeling development. The benefits of this generic n-equation framework are:

1. It is based on a numerical approach of proven algorithmic efficiency;
2. Coding is modularized such that different turbulence models can re-use a significant amount of source code;
3. Future models can be introduced in the same framework with reduced effort, following existing source 'templates';
4. Any algorithmic improvements can be extended to all models with reduced risk;
5. The approach promotes good software development practice which is essential for software of growing complexity.

A second two-equation turbulence model, the Shear-Stress Transport (SST) model due to Menter, has subsequently been implemented within this generic framework and is currently being validated, while the four-equation v^2-f model of Durbin, described in greater detail below, is similarly being developed.

Within the new generic n-equation framework, the auxiliary equation(s) for the turbulence models are solved decoupled from the mean flow, using an ADI scheme [1] which is adapted for three-dimensional domains via an approximately-factored 'delta' form. In delta form, the solution is recovered from the changes in the turbulence variable rather than from the variable itself. Using an effective strategy for treating the source terms, the production components are treated explicity (viz. they are lagged in time) while the destruction, convection, and diffusion terms alone are linearized for the implicit operator. The rationale is that linearization of the negative source components enhances the diagonal dominance of the implicit coefficient matrix, by increasing the magnitude of the central coefficient. In the later stages of convergence, linearization of the positive (production) terms can improve convergence but this approach is avoided as earlier on in the iterative process, the diagonal dominance of the matrix can be affected adversely. However, it should be noted that the chosen linearization strategy can readily be relaxed as necessary to suit the idiosyncracies of a

given turbulence model, without compromising the generic algorithmic framework. The turbulence field is solved decoupled from the mean flow after each complete multi-stage time update on the finest mesh in the multigrid cycle. On coarser meshes, the turbulence variable is effectively frozen while the turbulent viscosity is restricted to coarser meshes in the cycle via standard operators.

2.2.2 Implementation and Validation of the v^2-f Model

The v^2-f turbulence model is an advanced eddy-viscosity model. It can be regarded as an abbreviated Reynolds stress model consisting of three transport equations: for the turbulent kinetic energy k, the dissipation of the turbulent kinetic energy ϵ and for a scalar quantity $\overline{v^2}$. The eddy-viscosity is essentially given by

$$\mu_t = C_\mu \rho \overline{v^2} k/\varepsilon.$$

Close to solid walls, $\overline{v^2}$ represents the energy of the fluctuations normal to the wall. The production term in the $\overline{v^2}$ transport equation is a pressure-strain term. Its nonlocal dependency on the flow, in particular in the present of solid walls, is modeled with an elliptic relaxation equation for a quantity f. The equations of the standard v^2-f model can be found in [3]. The wall boundary conditions, which can also be found in that paper, are a critical issue in the implementation into TFLO. The v^2-f model is solved in TFLO in a separate set of subroutines, segregated from the mean flow. Multigrid is used for the mean flow and at each multigrid cycle model's subroutines are called on the finest grid. They return an updated value for the eddy viscosity and the turbulent kinetic energy. Only these two quantities are passed to the mean flow solver for the determination of the Reynolds stresses.

The model equations are solved in an implicit, pairwise coupled manner, with a cell centered finite difference scheme in delta form. The coupled solution of the k and ϵ equations and of the $\overline{v^2}$ and f equations allow an implicit treatment of the wall boundary condition. The diffusion terms are discretized with second order central differences. First order upwind differences are used for the discretization of the convective terms. At the current stage, the model is implemented to provide steady state solution by marching in time from an initial guess. The time derivative term is discretized with a first order forward difference. Local timesteps are used to accelerate convergence. These timesteps are determined by the spectral radius analysis from mean flow quantities, the eddy viscosity and the input CFL number. Multigrid is not used for the turbulence equations. The timesteps for the advancement of the turbulence variables have been weighted with a constant factor. The algebraic system of equations resulting from the implicit operator is solved with a three factored, approximate factorization scheme. Both the discretization of the equations and the solution algorithm are described in detail in [7].

Next, the development of a new factorization scheme is described. It makes a significant improvement in the robustness of the v^2-f implementation, and may generally improve the solution of RANS turbulence models. A particular difficulty with the v^2-f model arises from the f wall boundary condition. The value of f_w at the wall depends, on the value of $\overline{v^2}$ in the first cell above the wall. A small change of $\overline{v_1^2}$, in the first cell above the wall, changes the value f_w at the wall, which in turn, affects in an elliptic manner the f distribution across the entire boundary layer. An accurate solution of the models equations close to solid walls is therefore required.

The model equations can be written in matrix form notation as:

$$(I + S + L_\eta + L_\xi + L_\zeta)\Delta\phi = RHS \tag{2.1}$$

where L_γ, with $\gamma = \xi, \eta, \zeta$, contains the convection and diffusion terms in a grid oriented (ξ, η, ζ) coordinate system. S and $\Delta\phi$ contain the implicit source terms and the solution update, respectively. I is the identity matrix. RHS is the right hand side.

A computer-memory and CPU-time efficient algorithm for the solution of equation (2.1) is critical in large three-dimensional computations. Ordinary ADI algorithm developed for structured grids often converge better if certain directions of the computational coordinate system are normal to existing walls. Only one specific direction may be normal to a solid wall for turbulence models with stiff wall boundary conditions, as it is the case of the v^2-f model. Even in a multi-block concept, this directionality of the factorization algorithm may lead to severe constrains for the grid generation process for complex geometries, especially in wall corners of cavities, wing-body junctions of an aircraft or hub-blade junctions in turbomachines. An ADI algorithm requires an approximation for equation (2.1) which allows a split into a set of one-dimensional equations. As proposed in [6] the approximation of equation (2.1) with

$$(I + S + L_\eta)\,(I + S)^{-1}\,(I + S + L_\xi)\,(I + S)^{-1}\,(I + S + L_\zeta)\,\Delta\phi = RHS$$

allows a split into the three one-dimensional equations,

$$(I + S + L_\eta)\,\Delta\phi' = P \tag{2.2}$$

along η grid lines,

$$(I + S + L_\xi)\,\Delta\phi'' = (I + S)\Delta\phi' \tag{2.3}$$

along ξ grid lines, and

$$(I + S + L_\zeta)\,\Delta\phi''' = (I + S)\Delta\phi'' \tag{2.4}$$

along ζ grid lines. In contrast to earlier formulations [5], the source terms are treated implicitly in each computational direction. This leads to a consistent implicit operator on the left-hand side in each computational direction, and allows different computational directions to be normal to solid walls.

Flow over Flat Plate

The current implementation of the v^2-f model in TFLO has been tested for subsonic flow over a flat plate. The flow is computed for a Reynolds number of $6 \cdot 10^6$ and a Mach number of 0.2. Using a mesh of 64×96 cells, an average of 40 grid points lie within the boundary layer.

The TFLO implementation has been compared with an extensively validated CFL3D implementation. A computation of a flat plate boundary layer is shown in Fig. 2.1. The log law and profiles of the normalized turbulent quantities are shown for $x/c = 0.9$. The discrepancy in the results, which is noticable for the ϵ and f distribution in Fig.2.1b and 2.1d, can be traced back to the numerical dissipation of the mean flow solver. Although the turbulence transport equations are solved similarly in both codes, the mean flow is solved in TFLO with a Jameson-Schmidt-Turkel [4] dissipation scheme and in CFL3D with Roe upwind flux difference splitting [10]. The discrepancy is reduced by adjusting in TFLO the parameter for the fourth-order background dissipation (VIS4) from the default value of 1 to 0.3.

The convergence history of the L_2 norm of the density, turbulent kinetic energy and $\overline{v^2}$ for the TFLO computation with VIS4=1.0 is shown in Fig.2.1f. A five decade drop in the magnitude of the residual requires about 2000 iterations. This is usually required for this type of computation irrespective of the turbulence model.

Flow over Bachalo-Johnson Bump

Transonic flow over the Bachalo-Johnson bump [2] has been chosen to study the performance of the TFLO implementation of v^2-f in predicting shock-boundary layer interaction. This test case is an axisymmetric configuration. The inflow Mach number is 0.875 and the Reynolds number is $Re_c = 2.66 \cdot 10^6$. The axisymmetric bump model together with contour lines of pressure in an axial plane are shown in Fig.2.2. The computational mesh consists of 180×100 cells. Cells are clustered in the region of the shock.

The pressure distribution on the wall of the cylinder and bump computed with TFLO and CFL3D is shown in Fig.2.3. The x-coordinate is chosen such that the bump is located between 0 and 1. The shock location is closely related to the amount of separation downstream of the shock. TFLO, using the Jameson-Schmidt-Turkel scheme, predicts the shock location slightly upstream of that obtained with CFL3D. This results in a lower pressure at $x/c = 1$ indicating a larger separation zone.

2.2.3 Wall Function Implementation

In order to facilitate the use of TFLO by some of our industry partners, and for compatibility with their internal design/analysis processes, a wall-function capability is currently being added to TFLO. The main purpose of this wall function implementation is to reduce the computational cost by reducing number of grid points in the viscous layer. The wall function implementation provides boundary conditions for components of the the momentum equation, the $k - \omega$ model, and the energy equation. A baseline implementation of the wall-function approach was provided by GE Aircraft Engines. The TFLO implementation is partially completed as of August 2000.

2.2.4 Summary and Future Plans

The v^2-f model, Spalart-Allmaras, $k-\omega$ and Shear-Stress Transport model have been implemented in TFLO. The $v^2 - f$ model implementation has been successfully tested for two two-dimensional test cases, a subsonic flow over a flat plate and transonic flow over a axisymmetric bump. In both cases, the code performs in terms of accuracy and computational costs similarly to v^2-f implementations in other codes.

A new ADI scheme has been developed for the solution of RANS turbulence models. This scheme is less dependent on the order of sweeping directions. In case of the v^2-f model, it allows, in contrast to earlier ADI schemes, any computational direction to be normal to solid walls.

Future work includes the testing of TFLO / v^2-f for steady three dimensional flows. This may require the implementation of a two-factored scheme [7] or the adaptation of methods for reducing Approximate-Factorization errors [13].

The effect of numerical dissipation remains to be investigated. This will be facilitated by using the CUSP scheme [8] in TFLO. This scheme is reported to have a similar performance to the Roe scheme which is used in CFL3D.

A higher-order time integration scheme is required for unsteady computations. It is planned to implement a dual time stepping scheme in which a pseudo timestep is used for the inner iteration to a pseudo steady state.

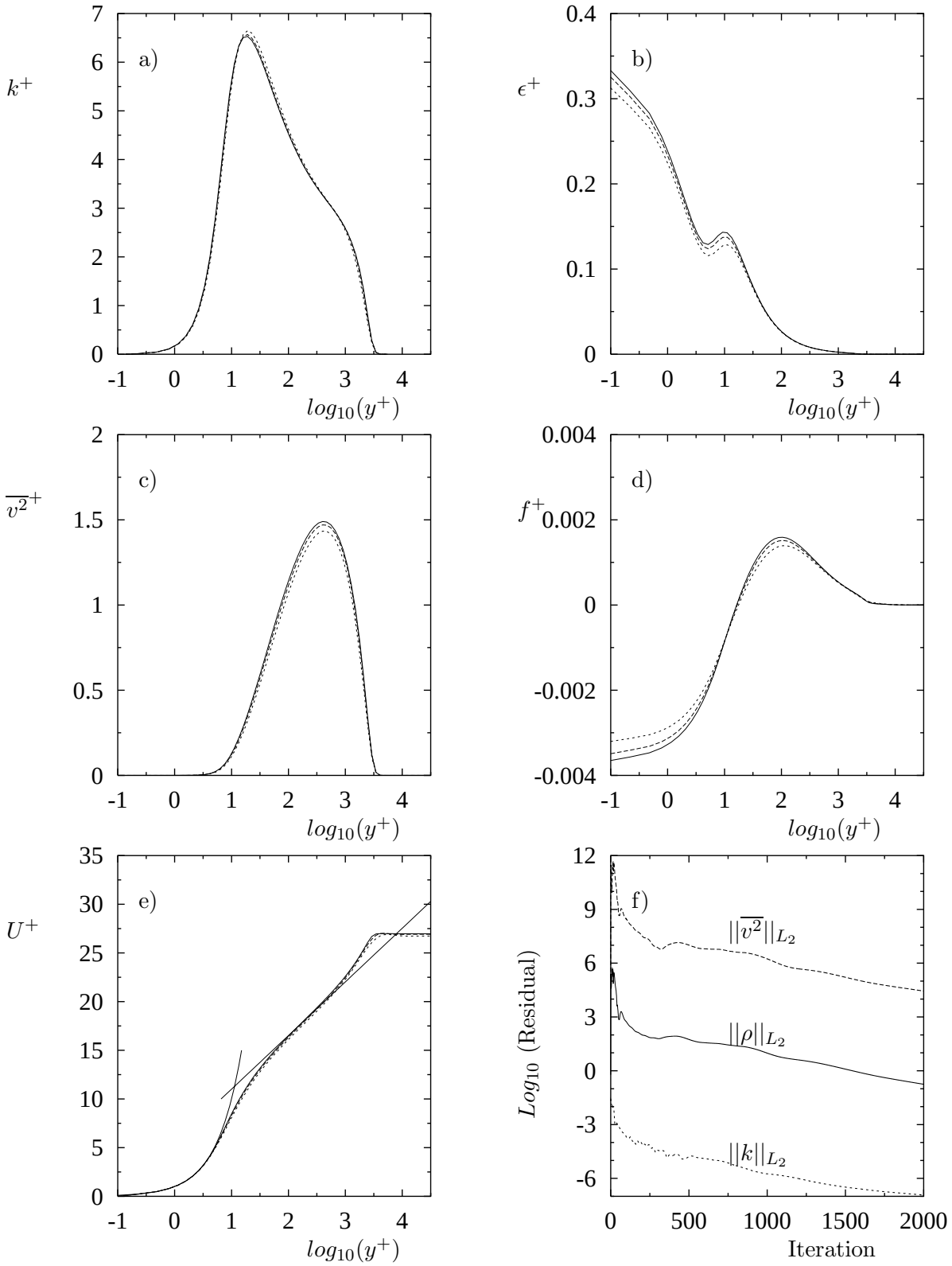


Figure 2.1: Flow over flat plate; v^2-f model; profiles of a) k, b) ϵ, c) $\overline{v^2}$, d) f, e) U^+ at $x/c = 0.9$ (——— : CFL3D; – – – – : TFLO, VIS4=0.3; ····· : TFLO, VIS4=1.0) and f) convergence history for computation with TFLO, VIS4=1.0

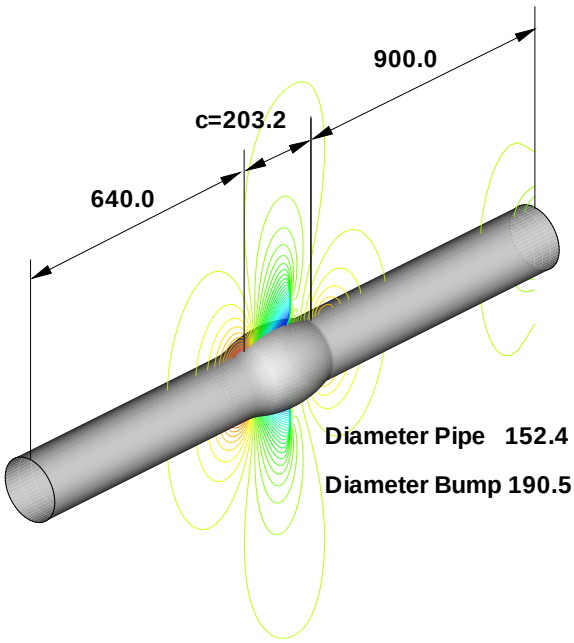


Figure 2.2: Bachalo-Johnson bump: geometry sketch.

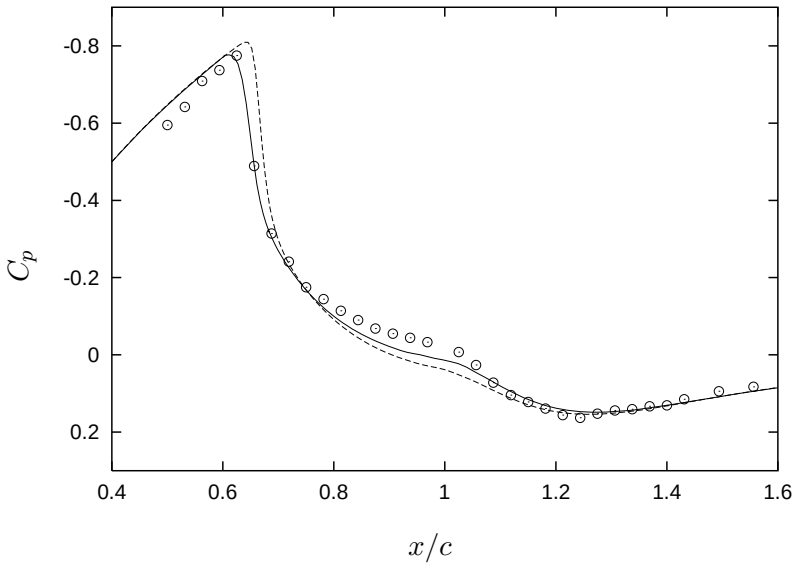


Figure 2.3: C_p-distribution for the Bachalo-Johnson bump using the v^2-f model: —— : TFLO; - - - - : CFL3D, ∘ : expt.

2.3 Large-Scale Simulations

The process of developing the capability to perform integrated, multi-component simulations of major or whole sections of a gas-turbine engine includes intermediate steps. These steps consist of validation, verification, and demonstration of the ability to compute the flow through major sections or entire components of the engine, such as the complete high- or low-pressure turbine or combustor. The validation and verification process consists of comparing details of the predicted steady- and/or unsteady-flows with both experimental data and results from similar numerical prediction techniques. These intermediate steps are tedious and time-consuming but are critical to establish the prediction accuracy and speed that is required to develop a useful engineering tool and a procedure that can ultimately step up to the larger, multi-component problems of interest.

Validation and verification of the TFLO code for turbomachinery configurations continued during the past year. The simulations performed during this phase of the development cycle have focused on off-design, unsteady-flow, and on entire components of the gas-turbine engine. For the TFLO code, a number of steady- and unsteady-flow turbomachinery simulations of both openly available and Pratt & Whitney proprietary configurations have been performed and compared with experimental data and other numerical solutions. These include:

ERCOFTAC Aachen $1 - 1/2$ Stage Turbine Rig
Pratt & Whitney High-Pressure Turbine Rig
Wright-Patterson $1 - 1/2$ Stage Compressor (SMI) Rig

The Aachen $1 - 1/2$ stage turbine has been computed with TFLO as well as with a number of similar codes from government and industry. The results of the steady-flow simulations were reported in last year's annual report as well as in Ref. [18]. A comprehensive set of results from both the Wright-Patterson $1-1/2$ stage compressor and the Pratt & Whitney high-pressure turbine simulations will be shown below to further illustrate the validation and verification efforts being performed for the TFLO turbomachinery code.

2.3.1 Pratt & Whitney High-Pressure Turbine Rig

The Pratt & Whitney $1 - 1/2$ stage transonic high-pressure turbine configuration [19] has been recently tested at the Ohio State University short-duration turbine-test facility. The experimental data collected are being used to validate and verify the TFLO code for transonic turbomachinery. This case is particularly difficult to predict due to highly loaded airfoils and shock/airfoil interaction. The turbine configuration is shown in Fig. 2.4.

Steady Flow

In addition to validating the TFLO code with steady and unsteady experimental pressure measurements, Pratt & Whitney's 3DFLOW code [23, 24, 25] was also run for this case [19]. Subsampled side and top views of the computational mesh used in the TFLO simulations are shown in Figs. 2.5-2.7. The computational grid dimensions used for each blade row are given in Tab. 4.1. The multi-stage steady flow was predicted with TFLO using 58 processors of the Lawrence Livermore Blue Pacific IBM-SP2 computer. Approximately 122 hours of wall clock (7068 hours of CPU time) were required for this steady solution.

The predicted results from the TFLO code at the mid-span section of the three blade-row configuration will be shown here as a sample of the validation and verification performed for this configuration. The set of results shown here represent only a fraction of the results that have been generated and analyzed in detail.

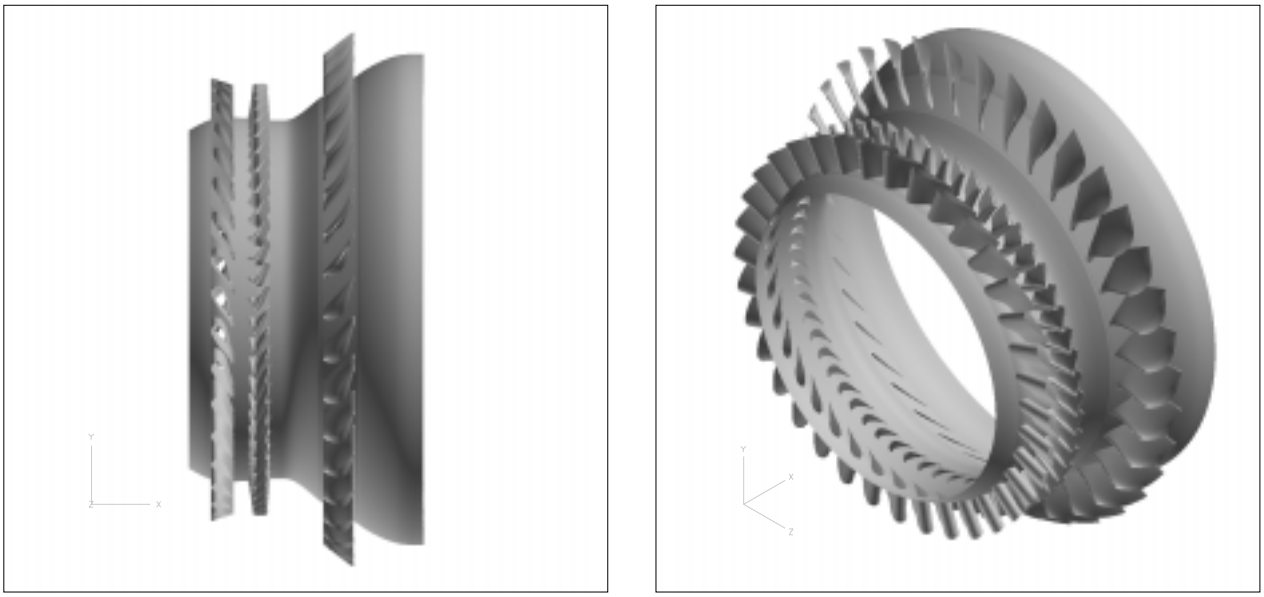


Figure 2.4: P&W High-Pressure Turbine Rig (HPT-Vane, HPT-Blade, and LPT Vane)

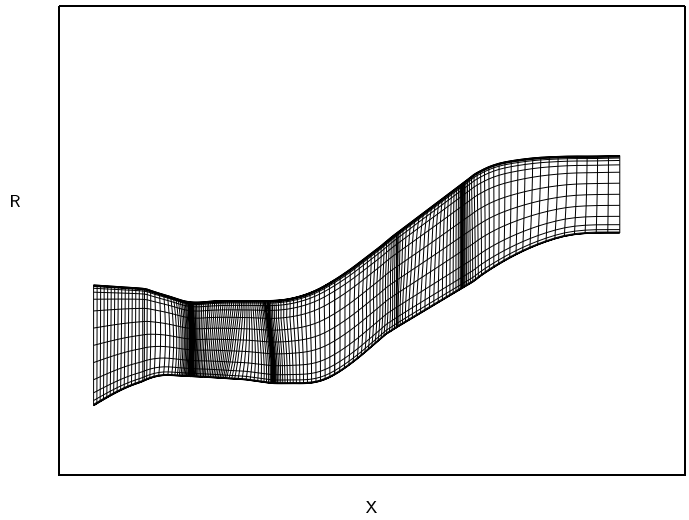


Figure 2.5: Computational Grid For P&W High-Pressure Turbine Rig (every fourth point shown).

Vane1	209 x 57 x 73
Blade	217 x 57 x 73
Blade Tip	73 x 17 x 17
Vane2	281 x 57 x 73
Total	2,962,924 Points

Table 2.1: Computational Grid Dimensions For P&W High-Pressure Turbine Rig Steady Flow Simulation

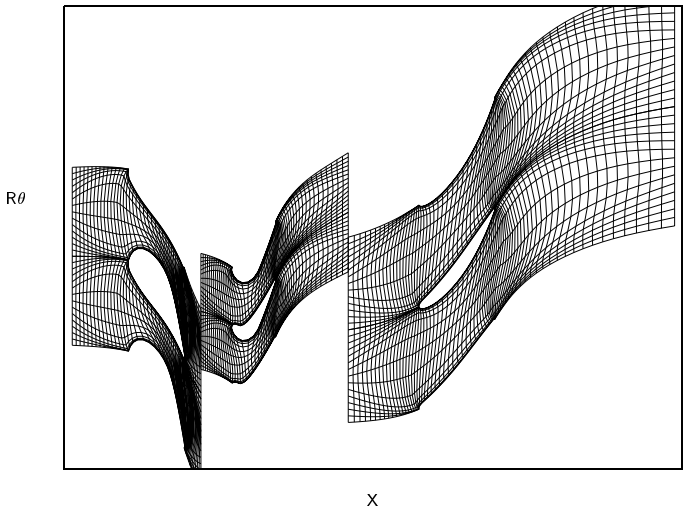


Figure 2.6: Mid-Span Blade-to-Blade Computational Grid For P&W High-Pressure Turbine Rig (every fourth point shown).

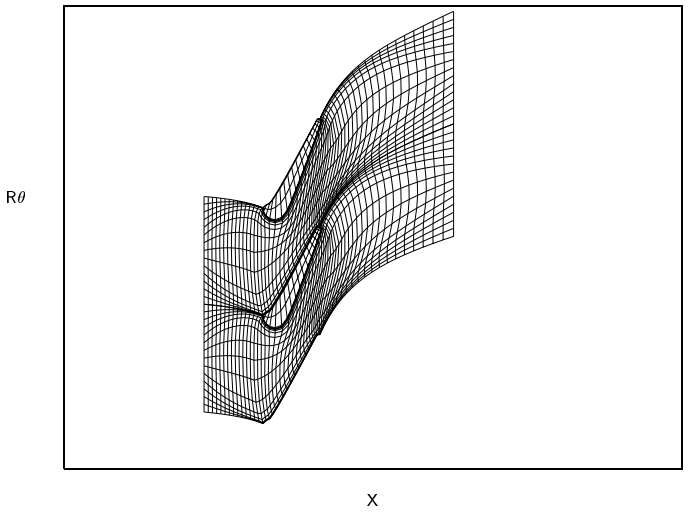


Figure 2.7: Blade Tip Computational Grid For P&W High-Pressure Turbine Rig (every fourth point shown).

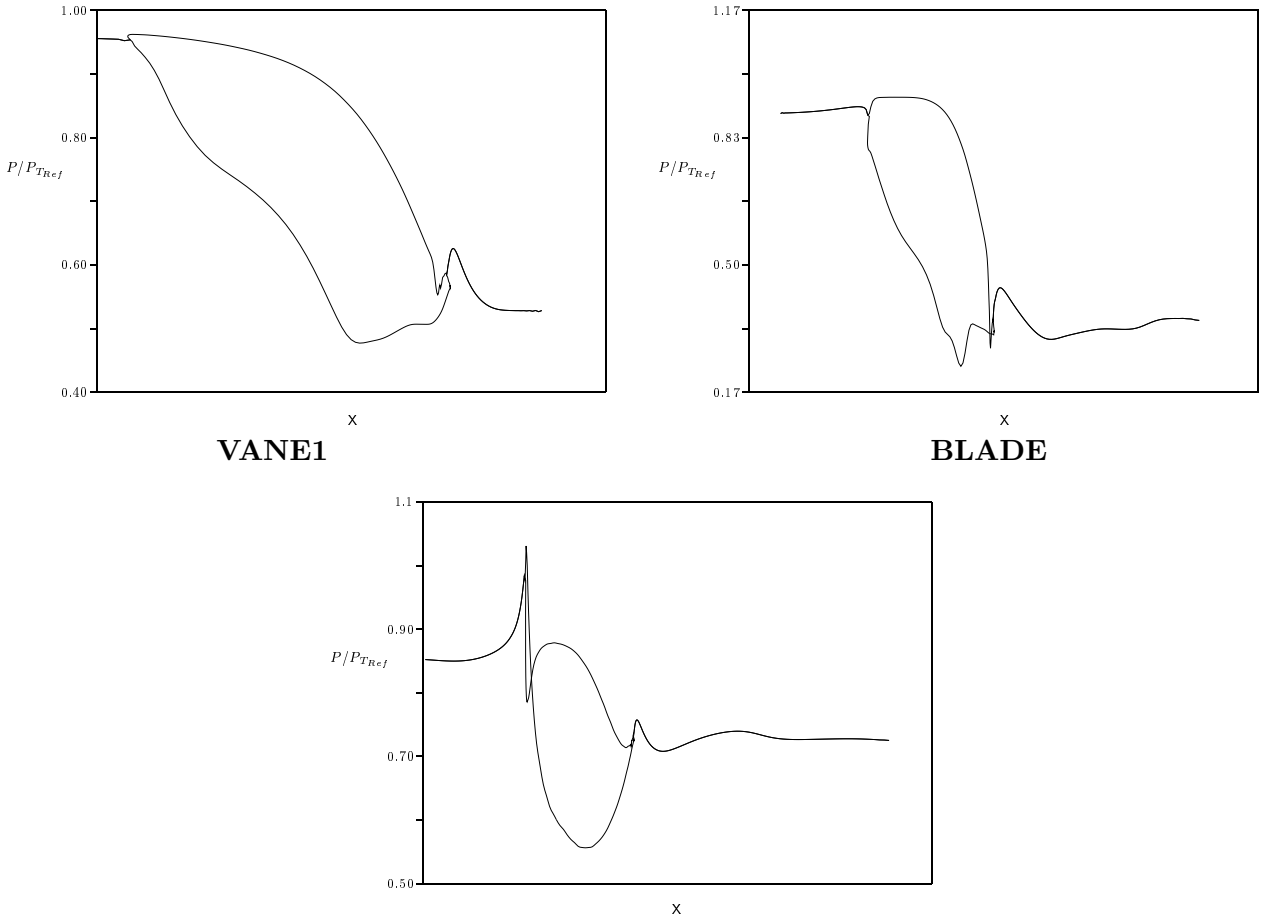


Figure 2.8: Mid-Span Pressure Distribution of P&W High-Pressure Turbine Rig

Figure 2.8 shows the predicted midspan pressure distribution of each blade row from the TFLO code. The aerodynamic loads (blade force) are determined primarily from integrals of the pressure distribution. Once airfoil loads are determined, structural analysis can be performed to determine the "steady" stress on each airfoil.

The trailing edge shock system of the blade, typical of transonic turbines, is illustrated in Fig. 2.9 with the TFLO predicted Mach number contours. The shock system is generated by the expanding supersonic flow around the trailing edge. The streamlines from the suction and pressure sides of the airfoil meet at the aft-end of the near-wake and the flow is re-directed to align itself with the streamwise direction. This re-direction of a supersonic flow is the source of the shock system. One leg of the trailing edge shock system emits from the pressure side of the near-wake and traverses across the passage where it strikes the suction side of the airfoil downstream of the throat. This leg of the shock reflects off of the suction side into the flow downstream of the trailing edge. The other leg of the shock system emits from the suction side of the near-wake where it immediately traverses downstream of the trailing edge. Both shocks of the system can and often interact with downstream blade rows causing unsteadiness in the pressure loading of the airfoil. Steady-flow analysis, such as the one shown here, are usually the first indication of possible shock/blade interaction. These solutions are also used as initial conditions for unsteady simulations that are executed to determine the amplitude of the unsteady aerodynamic loading on the airfoil. The unsteady flow solution for

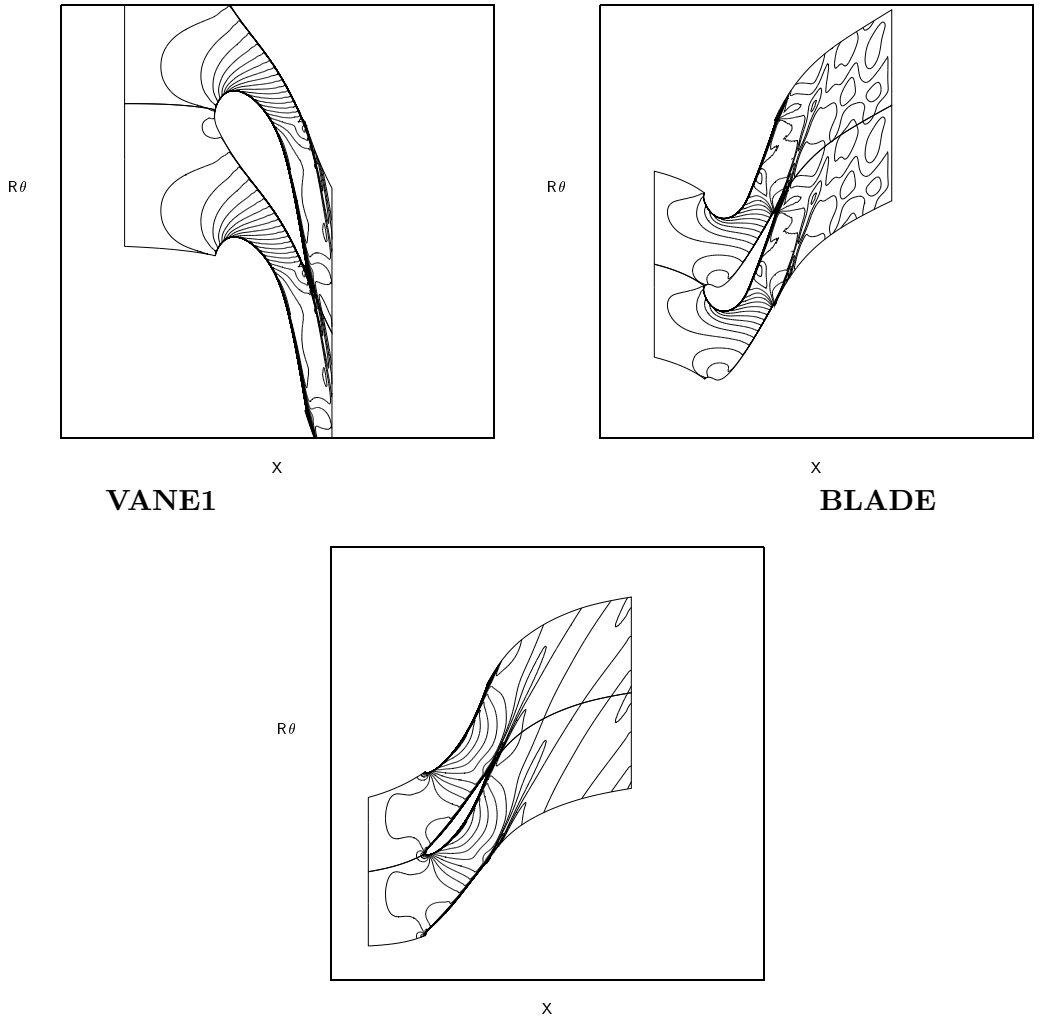


Figure 2.9: Mid-Span Mach Number Contours of P&W High-Pressure Turbine Rig

this configuration will be discussed later.

One of the most important capabilities of numerical tools like TFLO is the accurate prediction of aerodynamic performance. Since, in turbomachinery, work is either being applied to (for a compressor) or extracted from (for a turbine) the fluid, the prediction of the aerodynamic performance (*i.e.* efficiency) is dependent on the accurate computation of stagnation temperature and pressure change across each blade row. The loss in relative total pressure across each blade row, due to viscosity effects and the presence of shock waves, is one measure of how close the actual flow is to "ideal". Figure 2.10 shows the predicted relative total pressure contours in the axial planes behind each blade row from the TFLO prediction.

In the calculation of the efficiency of a particular stage of a turbine or compressor, the total temperature and total pressure changes across the blade-rows of each stage must be computed. In this calculation, the average total temperature and total pressure at the inlet and exit of each blade row is computed by first circumferentially averaging these conditions at a fixed radius and then radially averaging the circumferentially averaged conditions. In addition, circumferential averaging of the flow in a similar manner is performed to establish the "averaged" flow conditions that

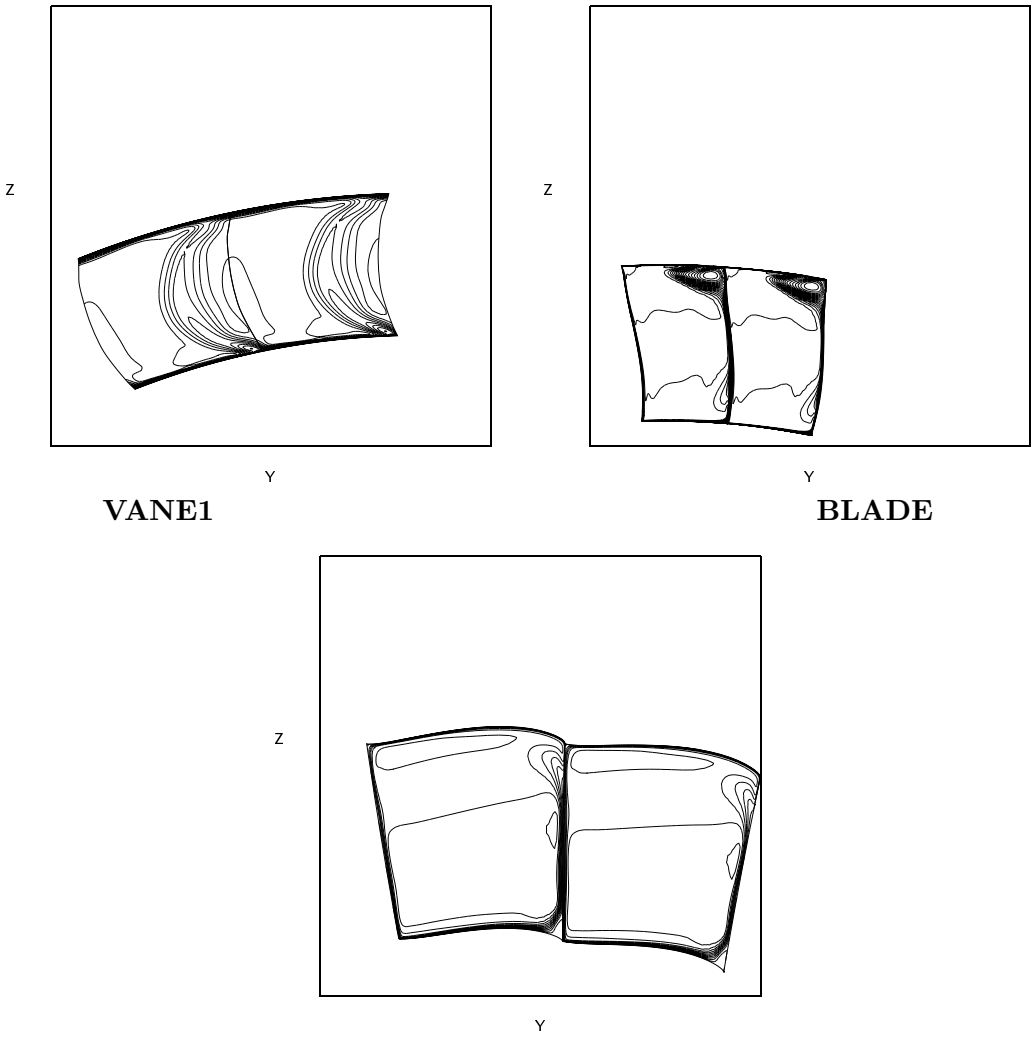


Figure 2.10: Exit Relative Total Pressure Contours of P&W High-Pressure Turbine Rig

Blade-Row	Relative Total Pressure Loss, $(P_{T_1} - P_{T_2})/P_{T_1}$, Difference (TFLO(CUSP)-3DFLOW)	Relative Total Pressure Loss, $(P_{T_1} - P_{T_2})/P_{T_1}$, Difference (TFLO(JST)-3DFLOW)
Vane-1	0.01504	0.00869
Blade	0.00750	-0.00090
Vane-2	0.00747	0.00343

Table 2.2: Difference in Predicted Relative Total Pressure Loss Between TFLO(JST) and TFLO(CUSP)

are communicated between blade-rows in a steady, multi-stage computation. This mixing-plane inter-blade boundary condition treatment is discussed in detail in the code development section. Figure 2.11 shows the TFLO prediction of the circumferentially-averaged relative total pressure. These distributions were generated from circumferential averages of the flow fields shown in Fig. 2.10 for each radial location.

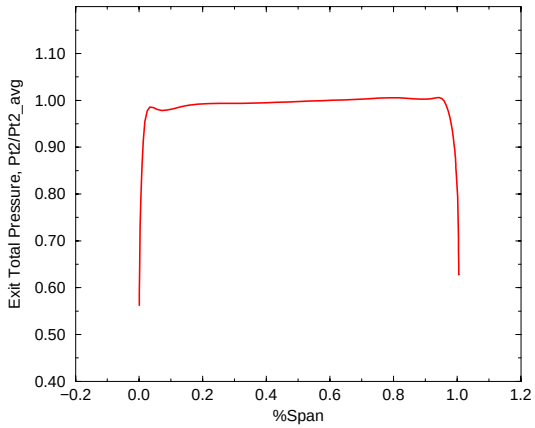
The change in total temperature (across the rotor) as well as the decrease in static pressure across all blade-rows is a function of the amount of circumferential flow turning produced by each blade-row. Viscous flows and shocks usually reduce the flow turning that a blade-row can produce compared to the ideal turning which is a function of the airfoil camber. Spanwise flow redistribution resulting from secondary flows and radial pressure gradients can also change the radial distribution of flow turning from ideal and/or design intent. The circumferentially-averaged exit flow angle behind each blade row of the P&W turbine rig predicted by TFLO is shown in Fig. 2.12.

Effect of the CUSP Dissipation Model

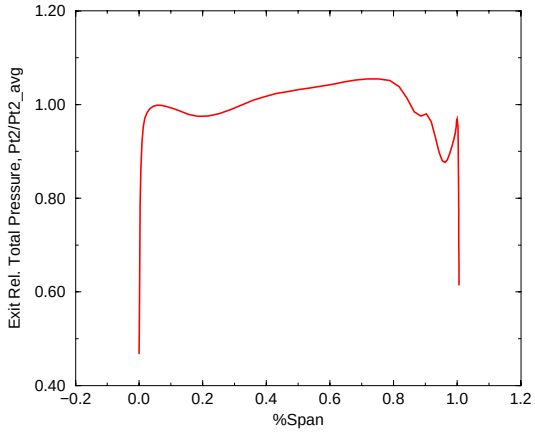
An investigation of an alternate dissipation model, known as the CUSP scheme, was also performed for this particular configuration. As explained in the description of the numerical schemes used in TFLO, the dissipation model is used to damp oscillatory modes in the numerical solution. The standard dissipation model used in TFLO is that of Jameson, Schmidt, and Turkel (JST) [20, 22]. The CUSP dissipation model was developed by Jameson [21] as a refinement of the JST scheme and has been proven both in theory and in practice [27, 28] to be superior to the JST scheme in terms of shock capturing and numerical accuracy.

Predicted pressure distributions from TFLO using the CUSP dissipation model are very similar to those shown in Fig. 2.8. The CUSP scheme resulted in slightly smoother pressure distributions and slightly less-defined shocks than those predicted with the JST scheme. This effect is caused by higher levels of dissipation produced by the current implementation of the CUSP scheme in TFLO, which are higher than theoretically expected. Figure 2.13 shows a comparison in the predicted Mach number contours of the TFLO code with both the JST and CUSP schemes. The clustering of the Mach number contours around the trailing edge shock system is less defined in the CUSP prediction.

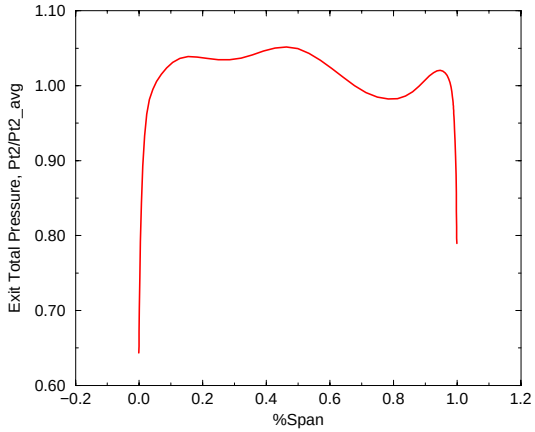
The higher level of dissipation associated with the current implementation of the CUSP scheme is also reflected in the predicted relative total pressure loss. Table 2.2 gives the predicted relative total pressure losses for each blade row for both the JST and CUSP schemes relative to the 3DFLOW procedure. The results show that TFLO with the JST dissipation scheme more closely predicts the same relative total pressure loss from 3DFLOW (which also uses the JST scheme). Unfortunately, the relative total pressure loss is very difficult to measure in an experimental rig so that there is no experimental measurement available for the actual relative total pressure loss.



VANE1

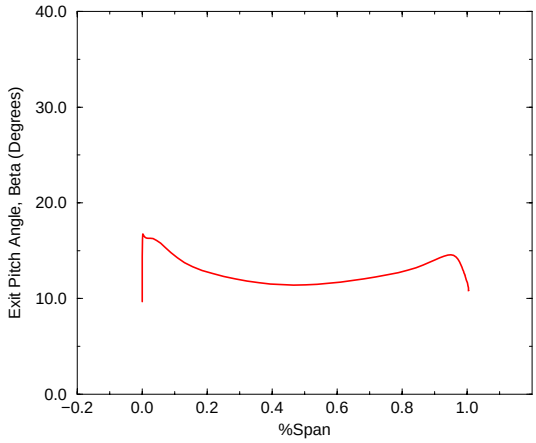


BLADE

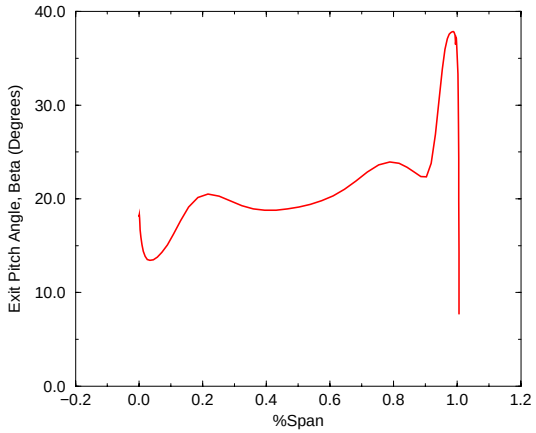


VANE2

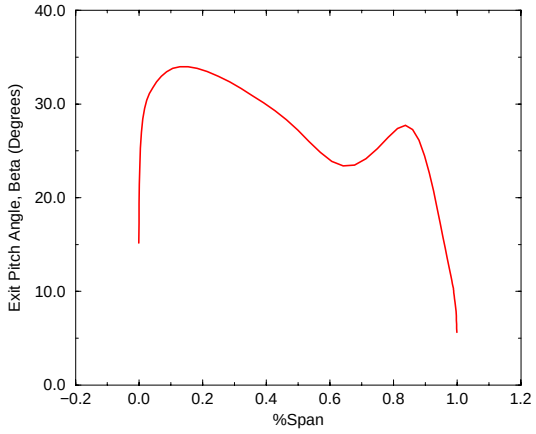
Figure 2.11: Exit Circumferentially-Averaged Relative Total Pressure Distributions of P&W High-Pressure Turbine Rig



VANE1



BLADE



VANE2

Figure 2.12: Exit Circumferentially-Averaged Relative Pitch Flow Angle Distributions of P&W High-Pressure Turbine Rig

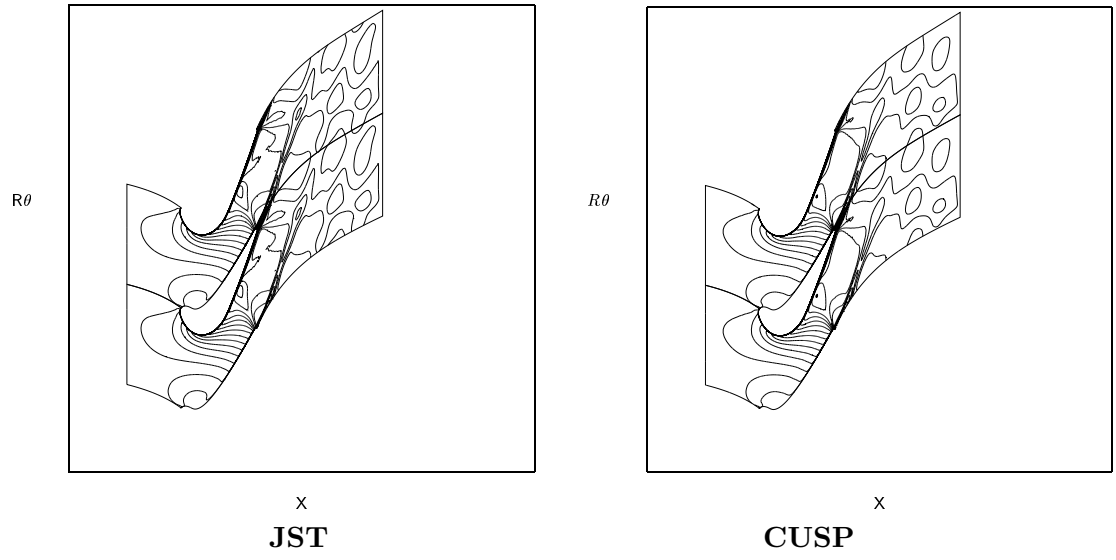


Figure 2.13: Mid-Span Mach Number Contours of P&W High-Pressure Turbine Rig Predicted by TFLO with JST and CUSP Dissipation Schemes

Vane1 (9)	209 x 57 x 73	7,826,841 Points
Blade (14)	217 x 57 x 73	12,641,118
Blade Tip (14)	73 x 17 x 17	295,358
Vane2 (9)	281 x 57 x 73	10,523,169
Total		31,286,486

Table 2.3: Computational Grid Dimensions For P&W High-Pressure Turbine Rig Unsteady Flow Simulation

Unsteady Flow

A large-scale simulation of the unsteady flow in the Pratt & Whitney 1-1/2 stage turbine rig is ongoing on ASCI platforms. In this simulation, a total of 9 1st and 2nd vane passages along with 14 blade passages were included in order to achieve a common circumferential pitch among the 3 blade rows. Clark *et al.* [19] found that simulation of the smallest fraction of the circumference with the actual blade count was critical to accurately predict the airfoil unsteady pressures. The present 9/14/9 vane/blade/vane simulation represents 1/4th of the entire circumference of the rig. A comparison between Pratt & Whitney's 3DFLOW code and experimental data for this configuration can be found in Ref. [19]. Preliminary results from the TFLO simulation are shown below. Approximately 20% of the overall unsteady solution has been obtained to-date.

Figure 2.14 shows contours of the pressure at one instant in time from the unsteady simulation along a cylindrical cutting surface located at mid-span. The pressure waves associated with the suction-side leg of the blade trailing edge shock system are illustrated in this figure. These pressure waves traverse downstream along the transition duct and interact with the downstream 1st vane (of the low-pressure turbine). A weak reflected pressure wave can be observed slightly ahead of the last vane. This upstream-running reflected pressure wave propagates back to the blade where it can interact and cause additional flow unsteadiness. These types of shock/pressure-wave/blade interactions can sometimes lead to large fluctuations in airfoil surface pressure loads and corresponding stress. This is why unsteady aerodynamic simulations, such as this, are important to

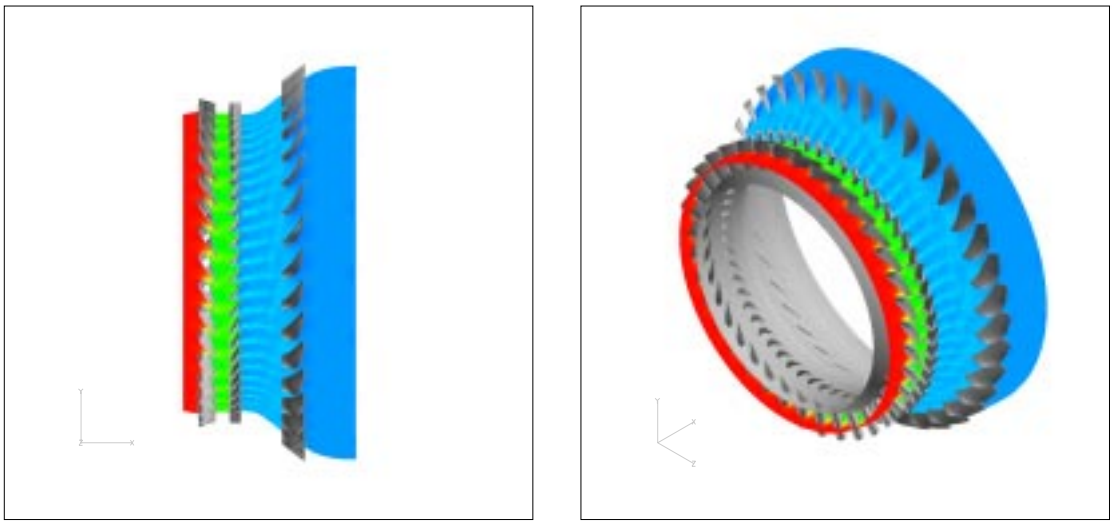


Figure 2.14: Instantaneous, Mid-Span Pressure Contours of P&W High-Pressure Turbine Rig Unsteady Flow Simulation

turbomachinery designers.

Contours of instantaneous entropy for the same unsteady simulation are shown in Fig. 2.15. As opposed to the pressure contours shown above that highlight the shock and pressure interaction, the entropy contours highlight the viscous wake/blade interaction. The wakes emanating from the trailing edge of each airfoil propagate downstream, where they interact with the next blade row. Since the velocity in the wakes is lower than that of the surrounding free stream, pressure fluctuations occur on the airfoil surfaces when the wakes pass over them. As with the shock/airfoil interaction, these pressure fluctuations can lead to degradation in aerodynamic performance and durability problemd for the airfoils.

Once the unsteady flow simulation has progressed to a point that the flow field is periodic in time, a complete animation of various flow quantities will be constructed. In addition, the minimum-to-maximum pressure envelope and the time-averaged pressure distributions will be compared with experimental data to evaluate the unsteady flow prediction capability of TFLO.

2.3.2 Wright-Patterson Stage Matching Investigation (SMI) Rig

Measurements were performed by Dr. William Copenhaver and his research team at Wright-Patterson Air Force Base on a $1-1/2$ stage compressor rig [26]. The main goals of his experimental research were to identify and investigate the flow structures from wake generators (WGs), the wake-blade interaction, and the wake-shock interaction in transonic compressor rotors.

The experimental hardware was designed such that the wake generator-to-rotor axial spacing (where the wake generator models a stator) can be set to three values denoted as *close*, *mid*, and *far*, as shown in Fig. 2.16. The spacings normalized by the WG chord are given in Table 2.4.

Wake Generators

The wake generators were designed with the intent of producing wakes typical of modern, highly loaded, low-aspect-ratio, front-stage compressor stators. To simplify the experiment, the wake generators were designed as uncambered airfoils produce no flow turning. Results from measurements of stator wakes from rig tests were used as the design target. These wakes were produced in a stator that had no bow or sweep, with a hub Mach number of 0.95, a hub D-Factor of 0.55, and a solidity

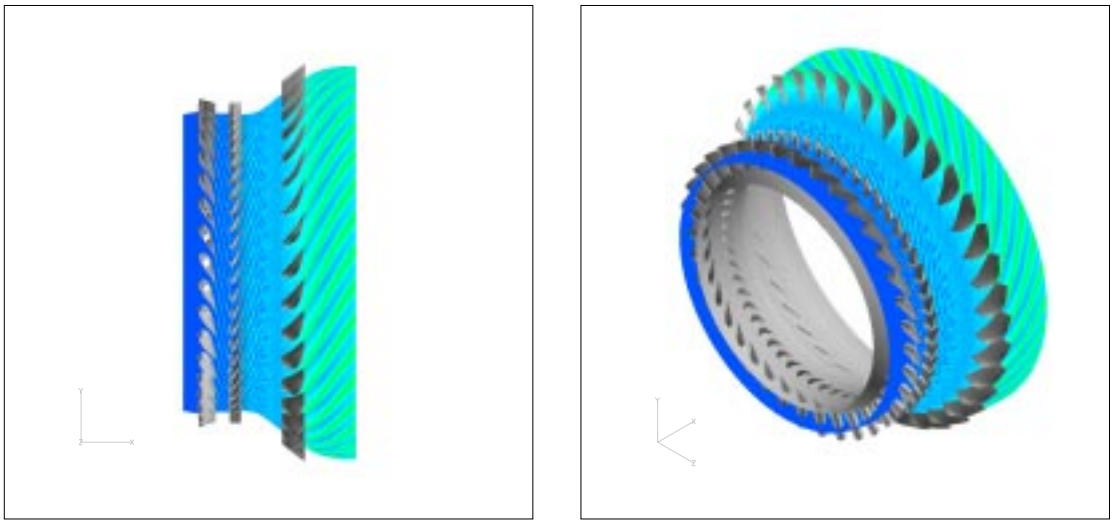


Figure 2.15: Instantaneous, Mid-Span Entropy Contours of P&W High-Pressure Turbine Rig Unsteady Flow Simulation

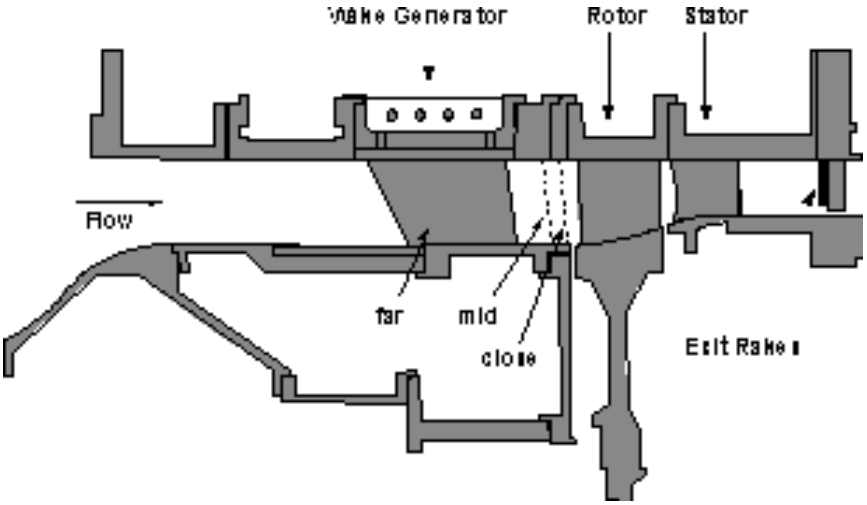


Figure 2.16: Stage Matching Investigation Rig Cross-Section

Spacing	x/c (mean)	x/c (hub)	x/c (tip)
Close	0.12	0.10	0.14
Mid	0.26	0.26	0.26
Far	0.56	0.60	0.52

Table 2.4: Wake Generator Axial Spacing (normalized by local WG chord)

of 1.64. For simplicity and for isolating the effects of various wake parameters, a 2D representation of the wake was desired.

The airfoils have a small LE radius with a relatively blunt trailing-edge (TE) radius. In the design process, this produced the optimum combination of profile and base drag for matching the desired highly loaded stator wake. Since the wake width and loss are strong functions of solidity, the solidity of the WGs was held constant from hub to tip. This results in a tapered airfoil chord along the radius, as seen in the Figure. The TE was swept to allow a constant non-dimensional mixing length from hub to tip. With no diffusion in the flow path, the end-wall losses were expected to be low. There is no clearance and no fillet at either the hub or the tip.

Compressor Stage

The rotor and stator were designed at Wright-Patterson. A summary of the aerodynamic design parameters is given in Table 2.5. A compressor map of the overall stage pressure ratio for the *clean*-inlet and 40-WG configurations is shown in Fig. 2.17.

TFLO Calculations, Rotor Only

The performance of an isolated rotor configuration was computed using TFLO. The back pressure was varied from *choke* to *stall* conditions to obtain a predicted rotor speed-line. Eight operating conditions were computed with back pressures shown in Table 2.6. Figure 2.18 shows the resulting total pressure ratio and adiabatic efficiency speedlines.

Details of Flow Field, Back Pressure = Baseline

Figures 2.19 and 2.20 show the pressure distributions on the blade surfaces and the relative Mach number contours in the blade-to-blade direction at two different spanwise locations (50% and 97% spanwise from hub). Dual shock waves appear under this condition, the inlet bow shock and the passage shock. In the tip gap region, flow leakage is found from pressure side to suction side mostly at the mid-chord location. In addition, a smaller amount of leakage can also be found right behind the leading edge, as shown in Figure 2.21.

Details of Flow Field, Back Pressure = +12%

Figures 2.22 and 2.23 show the pressure distributions on the blade surfaces and the relative Mach number contours in the blade-to-blade direction at two different spanwise locations (50% and 97% spanwise from hub). Compared to the baseline back pressure condition, the passage shock wave has been pushed forward and has merged with the inlet shock wave to become a single shock wave. This single shock wave departs from the leading edge of the blade, which indicates the operating condition is close to stall. In the tip gap region, the flow is found to leak from pressure side to suction side mainly at leading edge owing to the big pressure difference right after the inlet shock wave, as shown in Figure 2.24.

Details of Flow Field, Back Pressure = -10%

Figures 2.25 and 2.26 show the pressure distributions on the blade surfaces and the relative Mach number contours in the blade-to-blade direction at two different spanwise locations (50% and 97% spanwise from hub). Compared to the baseline back pressure condition, the passage shock wave is pulled further downstream. The inlet bow shock wave is also moved closer to the blade leading edge. In the tip gap region, the flow leaks from the pressure side to the suction side mainly in the vicinity of the mid-chord as shown in Figure 2.27.

Parameter	Rotor	Stator
Number of Airfoils	33	49
Aspect Ratio - Average	0.961	0.892
Inlet Hub/Tip Ratio	0.750	0.816
Flow Rate, lbm/sec	34.46	–
Tip Speed, Corrected ft/sec	1120	–
M_rel LE Hub	0.963	0.82
M_rel LE Tip	1.191	0.69
PR, Rotor	1.88	–
PR, Stage	–	1.84
D Factor, Hub	0.545	0.502
D Factor, Tip	0.530	0.491

Table 2.5: SMI Aerodynamic Design Parameters

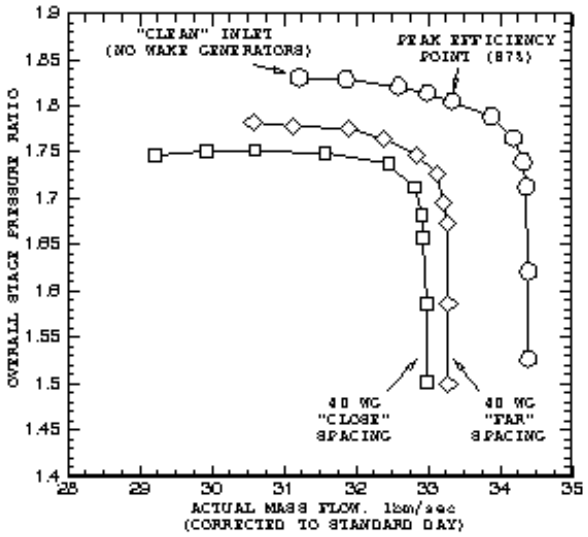


Figure 2.17: Overall Stage Pressure Ratio for the *Clean* Inlet and 40 Wake Generator Configurations

Index	Back Pressure
1	+15% (stalled)
2	+12%
3	+10%
4	+8%
5	+5%
6	baseline ($Pascal$) (107627.1568, 116797.1836)
7	-5%
8	-10%
9	-20%

Table 2.6: Back Pressure Settings for Speedline Calculation

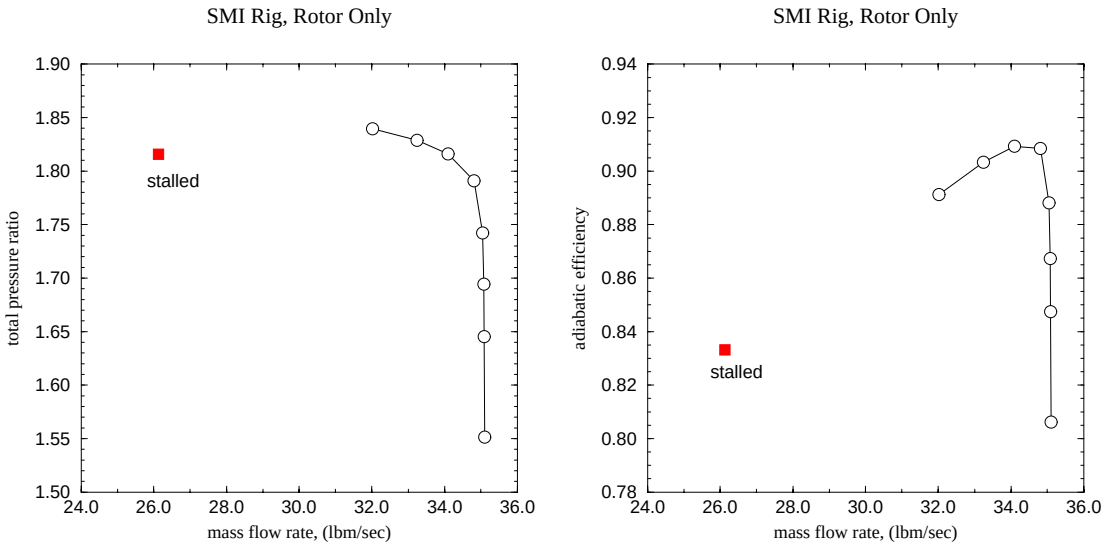


Figure 2.18: TFLO Predicted SMI Rotor Performance

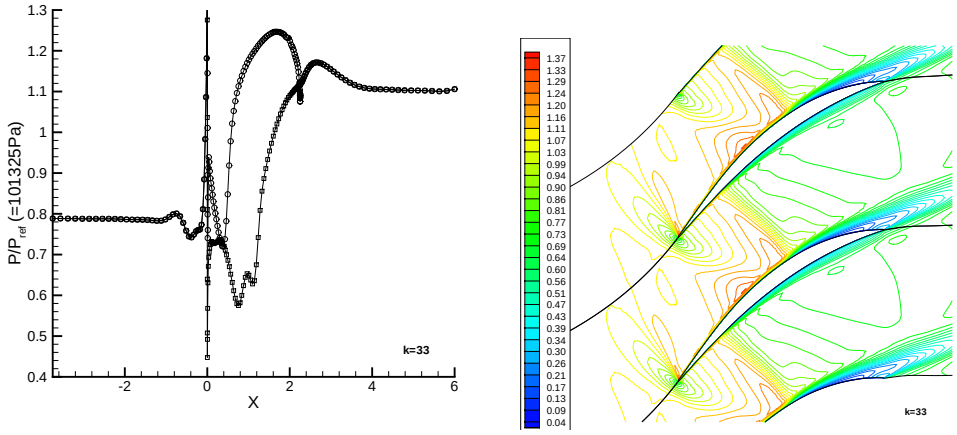


Figure 2.19: Static Pressure on Blade Surfaces and Relative Mach Contours at Mid-Span, baseline back pressure

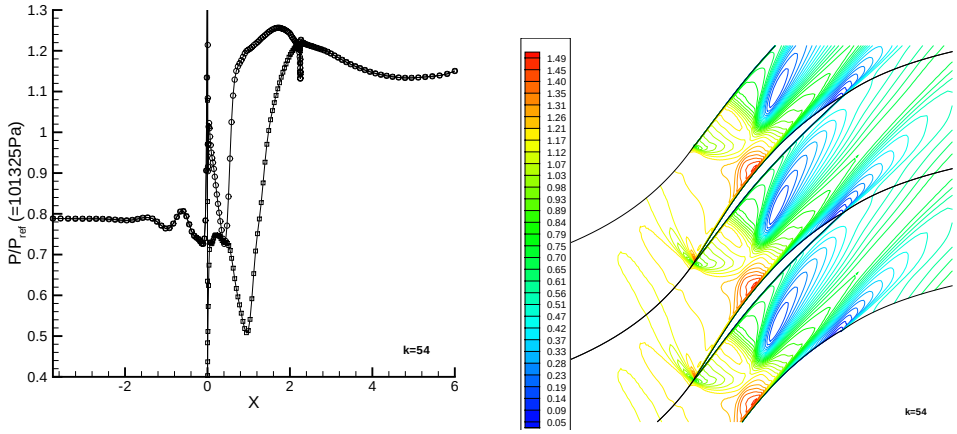


Figure 2.20: Static Pressure on Blade Surfaces and Relative Mach Contours at 97% Span from Hub, baseline back pressure

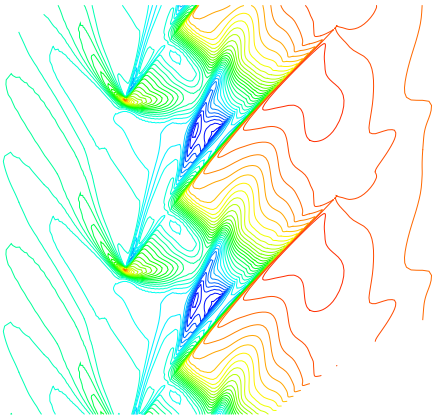


Figure 2.21: Static Pressure Map in the Tip Gap Region, baseline back pressure

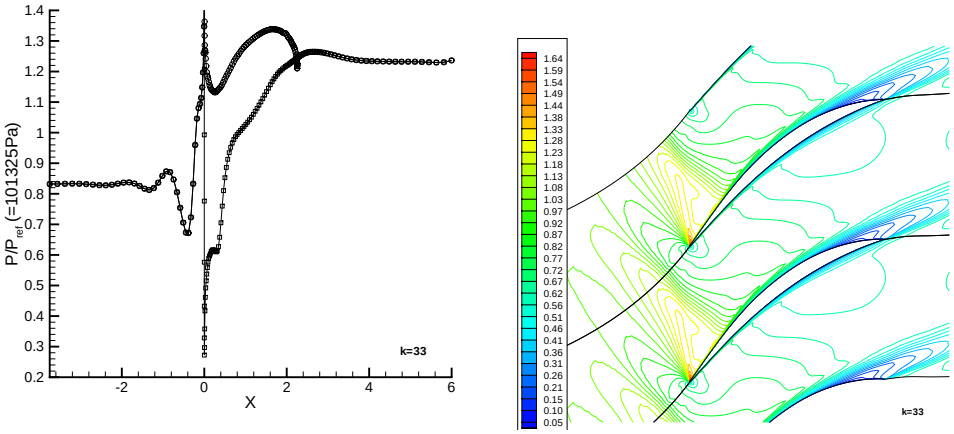


Figure 2.22: Static Pressure on Blade Surfaces and Relative Mach Contours at Mid-Span, +12% back pressure

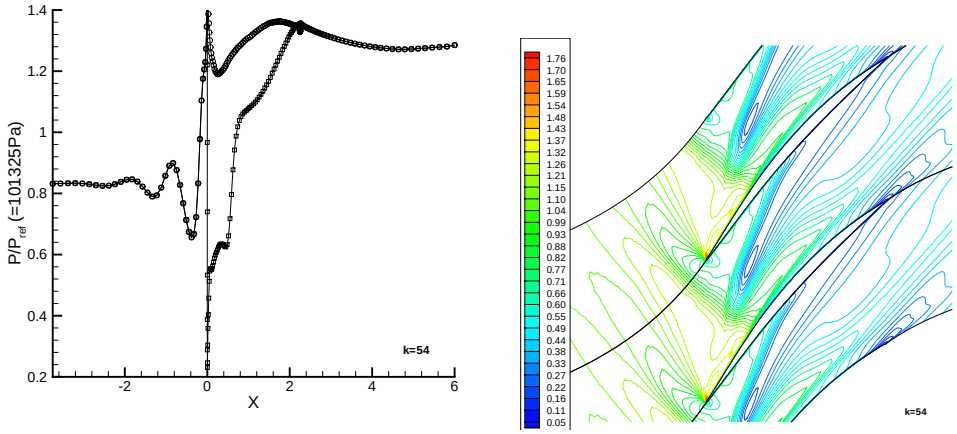


Figure 2.23: Static Pressure on Blade Surfaces and Relative Mach Contours at 97% Span from Hub, +12% back pressure

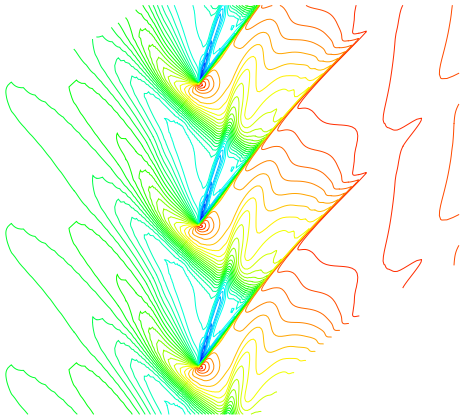


Figure 2.24: Static Pressure Map in the Tip Gap Region, +12% back pressure

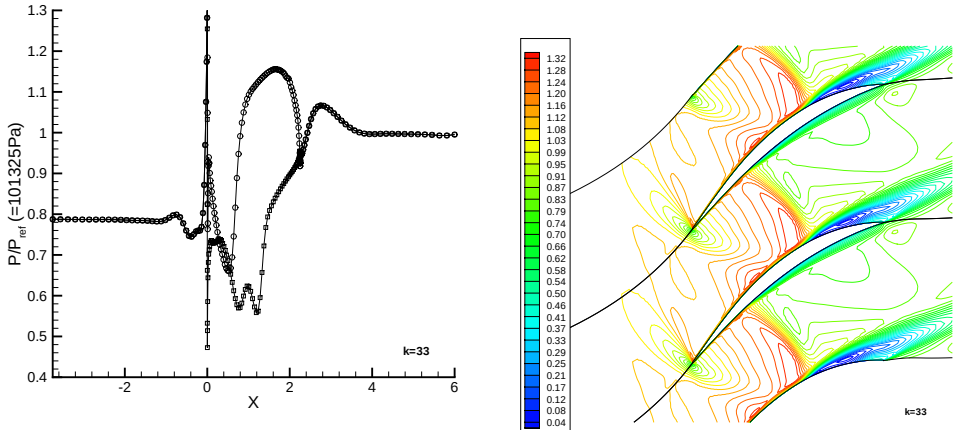


Figure 2.25: Static Pressure on Blade Surfaces and Relative Mach Contours at Mid-Span, -10% back pressure

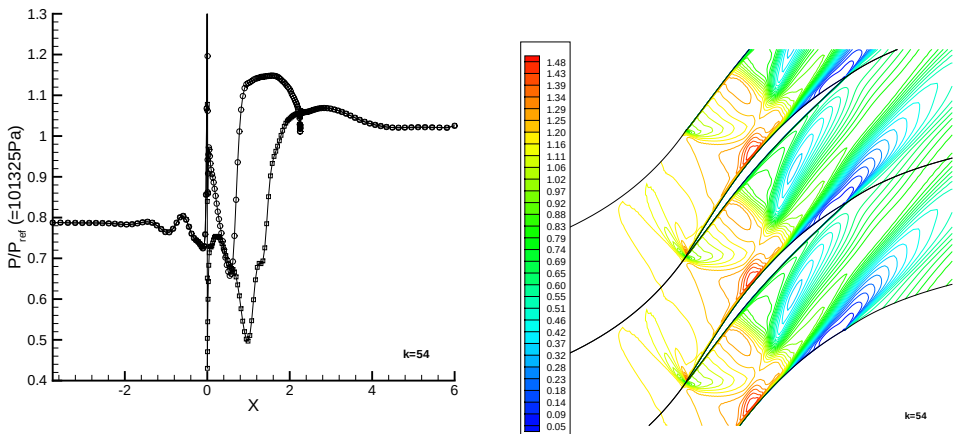


Figure 2.26: Static Pressure on Blade Surfaces and Relative Mach Contours at 97% Span from Hub, -10% back pressure

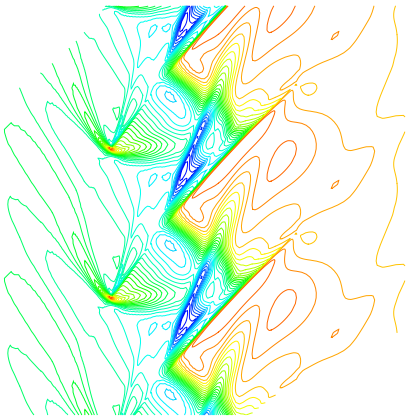


Figure 2.27: Static Pressure Map in the Tip Gap Region, -10% back pressure

2.4 Turbine Wakes and Transition Simulation

During FY 2000, the numerical algorithm for direct numerical simulation of transitional flow physics in a low-pressure turbine (LPT) passage has been implemented. The methodology was described in our 1999 Annual Technical Report.

The LPT is installed in jet engines upstream of the propelling nozzle, supplying power to the fan and first compressor stages. A feature of the LPT is that representative Reynolds numbers are in a range where DNS is practicable; also, the aspect ratio is relatively high and the Mach number relatively low. Our computations were performed on Nirvana using 57 million grid points. Analysis and visualization of the simulation results revealed new fluid mechanics phenomena, which possess both lasting scientific value and immediate engineering importance.

2.4.1 Flow System

Work in FY 2000 has focused on computing the evolution of an incompressible flow through a smooth turbine passage, as in figure 2.28 and figure 2.29. The geometry, termed T106, is the mid-span section of a Pratt & Whitney PW2037 rotor. When normalized by axial chord L, the blade pitch is 0.9306, and true chord length is 1.1647. Inlet and exit planes of the computational domain are at $x = -0.5$, and $x = 2.0$, respectively. The spanwise dimension is prescribed as 0.15.

Upstream, the flow velocity is $U_{ref} = 1.0$, and makes an angle $\alpha = 37.7^\circ$ with the x-axis. $Re = U_{ref}L/\nu = 1.48 \times 10^5$. The designed exit flow makes an angle of -63.2° with the x-axis. Upstream wakes traversing the inlet are representive of those that would be generated by circular cylinders moving in the y direction at $U_{cyl} = -1.2048$. The cylinders are equally spaced so that they cut through a xz-plane at a specified passing period $\mathcal{T} = 0.7724$. The mean flow properties of the wake are those produced of 0.01177 diameter cylinders positioned at $x = -0.33$.

The flow is turned by more than 100° , and accelerated by a factor of about 1.9, producing significant streamwise straining. The magnitude and direction of the principle axes of strain vary with position. These particular features — large rate of strain, varying orientation, incident wake — produce an intriguing vortical structure inside the turbine passage, which had not previously been discovered.

2.4.2 Simulation Approach

Numerical integration is by the fractional step method for unsteady, three-dimensional Navier-Stokes equations on general geometry, structured grids. The governing equations are discretized by finite volumes on a staggered mesh. The dependent variables are pressure and volume fluxes across the faces of the cells. Transformation between Cartesian velocity components and volume fluxes is made via area vectors and their reciprocals. Divergence-free flow is obtained by solving a discrete Poisson equation.

Convergence of the Poisson equation was accelerated by the multigrid method after Fourier transformation along the span. For each wavenumber, the decoupled, 2-D Helmholtz system is solved using a V-cycle multigrid subroutine. Cell-wise coarsening is adopted instead of the usual pointwise coarsening. Scalable parallelization is achieved with the OpenMP Fortran Application Program Interface. Simulations were performed on a 57 million point grid (1153,385,129) and on a 25 million point grid (769,257,129), in the streamwise, wall-normal and spanwise directions, respectively.

2.4.3 Results

Figure 2.30 shows the wall static-pressure coefficient. Experimental data were kindly made available to us by Dr. Peter Stadtmueller of the Universität der Bundeswehr München in Germany. Figure 2.31 shows contours of instantaneous velocity magnitude over one xy-plane which gives the locations of the migrating wake at this particular time.

When wakes are present, transition occurs near the leading edge, but the flow quickly relaminarizes on the suction surface. The tangential velocity component immediately adjacent to the blade is shown in figure 2.32 at one instant. Transition is induced by turbulent spots (Wu et al 1999, Wu & Durbin 2000a,b), which are seen to form near midchord, to grow, and to merge into the trailing edge, fully turbulent region. This is a bypass transition process.

Contours in figure 2.32 on the pressure-side surface display large amplitude spikes. They move slowly with time and are most apparent where strong streamwise stretching occurs. They are symptomatic of an intriguing phenomenon discovered by us using flow visualization for vortex identification. Figure 2.33 shows a three-dimensional, perspective view of negative λ_2 (see Wu & Durbin 2000c for detailed definitions) regions at one random instant. Fluctuating vortex filaments in the upstream wake form a layer of vortices embedded in the flow. Stretching aligns the vortex filaments towards the axis of principal stress and intensifies vorticity magnitude in that direction. Further stretching gradually collapses this layer of vortex filaments into circular tubes. As the cumulative effect of the rate of strain adds up, the circular tubes become progressively more distinct; they might be leg elements of extremely long hairpin vortices. With the turning of the turbine, wakes are gradually rotated counter clockwise, causing more and more of the leg to attach to the pressure surface (Figure 2.34). Thus the flattened primary vortices at the wall are continuously fed by the descending free-stream legs.

As the primary vortices descend towards the wall, they induce vortices with the opposite sense of rotation under the constraint of no-slip. The effect of such conter-rotating vortices on near-wall streamwise momentum transport is shown in figure 2.35. The mushroom shape of these structures is typical of low speed streaks produced by upwash of counter rotating vortex pairs. In between the vortices in a pair, low-momentum fluid is transported away from the wall, and is spread laterally as it flows up and over them.

It is known that the turbine pressure surface can run hotter by as much as several hundred degrees Fahrenheit than the suction surface. Of course, this is an issue in the high pressure turbine, while the flow here is that of a low pressure turbine; however, some of our observations may be relevant. Intense streamwise vortices, descending from passing wakes could enhance heat transfer, especially for the flow near the downstream half of the pressure surface. The tendency for a wake to be dragged parallel to the surface applies also to a material line aligned across the passage. If this were a streak of hot fluid, then the combination of dragging and entrainment into the near-wall vortices could heat the wall.

2.4.4 Simulation Plan

Work in FY 2001 will be concentrated on simulation of turbine cascade flow with continuous grid turbulence as inflow. The inlet turbulence intensity will be set to 8% using the isotropic, homogeneous turbulence data of Dr. Alan Wray of NASA/Ames. The computation will use a 90 million grid-point mesh and promises to be yet another land-mark contribution to turbulence simulation as well as to turbo-machinery engineering.

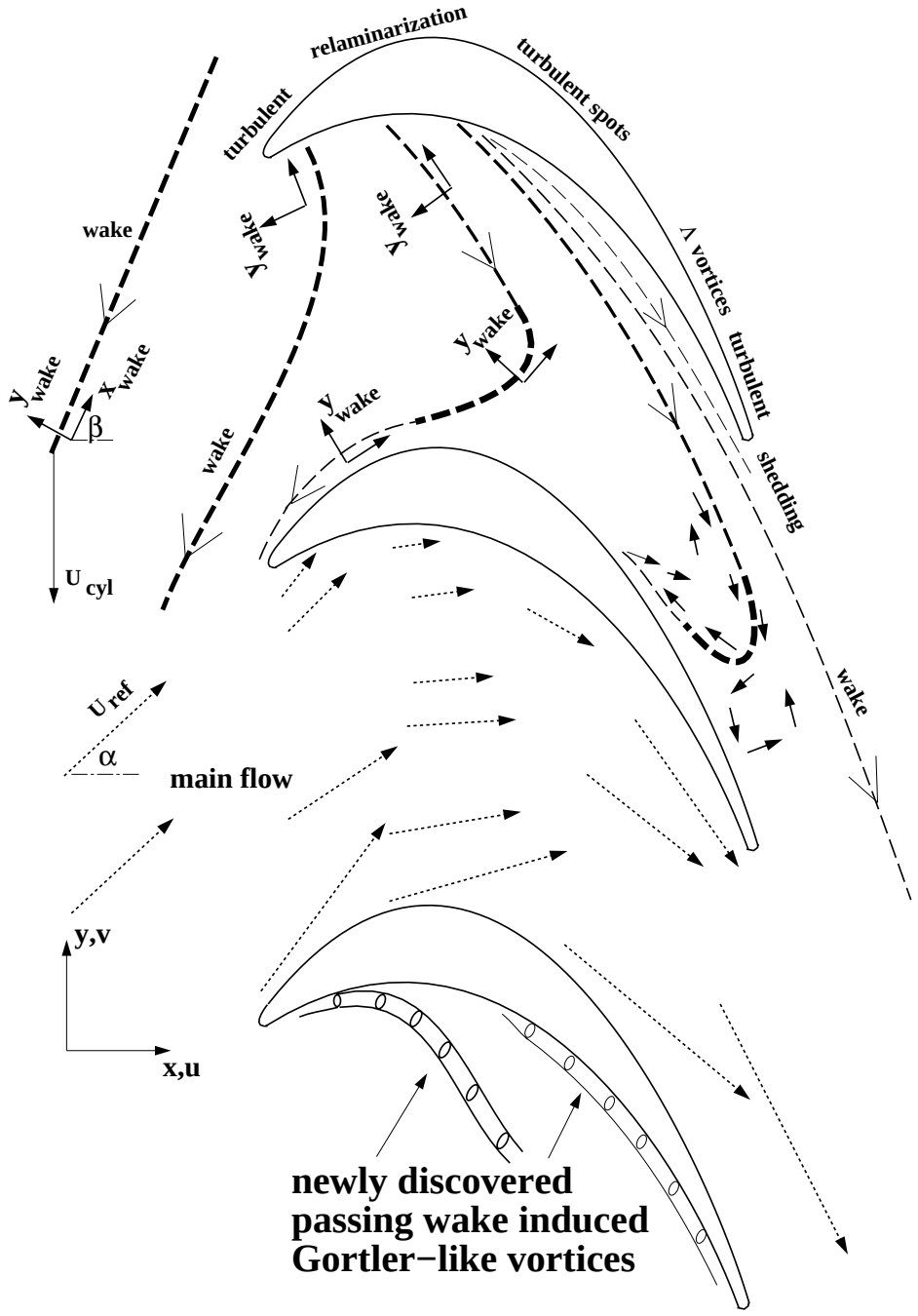


Figure 2.28: Sketch of the observed wake distortion, wake induced transition, primary type longitudinal vortices as well as flow velocity vectors in the present simulation.

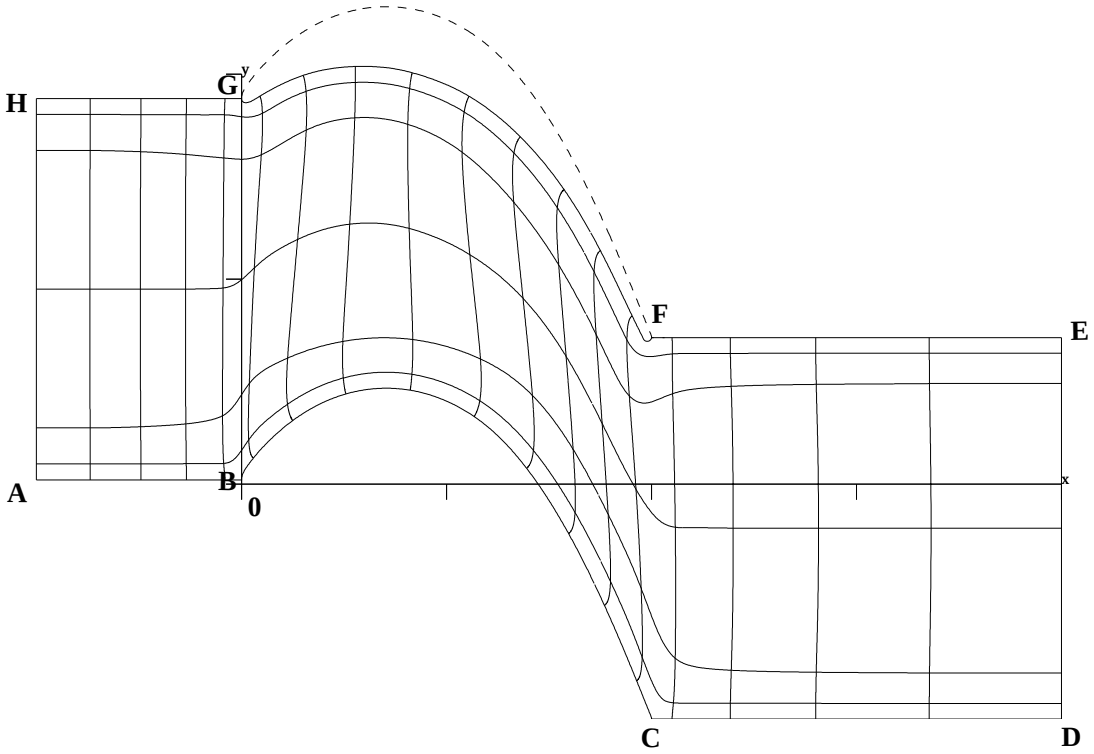


Figure 2.29: Cross-sectional view of the computational domain and one level of multigrid mesh (1153/64,385/64,129/129).

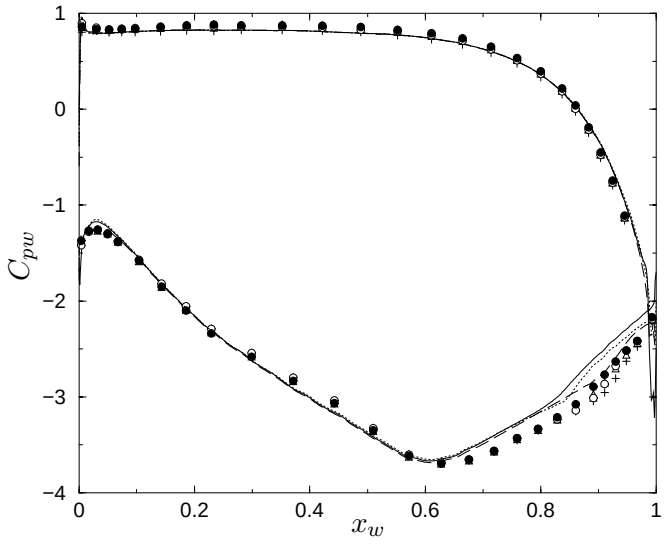


Figure 2.30: Time-averaged wall static pressure coefficient C_{pw}; present DNS: —— (nowake), —— (with wake, $769 \times 257 \times 129$ mesh), $\cdots\cdots$ (with wake, $1153 \times 385 \times 129$ mesh); all symbols: compressible flow experiments with varying inlet free-stream turbulence intensity and Mach number, but without upstream passing wake.

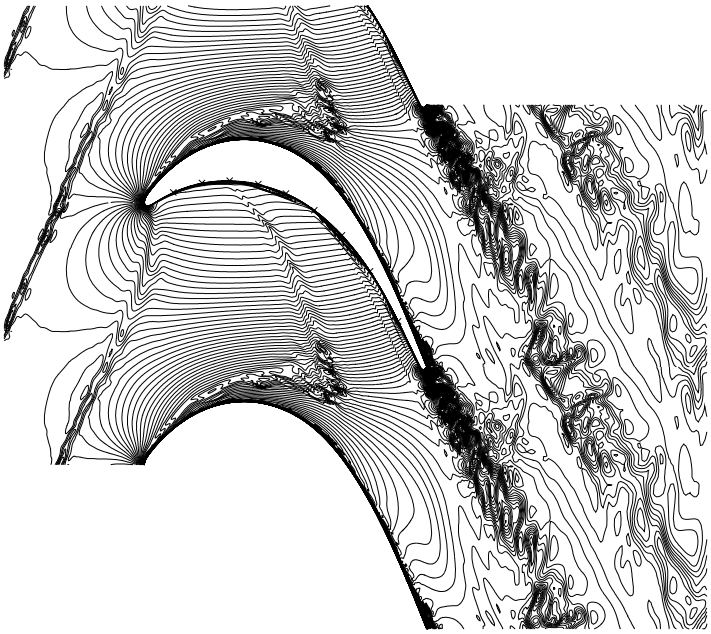


Figure 2.31: Velocity norm over the xy-plane of $z = 0.0$; $\times$ mark of $0.1 \leq x_w \leq 0.9$ with increment 0.1.

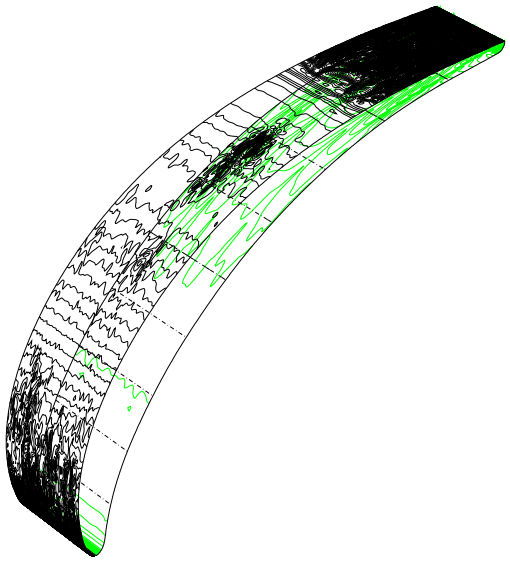


Figure 2.32: v velocity component over the suction (dark lines) and pressure (light lines) surfaces at one instant; $-\cdot-$ mark of $0.1 \leq x_w \leq 0.9$ with increment 0.1.

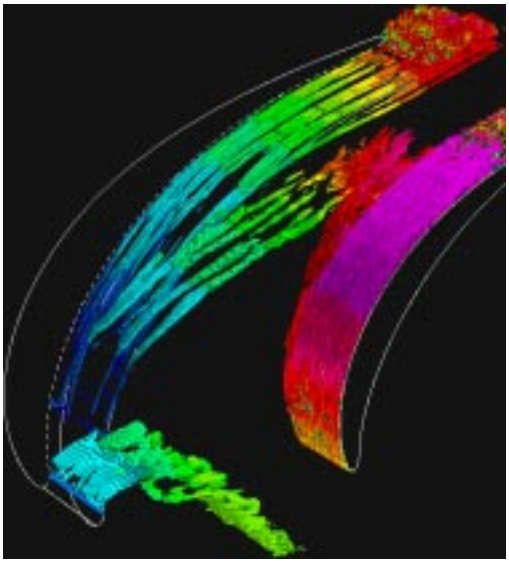


Figure 2.33: Regions of negative λ_2 inside the turbine passage.

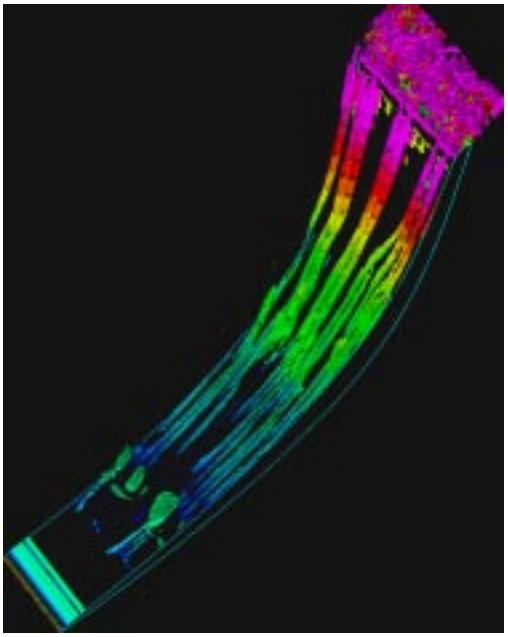


Figure 2.34: Regions of negative λ_2 near the pressure surface.

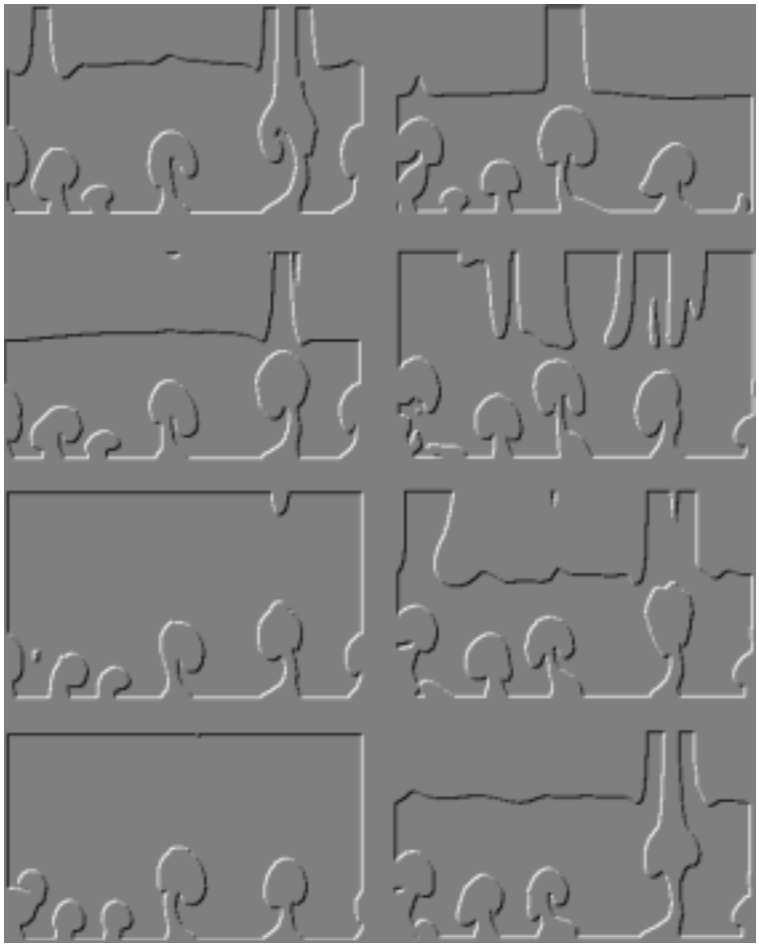


Figure 2.35: Time-sequence plots (8, from left column to right column) showing regions of negative fluctuating streamwise velocity (with respect to spanwise-averaged mean) in the plane perpendicular to the pressure surface at $x_w = 0.7$.

Bibliography

[1] R.M. Beam and R.F. Warming. An implicit finite-difference algorithm for hyperbolic systems in conservation law form. *J. Comp. Phys.*, 22:87–110, 1976.

[2] Bachalo, W. D. & Johnson, D. A. (1986), Transonic turbulent boundary layer separation generated on an axisymmetric flow model. AIAA J, 24, 437-443.

[3] Durbin, P.A. (1995), "Separated flow computations with the k-(e) -v^2 model", *AIAA J.*, **33**, p. 659.

[4] A. Jameson, W. Schmidt and E. Turkel (1981), "Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes", *AIAA Paper 8l-l259.*

[5] Krist, S., Biedron, R. & Rumsey, C. (1998), CFL3D User's Manual (Version 5.0), NASA/TM-208444.

[6] Kalitzin, G., Kalitzin, N., Wilde, A. (2000), "A Factorization Scheme for RANS Turbulence Models and SNGR Predictions of Trailing Edge Noise", *AIAA-2000-1982*

[7] Kalitzin, G. (1999), "Application of v^2-f turbulence model to transonic flows", *AIAA-99-3780*

[8] Liu, F., Jameson, A., and Jennions, I.K. (1998), "Computation of Turbomachinery Flow by a Convective-Upwind-Split-Pressure (CUSP) Scheme", *AIAA Paper 98-0969*

[9] C.Y. McNeil. Turbulence model implementation and validation. In *Center for Integrated Turbulence Simulations 1999 Annual Technical Report*, pages 13–18. Stanford University, September 1999.

[10] Roe, P. L. (1986), "Characteristic Based Schemes for the Euler Equations", A Numerical Review of Fluid Mechanics, pp. 337-365.

[11] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA Paper 92-0439*, 1992.

[12] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. *La Recherche Aérospatiale*, 1(1):5–21, September 1994.

[13] Steinthorsen, E. & Shih, T.I-P. (1993), Methods for Reducing Approximate-Factorization Errors in Two- and Three-Factored Schemes, SIAM J. Sci. Comput, 14-3, pp. 1214-1236.

[14] Wu, X., Jacobs, R.G., Hunt, J.C.R. & Durbin, P.A., 1999, "Simulation of boundary layer transition induced by periodically passing wakes", J. Fluid Mech., **398**, 109-153.

[15] Wu, X. & Durbin, P.A., 2000a, "Numerical simulation of heat transfer in a transitional boundary layer with passing wakes", ASME. J. Heat Transfer **122**, 248–257.

[16] Wu, X. & Durbin, P.A., 2000b, "Boundary layer transition induced by periodic wakes", J. Turbomachinery, *to appear*.

[17] Wu, X. & Durbin, P.A., 2000c, "Evidence of longitudinal vortices evolved from distorted wakes in a turbine passage", J. Fluid Mech., submitted.

[18] "Development and Validation of a Massively Parallel Flow Solver for Turbomachinery," Yao, J., Alonso, J., Jameson, A., and Liu, F., AIAA Paper 00-0882, 2000.

[19] "The Effect of Airfoil Scaling on the Predicted Unsteady Loading on the Blade of a 1 and 1/2 Stage Transonic Turbine and a Comparison with Experimental Results," Clark, J. P., Stetson, G. M., Magge, S. S., Ni, R. H., Haldeman, C. W., and Dunn, M. G., ASME Paper 2000-GT-00446, May, 2000.

[20] "Numerical Solutions of the Euler Equations by Finite Volume Methods with Runge-Kutta Time Stepping Schemes," Jameson, A., Schmidt, W., and Turkel, E., AIAA Paper 81-1259, January, 1981.

[21] "Analysis and Design of Numerical Schemes for Gas Dynamics 1, Artifical Diffusion, Upwind Biasing, Limiters and their Effect on Multigrid Convergence," Jameson, A., International Journal of Computational Fluid Dynamics, Vol. 4, pp. 171-218, 1995.

[22] "Analysis and Design of Numerical Schemes for Gas Dynamics 2, Artificial Diffusion and Discrete Shock Structure," Jameson, A., International Journal of Computational Fluid Dynamics, Vol. 5, pp. 1-38, 1995.

[23] "A Multiple-Grid Scheme for Solving the Euler Equations," Ni, R. H., AIAA Journal, Vol. 20, No. 11, pp. 1565-1571.

[24] "Prediction of 3-D Multistage Turbine Flow Field Using a Multiple-Grid Euler Solver," Ni, R. H. and Bogoian, J. C., AIAA Paper No. 89-0203, 1989.

[25] "Prediction of 3-D Unsteady Flow in Multi-Stage Turbomachinery Using an Implicit Dual Time-Step Approach," Davis, R. L., Shang, T., Buteau, J., and Ni, R. H., AIAA Paper No. 96-2565, 1996.

[26] "Study of Flow-Field Interactions in a Transonic Compressor Using DPIV," AIAA Paper 00–378, 2000.

[27] "A new High Resolution Scheme for Compressible Viscous Flows with Shocks", S. Tatsumi and L. Martinelli and A. Jameson, AIAA 33nd Aerospace Sciences Meeting, Reno, Nevada, January 1995.

[28] "Multigrid Unsteady Navier-Stokes Calculations with Aeroelastic Applications", J. J. Alonso and L. Martinelli and A. Jameson, AIAA Paper 95-0048.

Chapter 3

Combustor

3.1 Large-eddy Simulation of Gas Turbine Combustors

The large eddy simulation (LES) approach is used to simulate the combustor part of the gas turbine engine. LES was chosen because of its demonstrated superiority over RANS in predicting mixing, which is central to combustion. The combustor simulations have two major components - gas phase and sprays. The gas phase part of the project is developing a parallel, unstructured grid LES solver which will then be integrated with the models developed by the spray group.

This section discusses the gas phase part of the project. In the first two years, we developed a conservative numerical algorithm, that staggers the dependent variables to simulate incompressible flow on unstructured grids. At the end of the second year, the algorithm was extended to hybrid elements, the data structures were changed to compressed storage format to reduce memory use, a grid reordering technique was developed, and validation simulations were just being initiated.

Our progress in the last year is as follows:

- A more efficient version of the constant density algorithm was derived for hybrid grids.
- Severe memory bottlenecks in the pre-processor part of the solver were removed. The preprocessor was rewritten to store data out of core; as a result memory requirements of the order of gigabytes were reduced to the order of megabytes.
- The solver was validated for a variety of steady and unsteady flows.
- Simulations in the Pratt & Whitney geometry have been initiated; the unsteady flow in a subset of the overall geometry was simulated; the simulations retained the geometrical complexities of the full configuration.
- The dynamic Smagorinsky model for LES was extended to unstructured grids, and implemented.
- The algorithm was extended to low Mach number, variable density flows; validation is in progress.

3.1.1 Algorithm Improvements

Base algorithm

Recall that the constant density algorithm stores pressure at the centroids of the elements, and velocity at their faces. As shown in figure 3.1, only one component of velocity is stored and advanced

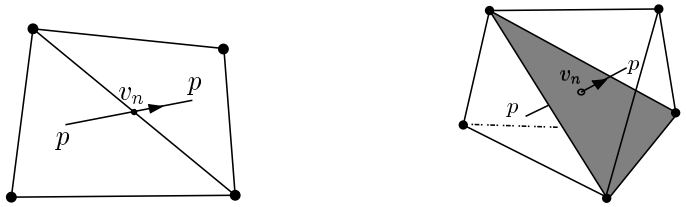


Figure 3.1: Positioning of variables in staggered algorithm.

in time; the other two components are reconstructed. The velocity component v_n satisfies,

$$\frac{\partial v_n}{\partial t} - (\vec{u} \times \vec{\omega}) \cdot \vec{n} + \frac{\partial}{\partial n}\left(\frac{\vec{u} \cdot \vec{u}}{2}\right) = -\frac{1}{\rho}\frac{\partial p}{\partial n} + \nu\left(\nabla^2 \vec{u}\right) \cdot \vec{n}. \tag{3.1}$$

Note that the convection term is written in terms of velocity and vorticity. The pressure - projection approach is used to ensure that the velocity field is discretely divergence-free; the resulting Poisson equation is solved using the conjugate gradient method with diagonal preconditioning. As shown in figure 3.1, v_n is not necessarily aligned with the face normal. As a result, the projection step involves the tangential velocities, which have to be reconstructed at every iteration. A more efficient formulation was therefore derived - the face normal velocity is now stored at the face centroid. Storing the normal component makes the projection step cheaper, while storing velocities at the centroid makes flux calculations more accurate.

The dynamic model

The dynamic Smagorinsky model [1] was extended to unstructured grids. Recall that the LES equations are

$$\frac{\partial \overline{u_i}}{\partial t} + \frac{\partial \overline{u}_i \overline{u}_j}{\partial x_j} = -\frac{\partial \phi}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 \overline{u}_i}{\partial x_j x_j} - \frac{\partial q_{ij}}{\partial x_j} \tag{3.2}$$

where q_{ij} denotes the anisotropic part of the subgrid-scale stress, $\overline{u_i u_j} - \overline{u}_i \overline{u}_j$, and the overbar indicates filtered variables. Note that the above equation can equivalently be written with the convection term in rotational form.

The details of the dynamic modeling procedure may be found elsewhere and are not repeated here. We only note that the Smagorinsky model assumes that

$$q_{ij} = -2C\overline{\Delta}^2 |\overline{S}| \overline{S}_{ij} \tag{3.3}$$

where $\overline{S}_{ij}$ denotes the filtered strain-rate tensor. Application of the dynamic procedure using the least - squares approach [17] yields the following expression for C:

$$C\overline{\Delta}^2 = -\frac{1}{2}\frac{L_{ij} M_{ij}}{M_{kl} M_{kl}} \tag{3.4}$$

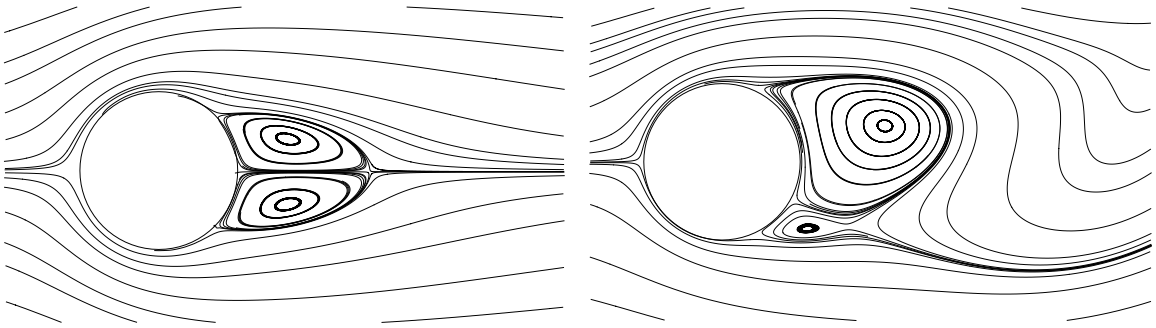
where

$$L_{ij} = \widehat{\overline{u}_i \overline{u}_j} - \widehat{\overline{u}}_i \widehat{\overline{u}}_j \tag{3.5}$$

and

$$M_{ij} = \left(\widehat{\Delta}/\overline{\Delta}\right)^2 \widehat{|\overline{S}| \overline{S}_{ij}} - \widehat{|\overline{S}| \overline{S}_{ij}} \tag{3.6}$$

The dynamic procedure requires definition of a test - filter, and the ratio of test to grid filter widths. The ratio of filter widths is commonly assumed to be 2; we do the same. We define the filter



(a) (b)

Figure 3.2: Streamlines in the immediate vicinity of the cylinder. $(a) : Re = 20, \quad (b) : Re = 100.$

width at the faces of the grid as $V^{1/3}$ where V denotes the average of the volume of the elements that straddle the face. This yields a filter width of $(\Delta_x \Delta_y \Delta_z)^{1/3}$ for a Cartesian grid. The test filter is assumed to be a top - hat filter, and uses information from the neighboring volumes to obtain test-filtered values of the velocity at the faces. In practice, the dynamic model constant C is usually averaged over homogeneous directions, should such directions be present. Other approaches have been proposed for flows without homogeneous directions; e.g. [18],[19]. The approach proposed by Ghosal [18] requires solution of a variational problem, and was successfully applied to flow over a backstep by [13]. The Lagrangian averaging proposed in [19] has been successfully applied to channel flow, but can exhibit sensitivity to the Lagrangian averaging time. In this report, we have chosen to filter the dynamic model coefficient in space instead of Lagrangian averaging. This implementation of the dynamic model is considered preliminary; the rationale for test-filtering the coefficients is that spatial filtering is consistent with the assumption in the dynamic procedure that the model coefficient does not vary over the test filter width.

Variable density flow

The constant density algorithm has been extended to variable density flow in the low Mach number limit. The dependent variable is now ρv_n instead of v_n, and the convection term is rewritten in terms of $\rho\vec{u}$, and its curl. Again, a pressure - projection approach is used to enforce the continuity equation. The energy equation may either be solved for, or temperature may be obtained by mapping from the scalars. Validation is in progress, and is composed of two parts - evolving the scalars for constant density flow, evolving the variable density equations while specifying the density, and combining the two. The first two validations have been completed; e.g., figure 3.6 shows scalar profiles in the region downstream of the splitter plate in a coaxial combustor. Similar computations in other simple laminar flows were performed.

3.1.2 Results

Several computations were carried out to validate the numerical method. Some of these are discussed below.

Flow over a cylinder

The flow over a cylinder has been studied extensively, by both experiments and computations. The flow is sensitive to Reynolds number, has attached and separated regions, and exhibits steady and unsteady regimes. It was therefore used for validation. Four simulations are planned: direct

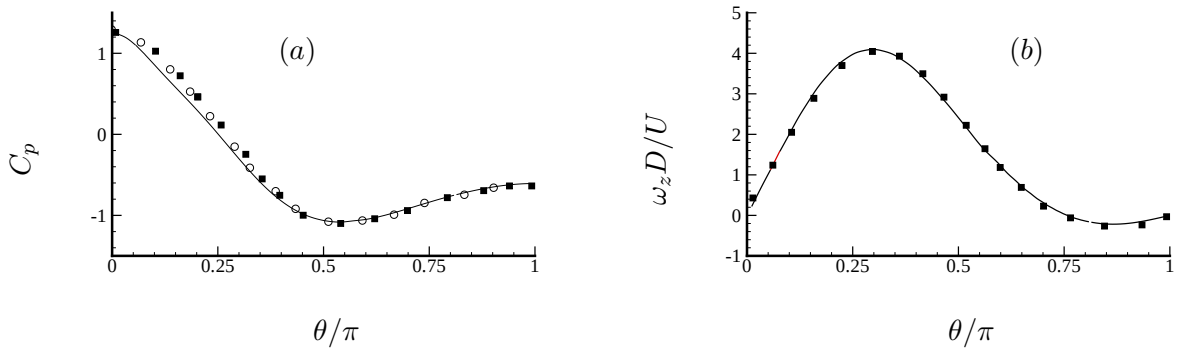


Figure 3.3: The vorticity and pressure coefficient (C_p) on the cylinder surface at $Re = 20$ are compared to experimental data. The solid lines are from the calculations, while the symbols are experimental data from [5]. (a): C_p, (b): vorticity.

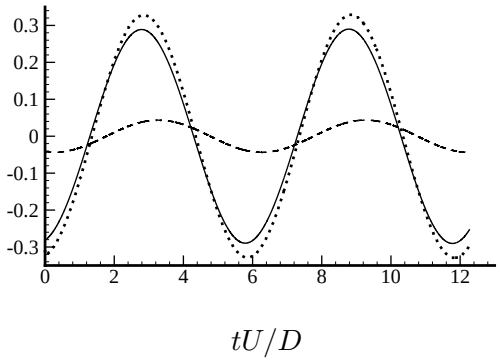


Figure 3.4: Lift coefficients at $Re = 100$. —— (C_{lp}), ---- (C_{lv}), ····· (C_l). The subscripts p and v stand for contributions from pressure and viscosity respectively.

numerical simulation at $Re = 20, Re = 100, Re = 300$, and LES at $Re = 3900$. The first two calculations are completed, and are reported here, while the latter two are in progress. Note that the flow at $Re = 20$ is two-dimensional and steady, that at $Re = 100$ is two - dimensional and unsteady, while those at $Re = 300$ and 3900 are three - dimensional and unsteady. Regardless of the regime and dimensionality of the flow, all simulations reported solve the three - dimensional unsteady equations.

Figure 3.2 shows streamlines in the immediate vicinity of the cylinder at $Re = 20$ and 100. The asymmetry in the presence of shedding is apparent. Quantitative validation is provided in figures 3.3 and 3.4, and tables 3.1 and 3.2, where good agreement with experiment and other computations is found.

Flow in a coaxial combustor

The flow in a coaxial dump combustor geometry has been extensively studied experimentally [3],[4] as well as computationally using LES [2]. Data for both cold and reacting flow are available. It is therefore used for validation purposes. Figure 3.5 shows a cross-section of the geometry. Two cases were considered. Prior to inclusion of the LES model in the code, laminar flow in the same geometry was computed. The objective of the laminar calculations was to validate the code by comparing to computations by Pierce (personal communication) using a structured solver in cylindrical coordinates. The three components of the coaxial geometry: core inlet, annular inlet, and test section were independently considered. The solutions in the inlets are trivial and are not

	Present results	Computations	Experiments
Drag coefficient	2.01	1.99	2.05
Pressure drag	1.18	–	1.22
Viscous drag	0.83	–	0.83
Separation angle	42.5°	43.8°	41.6°
Min. vel. in bubble	$-0.032U$	$-0.031U$	$-0.040U$
Position of min. vel.	$0.42D$	$0.42D$	$0.36D$

Table 3.1: Comparison to experiments ([6],[7],[5], and computation [9] for $Re = 20$.

shown here. Figure 3.6 compares the solution in the test section to results from Pierce, and good agreement is observed.

Currently LES in the same geometry is in progress at conditions corresponding to those of [3].

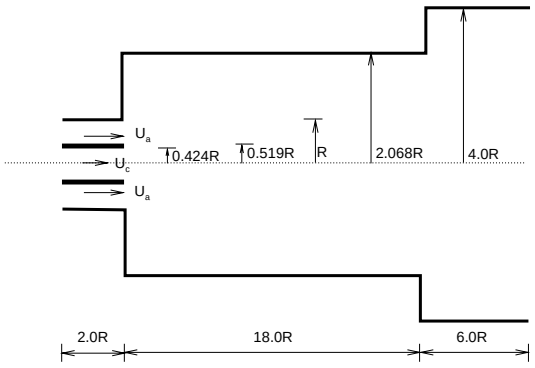


Figure 3.5: Cross-section of coaxial combustor.

Turbulent pipe flow

DNS of the turbulent flow in a pipe at Reynolds number (based on diameter and bulk velocity) of 5300 is in progress. Experimental [10], and computational i [11],[12],[13] are available for comparison. Pipe flow provides an important test of the algorithm's ability to predict wall - bounded flows. The simulations were initialized with random initial conditions, and at the time of this writing, turbulent statistics were not yet converged. An instantaneous snapshot of the flow is provided in figure 3.7. Also shown is a snapshot from ongoing LES of the same flow in figure 3.8.

	Present results	Computations	Experiments
Drag coeff. (max)	1.329	1.314 - 1.365	1.35
Pressure drag (max)	0.989	1.021	1.01
Viscous drag (max)	0.340	0.344	0.34
Lift. coef. (max)	0.328	0.314 - 0.328	–
Lift coef. (amp)	0.684	0.657	–
Pressure lift (max)	0.290	0.297	–
Viscous lift (max)	0.043	0.045	–
Strouhal no.	0.166	0.164 - 0.166	0.164
-Cp_{base}	0.77	0.73 - 0.74	0.72
-$Cp_{\text{stag.}}$	1.08	1.11	–
Sep. angle	116.1°	117.4°	117.0°

Table 3.2: Comparison to experiments [14],[15], and computation [9],[8],citekravchenko for $Re = 100$.

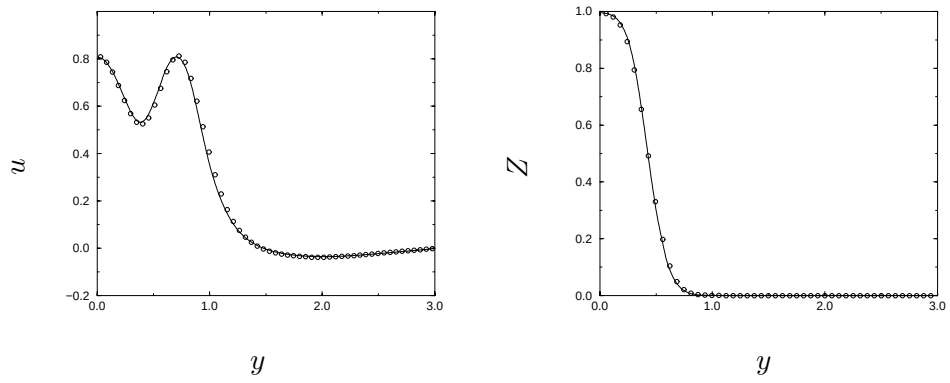


Figure 3.6: Laminar validation in the coaxial combustor geometry. The profiles at $x = 2$ are compared to results from Pierce (private communication) using a structured grid solver. (a): u, (b): Scalar —— (Pierce), ∘ (present results).

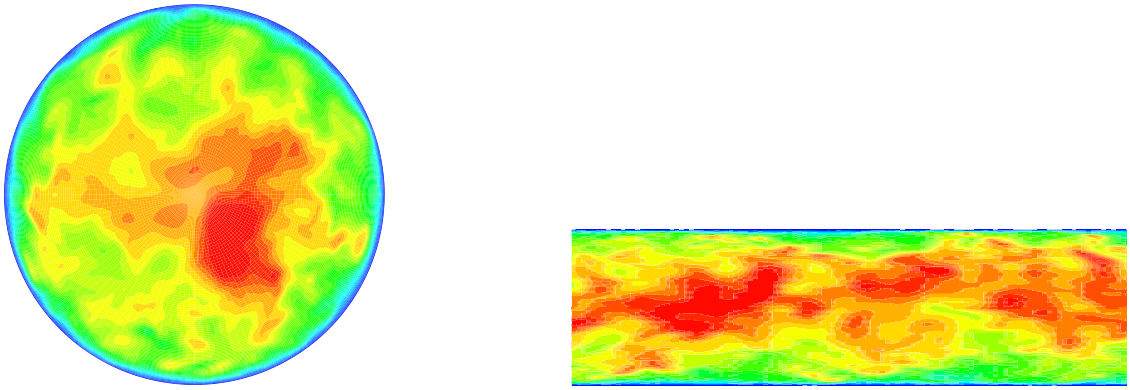


Figure 3.7: Contours of streamwise velocity in direct numerical simulation of flow in a turbulent pipe at $Re = 5300$.

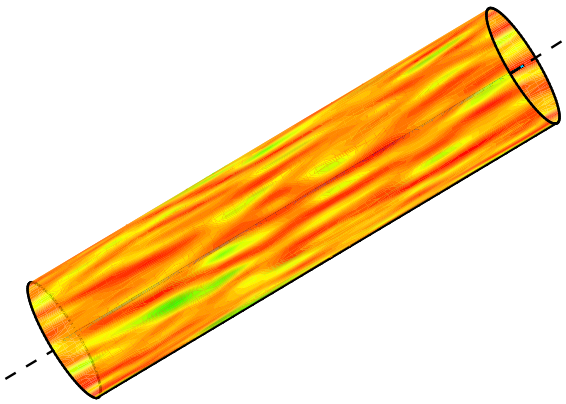


Figure 3.8: Contours of streamwise velocity in the immediate vicinity of the wall in LES of pipe flow at $Re = 5300$. The near-wall streaks are apparent.

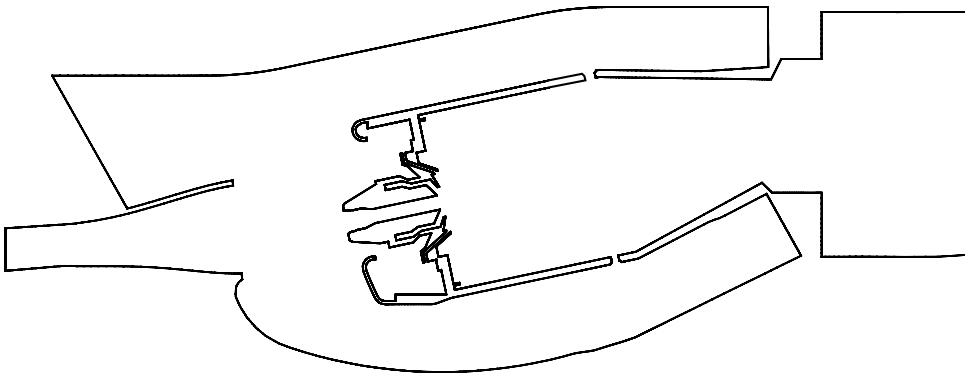


Figure 3.9: Cross-section of combustor

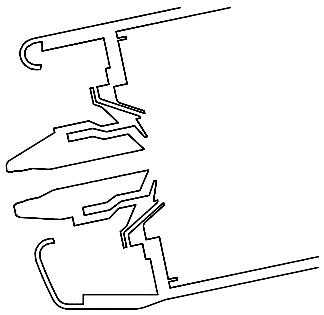


Figure 3.10: A closer view of the nozzle.

Flow in Pratt & Whitney combustor

Our goal is to perform LES in an industrial gas turbine combustor as part of the overall integrated simulation. A step in that direction is to simulate cold flow and then reacting-flow, in an industrial combustor. Pratt & Whitney have provided us with a combustor geometry for which they have data. Simulating flow in the real combustor geometry tests the algorithm, the code, and the grid generating capability.

Our first step has been to assess our ability to handle the intricate internal details of the combustor geometry, and to assess the resolution requirements in the different regions of the flow. Towards that end, we have extracted from the original three dimensional geometrythe midplane containing important features such as the pre-diffuser, outer diffuser, fuel and air nozzle, and injection holes (figures 3.9, 3.10). Ths midplane was extruded in the spanwise direction, and a completely hexahedral grid of about half a million elements was generated. Figure 3.11 illustrates a portion of the grid.

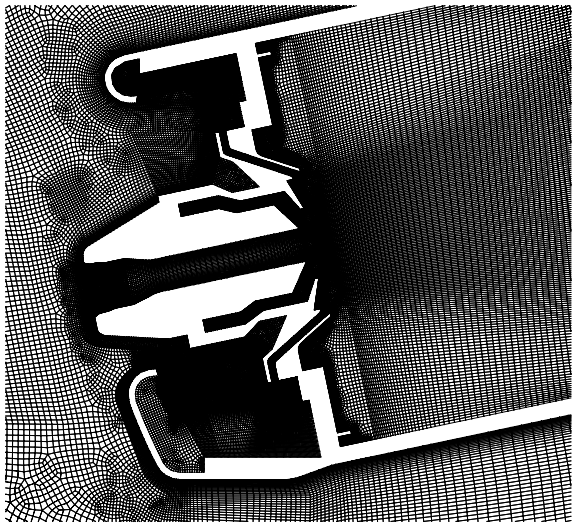


Figure 3.11: A closer view of the grid used in the nozzle

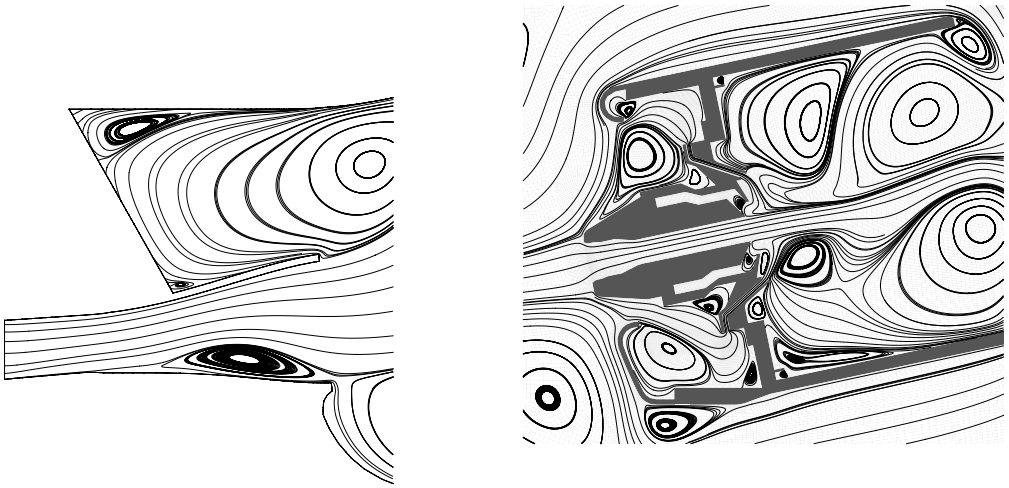


Figure 3.12: Streamlines in the prediffuser and nozzle regions respectively.

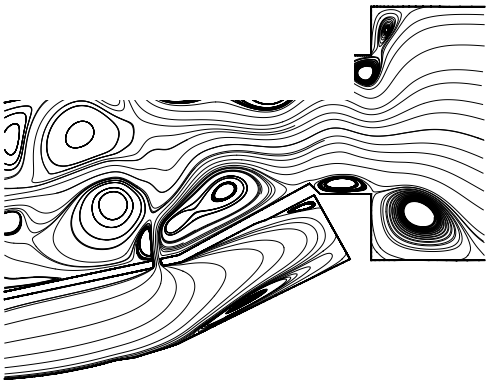


Figure 3.13: Streamlines illustrating flow around the dilution holes

Plug flow at a Reynolds number of around 3000 was specified at the inlet, the calculations were started from rest, and monitored for qualitative accuracy. Figures 3.12, and 3.13 show instantaneous streamlines in the pre-diffuser, nozzle, and dilution hole regions respectively. The computation is seen to capture separation, reattachment and recirculation regions at both large scales (such as in the diffuser and downstream of the nozzle) and small scales (*e.g.* small corners around the nozzle). Also, the evolution of the flow was tracked, and unsteady details such as starting vortices, separation at sharp edges were seen to be captured. Animation shows a flow that appears periodic and driven by periodic shedding in the pre-diffuser. This is consistent with the presence of sharp edges and the absence of three-dimensional features.

Our next step is to generate a suitable grid for the three-dimensional geometry, and then simulate scalar mixing in the full configuration.

Parallel performance

The constant-density solver was tested on the SGI Origin 2000, and found to scale quite well with the number of processors. Figure 3.14 shows results from calculations on two different grid sizes, 64000 and 216000 nodes. Note that these grids are quite small as compared to the grids commonly used for turbulence simulations. The computations with 64000 nodes scales linearly almost up to 32 processors, at which point there are only 2000 nodes per processor. The simulations with 216000 nodes is seen to scale linearly far beyond. An empirical rule of thumb used in our calculations is to partition the grid into as few as 5000 elements per processor if a sufficient number of processors is available.

3.1.3 Summary and Future Plans

Our progress in the last year is as follows:

- A more efficient version of the constant density algorithm was derived for hybrid grids.
- The constant density algorithm was implemented for hybrid grids, and validated for a variety of steady and unsteady flows.

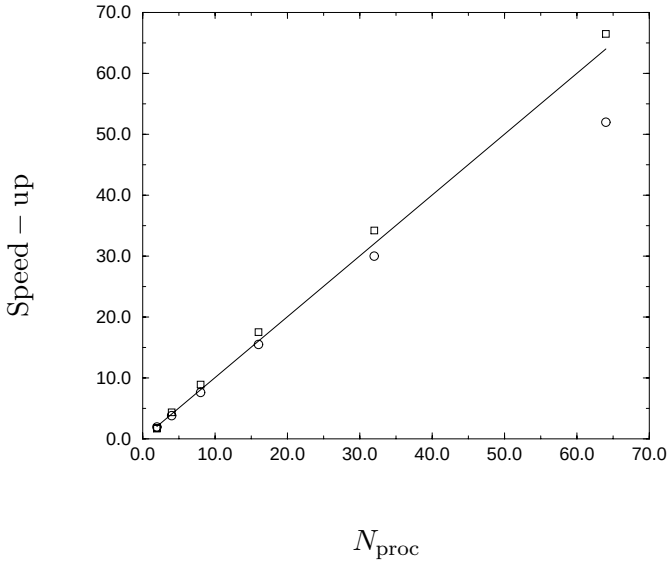


Figure 3.14: Results of a scaling study on the Origin 2000. Two different grid sizes were considered: 64000 nodes (○), and 216000 nodes (□).

- Simulations in the Pratt & Whitney geometry have been initiated; the unsteady flow in a subset of the overall geometry was simulated.
- The dynamic Smagorinsky model was extended to unstructured grids, and implemented in the code.
- The algorithm was extended to low Mach number, variable density flows; validation is in progress.

Our plans for the next year are as follows:

- Simulate cold flow in the complex combustor geometry.
- Implement unsteady flamelet model; complete validation in the presence of heat release.
- Initiate integration with the compressor and turbine simulations by TFLO.
- Initiate inclusion of spray models.

3.2 Combustion Models for Large-Eddy Simulations

In this section we present two different approaches for modeling combustion in Large-Eddy Simulations: the Lagrangian Flamelet Model and the Flamelet/Progress-Variable approach. Both methods are applied in LES and the results are compared to experimental data showing significant improvement over preliminary results presented in the previous annual report [20]. Further extensions of the models are needed to incorporate an accurate description of lifted flames, local extinction phenomena, premixed and partially premixed combustion. Accordingly, in addition to the aforementioned methods, a level-set method for premixed combustion, also known as the G-equation model, has been formulated for LES and has been applied to a turbulent Bunsen flame. Also some theoretical studies have been performed investigating local extinction and reignition processes in non-premixed turbulent combustion.

3.2.1 Lagrangian Flamelet Model

The Lagrangian Flamelet Model [21, 22] has been applied successfully in RANS calculations of turbulent jet diffusion flames [23, 24]. The model follows the conserved scalar approach with a presumed pdf of the mixture fraction and uses the flamelet ideas of Peters [25, 26]. The filtered values of all reactive scalars ϕ_i are given by an expression of the form

$$\widetilde{\phi}_i = \int_0^1 \phi_i(Z) P(Z) dZ \,. \tag{3.7}$$

The pdf $P(Z)$ of the mixture fraction Z is presumed to be a β-function. The functional dependence of the reactive scalars on the mixture fraction required for the integration of Eq. (3.7) is determined by the solution of the unsteady flamelet equations.

The flamelet equation for the species mass fractions Y_i can be written as

$$\frac{\partial Y_i}{\partial \tau} - \frac{\chi}{2}\frac{\partial^2 Y_i}{\partial Z^2} - \dot{w}_i = 0 \,, \tag{3.8}$$

where τ is a Lagrangian flamelet time, $\dot{w}_i$ is the chemical source term, and χ the scalar dissipation rate.

In Eq. (3.8) the Lewis numbers of all chemical species have been assumed to be unity. This assumption is discussed in Pitsch [24] and seems to be valid for the present configuration.

The consideration of the time dependence certainly adds some complexity to the problem, but it has been shown that it is essential to consider the unsteadiness, if for instance radiation cannot be neglected or NO formation is considered [23].

Together with the corresponding energy equation, Eq. (3.8) can readily be solved, provided the scalar dissipation rate is known as a function of the mixture fraction and the flamelet time τ can be related to the physical space coordinates. Then, the solution can be used to provide the remaining unknown quantities such as density and temperature.

The basic idea of the model is that flamelets are introduced at the inflow boundary and move downstream, essentially by convective transport. For a downstream jet flow, we conditionally average the scalars over planes downstream from the nozzle, and hence these averages depend only on the downstream distance. Since flamelets are associated with the reaction zone, which is located in the vicinity of the stoichiometric mixture, the Lagrangian flamelet time can be related to the downstream distance x by

$$\tau = \int_0^x \left\langle \widetilde{u} | \widetilde{Z}_{\mathrm{st}} \right\rangle^{-1} dx' \,. \tag{3.9}$$

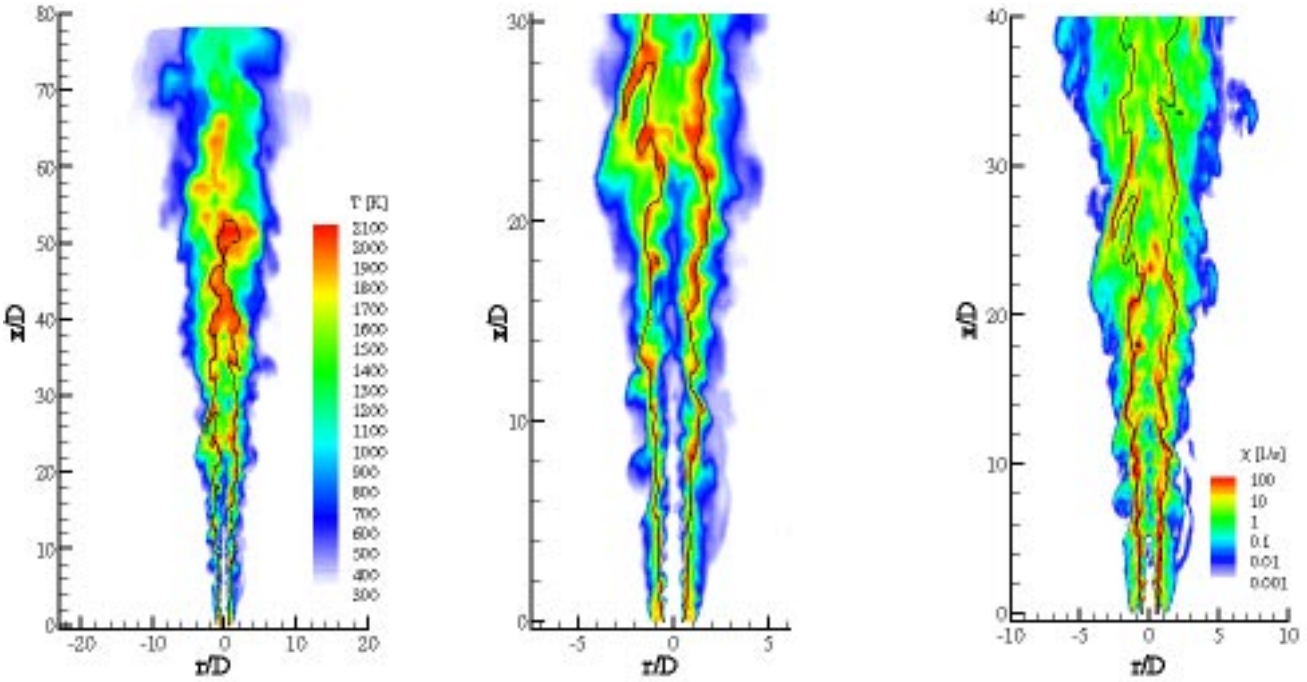


Figure 3.15: Lagrangian Flamelet Model: Instantaneous temperature field for entire jet (Left), zoom in near field region(Center). Scalar dissipation rate distribution (Right). Black line represents the contour of stoichiometric mixture fraction

For the evaluation of the conditional mean scalar dissipation rate, which is needed to solve the flamelet equations, a newly developed model described in Pitsch *et al.* [24] has been used. This model can account for local phenomena such as flame stabilization by a pilot flame.

The configuration used for validation of the Lagrangian Flamelet Model is a piloted methane/air jet diffusion flame (Sandia Flame D) [27, 28]. The fuel is a 25 % methane, 75 % air mixture, which has been diluted with air in order to minimize the formation of polycyclic aromatic hydrocarbons and soot. The fuel nozzle is enclosed by a broad, slightly lean pilot flame and coflowing air. The Reynolds number based on the fuel stream is Re = 22400. The flame has been experimentally investigated by Barlow and Frank [27, 28], who performed Rayleigh measurements of temperature, and Raman and LIF measurements to obtain mass fractions of chemical species and the mixture fraction.

An instantaneous temperature and scalar dissipation rate distribution from the simulation is given in Fig. 3.15. It can be observed that regions of very broad high temperature zones around the stoichiometric contour are alternating with regions of very narrow high temperature distribution. Similarly, regions of high scalar dissipation rate occur in layer like structures, which align with the reaction zone and always appear in a certain angle to the jet axis. These observations have also been made in experiments in cold jets by Feikema *et al.* [29].

Figure 3.16(a) shows that the mean mixture fraction is well predicted by the simulation along the centerline. The mixture fraction rms (including both resolved and sub-grid contributions) is also given in Fig. 3.16(a). The experiment is slightly underpredicted in the far field of the jet. The underprediction for $x/D < 10$ might be attributed to experimental uncertainties, since the rms must go to zero close to the nozzle. This conclusion is supported by the temperature rms in Fig. 3.16(b). Figure 3.16(a) also shows the subgrid-scale mixture fraction rms, which is shown to be much smaller than the total rms, indicating that the major part of the turbulent mixture-fraction fluctuations is resolved by the LES.

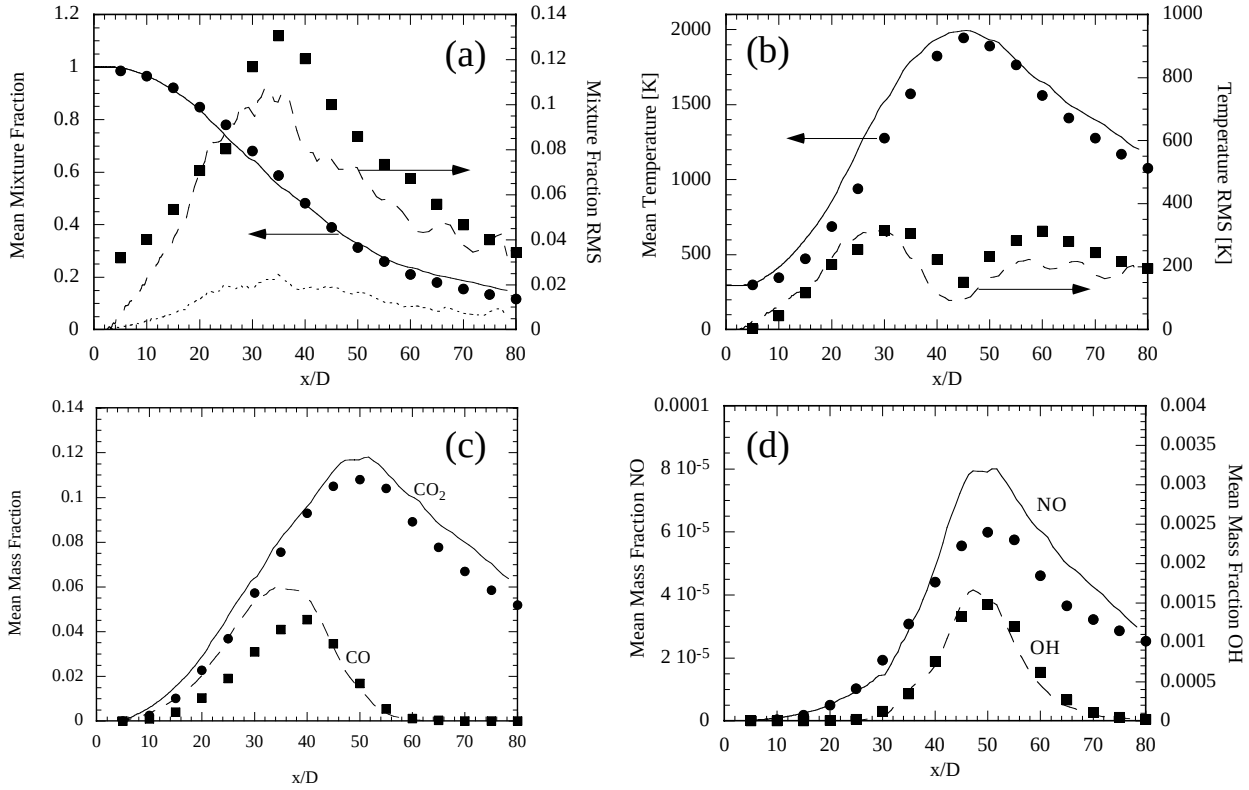


Figure 3.16: Lagrangian Flamelet Model: Comparison of calculated and experimental data along the centerline; (a) mean, total RMS, and subgrid-scale RMS (dotted line) of mixture fraction; b) mean and RMS of temperature; c) mean CO and CO_2 mass fractions; d) mean NO and OH mass fractions (Lines: Calculation, Symbols: Experiments)

The calculated mean and rms temperature along the centerline are shown in Fig. 3.16(b). Both agree well with the experimental data. Even the decrease in the temperature rms in the vicinity of the peak mean temperature is well represented by the simulation. The slight overprediction in mean temperature up to $x/D = 40$ has been found to be caused by a partially premixed heat release. The effect is also apparent in the experiments, but occurs in the calculations much closer to the nozzle.

The figures Fig. 3.16 show centerline profiles for species mass fractions of CO_2 and CO, NO and OH. In general all profiles agree well with the experimental data. Small discrepancies beyond $x/D > 60$ can mainly be attributed to the overprediction of the mixture fraction. In the rich part of the flame ($x/D < 40$), CO_2 and CO are overpredicted. This can be attributed to the partial premixing of the fuel with air, which causes substantial chemical reactions to occur in the rich premixed part of the jet. The OH radical mass fraction shown in Fig. 3.16(d) is predicted very accurately, and the agreement of predicted and measured NO mass fractions is very good. The analysis of the formation of nitric oxide shows that only approximately one-third of the total NO is formed by the thermal path, and that the N_2O path contributes approximately 10 %. The formation of NO in this case is therefore dominated by the prompt path.

More detailed information about modeling issues, simulation, and discussion of the results can be found in Pitsch and Steiner [21, 22].

3.2.2 Prediction of a Lifted Flame Using the Flamelet/Progress-Variable Approach

This approach, developed by Pierce [30], is based on "quasi-steady" flamelets [25], in which the local flame state undergoes unsteady evolution through a sequence of stationary solutions to the flamelet equations. A detailed description of the method is given in Moin *et al.* [31].

In the Flamelet/Progress-Variable approach, scalar transport equations are solved for both mixture fraction and a chemical progress variable. The progress variable is a non-conserved physical scalar, such as product mass fraction, that is chosen as an indicator of the overall flame state. All other chemical variables in the system are related to the mixture fraction and progress variable scalars through a flamelet library. This differs from traditional steady flamelet approaches in which chemical variables are related to mixture fraction and its dissipation rate. The addition of a progress variable gives the flamelet model a more unsteady, dynamic response.

A flamelet library is first constructed for the given combustor conditions by looking for stationary solutions to the one-dimensional reaction-diffusion equations. The unstable lower solution branch is included so that the complete range of flame states, from completely extinguished (mixing without reaction) to completely reacted (equilibrium chemistry), is represented in the library. Arbitrarily complex chemical kinetic mechanisms, as well as differential-diffusion effects, can be included. The result is a complete set of flame states, given in terms of mixture fraction and a single flamelet parameter related to the imposed strain rate.

One of the combustion variables, or some combination of the variables, is chosen to serve as the progress variable. This would be a quantity that is representative of the overall gross flame behavior and that varies monotonically with the flame state so that its value uniquely determines it. Transport equations are solved for both mixture fraction and the progress variable. The progress variable equation contains a reaction source term that must be closed. This is accomplished by assuming that each subgrid state is represented by a single flamelet with mixture-fraction fluctuations given by the β-distribution [32, 33] with the subgrid variance obtained dynamically using the method of Pierce and Moin [34]. This is a reasonable assumption in the absence of further information about the subgrid state.

The final step relates the flamelet state to the filtered value of the progress variable, which is known from its transport equation. If the progress variable is defined such that it is a monotonic function of the flamelet parameter, then the latter can be eliminated from the problem. The result is a closed specification of the chemical system in terms of two variables: the mixture fraction and a single progress variable. The main advantage of this new approach over classical steady flamelets (*e.g.* Cook *et al.* [35]) is that it allows for better representation of ignition and extinction phenomena including flame lift-off, flowfield history, and large scale unsteadiness.

The Flamelet/Progress-Variable approach has been used for large eddy simulation of methane-air combustion in a coaxial jet combustor (Fig. 3.17). The configuration used corresponds to the experiment of Owen et al. [36]. An instantaneous snapshot of the mixture fraction and product mass fraction fields is presented in Fig. 3.18. The simulation correctly predicts a highly unsteady, asymmetric lifted flame as was observed in the experiment. At the particular instant shown, the flame in the upper part of the flow is lifted farther down stream than the rest of the flame due to higher fuel velocity and concentration emanating from the upper part of the fuel jet. Earlier calculations with fast chemistry [32] or the steady flamelet approach [35] were unsuccessful in capturing the lifted flame phenomenon. Predictions of radial profiles of mean mixture fraction and product mass fraction ($Y_P = Y_{CO2} + Y_{H2O}$) and their comparison with experimental data are shown in Fig. 3.19.

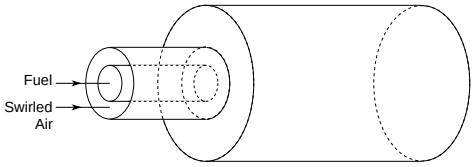


Figure 3.17: Schematic of a coaxial jet combustor.

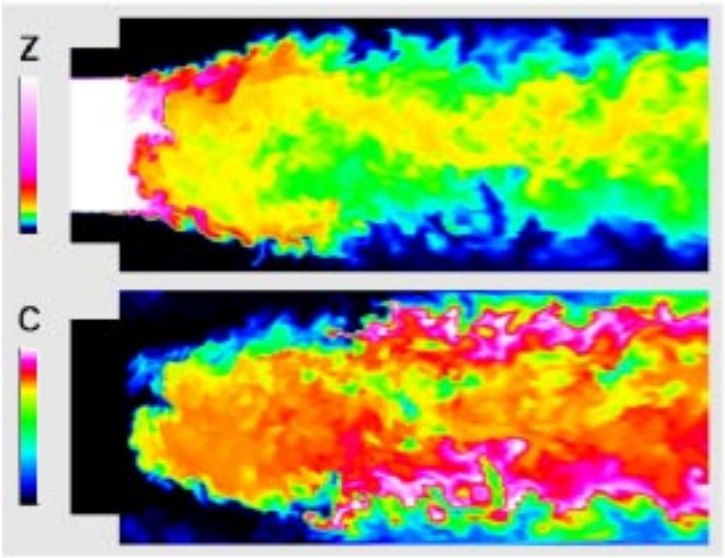


Figure 3.18: Flamelet/Progress-Variable approach: Instantaneous snapshot of mixture fraction (top) and product mass fraction (bottom).

3.2.3 Investigation of Scalar Dissipation Rate Fluctuations in Nonpremixed Turbulent Combustion

The scalar dissipation rate is the most important parameter appearing in the modeling of nonpremixed turbulent combustion. For infinitely fast chemistry, it can be shown that the scalar dissipation rate is directly proportional to reaction rate. The influence of fluctuations of the scalar dissipation rate on the chemical flame structure is neglected in most modeling approaches, even though these are known to be of great importance at conditions near ignition and extinction. In order to study these phenomena the influence of stochastic fluctuations of the scalar dissipation rate on the solution of the flamelet equations has been investigated. Assuming a one-step irreversible reaction the system can be described by the solution of the temperature equation. By modeling the diffusion term in the flamelet equation this can be written as an ordinary stochastic differential equation (SDE). In addition an SDE is derived for the scalar dissipation rate. From these two equations, a Fokker-Planck equation can be obtained describing the joint probability of temperature and the scalar dissipation rate. The equation is analyzed and numerically integrated using a fourth order Runge-Kutta scheme. The influence of the main parameters, which are the Damköhler number, the Zeldovich number, the heat release parameter, and the variance of the scalar dissipation rate fluctuations are discussed. A detailed description of the derivation of the equations and the results of the study can be found in Pitsch and Fedotov [37]

3.2.4 Flamelet Modeling of Local Extinction-Reignition

Direct numerical simulations (DNS) have been performed to investigate the applicability of local, instantaneous flamelet modeling in the presence of local extinction and reignition [38]. A spatially homogeneous, temporally decaying velocity field is considered. The conserved scalar is initialized in blobs, the initial field of the reacting scalars obeys the quasi-steady flamelet solution. Due to strain the scalar dissipation rate first increases leading to local extinction events and decreases later when molecular mixing overcomes the straining. The decrease of the scalar dissipation rate leads to the gradual reignition of the individual flamelets. To study the process of local extinction-

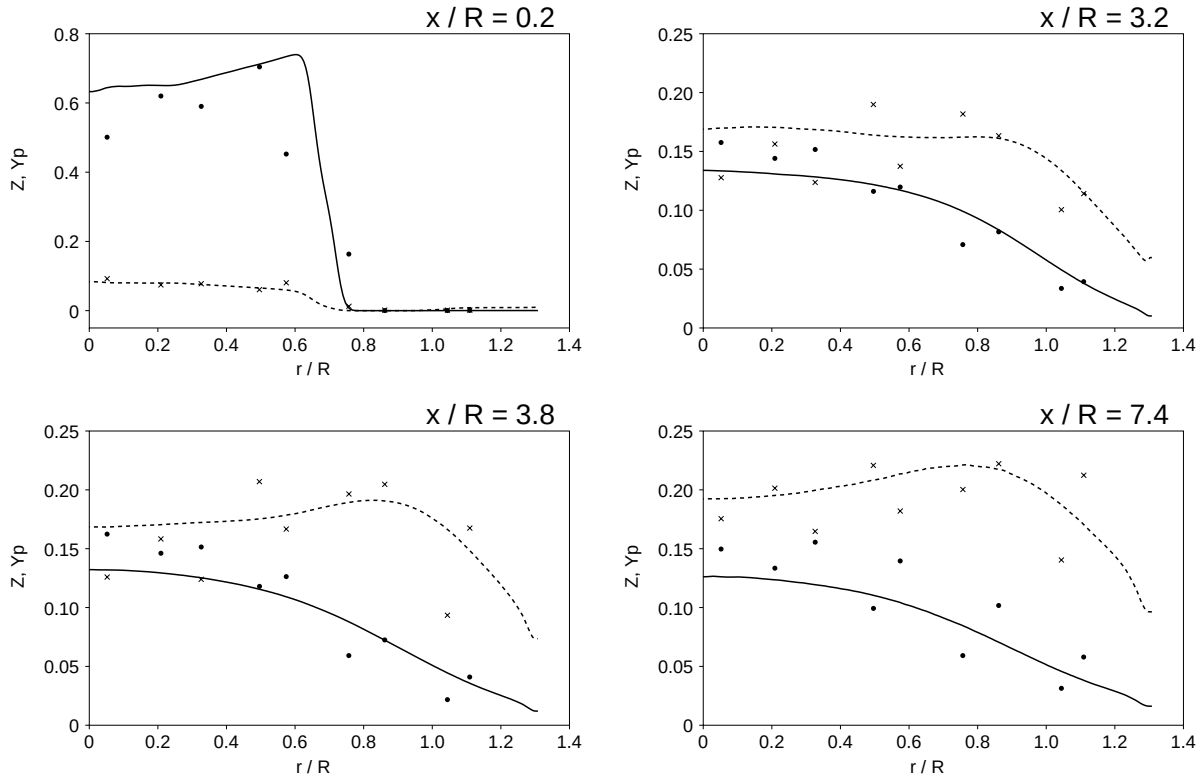


Figure 3.19: Flamelet/Progress-Variable approach: Radial profiles of time-average mixture fraction (—) and product mass fraction (- - -) in the coaxial jet combustor and comparison with experimental data [36] (symbols) at several axial locations, x/R.

reignition the time evolution of the temperature and the scalar dissipation rate at selected points on the stoichiometric surface has been tracked and compared with the data from flamelet predictions. There is a wide category of cases when the reignition process cannot be predicted without accounting for the interaction between the individual flamelets.

3.2.5 Large-Eddy Simulations of Premixed Turbulent Combustion

In premixed combustion, fuel and oxidizer are initially premixed and a flame front, driven by diffusion and heat release from chemical reactions, propagates through this mixture. The physical processes here are very different from those of diffusion flames, where fuel and oxidizer are initially unmixed and the diffusion of fuel and oxidizer into the reaction zone is of critical importance. Therefore different modeling strategies are needed. In the case of partially premixed combustion, models for non-premixed and premixed combustion will be combined.

The G-equation has been introduced by Williams [39] as

$$\frac{\partial G}{\partial t} + \boldsymbol{v} \cdot \nabla G = s_L |\nabla G|, \tag{3.10}$$

where t is the time, $\boldsymbol{v}$ the velocity vector, and s_L is the laminar burning velocity. G is a non-reactive scalar describing the propagation of a premixed flame. The instantaneous location of the flame is given by an iso-scalar surface $G = G_0$, which divides the flow field into burnt and unburnt regions. The laminar burning velocity S_L may be modified from that of an unstreched premixed flame by

the influence of stretch. A modified expression for small curvature and strain has for instance been given by Pelce and Clavin [40] as

$$s_L = s_L^0 - s_L^0 \mathcal{L}\kappa - \mathcal{L}S\,, \tag{3.11}$$

where s_L^0 is the burning velocity of an unstreched premixed flame, κ is the curvature, S is the strain rate, and $\mathcal{L}$ is the Markstein length, which can be determined following for instance Clavin and Williams [41, 42].

Since the laminar burning velocity s_L is a property of the flame, Eq. (3.10) is valid for $G = G_0$ only. Away from the front, the G-field has to be defined differently. A common approach is to use the constraint

$$|\nabla G| = 1, \tag{3.12}$$

which makes the G-field a function describing the distance from the flame.

Peters [43, 44, 45] has derived modeled equations for the Reynolds averaged mean and the variance of G for the corrugated flamelet [43] and the thin reaction zones regime [44]. He also provided expressions for the turbulent burning velocity, which appears as the major unclosed term in the equations.

In the derivation of the filtered G-equation we start with an expression for the sub-grid pdf of G

$$P(G^*, \boldsymbol{x}, t) = \int_{\mathcal{V}} F\left(\boldsymbol{x} - \boldsymbol{x}'\right) \delta(G(\boldsymbol{x}', t) - G^*) d\boldsymbol{x}'\,, \tag{3.13}$$

where F is the spatial filter function. From this, the mean value of G can be defined as

$$\overline{G}(\boldsymbol{x}, t) = \int_{-\infty}^{\infty} G^* P(G^*, \boldsymbol{x}, t) dG^*\,. \tag{3.14}$$

Spatial filtering of Eq. (3.10) leads, with Eqs. (3.13) and (3.14) and similar arguments as in Peters [45], to

$$\rho \frac{\partial \widetilde{G}}{\partial t} + \overline{\rho}\widetilde{\boldsymbol{v}} \cdot \nabla \widetilde{G} = (\overline{\rho} s_T^0) |\nabla \widetilde{G}| - \overline{\rho} D_t \widetilde{\kappa} |\nabla \widetilde{G}|, \tag{3.15}$$

where D_t is the sub-grid diffusivity and the turbulent burning velocity s_T^0, introduced as

$$s_T^0 = s_L^0 \frac{\widetilde{|\nabla G|}}{|\nabla \widetilde{G}|}\,, \tag{3.16}$$

can be modeled by

$$\frac{s_T^0 - s_L^0}{s_L^0} = -\frac{a_4 b_3^2}{2b_1}\frac{\Delta}{l_F} + \left[\left(\frac{a_4 b_3^2}{2b_1}\frac{\Delta}{l_F}\right)^2 + a_4 b_3^2 \frac{v'\Delta}{s_L^0 l_F}\right]^{1/2}. \tag{3.17}$$

Here, Δ is the filter size, l_F is the laminar flame thickness, and a_4, b_1, and b_3 are constants given in Peters [45] for the Reynolds averaged model. For LES a_4 can be evaluated to be $a_4 = 1.37$ [21]. The other constants might differ for LES and have to be determined by DNS.

Intantaneous temperature and G-field distributions of a preliminary LES of a turbulent premixed Bunsen flame are shown in Fig. 3.20. Results of these simulations will be compared to experimental data in the future.

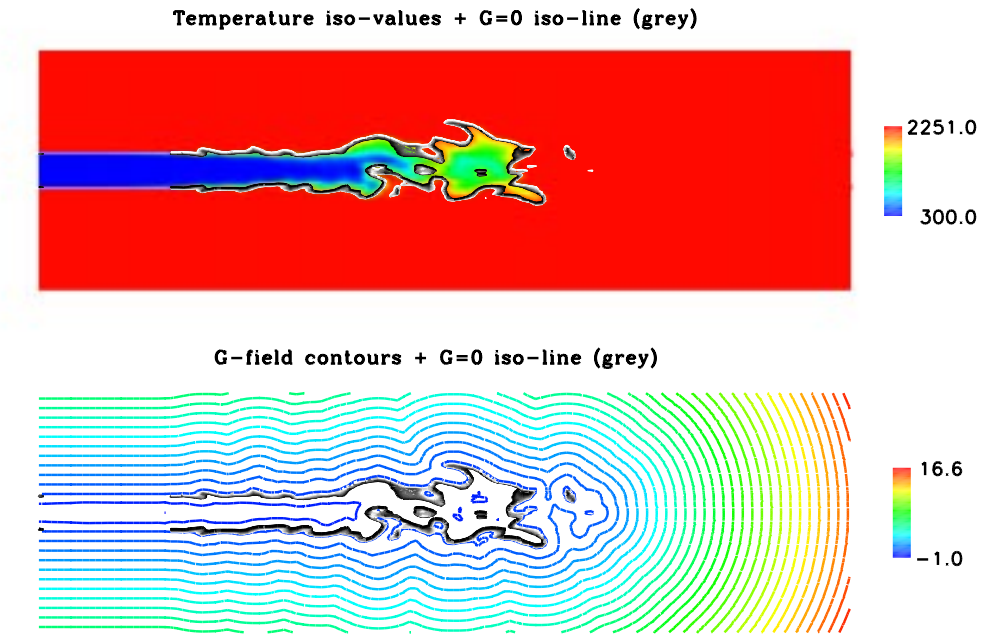


Figure 3.20: Instantaneous G and temperature fields of a turbulent premixed Bunsen flame

3.3 Spray Simulation and Modeling

The objective of the spray simulation effort is to develop the technologies required to perform a highly-accurate large eddy simulation (LES) of reacting multiphase phenomena typically encountered in gas-turbine combustors. This effort involves 1) the implementation of a consistent theoretical framework, 2) the development of appropriate massively parallel software, and 3) the development of improved constitutive subgrid-scale models. The approach involves a hierarchy of studies at realistic conditions using relevant experiments for validation purposes and direct numerical simulation (DNS) to focus on the key modeling issues. The goal is to develop and validate appropriate Lagrangian particle models and provide a state of the art numerical framework which will facilitate the integration of these models into the unstructured combustor code currently under development.

In addition to LES, two core DNS efforts are currently being pursued to support the model development effort. The first involves using Chimera grids to characterize the anisotropic turbulence phenomena produced by drops and heterogeneous drop clusters. The second involves using a finite-element Arbitrary-Lagrangian-Eulerian approach to characterize the effects of drop deformation processes. The goal is to improve subgrid-scale drop models used in the LES formulation at conditions where current models fail by direct treatment of the observed multiple-time, multiple-length scale anomalies which are prevalent at typical engine operating conditions. The detailed DNS data, coupled with comparisons of LES results with experiments, is currently being used to establish improved constitutive models at the relevant conditions. Details relevant to the LES and DNS are outlined in subsequent sections.

3.3.1 Established Milestones

Progress from the onset of the project through FY99 are systematically outlined in the CITS 1998 [46] and 1999 [47] Annual Technical Reports. The culmination of milestones achieved over this period have facilitated 1) the development of "generalized" massively parallel Lagrangian particle modules which will ultimately be integrated into the unstructured combustor code, 2) the integration and testing of these modules in a structured solver with similar numerics to those being used for the unstructured solver, and 3) a clear quantitative description of what the relevant issues are with regard to the spray modeling effort. In particular, the validity of existing models for drop vaporization rate and drag in the dilute spray regime has been established along with the computational feasibility of obtaining high-fidelity LES solutions with well posed, non-ambiguous boundary conditions and model constants. The capability to handle large-scale massively parallel problems in the Lagrangian frame has also been established.

3.3.2 Current Status

Efforts in FY00 have focused on advancing the treatment of multiphase physics to the next level of complexity. Emphasis is currently being placed on the development and validation of constitutive subgrid-scale models applicable in the hollow-cone "dense" and "intermediate" spray regimes exhibited by gas-turbine fuel injectors. A photograph which illustrates these regimes is given in Fig. 3.21 with an outline of the key processes which occur in these regimes. Given the systematic set of results established by the end of FY99, the focus and goals of current interest are:

1. Begin to systematically validate a unified framework in fully-coupled, high-fidelity LES's by comparing results with companion experiments.

2. Obtain correlations for drop vaporization, heat transfer, and drag applicable over a range of pressures from atmospheric to supercritical.

1. "Dense" regime:
   - Liquid injection.
   - Sheet, filament, lattice formation.
   - Primary breakup.
2. "Intermediate" regime:
   - Secondary breakup.
   - Particle deformation, coalescence.
3. "Dilute" regime:
   - Vaporization.
   - Mixing.
   - Combustion.

Figure 3.21: Regimes and key processes of interest.

3. Investigate issues related to high mass-loading interphase (two-way) coupling and obtain a validated scheme for the LES methodology.
4. Combine the interphase coupling model with an appropriate characterization of secondary breakup mechanisms.
5. Derive a primary breakup model which is consistent with the Lagrangian tracking method and maintains film integrity near the injector orifice.

Each of these tasks are leading dissertation level research topics being pursued in various academic, industrial and government laboratories. In FY00 the Stanford effort has focused on each individually. Task 1 represents the key unifying step, which will facilitate the development of the next generation, fully-integrated simulation capabilities. Tasks 2–5 represent the key modeling issues. Progress associated with these tasks is outlined below.

3.3.3 Conditions of Interest

Research throughout FY00 has focused on liquid-hexane–air systems over a range of temperatures from 300 to 2500 K and pressures from 1 to 100 atm. This fuel–air combination was selected to match that used in the Stanford Thermosciences Division (TSD) liquid-fueled combustor [48]. Collaboration with this project is currently serving as the basis for Task 1. Further details regarding the coordinated experimental and numerical effort are given in Section 3.3.5. The range of temperatures selected is consistent with the combustion processes encountered in both the TSD liquid-fueled combustor and in gas-turbines in general. The range of pressures selected is a generalization from the experiment, which will be initially conducted at a nominal pressure of 1 atm, and eventually at 30 atm. This range of pressures is consistent with the current and projected operating envelopes of all contemporary and future generation combustors.

Figure 3.22 shows contours of the density of hexane over the relevant ranges of temperature and pressure. This plot shows the key thermodynamic regimes which are currently being considered as

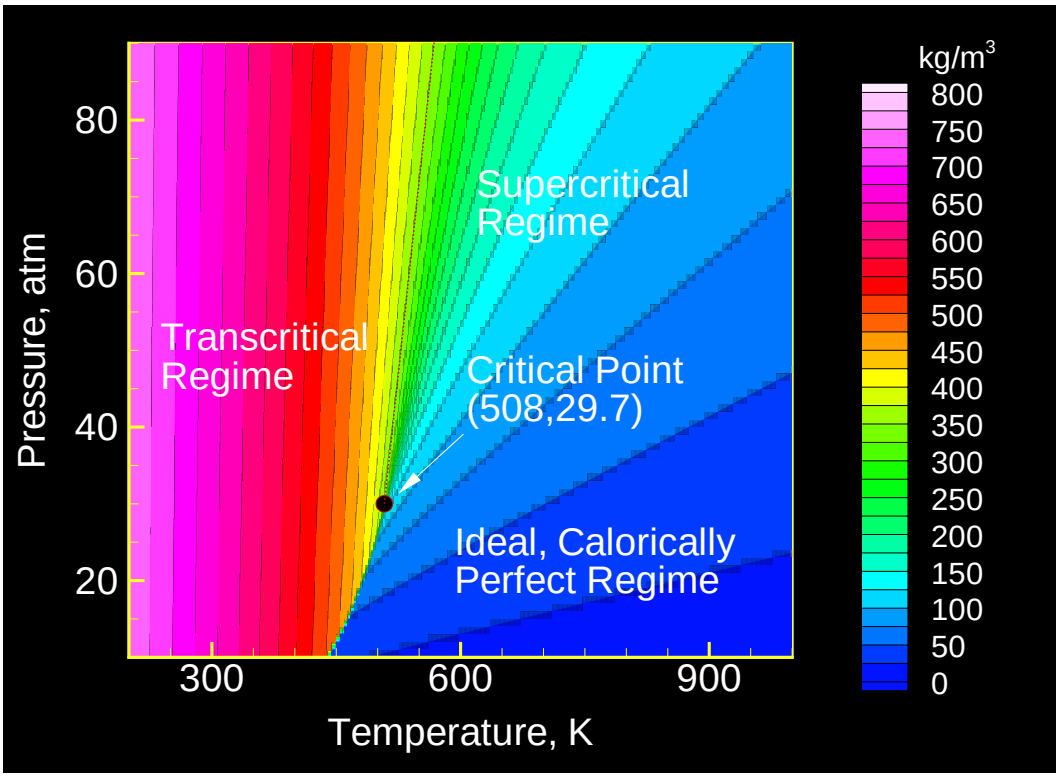


Figure 3.22: Density of hexane as a function of pressure and temperature with the key thermodynamic regimes of interest highlighted.

part of the modeling effort. In particular, Task 2 in Section 3.3.2 was addressed as part of the CTR Eighth Biennial Summer Research Program held at Stanford on July 2–28, 2000. Here the first step towards the development of a unified high-pressure drop model for drop vaporization, heat transfer, and drag was established [49]. Since this work is being summarized as part of the CTR Summer Program Proceedings, the details will not be repeated here. As part of the work done in preparation for this project, however, it was possible to determine the relevant deformation and breakup regimes over the full range of thermophysical conditions and gas liquid interface anomalies to be encountered. These results are plotted in Fig. 3.23.

There have been numerous studies of reacting and non-reacting sprays emphasizing the dilute region in the far field away from the atomizing fuel jet. In this region, observations and modeling issues are relatively tractable and many features of sprays are well understood. The dense spray region in the vicinity of the conical liquid sheet depicted in Fig. 3.21, however, is less understood. One particularly relevant set of phenomena is that associated with secondary breakup. Secondary breakup processes are often the rate controlling mechanism in atomizing sprays, much the same way that drop vaporization is a rate controlling mechanism in the dilute regime.

Existing observations of secondary breakup at conditions typically encountered in gas turbine engines have been shown to be largely functions of the Weber number *(We)* and the Ohnesorge number *(Oh)*. These dimensionless quantities characterize the ratios of drag and liquid viscous forces, respectively, relative to surface tension forces acting on a given drop in a particular thermophysical state and ambient environment. The precise relationship between the Weber number and the Ohnesorge number is the subject of ongoing research. Hsiang and Faeth [50] (among others) have shown the many forms of deformation and breakup which can occur. The typical regimes are shown in Fig. 3.24.

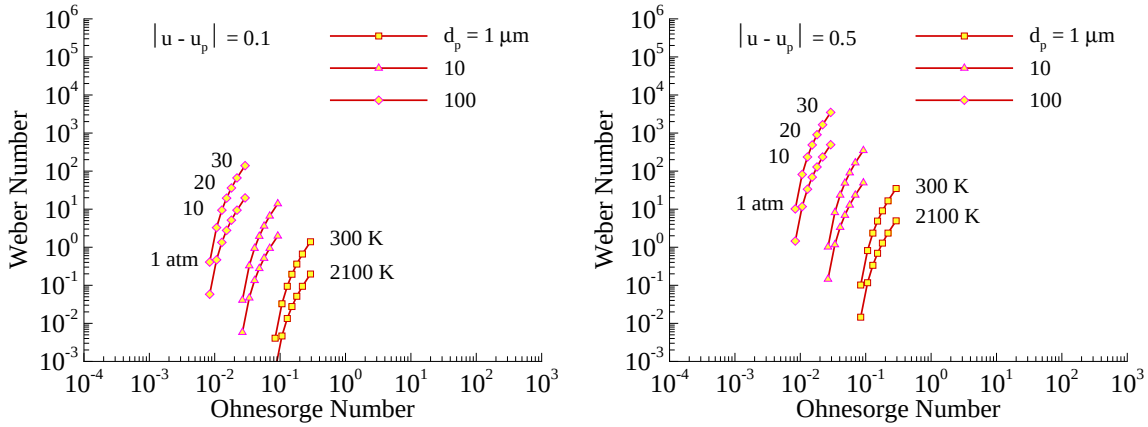


Figure 3.23: Drop deformation and breakup characteristics of hexane–air systems for dimensionless slip velocities of 0.1 and 0.5, respectively.

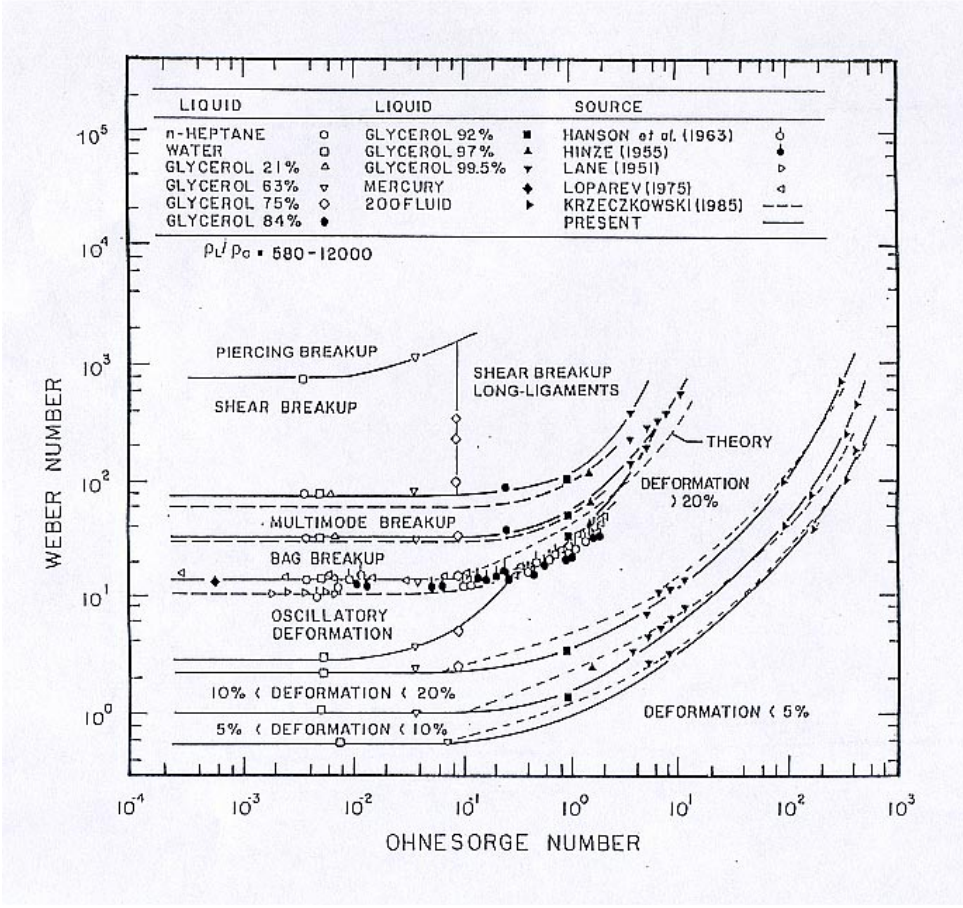


Figure 3.24: Drop deformation and breakup regime map.

Task 3 in Section 3.3.2 is aimed at providing a unified treatment of interphase coupling phenomena, and is directly related to the model development process associated with Tasks 4 and 5. Here, the experiments conducted by Kulick, Fessler and Eaton [51] are being used as the basis for joint comparisons between measured data and LES results. Details regarding this coordinated experimental and numerical effort are given in Section 3.3.5. As part of Tasks 4 and 5 a parametric study which was begun in FY99 was performed to identify the primary particle deformation and breakup regimes relevant to the current problem and thermodynamic regimes of interest. Because of the wide range of markedly different phenomenological regimes, it was imperative to perform this study 1) to insure that the DNS methods currently being developed can be used to perform parametric studies in the relevant regimes and 2) to provide the appropriate envelope of conditions in parameter space relevant to model development. Figure 3.23 represents a sample of the end results. Here relatively accurate estimates of the deformation and breakup characteristics which occur over relevant ranges of particle diameters, pressures, temperatures and the dimensionless particle slip velocity $|\mathbf{u} - \mathbf{u}_p|$ is given.

3.3.4 Numerical Framework

To facilitate the model development process, a numerical framework developed by Oefelein [52] is being used. This scheme employs the staggered methodology in generalized curvilinear coordinates and the dual-time stepping methodology using a unified all Mach number preconditioning technique based on the references cited therein. The algorithm accommodates fully implicit time advancement using a fully explicit multistage scheme in pseudo-time, and exhibits excellent parallel scalability. The implicit formulation is A-stable, which allows one to set the time step based solely on accuracy considerations. It accommodates any arbitrary equation of state, handles the thermodynamic non-idealities and transport anomalies typically encountered at elevated chamber pressures, and provides full thermophysical coupling (compressible and incompressible) over a wide range of conditions. It also accommodates intermediately complex geometric features while maintaining the accuracy of structured spatial stencils. The parallel paradigm employed for this framework and the Lagrangian spray modules is as follows:

- Distributed-memory message-passing using *MPI*.
- Single-Program–Multiple-Data *(SPMD)* model.
- Structured multiblock domain decomposition.
- Point-to-point at block interfaces.
- Static load balancing via preprocessing of the grid.

The primary calculations being performed with this framework are outlined in the next section.

3.3.5 Baseline Simulations

Stanford TSD Liquid-Fueled Combustor

The Stanford Thermosciences Division (TSD) liquid-fueled combustor [48] is a hexane–air (and methanol–air) experiment funded by NASA to provide detailed datasets for validation of liquid spray combustion processes typically encountered in gas turbine engines. In particular, the experiment is designed to allow for validation of interphase mass, momentum and energy transfer in both non-reacting and reacting environments. Similarly, it allows for validation of primary breakup, secondary breakup, and drop dispersion effects. The model combustor is similar to a lean, direct-injection (LDI) combustor with two co-annular, counter-swirled air streams.

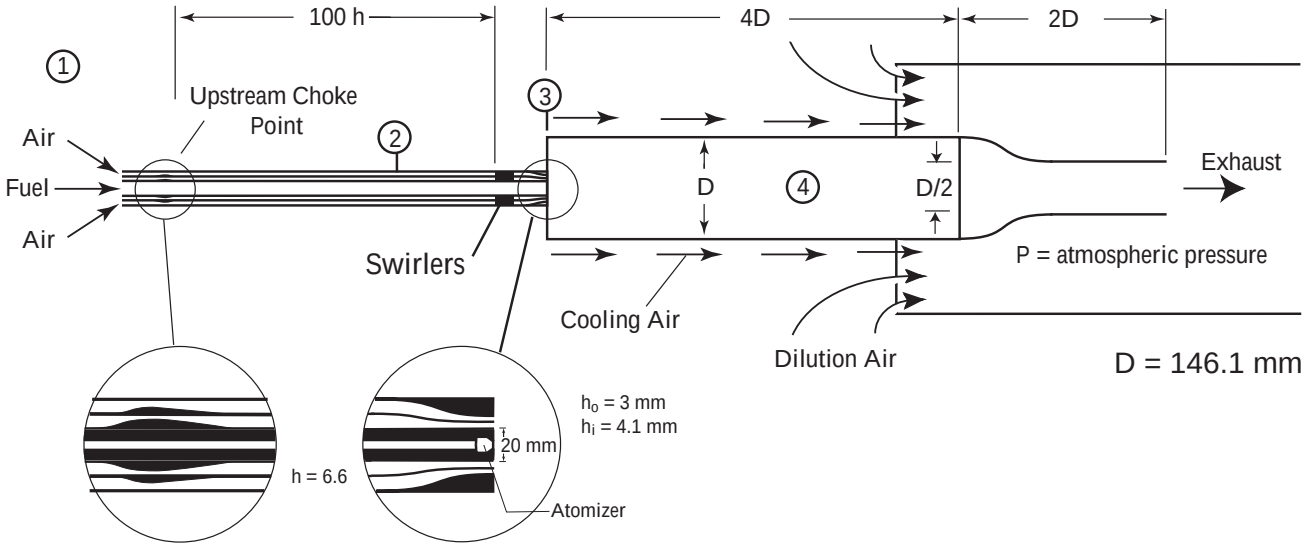


Figure 3.25: Schematic of the Stanford TSD liquid-fueled combustor.

Table 3.3: Nominal flow conditions at the stations indicated in Fig. 3.25.

1	**2**	**3**	**4**
$\dot{m}_a = 60.4\ g/s$ (50/50 Split)	$p = 1\ atm\ (\rightarrow 30\ atm)$	$U_o = 82.89\ m/s$	$Re_{\text{ref}} = 52043$
$\dot{m}_f = 2\ g/s$ (Hexane)	$T = 300\ K$	$U_i = 82.59\ m/s$	$U_{\text{ref}} = 82.72\ m/s$
$\Phi = 0.5\ (T_f = 1513\ K)$	$\nu = 15.89 \times 10^{-6}\ m^2/s$	$U_f \sim 15\ m/s$	$\delta_{\text{ref}} = 10\ mm$

A schematic of the experimental apparatus is given in Fig. 3.25. The geometric configuration includes an injector section with a long flow development region, a cylindrical combustor section, and a converging nozzle at the exit. The overall configuration has been designed to accommodate the specification of well-posed, numerically compatible, boundary conditions at both the inlet and exit. The nominal flow conditions corresponding to stations 1–4 in Fig. 3.25 are given in Table 3.3.

In FY00, efforts related to this combustor were focused on performing massively parallel simulations of the flow in the entire geometry, including the inlet ducts. A representative curvilinear multiblock grid is given in Fig. 3.26. This grid corresponds identically to the geometric features given schematically in Fig. 3.25. The baseline grid is composed of 4.703488×10^6 hexahedral cells. The current configuration has been decomposed into 832 blocks. This calculation will precede the acquisition of experimental measurements and will serve as the baseline for the implementation of Tasks 1–5 in Section 3.3.2.

Turbulent Particle-Laden Channel Flow

Task 3 in Section 3.3.2 focuses on the modeling issues related to high mass-loading interphase coupling and preferential concentration processes. Three particularly relevant effects induced by interphase coupling are 1) turbulence modulation which involves the damping of gas phase turbulence by "particulates" accommodating to turbulent motion; 2) turbulence generation, which involves the production of gas phase turbulence due to the presence of particle wakes; and 3) liquid deformation and breakup processes and the resultant effect on interphase exchange processes. These effects are relevant in all of the regimes depicted in Fig. 3.21.

As part of an effort to treat these phenomena systematically a series of LES calculations are being performed and compared to the experimental data acquired at Stanford by Kulick, Fessler

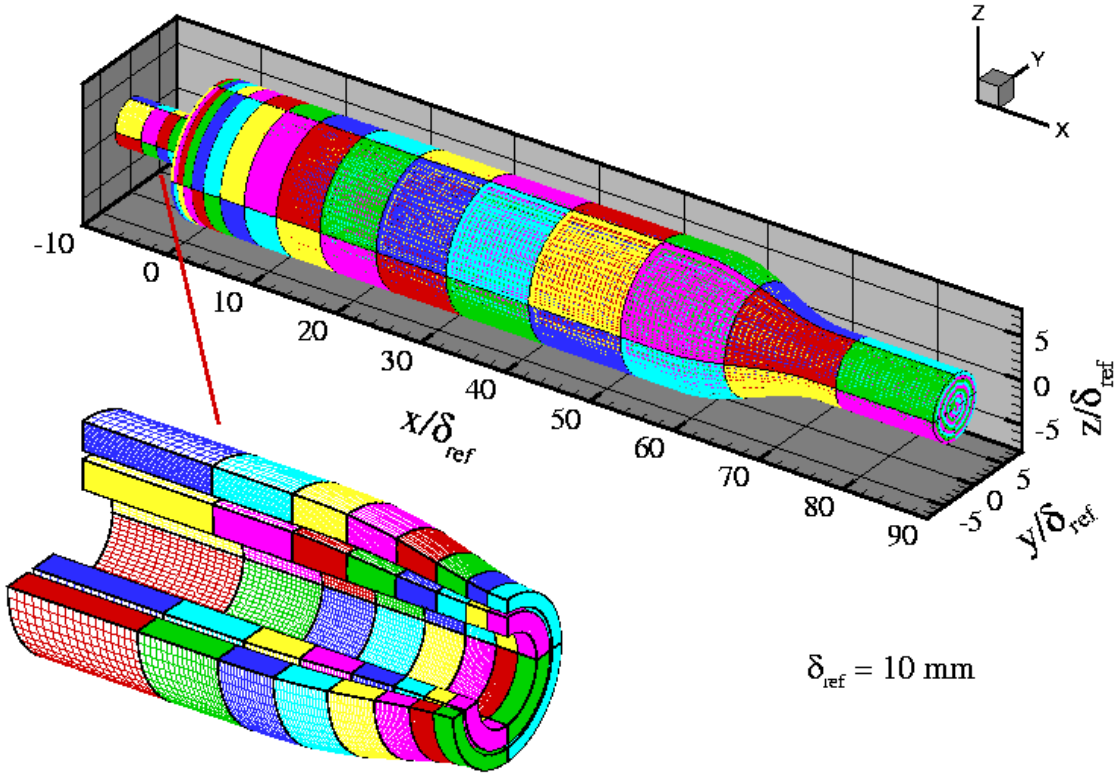


Figure 3.26: Primary curvilinear multiblock grid, 4.703488×10^6 hexahedral cells, 832 blocks.

Table 3.4: Nominal flow conditions.

h	20 mm
U_{cl}	10.5 m/s
Re_h	13800
u_τ	0.49 m/s
Re_τ	645

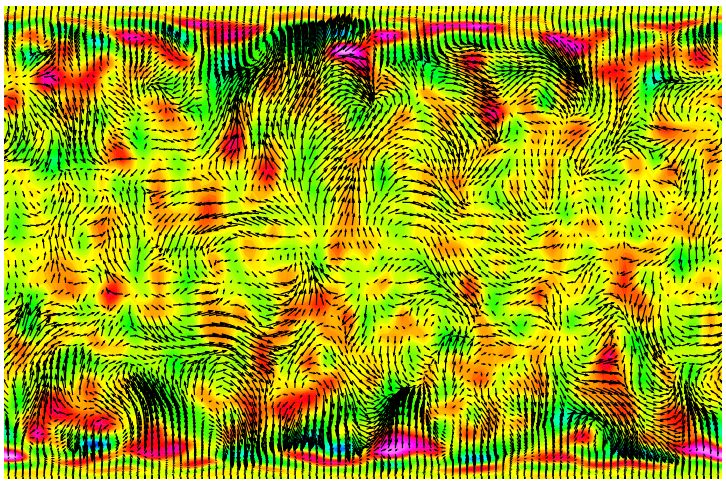


Figure 3.27: Velocity vectors and contours of streamwise component of vorticity in y – z plane.

and Eaton [51]. These experiments characterize the interactions between various particle loading conditions and the fluid turbulence in the well-defined confines of a turbulent channel. Particles are selected to respond to some, but not all scales of turbulent motions. Gas phase velocities were measured to investigate the means by which particles attenuate turbulence. The nominal flow conditions in the channel are summarized in Table 3.4.

LES calculations for this case are currently in progress. In addition to the direct impact on the modeling effort, this case provides a well defined way to demonstrate parallel speedup, performance and scaling issues with and without particles. The domain dimensions in the streamwise, transverse (wall-normal), and spanwise directions, respectively are $6h \times 2h \times 3h$. The primary grid is composed of 100^3 hexahedral cells.

Figure 3.27 shows the instantaneous velocity vectors and contours of the streamwise component of vorticity in y – z plane. Calculations are currently being performed with; a) no particles, b) 25 μm lycopodium particles ($\rho = 700\ kg/m^3$), c) 70 μm copper particles ($\rho = 8800\ kg/m^3$) present in the flow.

Efforts with regard to both experiments will continue in FY01 with emphasis placed on model development and validation.

3.3.6 Future Tasks Related to LES

High-fidelity LES calculations will continue throughout FY01 with emphasis placed on the following:

- Continue to perform systematic comparisons of LES calculations with experimental data acquired from the TSD liquid-fueled combustor.
  - Address issues related to modeling primary and secondary breakup processes.

  - Use data as baseline to develop a systematically validated and unified framework for use in full-component simulations.

- Continue to perform systematic comparisons of LES calculations with turbulent particle-laden flow data acquired by Eaton et al.
  - Address issues related to modeling preferential concentration and high mass-loading interphase coupling processes.
  - Use data develop and validate improved interphase coupling model(s).

- Begin to address issues related to the integration of the Lagrangian spray modules into the unstructured combustor code.

3.4 Simulations of Deformable Liquid Drops

Sub-grid scale drop models used in large-eddy simulations of spray flows are most accurate when the drop Weber number is much less than one. As the Weber number increases, the accuracy of the models decreases because the drop deviates from a spherical shape. At Weber numbers greater than one, the drop may break into smaller drops. To improve the modeling of these phenomena, simulations of isolated drops are needed which are accurate enough to provide quantitative information.

In the past two years, we have developed an Arbitrary-Lagrangian-Eulerian (ALE) spectral element method to simulate drop phenomena. This approach has several advantages over currently available methods for simulating droplets. First, the spectral elements give a high-order of accuracy. This allows accurate solutions to be obtained with fewer degrees of freedom than second-order accurate methods. Second, because the mesh deforms with the drop interface, accurate implementations of the interface jump conditions are possible. This drastically reduces the resolution requirements for obtaining quantitative information compared to fixed-grid methods such as Tryggvasson's method [65] or the level-set method [64]. It also allows us to study surface phenomena such as contaminant induced Marangoni effects, which can not be studied with a fixed grid method. These effects can have a strong role in the behavior of liquid drops. Third, an unstructured mesh is used in the spectral element formulation. This allows us to locally adapt and restructure the mesh to resolve the features of the problem being studied. This becomes increasingly important for high deformation problems because regions of high interface curvature and large flow gradients develop.

In the following, we begin with a brief outline of the numerical method and a validation of the method for problems relevant to deformable liquid drops. There are several new features of this method which may make calculations with spectral elements more feasible for practical problems, and we will point these out in the discussion. We then present our latest findings in our studies of liquid drops. These findings have helped us evaluate the current drag models in use and understand what aspects of the models need improvement. We have also begun to look at secondary break-up of droplets. Current models of secondary break-up are rather rough, (see [62]), and much work is needed in this area.

3.4.1 Formulation/Numerical Method

We are interested in simulating the flow of two immiscible fluids separated by an interface which has a variable surface tension, σ. The fluids are assumed to be incompressible with constant densities ρ_1 and ρ_2 and viscosities μ_1 and μ_2. The surface tension is assumed to vary linearly with the concentration of a contaminant which is bound to the surface of the droplet. The surface tension equation of state is thus of the form $\sigma = \sigma_0(1 - \beta C)$ where β is a sensitivity parameter, σ_0 is the uncontaminated surface tension, and C is the contaminant concentration. This equation of state is valid for small βC [53]. The contaminant is assumed to obey a convective-diffusive equation formulated on the surface of the drop. This introduces another parameter, D_C, the diffusivity of the contaminant. Because the contaminant is an unknown substance, the best we can do is parametrically vary these parameters and determine their influence on drop behavior.

The simulations are performed using an ALE method in which the mesh deforms with the interface. This allows accurate implementation of the interface jump conditions for conservation of mass and momentum through the interface. It also allows us to formulate the equation governing the contaminant on the subset of the mesh which is coincident with the interface. The governing equations are discretized using a fully-implicit time integration scheme and the Streamwise-Upwind-Petrov-Galerkin Variational Method [59]. The basis functions for the variational method are the variable-order, triangular spectral element basis developed by Dubiner [57].

One of the important new developments in this work is the implementation of a spectral-multigrid scheme for solving the implicit governing equations. This scheme is essentially a generalization to spectral elements of the multigrid scheme developed by Jameson [60, 54]. Because both steady-state and unsteady solutions can be obtained efficiently, it makes spectral elements an attractive method for problems requiring high accuracy. Also, because spectral-elements retain their accuracy on skewed meshes and hybrid meshes [63], this may be an important advancement for calculations of the flow over complex geometries. A paper on the method is in preparation for submission to *Computational Methods in Applied Mechanics and Engineering.*

3.4.2 Validation

To validate the above method, we begin with the problem of flow over a solid sphere at a Reynolds number of 100 where the Reynolds number is based on the sphere diameter. This problem will lay the foundation for our studies of deformable droplets since in the limit of high liquid to gas viscosity ratio, μ_1/μ_2, and low Weber number, $\rho_2 U_\infty^2 d/\sigma$ where d is the droplet diameter, the behavior of liquid droplet is nearly indistinguishable from that of a solid sphere.

A typical axisymmetric mesh used for these calculations is shown in Fig. 3.28-Left. This mesh has 16 element edges along the surface of the sphere. At the lower boundary of the mesh an inflow condition is enforced with $(u, v) = (0, 1)$ such that the flow is from bottom to top. At the centerline, u is set to zero and the total flux through the centerline in the continuity and v-momentum equations is set to zero. At the downstream boundary and the boundary to the right of the domain, the total stress is set to zero. The downstream boundary is from $(0, 15)$ to $(10, 10)$, and the inflow boundary is from $(0, -10)$ to $(10, -5)$ where the units are sphere diameters.

Simulations are performed using spectral element basis functions of polynomial degree P of 1, 2, and 4 and with meshes having 4 to 64 element sides on the sphere. With 16 elements edges on the sphere, there are 17, 33, and 65 pressure modes along the surface of the sphere for P equal to 1, 2, and 4 respectively. The number of pressure basis functions on the sphere surface will be denoted as N. The number of global degrees of freedom goes as N^2 independent of the polynomial degree. We note that when the polynomial degree is equal to one, the method corresponds to the standard linear finite element method.

In Fig. 3.28-Right, we plot the relative error in C_D versus the number of basis functions on the surface of the sphere. The C_D calculated from the simulation with $N = 129$ and $P = 4$ is used as the exact value of drag. The straight lines are power law curve fits to the data. The exponent of the fit for P equal to 1, 2, and 4 was -0.9, -3.1, and -5.2 respectively. These numbers are fairly sensitive to the detailed structure of each mesh. The general trend however shows that the spatial scheme attains the desired convergence rates. This plot shows the advantage of using spectral elements to obtain accurate solutions. For an accuracy of 1%, an approximate factor of 6^2 fewer degrees of freedom are needed when using polynomials of degree of 4 compared to using polynomials of degree 1.

As a final check on the drag results for flow over a sphere, we generate a C_D curve versus Reynolds number. The calculations are performed with $N = 33$ and $P = 2$ which should give an error of less than 1%. The Reynolds number range is from 1 to 200. Calculations are not performed for $Re > 200$ because at $Re = 210$ the flow transitions to a non-axisymmetric solution [61]. Comparing to the experimental curve fit given by [55], we find that the largest difference between our result and the curve fit is 3% over the entire range of Reynolds number.

To test the two-fluid aspects of the formulation, we calculate the quasi-steady solution for a falling liquid drop in a gaseous environment. The liquid to gas density and viscosity ratios are $\rho_1/\rho_2 = 100$ and $\mu_1/\mu_2 = 100$ where fluid 1 is the liquid and fluid 2 is the gas. The body force on the drop is determined as part of the iterative process such that the terminal Reynolds number

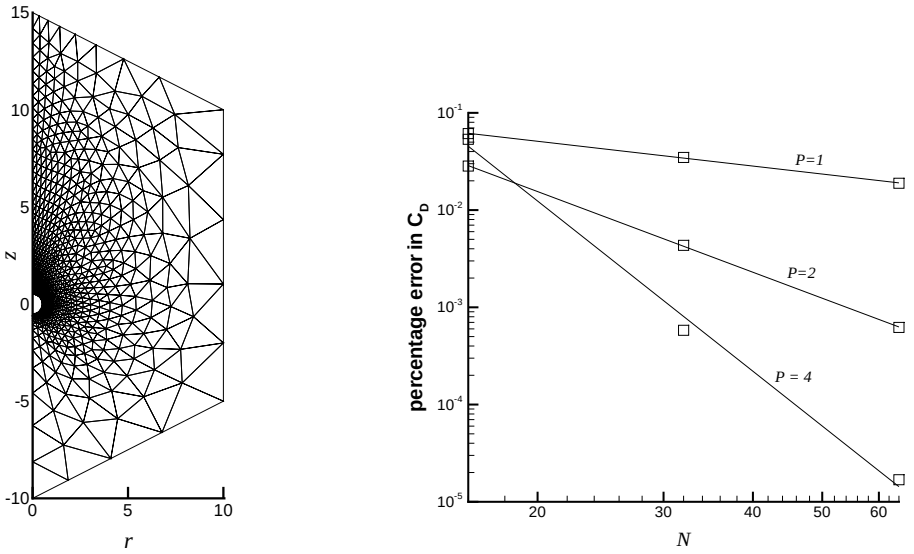


Figure 3.28: Left: Typical mesh used for flow over a sphere calculations. Right: Relative error in the drag coefficient at $Re = 100$. Solid line – $M = 1$, Dashed line – $M = 2$, Dotted line – $M = 4$.

based on the gas properties is 60. The terminal Weber number of the drop is 4.0. This case was previously calculated by Dandy and Leal [56].

The geometry and boundary conditions used for the calculation are the same as that used for the solid sphere. Fig. 3.29 shows a comparison of the pressure contours and drop shape calculated using a mesh with 16 element sides along the interface and $P = 2$ to a calculation on the same mesh with $P = 4$. Qualitatively the contours are nearly indistinguishable. The coefficient of drag on the drop, based on the frontal area of the undeformed drop, $\pi d^2/4$, is 1.713 for $P = 2$. For the case with $P = 4$, the drag is 1.702 which is a 0.6% change. The results of Dandy and Leal [56] predict 1.66, which is within 3% of our results, confirming that our method can predict the behavior of two-fluid problems accurately.

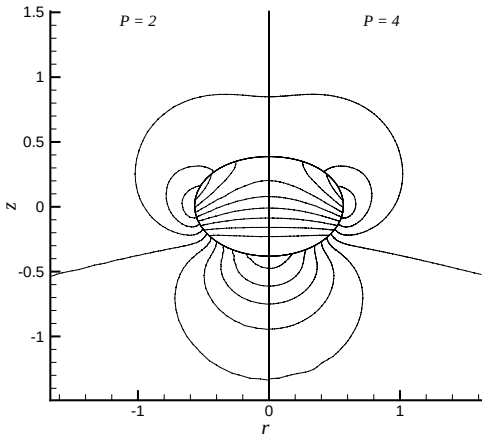


Figure 3.29: Pressure contours at an increment of 0.1 and droplet shape calculated with $P = 2$ on the left and $P = 4$ on the right

3.4.3 Hexane Drops

We now present some new results on the behavior of liquid drops in a gas flow field. To perform the simulations, we choose properties corresponding to a hexane drop in air at one atmosphere. If we make quantities non-dimensional by the diameter of the drop, a characteristic velocity U_c, and the density of air, the non-dimensional parameters are the liquid to gas density and viscosity ratios $\approx 500, 15$, the Reynolds number Re, and the Weber number We. It will be convenient to also define the Ohnesogre number, $Oh = \mu_2/\mu_1 Re^{-1}\sqrt{\rho_1/\rho_2 We}$. We note that if the fluid properties are fixed, Oh varies only with the diameter of the drop.

For our initial simulations, we examine the steady drag on a drop in a uniform flow. For a steady solution to exist, a body force must be applied to the drop otherwise the drop would accelerate until it reached the free-stream velocity. For a given Re, the force is solved for as part of the iterative process. To compare the results for a drop to those of a sphere, drag results are obtained at fixed Oh for various Re, We. In physical space, this corresponds to holding the diameter of the drop fixed while varying the free-stream velocity.

Figure 3.30 shows the percentage change in C_D as compared to the solid sphere case versus Weber number. The cases shown are a 10, 50, and 100 μm drops which correspond to Ohnesogre numbers of 2.7e-2, 1.2e-2, and 8.5e-3 respectively. In all cases, the drag on the liquid droplet was less than that of the solid sphere. For small Weber numbers, the deviation between the solid sphere and the liquid droplets is around 3%. This is again because there is an approximate 3% difference between the experimental curve fit and our numerical results for a solid sphere. This difference is therefore negligible. As the Weber number approaches unity however the deviation from the solid-sphere drag becomes significant. This effect is greater for larger diameter drops with the maximum difference being 20%.

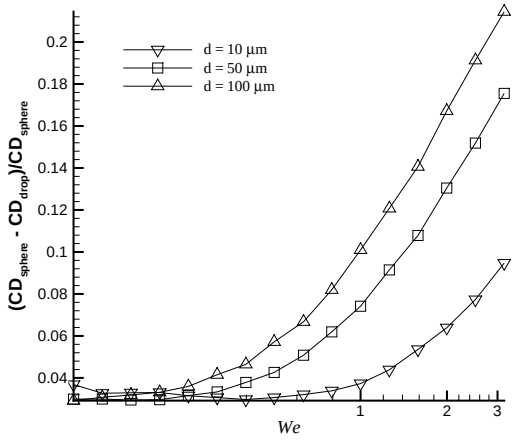


Figure 3.30: Deviation of the drag from the solid sphere model

The reason for the reduction in drag is shown by Figure 3.31-Left. This figure shows the drop shape as well as the v-velocity contours and streamlines in a region near the drop for a 50 μm drop at a Weber number of 3.2. The ratio between the streamwise length and cross-subsectional diameter of the drop is 1.5. Thus, the strong difference in drag coefficients is due to the streamlined shape of the drop. We note that streamlined drop shapes are not usually seen in experiments. The reason for this is that experimental drops usually have some level of surface contaminant.

To investigate the effect of surface contaminants, we recalculate the case shown in Figure 3.31-Left with contaminant-induced Marangoni effects. The nondimensional sensitivity parameter of

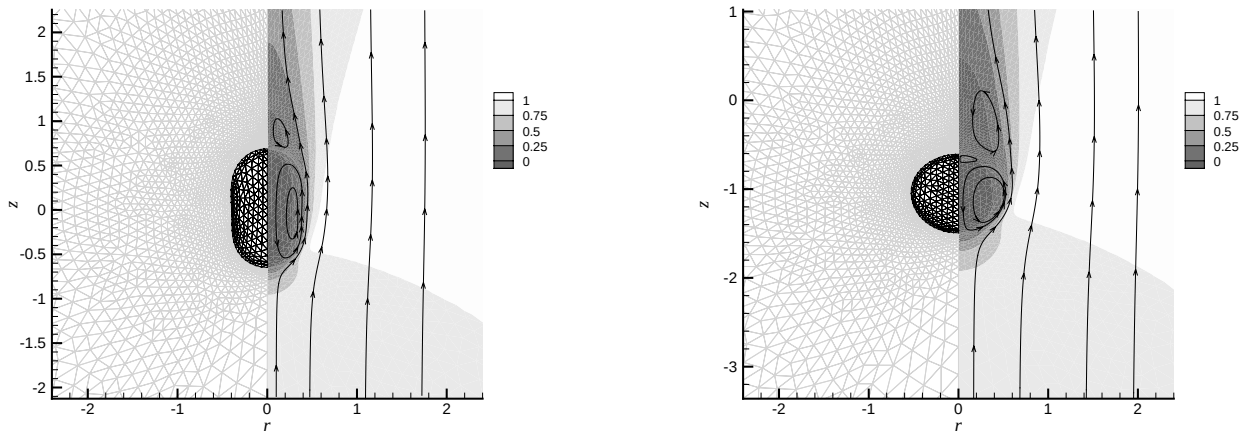


Figure 3.31: Left: v-velocity contours, streamlines, and drop shape at $Re = 100$. Contours are from 0 to 1 by 0.25. Right: Same as the left except with a 10% surface tension variation due to contaminant.

the surface tension to the contaminant is taken as 0.1 such that a uniform distribution causes a 10% decrease in the surface tension. The Peclet number of the contaminant is 100 based on the free-stream velocity, the diameter of the drop, and the contaminant diffusivity. The maximum tangential surface velocity is about 7% of the free-stream velocity so the surface Peclet number is actually $O(1)$. As was done previously, figure 3.31-Right shows drop shape as well as the v-velocity contours and streamlines. Clearly the contaminant has a very strong effect on the deformation of the droplet. The drop changes from a prolate shape to an oblate shape and the drag changes from 15% less to 10% greater than the solid-sphere drag. We also see a reverse recirculation zone at the rear of the drop. This has been seen in experimental visualizations of falling liquid droplets [55] and is called the "stagnant-cap region." Because of this strong dependence on contaminants, without further experimental and numerical investigation it will be impossible to provide a more reliable drag model than the standard solid-sphere drag model.

Beyond a Weber number of 3.0, a critical transition in the stability of the drops is reached and stable steady solutions cease to exist. For Ohnesogre numbers less than unity, this transition is from a quasi-steady solution to periodic oscillations of the droplet. To show this, an unsteady calculation with fixed drop body force is performed in which the terminal velocity corresponds to approximately $We = 4.0$. Figure 3.32 shows the drop shape at its maximum and minimum streamwise length over one period. The period of oscillation based on the free-stream velocity and the drop diameter is approximately 46.8. Using the analytic formula given above for the period of a inviscid drop in a vacuum, we get a prediction of 37.3. Thus, it is roughly the leading-order mode of oscillation which is being excited.

This transition in stability has some analog in experimental results. Hsiang and Faeth [58] have compiled data from shock tube experiments with drops which shows that, for a fixed Oh, there is transition from steady deformation to oscillatory deformation at a Weber number between 2 and 3. Although the physical setup is much different, this is qualitatively similar to the behavior seen here.

To further investigate this transition in stability, we plot in figure 3.33 the Weber number at the transition point versus Ohnesogre number. On the plot, three regimes are shown, the stable regime, the oscillatory regime, and the diffusive break-up regime. The stable regime and the oscillatory regime have already been discussed. The diffusive break-up regime occurs above the

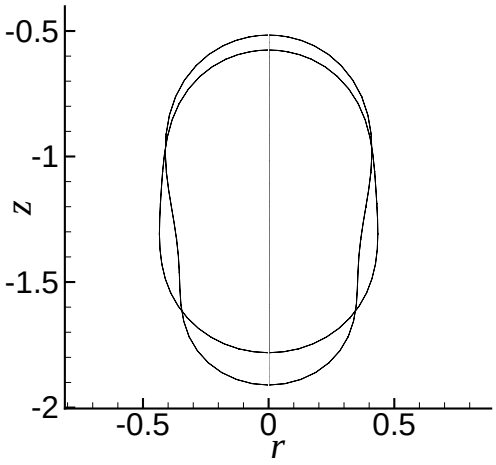


Figure 3.32: Unsteady oscillations of drop shape at $Re = 112$, $We = 4$.

critical transition when the Ohnesogre number is greater than one. In this limit, viscous forces are strong relative to the restoration force of surface tension which means the droplet is overdamped and does not oscillate. The transition in stability in this regime therefore indicates that the droplet is entering a diffusive (non-oscillatory) break-up process.

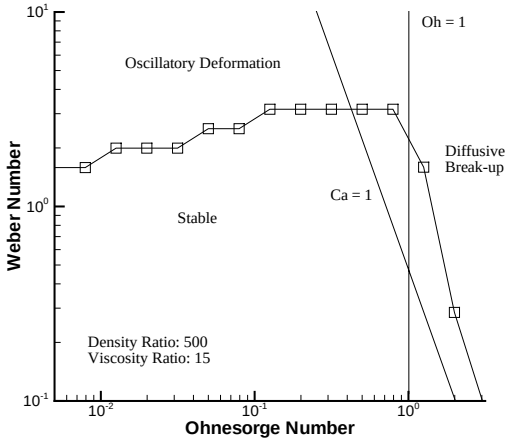


Figure 3.33: Critical limits of drop stability.

Because viscous forces dominate convective forces in the diffusive break-up regime, the relevant parameter is no longer the Weber number. In this limit the Capillary number, $\mu_g U/\sigma$, is the leading order predictor of the transition regime. The line of capillary number equal to one is shown on the figure and gives a reasonable estimate of the transition to diffusive break-up. Determining these transition limits is the first step in developing better secondary break-up models to be used in spray-simulations. Because the limits shown are found by holding the droplet at a fixed velocity, they will be valid only when the droplet acceleration time due to drag is long relative to the deformation time of the droplet. This should be roughly true because ratio of the dynamic viscosity of the liquid to the gas is large however further work is needed to verify these limits in the full unsteady case. Further work is also needed to investigate the diffusive break-up regime. As the operating pressure

of gas-turbine engines is increased towards the critical pressure, the drop surface tension decreases which results in larger Ohnesogre numbers. Compared to the low Ohnesogre limit, there has been little experimental or numerical investigation of this limit.

3.4.4 Summary and Future Plans

Our progress in the last year is as follows:

- The spectral element algorithm for simulating two-fluid incompressible flow has been completed.
- A surface contaminant model has been implemented.
- Results for drop drag and deformation have been obtained. Results indicate that without a better characterization of the contaminants in the application, improved drag modeling is impossible.
- Several studies of droplet stability have been completed.

Our plans for the next year are as follows:

- Continue to use the code to investigate drop behavior – specifically look at unsteady flow effects, contaminant effects, and pressure and temperature effects on droplet drag and break-up.
- Use this information to develop improved secondary break-up models for drops.

3.5 Chimera Method for Heterogeneous Particle Clusters

The objective of this work is to provide a fully resolved data set that can be used as a validation and development tool for momentum coupling models in the regime in which the drop size is on the order of the Kolmogorov length scale, but deformation and internal circulation effects are not important. Currently, there are no experimental or computational data sets that provide the information required to develop suitable subgrid-scale momentum-coupling models for LES. We plan to perform DNS quality simulations of a small region of three-dimensional turbulence containing a heterogeneous distribution of spherical particles. Computations of this flow are extremely complex due to the wide range of scales introduced by the presence of the particles. The computational method used for these simulations must accurately capture this wide range of scales in order for the results to be useful for model development. This method is likely to find additional applications in situations where physical processes at widely disparate scales are important.

The numerical method chosen for these simulations utilizes overset grids as shown in Figure 1. Each particle has its own body-fitted grid to completely resolve the flow to the particle surface. A coarse background grid, capable of resolving the desired scales of turbulence in the absence of particles, lies underneath the particle grids and passes boundary information to the edge of each particle grid. The solution obtained on each particle grid is used to provide information to coarse grid nodes at the edge of a hole region in the background grid. In this way, two-way momentum coupling between the phases is achieved.

The proposed overset grid method provides the necessary flexibility for coupled multiphase flow simulations. Though the particles will initially be fixed, moving particles can also be computed without the expense of regridding. In addition to solid particles, non-deforming spherical drops can also be simulated and heat and mass transfer between the phases can be included. Different time advancement and spatial discretization schemes can be used on the background and particle grids. For solid particles in turbulence, the background grid will be time advanced explicitly and the particle grids will be time advanced using a semi-implicit method to alleviate timestep restrictions. The use of overset grids also provides a natural parallelization for the problem as each particle grid can be placed on a separate processor and passed boundary information from the background grid. Another benefit of the the overset grid method is that all grids are structured allowing the use of specialized computational techniques.

The turbulent simulations require a background solver, a particle grid solver, and a coupling algorithm. The particle grid solver has been validated and optimized for single processor performance. The work in the past year has focused on the development of the coupling algorithm. The majority of incompressible turbulent simulations use a fractional step algorithm to time advance the velocity field and satisfy the continuity equation. Implementation of fractional step is straight forward on a single grid, but it becomes more complex when an overset grid is used. We rigorously derived a fractional step algorithm suitable for solution of the incompressible Navier-Stokes equations on overset grids as an approximate matrix factorization. The first step of the algorithm advances the velocity field using the momentum equations. The second step solves a Poisson equation for the pressure. The final step projects the velocity field onto a divergence free subspace using the solution of the Poisson equation.

Coupling between the grids in the first step of the fractional step algorithm depends on the time advancement schemes chosen for the individual grids. If all grids use explicit time advancement, then the interior nodes on each grid can be updated without any information from other grids. After updating the interior nodes, the pseudo-boundary node velocity values are interpolated from other grids. If all grids use implicit time advancement, then the momentum equations for the interior nodes and the interpolation equations for the pseudo-boundary nodes on all grids must be solved simultaneously. Solution of this system would be extremely expensive for the turbulent simulations,

so semi-implicit advancement will be used only on the particle grids. The interior nodes on the background grid can be advanced explicitly without information from the particle grids. Then, the pseudo-boundary values on the particle grids can be interpolated from the background grid. The updated values on the pseudo-boundary of the particle grids provide the necessary information for semi-implicit advancement on the particle grids. Finally, the updated solution on the particle grids is used to interpolate the velocity at the pseudo-boundary nodes on the background grid. Note that regardless of the time advancement schemes on the different grids, the pseudo-boundary values on one grid cannot be used to interpolate to another grid. This can be avoided easily by ensuring that there is sufficient overlap of the grids.

Though initially it was believed that the pressure fields on each grid could be determined independently, the derived fractional step algorithm requires that the solution of the pressure field be fully coupled. One possibility for solving the system is to use an Alternating Schwarz algorithm that solves first on the background and then solves on the particle grids using interpolated values from the current solution on the background grid. The procedure is repeated until convergence at which time the computed pressure field is used to project the velocity field on all grids. However, the Alternating Schwarz algorithm converges extremely slowly making it prohibitively expensive for large systems. Therefore, a multigrid algorithm will be used for the turbulent simulations.

It is well known that the system of equations for the pressure field in incompressible flow is singular. One way to remove the singularity in a single grid code is to replace one of the equations by an equation that sets the level of the pressure. However, if the system of equations does not satisfy the discrete analog of the solvability condition for a Poisson equation with Neumann boundary conditions, then an error in the solution will appear at the node corresponding to the replaced equation. In an overset grid code, the discrete analog of the solvability condition will not be satisfied because of interpolation errors. In order to make the system solvable, the right hand side of the Poisson equation is adjusted, introducing an error in each equation that leads to non-zero divergences in the velocity field.

The proposed fractional step algorithm has been tested on the Taylor vortex array; a time-dependent, 2-D analytical solution of the full Navier-Stokes equations. A sample grid used for this study is shown in Figure 2. An off-center hole is cut out of the background square grid and then covered with another square grid with a slightly smaller grid spacing. Note that the grid points do not lie on top of one another at the boundary of the fine grid. A staggered grid arrangement is used for the dependent variables and 2nd order finite differences are used for computation of derivatives. In order to be consistent, bi-quadratic interpolation is used for the pseudo-boundary node interpolation. Time advancement is performed with fully explicit 3rd order Runge-Kutta on the background grid and semi-implicit 3rd order Runge-Kutta/Crank-Nicholson on the fine grid. Contours of u_x are shown in Figure 2. There are no visible discontinuities in the solution at the boundary of the fine grid indicating that information is being transferred accurately from grid to grid. In addition, the computed solution accurately follows the known analytical solution where the vortices decay without distortion.

Both spatial and temporal accuracy must be maintained by the coupling algorithm if the results are to be treated as a true DNS. To test the spatial order of accuracy, the RMS errors of the velocity field on the fine grid were monitored until t=1.0 (2τ). Time histories for the RMS error in u_x are shown in Figure 3 for three different grid arrangements. For a 2nd order accurate scheme, the RMS error should decrease by a factor of four as the number of grid points is doubled in all directions. Figure 3 shows the correct general behavior, but changes in the interpolation locations as the grids are refined cause the RMS error to decrease by more than a factor of four at some points in time and less than a factor of four at other points in time. In order to check the time accuracy, the $(15\times15)\cup(7\times7)$ grid was run until t=0.5 with a timestep of 5e-5. This solution was called the exact solution and RMS differences between this solution and solutions computed with larger timesteps

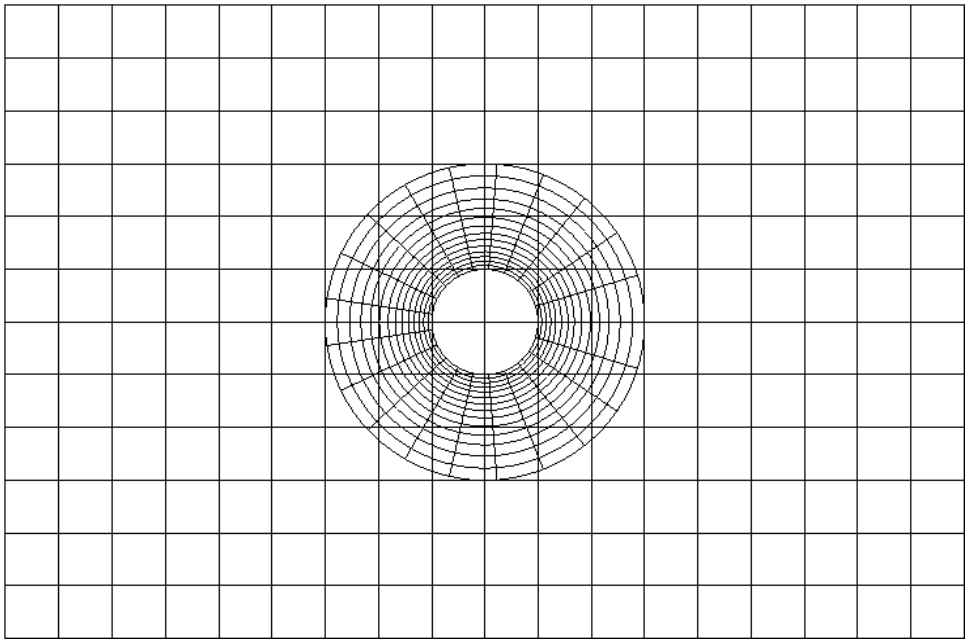


Figure 3.34: Overset grid arrangement

were calculated. Figure 4 shows the behavior of the RMS differences as the timestep is reduced. The code is clearly 2nd order accurate in time as is expected since Crank-Nicholson is used for some of the viscous terms on the fine grid. For the $(15 \times 15) \cup (7 \times 7)$ grid, the discrete divergences at t=1.0 are 1.8e-9 and 8.3e-10 on the background and fine grids respectively. Though the continuity equation requires that the discrete divergence be machine zero, in practice the Poisson equation is usually converged to a certain level of accuracy which makes the discrete divergence small, but not necessarily machine zero. For the Taylor vortex problem, the incompatibility between the grids is small enough that the discrete divergence remains small.

The coupling algorithm is currently being tested in a simulation of flow over a circular cylinder at a Reynolds number of 100. A portion of the grid used for this simulation is shown in Figure 5. This overset grid arrangement is very similar to the desired configuration for particles in turbulence including a coordinate transformation in the interpolation of velocity components. A hole is cut out of the background cartesian grid and a cylindrical grid covers the hole and resolves the flow to the surface of the cylinder. Staggered grids are used with 2nd order finite differences and bi-quadratic interpolation. Fully explicit time advancement is used on both grids. Streamwise velocity contours after three domain flow-through times are shown in Figure 5. By this time the cylinder has begun to shed vortices. Once again, the contours are smooth across the grid interface indicating that the coupling algorithm can handle this type of unsteadiness accurately. The discrete divergences at t=60 are 1.5e-5 and 5.6e-4 on the background and fine grids respectively. These divergences are orders of magnitude larger than in the Taylor problem since the cylindrical and square grids are less compatible than a pair of square grids. Tests are currently underway to verify the spatial and temporal accuracy of the coupling for this flow configuration where the discrete divergence is much further from machine zero.

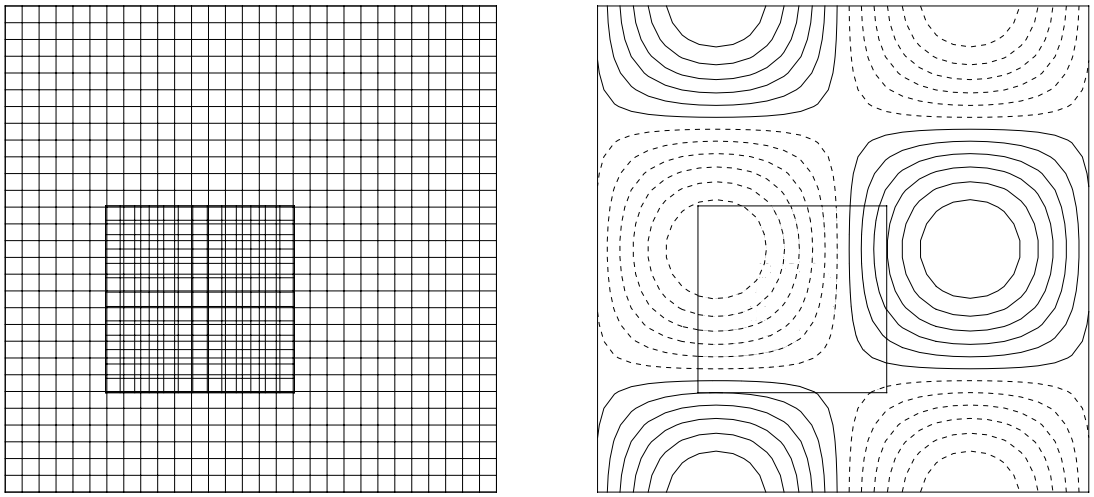


Figure 3.35: Taylor vortex $(30 \times 30) \cup (14 \times 14)$ grid and u_x contours at t=2τ

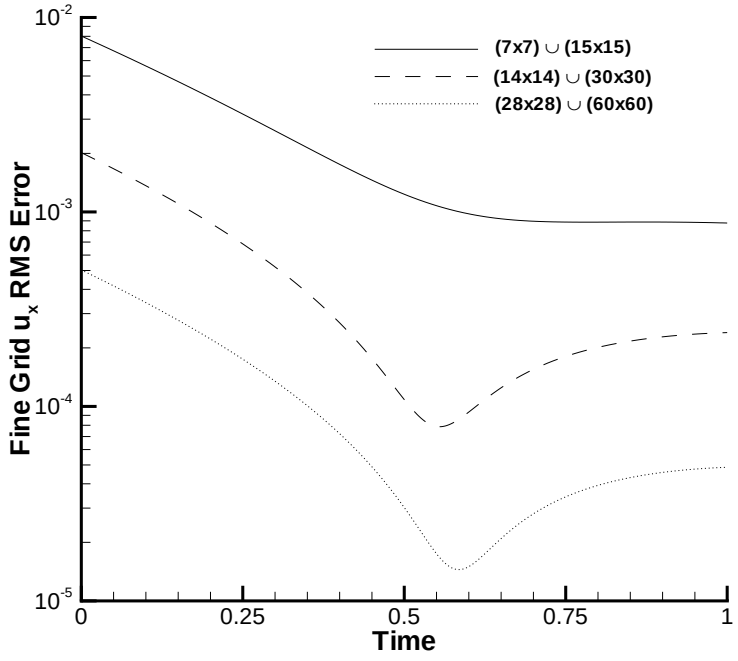


Figure 3.36: Time histories of u_x RMS error for Taylor vortex

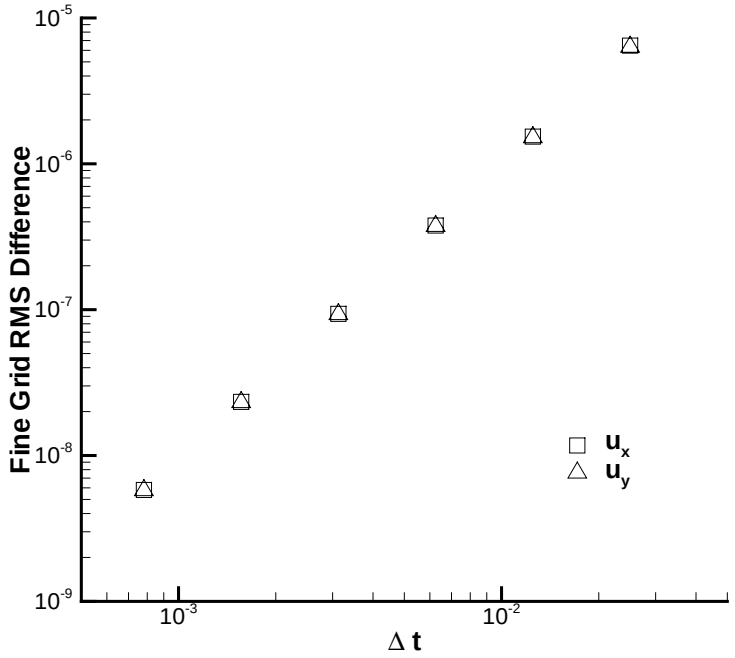


Figure 3.37: Time accuracy for Taylor vortex on $(15 \times 15) \cup (7 \times 7)$ grid

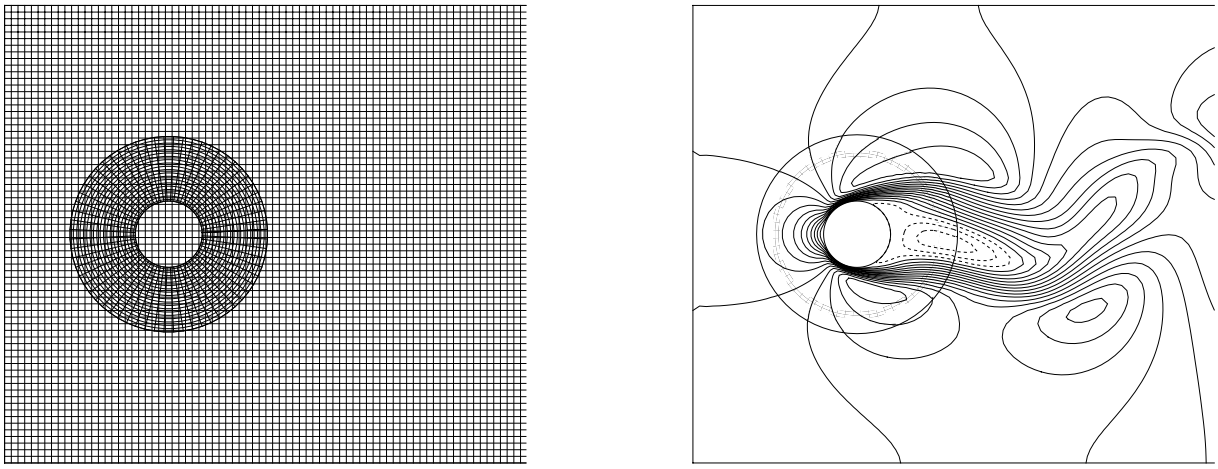


Figure 3.38: Cylinder $(175 \times 70) \cup (20 \times 64)$ grid and u_x contours at t=60

3.6 SCCM

The Computational Mathematics component have contributed the fundamental study of the fully-coupled particle-fluid flow problems arising from the spray simulations, and developed numerical techniques associated with spray as well as simulations involving interface. Regarding high performance computing effort, we have offered a new parallel computing course and hosted a special high performance computing seminar series with presentations from computational scientists at the DOE labs and other ASCI alliance.

3.6.1 Collision Detection

The research work in this area is a joint collaboration between Andrew Stuart, Hersir Sigurgeirsson and Wing-Lok Wan [70]. The ultimate goal is to simulate the fully-coupled problem of colliding particles immersed in a fluid governed by the Navier-Stokes equations. However, to make the problem more tractable, we have broken the path down into three model problems of increasing complexity:

1. **Billiards.** Particles move in straight lines between collisions, so their positions y_j are given by

$$\begin{aligned} m_j \ddot{y}_j &= 0, \quad j = 1, \ldots, n, \\ &\quad \text{plus elastic collisions,} \end{aligned}$$

   where m_j is the mass of particle j.

2. **Particle-laden flow.** Particles move in, but do not affect, a fluid according to Stokes law. Their positions are given by

$$\begin{aligned} m_j \ddot{y}_j &= \alpha[u(y_j, t) - \dot{y}_j], \quad j = 1, \ldots, n, \\ &\quad \text{plus elastic collisions,} \end{aligned}$$

   where u is the fluid velocity field, a solution of the Navier-Stokes equations, and α is a constant depending on the fluid viscosity, ν, and the particle radius, r; $\alpha = 6\pi\nu r$.

3. **Fully-Coupled.** Particles move in and exchange momentum with a fluid. For simplicity we take a simple diffusion equation rather than the full Navier-Stokes equations. Note that this simplified model does not include the pressure, and hence cannot conserve mass. The velocity of the fluid, u, and the particle positions are then given by

$$\begin{aligned} u_t &= \Delta u + f - \sum_{j=1}^{n} \delta_\epsilon(x - y_j)\mathcal{F}_j, \\ m_j \ddot{y}_j &= \mathcal{F}_j, \quad j = 1, \ldots, n, \\ &\quad \text{elastic collisions.} \end{aligned}$$

   Stokes Law:

$$\mathcal{F}_j = \alpha(u(y_j, t) - \dot{y}_j), \quad j = 1, \ldots, n.$$

We have designed and implemented an algorithm to solve model problem 3, the fully-coupled system of partial differential equations and particles. Below we give the detail of the algorithm and its computational cost. We also briefly mention some issues related to the numerical stability of the algorithm.

Algorithms described in the molecular dynamics literature can be used to simulate model problem 1. We have extended such an algorithm to simulate model problems 2 and 3, and described them in a previous report. In this report we therefore focus on the computational cost of the algorithms when applied to model problems 2 and 3.

To briefly review, the standard algorithm for collision detection of model problem 1 is event driven, *i.e.* proceeds from collision to collision. To reduce the number of collision checks, one divides space into equally sized cells and then only considers collisions between particles in neighboring cells. To keep track of which cell a particle belongs to, one detects, in addition to collisions, when a particle moves from one cell to another. The algorithm then consists of repeatedly computing next transfer and collision events for every particle and handling them in order of occurrence, by use of an event queue. The optimal choice of cell-size trades off the cost of these two effects.

To extend the algorithm to simulate mode problems 2 and 3 we introduce a time step Δt and integrate the equations numerically. We then assume a simple motion, for example linear, within each time step and apply the algorithm for model problem 1 to the piecewise linear paths.

We have rigorously analyzed the complexity of the algorithm for model problem 1, under justifiable statistical assumptions on the particle positions and velocities. Our result is that with the optimal choice of cell size, which is a by-product of our complexity analysis, the complexity is

$$W_1 = \mathcal{O}(n_c \log n),$$

where n_c is the total number of collisions, and n the number of particles. This is close to optimal since an event driven algorithm necessarily has complexity $\Omega(n_c)$.

Although our assumptions are not satisfied for model problem 2, we can apply the results, with appropriate additions, to gain some insight into the computational cost. At the beginning of each time step we now additionally have to integrate the path of every particle, compute the next collision and transfer time for each of them, and set up the event queue, adding $nk \log n$ to the complexity, where $k = T/\Delta t$ is the total number of time steps taken. Our analysis suggests the same choice of cell size for optimal performance, and the complexity is now

$$W_2 = \mathcal{O}((n_c + nk) \log n).$$

From this we notice that in order for the overhead added by the time integration not to be dominant we need $k = \mathcal{O}(n_c/n)$, *i.e.* the time step should not exceed the mean time between collisions.

For model problem 3, we have the additional task of solving a PDE. We use a method implicit in the diffusion term and linearly implicit in the delta sources. Our choice of method is dictated by a numerical instability in a method that is explicit in the nonlinear terms, which results when many particles cluster together. The implicit method appears to cure this instability. To fully understand this numerical instability rigorous mathematical underpinnings for the PDE/ODEs are needed.

The relative cost of the two parts, the collision detection on the one hand and the solution of the PDE on the other, is now of interest. We consider a situation where the number of particles scales like the number of mesh points and, since we use a method implicit in the diffusion term, we will take the time step, Δt, to scale like the space step, Δx. Thus, if $N = (\Delta x)^{-d}$ is the total number of mesh points, we take $n = \Theta(N)$ and $\Delta t = \Theta(N^{-1/d})$.

To solve the linear equations arising in the PDE, we use the conjugate gradient (CG) method, preconditioned by the solution with explicit treatment of the delta source terms. This preconditioning can be performed using the FFT in $\Theta(N \log N)$ operations. Hence, if the number of CG iterations is bounded independently of N, then the total cost of the linear solver is $\Theta(N \log N)$. In practice, only a few CG iterations are used each time step and the number of iterations is independent of N. Thus, the cost of solving the PDE over the time interval $[0, T]$ is

$$\Theta(N^{1+1/d} \log N).$$

We study such Fourier-based methods as they minimize the cost of solving the PDE (for an implicit method) and allow us to assess the additional relative cost of collision detection in a worst case setting; for PDEs where Fourier methods cannot be employed we anticipate a lower relative cost for collision detection.

Under our previous assumptions on the particle distribution, the total cost of the collision detection is

$$\mathcal{O}(n^{1+1/d} \log n).$$

Thus, as $n = \Theta(N)$, that is the number of mesh points and the number of particles are kept proportional as they are increased, the cost of collision detection and PDE solution are comparable to one another, and the total complexity is

$$W_3 = \mathcal{O}(n^{1+1/d} \log n).$$

In particular, the cost of collision detection does not add asymptotically to the cost of solving the coupled problem. Experiments on the fully coupled algorithm confirm the main result of this analysis, that the two parts are comparable.

3.6.2 Fast Numerical Techniques

We develop efficient numerical solution methods for the linear systems arising from, for instance, the spray simulation, interface problems, wave equations. The preconditioned conjugate gradient linear solver and the multiple right-hand side solver are the joint work of Gene Golub and Wing-Lok Wan; the multigrid method for solving interface problems are developed by Wing-Lok Wan; and the hyperbolic multigrid methods are joint work of Antony Jameson and Wing-Lok Wan.

Preconditioned conjugate gradient

In each time step of the above computations, we need to solve a linear system of the form:

$$(A + N)x = b, \tag{3.18}$$

where A is the discrete Laplacian, and N is a rank n matrix arising from the coupling between the fluid and the particles. Thus, we can consider it as a rank n modification of the standard discrete Laplacian which can be solved efficiently by FFT in $\Theta(N \log N)$ operations. If we precondition the linear system by A and solve the resulting system by conjugate gradient, it will converge in at most $n+1$ steps since $\text{rank}(N) = n$. This has been implemented in the fully coupled PDE-ODE system together with collision detection.

Multiple right-hand sides

In time dependent problems as well as in other applications such as wave scattering, one needs to solve linear systems with multiple right-hand sides:

$$Ax^{(i)} = b^{(i)}, \qquad i = 1, \ldots, s.$$

The goal is to speed up the entire solution process without actually solving the linear system s times. When A is symmetric positive definite, an efficient Galerkin projection method has been analyzed by Chan and Wan [66]. We generalized the method to the case when A is complex symmetric. The projection method generates a Krylov subspace from a set of direction vectors obtained by solving one of the systems, called the seed system, and then projects the residuals of other systems orthogonally onto the generated Krylov subspace to compute an approximate solution. The whole

process is repeated with another unsolved system as a seed until all the systems are solved. In the real case, we proved two observations: (1) super-convergence behavior of the CG process using the projected initial guess compared with the usual CG process using a random initial guess, (2) r number of restarts is required to solve all the systems if the right-hand side matrix is rank $r \ll s$.

In the complex case, CG cannot be applied since the linear system may not be positive definite. We developed a new complex conjugate gradient method, analogous to BiCG in the real nonsymmetric case. The derivation is based on the tridiagonalization of A by a unitary matrix, guaranteed existence by the Takagi SVD theorem. The resulting tridiagonal matrix consists of only real entries along the upper and lower diagonals. To solve complex linear system, we approximate the solution by:

$$x_k = x_0 + Q_k T_k^{-1} Q_k^H r_0,$$

where T_k is tridiagonal, $Q_k^H Q_k = I_k$, and they are intermediate quantities obtained in the kth step of the tridiagonalization of A. If we implicitly factorize $T_k = L_k D_k L_k^T$, we can update x_k recursively by

$$x_k = x_{k-1} + \alpha_k p_k,$$

where α_k is a scalar and p_k is a vector. Thus we do not need to store both the matrix T_k and Q_k. We combined this iterative method with the Galerkin projection technique to obtain an efficient solver for solving complex symmetric systems with multiple right-hand sides. In a numerical experiment from real applications, only a total of 70 matrix-vector multiplications is required by our new method versus over 3000 matrix-vector multiplications if the right-hand sides were solved independently.

Multigrid for interface problems

Interface problems arise in a wide range of applications, for instance, in spray simulations where interfaces occur between the gas and the liquid gasoline. Traditional multigrid methods are efficient for PDEs with smooth coefficients, but can be extremely slow for interface problems due to the large jump in discontinuity at the interface. To recover the usual fast convergence of multigrid, the mainstream approach is to derive sophisticated interpolation operators [67]. Here, we developed an interface preserving coarsening [71], a geometric technique for handling discontinuous coefficients. In contrary to standard full coarsening, we select coarse grid points so that all the coarse grids are aligned with the interfaces for regular interface problems on Cartesian grids, and that the interfaces are resolved as much as possible for irregular interface problems. As a result, multigrid with linear interpolation is sufficient to obtain fast convergence. We show empirically that the convergence rate of the resulting multigrid method is independent of the mesh size and the size of the jump at the interface. Consider the following model problem commonly used to test multigrid convergence:

$$-\nabla \cdot a(x,y) \nabla u = 1,$$

where

$$a(x,y) = \begin{cases} a^+ & 0.25 - h \le x \le 0.75 - h \quad \& \quad 0.25 - h \le y \le 0.75 - h, \\ a^- & \text{otherwise.} \end{cases}$$

We fix $a^- = 1$ and vary a^+ from 10 to 10^4. The square interface is shifted by one fine grid point so that the interface does not align with any coarse grid if standard coarsening were used. In this example, we compare our method with AMG [68] which utilizes the strong/weak coupling for the algebraic coarsening instead of the interface location.

The convergence results are shown in Table 3.5. The new method compares favorably with AMG, but we do not need a setup phase for the construction and storage of a sophisticated interpolation operator.

	New MG Method			AMG			Standard MG		
h	10	10^2	10^4	10	10^2	10^4	10	10^2	10^4
1/16	5	5	6	7	7	7	14	*	*
1/32	5	6	6	7	7	8	14	*	*
1/64	6	6	6	8	9	9	14	*	*
1/128	6	6	6	8	9	10	14	*	*

Table 3.5: Number of V-cycles for a 2D square interface problem.

Multigrid for hyperbolic wave equation

In the approach of Jameson, Schmidt and Turkel [69], multigrid is used to accelerate the evolution of a hyperbolic system to a steady state by using large time step on coarse grids so that (low frequency) errors can be rapidly propagated away through the outer boundary. For instance, suppose the coarse mesh size is doubled from the current mesh, then a 3-level multigrid cycle consisting of one step on each mesh represents a total advance of

$$\Delta t + 2\Delta t + 4\Delta t = 7\Delta t$$

fine grid steps, where Δt is one time step on the fine mesh. This accelerated wave propagation property can be proved analytically.

In complement to the fast wave propagation viewpoint, the objective here is to accelerate the propagation on multiple grids while preserving TVD and monotonicity via upwind interpolation. As is well known, these two are fundamental properties of discretization schemes for solving hyperbolic equations.

To facilitate understanding, we primarily focus on the one-dimensional linear wave equation:

$$u_t + u_x = 0,$$

with first order (upwind) discretization scheme

$$u_i^{n+1} = u_i^n - \lambda(u_i^n - u_{i-1}^n),$$

where $\lambda = \Delta t/\Delta x$. In the two grid setting, Scheme 1 propagates the wave by one step on the fine grid and then restricts the residual on the coarse mesh based on cell averaging. The wave is then propagated by a bigger step on the coarse mesh. The coarse grid wave is interpolated back based on the upwind direction. The algorithm is summarized as follows:

Algorithm: Scheme 1 (two-grid)

$$\left.\begin{aligned} \overline{u}_i^{(1)} &= u_i^n - \lambda(u_i^n - u_{i-1}^n) \\ \overline{u}_i^{(2)} &= \overline{u}_i^{(1)} - \lambda(\overline{u}_i^{(1)} - \overline{u}_{i-2}^{(1)}) \\ \tilde{u}_i^{(1)} &= \overline{u}_i^{(2)} \\ \tilde{u}_{i-1}^{(1)} &= \overline{u}_{i-1}^{(1)} + \overline{u}_{i-2}^{(2)} - u_{i-2}^n \end{aligned}\right\} i = 0, 2, 4, \ldots$$

$$u_i^{n+1} = \tilde{u}_i^{(1)}$$

The extension to multiple grids is straightforward. Our analysis shows that Scheme 1 preserves monotonicity and TVD for two grids, but not true for multiple grids.

Theorem

Let u^n and u^{n+1} be the solution at the current and next time step, respectively. Suppose $u_i^n - u_{i-1}^n \geq 0$, for any i, and two-grid is used. Then

$$u_i^{n+1} - u_{i-1}^{n+1} \geq 0 \qquad \text{and} \qquad \text{TV}(u^{n+1}) \leq \text{TV}(u^n),$$

where $\text{TV}(u^n)$ denotes the total variation of u^n.

Scheme 2 restricts the wave on the fine mesh to all coarse coarse grids and then propagates the waves on all grids at the same time. The two-grid algorithm is as follows:

Algorithm: Scheme 2 (two-grid)

$$\left.\begin{array}{rcl} \overline{u}_i^{(1)} &=& u_i^n - \lambda(u_i^n - u_{i-1}^n) \\ \overline{u}_i^{(2)} &=& u_i^n - \lambda(u_i^n - u_{i-2}^n) \\ \tilde{u}_i^{(1)} &=& \overline{u}_i^{(2)} \\ \tilde{u}_{i-1}^{(1)} &=& \overline{u}_{i-1}^{(1)} + \overline{u}_{i-2}^{(2)} - u_{i-2}^n \end{array}\right\} i = 0, 2, 4, \ldots$$

$$u_i^{n+1} = \tilde{u}_i^{(1)}$$

In this case, we prove that Scheme 2 preserves monotonicity and TVD for *any* number of coarse grids.

Theorem

Suppose $u_i^n - u_{i-1}^n \geq 0$, for any i, and k-grid is used. Then

$$u_i^{n+1} - u_{i-1}^{n+1} \geq 0 \qquad \text{and} \qquad \text{TV}(u^{n+1}) \leq \text{TV}(u^n).$$

We note that the wave propagation is faster using Scheme 1 than Scheme 2.

The extension of the above schemes to higher dimensions, however, is not obvious. If the support of the wave on the fine mesh is on the noncoarse grid points only, then the restricted coarse grid function becomes identically zero. In light of this observation, we extended the one-dimensional schemes to two dimensions in conjunction with semi-coarsening techniques. Fast wave propagation was achieved.

Front Tracking/Capturing Computational Workbench

A computational workbench (problem solving environment) to develop robust second order accurate numerical algorithms describing incompressible fluid flows possessing a moving interface between different materials along which surface tension and other interfacial source terms are allowed to operate is currently being developed by SCCM student Doug Enright in consultation with Joel Ferziger (ME). The computational workbench will allow the user to use a variety of front tracking/capturing algorithms, including Peskin and Tryggvason's immersed boundary method and Sussman, Smereka, and Osher's level set implementation in an one-fluid Eulerian simulation. The surface tension forces calculated from the front are distributed to a single rectangular flow grid as a body force and the resulting fluid velocity is calculated. The calculated fluid velocity is then interpolated back onto the front and the front is moved accordingly. Incompressibility of the fluid is enforced during the pressure calculation stage of the simulation. A fractional step method is used to advance both the fluid and the front in time.

During the past year, the underlying hybrid C/FORTRAN software code for the computational workbench was produced in stages with validation tests performed at major milestones to ensure the proper functioning of the workbench. Some of the validation tests have included driven cavity

flow with results compared to those reported by Ghia et al. and a variety of interface stretching tests suggested by LANL scientists W.J. Rider and D.B. Kothe and D. Juric of Georgia Tech. Currently surface tension driven flows simulated by the workbench are being validated against a variety of theoretical results described by Lamb (1932) for two dimensional flows including standing capillary waves and small scale oscillations of the cross sectional section of a circular jet. After this series of validation tests are complete, work will progress along two paths, with novel algorithmic research occurring at the same time the computational capability of the workbench is extended to allow for the calculation of three dimensional axisymmetric flows. Simulation of high deformation droplet flows along with possible breakup will be performed in order to provide some challenging tests for the higher order accurate algorithms developed.

3.6.3 Parallel Numerical Methods/Intro. to Parallel Programming

During the Winter Quarter of the 1999-2000 academic year, the SCCM program offered an introductory course in parallel programming and parallel numerical algorithms. The course was taught by ASCI supported faculty including Juan Alonso (AA/SCCM), Justin Wan (SCCM), and Massimiliano Fatica (Center for Turbulence Research) and was well received by a number of high performance computing students and users at Stanford University, SLAC, and NASA/Ames. The course was intended to provide a computational scientist with an overview of current parallel computing architectures (distributed and shared memory) and programming environments (MPI/OpenMP) and how to analyze the performance of parallel programs and learn about ways to achieve parallel scalability as illustrated by examining several important numerical algorithms including the Fast Fourier Transform, graph partitioning, various matrix solution techniques, and Fast Multipole methods. The students completed several parallel programming projects and completed an in-depth parallel programming project at the end of the quarter.

3.6.4 DOE/HPC Seminar Series

The SCCM program hosted a series of seminars during the academic year that covered a variety of high performance computing and parallelization issues with various DOE laboratory scientists. Some of the speakers in the series included Tamara Kolda, Paul Boggs, and Ray Tuminaro of SNL(CA); Esmond Ng of LBNL/NERSC; Steve Ashby, Edmond Chow, and William Bosl of LLNL/CASC; Jack Dongarra of UTK/ORNL; and Mike Heath of the UIUC ASCI Center.

Bibliography

[1] Germano M., Piomelli U., Moin P. and Cabot W.H.,1991, A dynamic subgrid-scale eddy viscosity model, Physics of Fluid A,3,1760.

[2] C.D. Pierce & P. Moin, 1998, Large eddy simulation of a confined coaxial jet with swirl and heat release, *AIAA Paper* 98-2892.

[3] R. Roback & B.V. Johnston, 1983, Mass and momentum turbulent transport experiments with confined swirling coaxial jets, *NASA CR* 168252.

[4] M. Sommerfeld & H.H. Qiu, 1991, Detailed measurements in a swirling particulate two -phase flow by a phase - doppler anemometer. *Int. J. of Heat and Fluid Flow* 12(1):20 - 28.

[5] F. Nieuwstadt & H.B. Keller, 1973, Viscous flow past circular cylinders. *Computers and Fluids* 1:59 - 71.

[6] S. Taneda, 1956, Experimental investigation of the wakes behind cylinders and plates at low Reynolds numbers. *J. Phys. Soc. Japan* 11: 302

[7] M. Coutanceau & R. Bouard, 1977, Experimental determinatin of the main features of the viscous flow inthe wake of a cylinder in uniform translation. Part 1. Steady flow. *J. Fluid Mech.* 79: 231 - 256.

[8] R. Mittal, 1999, Private communication

[9] P. Beaudan & P. Moin, 1994, Numerical investigations on the flow past a circular cylinder at sub - critical Reynolds number. *Report TF - 62*, Mechanical Engineering Dept., Stanford University, Stanford, California.

[10] J. Westerweel, R.J. Adrian, J.G.M. Eggels & F.T. Nieuwstadt, 1992, Measurements with particle image velocimetry on fully developed turbulent pipe flow at low Reyolds number. *Proc. 6th Int. Symp. on Aplln. of Laser Tech. to Fluid Mech.*Lisbon, Portugal.

[11] J.G.M. Eggels, F. Unger, M.H. Weiss, R.J. Westerweel, R. Friedrich & F.T.M Nieuwstadt, 1994, Fully developed turbulent pipe flow: a comparisonbetween direct numerical simulation and experiment. *J. Fluid Mech.* 268: 175 - 209.

[12] P. Loulou, 1996, Direct numerical simulation of incompressible pipe flow using a B - spline spectral method. *Dissertation*, Stanford University, Stanford, California.

[13] K. Akselvoll & P. Moin, 1995, Large -eddy simulation of turbulent confined coannular jets and turbulent flow over a backward facing step. *Report TF - 63*, Mechannical Engineering Dept., Stanford University, Stanford, California.

[14] C.H.K Williamson, 1996, Vortex dynamics in the cylinder wake. *Ann. Rev. Fluid Mech.* 28: 477 - 539.

[15] R.D. Henderson, 1995, Details of the drag curve near the onset of vortex shedding. *Phys. Fluids* 7(9): 2102 - 2104.

[16] A.G. Kravchenko & P. Moin, 1998, B-spline methods and zonal grids for numerical simulations of turbulent flows. *Report TF - 73*, Mechanical Engineering Dept., Stanford University, Stanford, California.

[17] D.K. Lilly, 1992, A proposed modification of the Germano subgrid-scale closure method. *Phys. Fluids A* 4(3).

[18] S. Ghosal, T.S. Lund, P. Moin & K. Akselvoll, 1994, A dynamic localization model for large-eddy simulation of turbulent flows. *J. Fluid Mech.* 282: 1-27.

[19] C. Meneveau, T.S. Lund, W.H. Cabot, 1996, A Lagrangian dynamic subgrid-scale model of turbulence. *J. Fluid Mech.* 319: 353 - 385.

[20] Center for Integrated Turbulence Simulations. ASCI 1999 Technical Report (1998)

[21] H. Pitsch and H. Steiner. Large-eddy simulation of a turbulent piloted methane/air diffusion flame (Sandia flame D). Accepted for publication in *Phys. Fluids*, to appear Oct. 2000.

[22] H. Pitsch and H. Steiner. Scalar mixing and dissipation rate in large-eddy simulations of non-premixed turbulent combustion. Accepted for publication in *Proc. Comb. Inst.*, **28**, 2000.

[23] H. Pitsch, M. Chen, and N. Peters. Unsteady flamelet modeling of turbulent hydrogen/air diffusion flames. *Proc. Comb. Inst.*, **27**, pp. 1057–1064, (1998)

[24] H. Pitsch. Unsteady flamelet modeling of differential diffusion in turbulent jet diffusion flames. Accepted for publication in *Comb. Flame*, 1999.

[25] N. Peters. Laminar Diffusion Flamelet Models in Non-Premixed Turbulent Combustion. *Prog. Energy Combust. Sci.* **10**, 319–339 (1984)

[26] N. Peters. Laminar flamelet concepts in turbulent combustion. In *Proc. Comb. Inst.*, **21**, pp. 1231–1250, (1987)

[27] R. S. Barlow and J. H. Frank. Effect of turbulence on species mass fractions in methane/air jet flames. *Proc. Comb. Inst.*, **27**, pp. 1087–1095 (1998)

[28] N. Smith, J. Gore, and J. Kim. www.ca.sandia.gov/tdf/Workshop.html, 1998.

[29] Feikema, D. A., Everest, D., Driscoll, J. F., AIAA J., 34, pp. 2531–2538, 1996.

[30] C. Pierce and P. Moin. The Progress-Variable Approach for Large Eddy Simulation of Non-Premixed Combustion. *Bulletin of the Am. Phys. Soc.* vol. 44, no. 8, p. 44 (1999)

[31] P. Moin, C. Pierce, and H. Pitsch. Large eddy simulation of turbulent, nonpremixed combustion. *Advances in Turbulence VIII* , (Dopazo et al, Eds.), Proceedings of the Eighth European Turbulence Conference, Barcelona, pp. 385-392 (2000)

[32] A. W. Cook and J. J. Riley. A subgrid model for equilibrium chemistry in turbulent flows. *Phys. Fluids* **6**, 2868–2870 (1994)

[33] J. Jiménez, A. Liñán, M. M. Rogers, and F. J. Higuera. A-priori testing of subgrid models for chemically reacting non-premixed turbulent shear flows. *J. Fluid Mech.* **349**, 149–171 (1997)

[34] C. Pierce and P. Moin. A dynamic model for subgrid-scale variance and dissipation rate of a conserved scalar. *Phys. Fluids* **10**, 3041–3044 (1998)

[35] A. W. Cook, J. J. Riley, and G. Kosály. A Laminar Flamelet Approach to Subgrid-Scale Chemistry in Turbulent Flows. *Combust. Flame* **109**, 332–341 (1997)

[36] F. K. Owen, L. J. Spadaccini, and C. T. Bowman. Pollutant formation and energy release in confined turbulent diffusion flames. *Proc. Comb. Inst.*, **16**, p. 105, (1976)

[37] H. Pitsch and S. Fedotov. Investigation of Scalar Dissipation Rate Fluctuations in Non-Premixed Turbulent Combustion Using a Stochastic Approach. *Comb. Theory Modeling*, submitted (2000)

[38] P. Sripakagorn, G. Kosály, and H. Pitsch Flamelet Modeling of Local Extinction-Reignition. *Bulletin of the Am. Phys. Soc.*, submitted (2000)

[39] F. A. Williams. Turbulent Combustion. in *The Mathematics of Combustion*, (J. Buckmaster, Ed.), pp. 97–131 (1985)

[40] P. Pelce and P. Clavin. Influence of hydrodynamics and diffusion upon the stability limits of laminar premixed flames. *J. Fluid Mech.*, **124**, pp. 219–237 (1982)

[41] P. Clavin and F. A. Williams. Effects of molecular diffusion and of thermal expansion on the structure and dynamics of premixed flames in turbulent flows of large scale and low intensity. *J. Fluid Mech.*, **116**, pp. 251–282 (1982)

[42] F. A. Williams. *Combustion Theory.* Second Edition, The Addison-Wesley Publishing Company, Menlo Park (1985)

[43] N. Peters. A spectral closure for premixed turbulent combustion in the flamelet regime. *J. Fluid Mech.*, **242**, pp. 611–629 (1992)

[44] N. Peters. The turbulent burning velocity for large scale and small scale turbulence. *J. Fluid Mech.*, **384**, pp. 107–132 (1999)

[45] N. Peters. *Turbulent Combustion.* Cambridge University Press, Cambridge (2000)

[46] CITS ANNUAL TECHNICAL REPORT. Center for Integrated Turbulence Simulations. Stanford University, Department of Mechanical Engineering, Stanford, California 94305. October 1998.

[47] CITS ANNUAL TECHNICAL REPORT. Center for Integrated Turbulence Simulations. Stanford University, Department of Mechanical Engineering, Stanford, California 94305. October 1999.

[48] C. F. Edwards and C. T. Bowman. Stanford Thermosciences Division Liquid-Fueled Combustor Experiment, In Progress. Stanford University, Department of Mechanical Engineering, Stanford, California 94305. Contract Number NASA NAG2-1219, NASA Ames Research Center, Technical Monitor: Dr. N. N. Mansour.

[49] J. C. Oefelein and S. K. Aggarwal. A Unified High-Pressure Drop Model for Spray Simulations. *Proceedings of the Eighth Biennial Summer Research Program, 2000*, Center for Turbulence Research, Stanford University, Stanford, California 94305, 2000.

[50] L. P. Hsiang and G. M. Faeth. Near-Limit Drop Deformation and Secondary Breakup. *International Journal of Multiphase Flow*, 18(5):635–652, 1992.

[51] J. D. Kulick, J. R. Fessler and J. K. Eaton. On the Interactions Between Particles and Turbulence in a Fully-Developed Channel Flow in Air. Technical Report MD-66, Department of Mechanical Engineering, Stanford University, Stanford, California 94305, 1993.

[52] J. C. Oefelein. *Simulation and Analysis of Turbulent Multiphase Combustion Processes at High Pressures*. PhD thesis, The Pennsylvania State University, University Park, Pennsylvania 16802, May 1997.

[53] A. W. Adamson. *Physical Chemistry of Surfaces*. Wiley-Interscience Publications, New York, 1990.

[54] A. Belov. *A new multigrid-driven algorithm for unsteady incompressible flow calculations on parallel computers*. PhD thesis, Princeton University, June 1997.

[55] R. Clift, J. R. Grace, and M. E. Weber. *Bubbles, Drops, and Particles*. Academic Press, 1978.

[56] D. S. Dandy and L. G. Leal. Buoyancy-driven motion of a deformable drop through a quiescent liquid at intermediate Reynolds numbers. *J. Fluid Mech.*, 208:161–192, 1989.

[57] M. Dubiner. Spectral methods on triangles and other domains. *J. Sci. Comput.*, 6(4):345–390, 1991.

[58] L. P. Hsiang and G. M. Faeth. Near limit drop deformation and secondary breakup. *Int. J. Multiphase Flow*, 18(5):635–652, 1992.

[59] T. J. R. Hughes and M. Mallet. A new finite element formulation for computational fluid dynamics: Iii. the generalized streamline operator for multidimensional advective diffusive systems. *Comp. Meth. App. Mech. Eng.*, 58:305–328, 1986.

[60] A. Jameson. Multigrid algorithms for compressible flow calculations. In U. Trottenburg and W. Hackbusch, editors, *Proceedings of the Second European Conference on Multigrid Methods*, volume 1228 of *Lecture Notes in Mathematics*, pages 166–201. Spring-Verlag, 1986.

[61] T. A. Johnson and V. C. Patel. Flow past a sphere up to a Reynolds number of 300. *J. Fluid Mech.*, 378:19–70, 1999.

[62] P. J. O'Rourke and A. A. Amsden. The tab method for numerical calculation of spray droplet breakup. Technical Report 872089, SAE, 1987.

[63] S. Sherwin. Hierarchical hp finite elements in hybrid domains. *Finite Elements in Analysis and Design*, 27:109–119, 1997.

[64] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114:146–159, 1994.

[65] S. O. Unverdi and G. Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comput. Phys.*, 100:25–37, 1992.

[66] Tony F. Chan and Wing Lok Wan. Analysis of projection methods for solving linear systems with multiple right-hand sides. *SIAM J. Sci. Comput.*, 18:1698–1721, 1997.

[67] Tony F. Chan and Wing Lok Wan. Robust multigrid methods for nonsmooth coefficient elliptic linear systems. Technical Report 99-08, SCCM Program, Stanford University, 1999. To appear in the special issue of *J. Comput. Appl. Math.* on *Numerical Analysis in the 20th Century.*

[68] GMD software. `http://mgnet.ccs.uky.edu/mgnet/Codes/gmd`.

[69] A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time stepping schemes. AIAA paper 81-1259, 1981.

[70] Andrew Stuart, Hersir Sigurgeirsson, and Wing Lok Wan. Collision detection for particles in a field. Submitted to *J. Comput. Phy.*, 2000.

[71] Wing Lok Wan. Interface preserving coarsening for elliptic problems with highly discontinuous coefficients. Technical Report 00-04, SCCM Program, Stanford University, 2000. To appear in *J. Numer. Lin. Alg. Appl.*

Chapter 4

Multi-Component Integrated Simulations

4.1 Motivation

The goal of the Multi-Component Integrated Simulation Project is to couple several components of a gas-turbine engine into a single simulation that will eventually include the entire compressor, combustor, and turbine flow paths. The prediction of multi-component phenomena, such as compressor/combustor instability, combustor/turbine hot-streak migration, and main/secondary flow ingestion/interaction, can be improved by directly coupling the simulations of all components. The current effort has focused on coupling the gas-turbine combustor and turbine. The hot-gas flow exiting the combustor is the key phenomena responsible for the life (durability) of the turbine. This is particularly true for the high pressure turbine blade(s) and turbine disks. This hot-gas usually exits the combustor with a radial and circumferential total temperature profile. Since the total pressure exiting the combustor is more uniform, the higher total temperature region of the profile acts like a hot jet with higher velocity than the gas on the outer portions of the combustor profile. This high velocity, hot-gas jet from the combustor creates a flow incidence variation on the turbine rotor blades so that the hot-gas jet from the combustor usually migrates to the pressure side of the turbine rotor passage. This effect causes a significant time-averaged temperature difference across the turbine rotor and as a result, a severe thermal load on the blade(s) which can lead to accelerated fatigue. In addition, the hot-gas flow from the combustor can be ingested into the disk regions under the hub through the gaps between the stators and rotors in the turbine. The accurate prediction of the interaction between the combustor and turbine, thus can be very important in terms of increasing the life cycle of the turbine.

Since the coupling of gas-turbine components involves the coupling of multiple codes that simulate different flow physics, a second goal of the Multi-Component Integrated Simulation Project is to develop the capability/software to link multiple high-fidelity analyses together. Communication between the various analyses required to perform multi-component simulations can take several forms depending on the flow physics, temporal requirements, and numerical stability. The development of a multi-code interface that provides the required accuracy and numerical stability for multi-component gas-turbine simulations is a critical element of this effort. Moreover, the coupling between codes must be done in a way that parallel computing efficiency and scalability of the integrated code is preserved.

Finally, another important motivating factor for attempting a simulation of coupled components is to develop the means to perform very-large-scale calculations in terms of data handling, parallel computing, and engineering data processing on a routine basis. It is expected that simulations like those mentioned above will require computational grid densities approaching 200 million grid

points with associated restart binary files of approximately 50 Gigabytes. Thousands of these restart binary files will be generated for an unsteady-flow simulation. This scope of simulation has never been attempted before in the gas-turbine industry due to the large resource requirement. The development of techniques to efficiently generate, store, compute, and post-process this vast amount of data will be important to help the gas-turbine industry transition to multi-component simulations.

4.2 Approach

Most of the research at Stanford for combustion simulation has focused on large eddy simulation (LES) and is not mature enough at this time to use in a multi-component simulation. Since the turbomachinery Reynolds-Averaged Navier-Stokes code developed at Stanford, known as TFLO, does not have the capability to perform reacting-flow simulations for combustion, a separate Reynolds-Averaged, chemically-reacting Navier-Stokes procedure from the NASA-Glenn Research Center, known as the National Combustion Code (NCC), is being used initially to simulate the combustor portion of the engine. Members of the combustion research group at NASA-Glenn Research Center (GRC) have been invited to participate in the university/industry/government team performing this integrated simulation now known as the "Stanford Integrated Multi-Component Gas-Turbine Simulation Team".

The NCC code is typical of the Reynolds-Averaged Navier-Stokes solution procedures used in industry today. It has the capability to perform steady and unsteady solutions of non-reacting and reacting flows. The numerical algorithm consists of a 4-stage Runge-Kutta scheme along with a dual-time-step procedure for unsteady flows. The NCC code uses an unstructured-grid data structure that can be domain decomposed to take advantage of parallel computing. The NCC code has been benchmarked at NASA-GRC for a number of model problems. In the current effort, the NCC code has been applied to a combustor of a Pratt & Whitney's commercial engine. Details of the NCC combustor simulation will be described below in Section 4.3.

The TFLO code, developed at Stanford, has the capability to perform steady and unsteady solutions in multi-stage turbomachinery. As previously described, the numerical algorithm used in TFLO consists of a 5-stage Runge-Kutta procedure with multiple-grid, grid sequencing, implicit residual smoothing and enthalpy damping convergence acceleration procedures. It also uses the dual time-stepping technique for the solution of unsteady flows. The TFLO code uses a multi-block, structured-grid data structure with point-matched grids between non-sliding blocks. The TFLO code has been used in the Integrated Multi-Component Gas-Turbine Simulation project to compute the coupled high- and low-pressure turbine components of the same Pratt & Whitney commercial engine.

The integration layer of the NCC and TFLO codes is currently being debugged on several model problems. The code coupling effects will be discussed later in this section. Once the two codes have been interfaced, the combustor and entire turbine components will be coupled and compared with the baseline, isolated simulation results to quantify the coupling effects in terms of aerodynamic performance, computational resources, and numerical issues.

4.3 Baseline Isolated Component Simulations

The simulation of the isolated combustor and turbine components is an important step in the multi-component integration. These cases are not only used to quantify the coupling effects, but can also be used as initial conditions for the multi-component simulation.

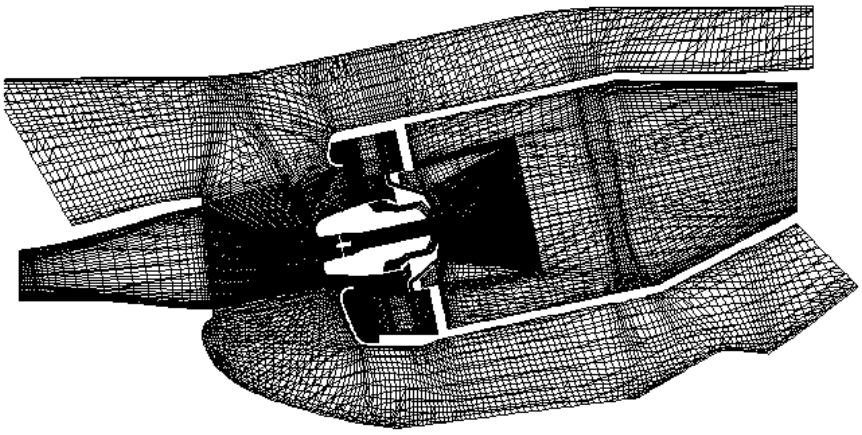


Figure 4.1: Computational Grid for Initial Pratt & Whitney Combustor Simulation

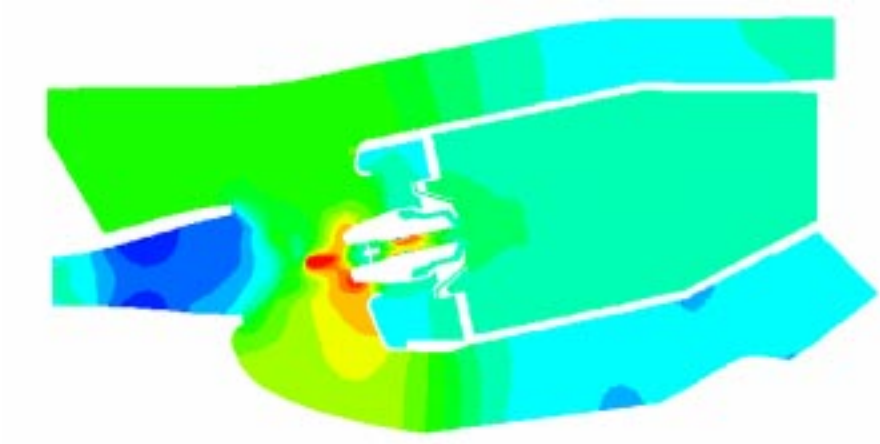


Figure 4.2: NCC Preliminary Prediction of Static Pressure Along Mid-Circumferential Slice for Pratt & Whitney Combustor

4.3.1 Pratt & Whitney Combustor

The NCC is currently being used to compute the steady, non-reacting flow for the Pratt & Whitney combustor. Once the non-reacting flow simulation has been completed, chemistry will be added and the reacting-flow solution will be obtained.

Steady, Non-Reacting Flow

Figure 4.1 shows a mid-circumferential slice of the computational grid through the Pratt & Whitney combustor. The combustor consists of a pre-diffuser at the combustor inlet, the fuel nozzle sting, fuel nozzle, burner and burner-bypass. The computational grid shown in Fig. 4.1 was generated as a multi-block structured grid and subsequently converted to an unstructured file for the NCC. A velocity profile corresponding to the exit of the compressor was specified at the inlet to the combustor. The static pressure corresponding to the inlet of the turbine was held constant at the exit of the burner. The mass flow that exits the burner-bypass at the inner and outer diameters was specified.

The static pressure contours from a preliminary non-reacting, steady flow simulation of the combustor are shown in Fig 4.2. The computational grid used in this simulation, shown in Fig. 4.1,

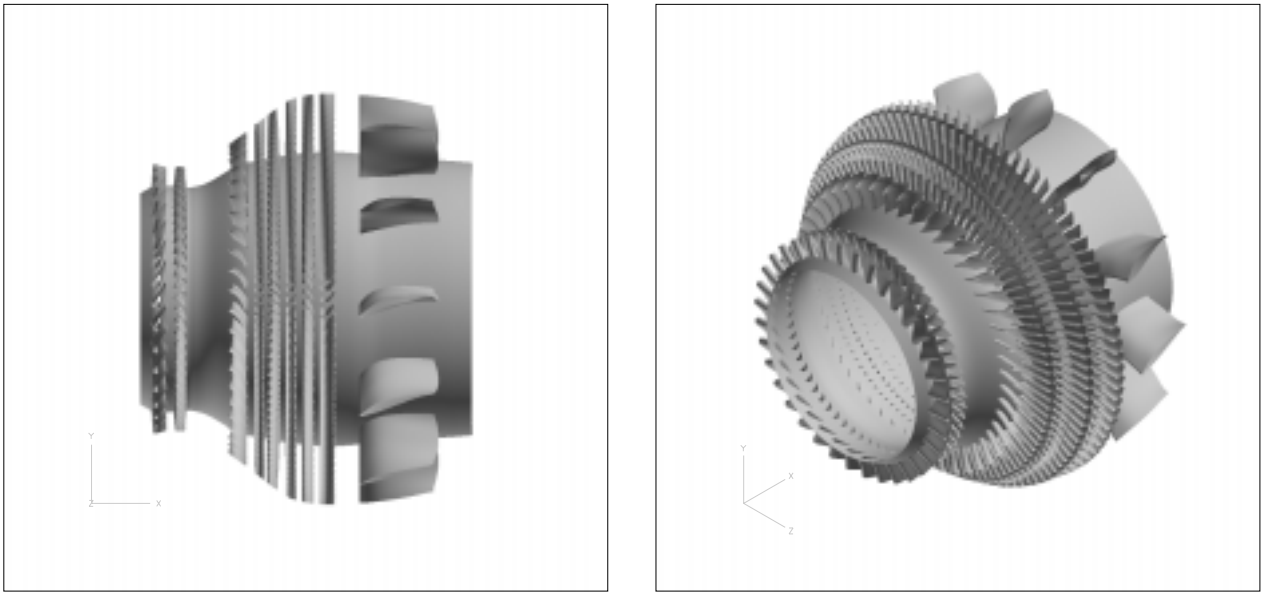


Figure 4.3: Pratt & Whitney Turbine

has regions of large skew and large stretching. This has led to numerical stability problems with the NCC which have forced the use of small local time-steps and therefore have resulted in long solution times. The pressure contours shown in Fig. 4.2, although not converged to a final steady-state solution, indicate that the pressure field is smooth and continuous over the domain. Work is now underway to improve the computational grid to reduce or eliminate severe grid skewness and stretching problems. In addition, various strategies are continuing to be explored with the developers of NCC using the existing computational grid.

4.3.2 Pratt & Whitney High- and Low-Pressure Turbine

The first integration of multiple components in this effort has focused on the coupling of the high- and low-pressure turbines for a Pratt & Whitney commercial engine. Figure 4.3 shows the single-stage high pressure turbine (vane and blade), followed by a transition duct, then followed by three stages of the low-pressure turbine and the exit guide vane. The low-pressure turbine rotors spin in the opposite direction of the high-pressure rotor. The special multi-code interface treatment, to be discussed further below, was not required to couple the high- and low-pressure turbines since both were computed with the TFLO code. Instead, the mixing-plane and sliding grid interface treatment built within the TFLO code itself was used. A steady-flow simulation has been completed using the TFLO code and is currently being compared with engine data as well as with Pratt & Whitney's 3DFLOW code predictions. In addition, the TFLO code is being used to simulate the unsteady flow through the entire high- and low-pressure turbines. Results from the steady flow and preliminary results from the unsteady flow simulations will be shown below.

Steady Flow

The computational mesh in the axisymmetric plane used for the TFLO code is shown in Fig. 4.4. The computational grid in the blade-to-blade as well as the blade tip region is similar to that previously shown for the Pratt & Whitney turbine rig. The computational grid dimensions used for each blade row is given in Tab. 4.1. A single passage for each blade row is used in the steady flow simulation. The multi-stage steady flow was predicted with the TFLO code using 196 processors of

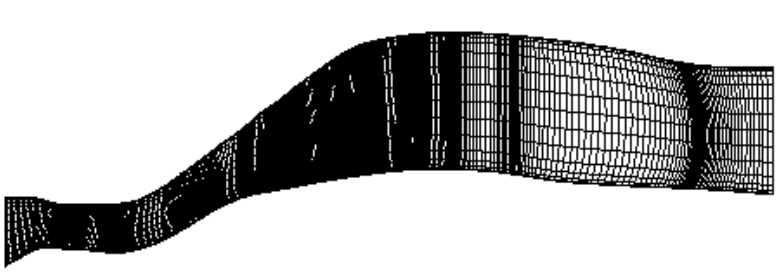


Figure 4.4: Computational Grid (Axisymmetric Plane) For Pratt & Whitney Turbine (every fourth point shown)

Vane1	185 x 57 x 81
Blade1	265 x 57 x 81
Blade Tip	89 x 33 x 17
Vane2	241 x 57 x 81
Blade2	153 x 57 x 81
Vane3	169 x 57 x 81
Blade3	153 x 57 x 81
Vane4	169 x 57 x 81
Blade4	185 x 57 x 81
EGV	313 x 57 x 81
Total	8,512,890 Points

Table 4.1: Computational Grid Dimensions For Pratt & Whitney Turbine Steady Flow Simulation

the Lawrence Livermore Blue Pacific IBM-SP2 computer. Approximately 170 hours of wall clock (33,250 hours of CPU time) were required for the TFLO steady solution.

The Pratt & Whitney 3DFLOW code was also executed for this case using the same computational grid in order to provide a consistency check. Comparisons between the two codes in terms of the pressure distributions, relative total pressure losses, as well as stage and overall efficiency are currently underway.

Figure 4.5 shows a visualization of the predicted surface entropy contours from the TFLO simulation. Entropy is used here to highlight areas of pressure and temperature change that can be attributed to losses in aerodynamic performance. As shown in Fig. 4.5, the contours of entropy are not continuous across the inter-blade mixing planes. The flow between blade-rows is circumferentially mixed in order to provide circumferentially uniform boundary conditions for each blade row. In essence, the spatial "average-passage" (without unsteady and body-force effects) flow for each blade row is modeled with this treatment. The steady flow mixing plane treatment allows each blade row to adjust its flow based upon its adjacent upstream and downstream blade row flows until the entire turbine comes to equilibrium.

Unsteady Flow

The unsteady simulation of the Pratt & Whitney high- and low-pressure turbine components is currently underway. This is the first time that the unsteady-flow for an entire turbomachinery

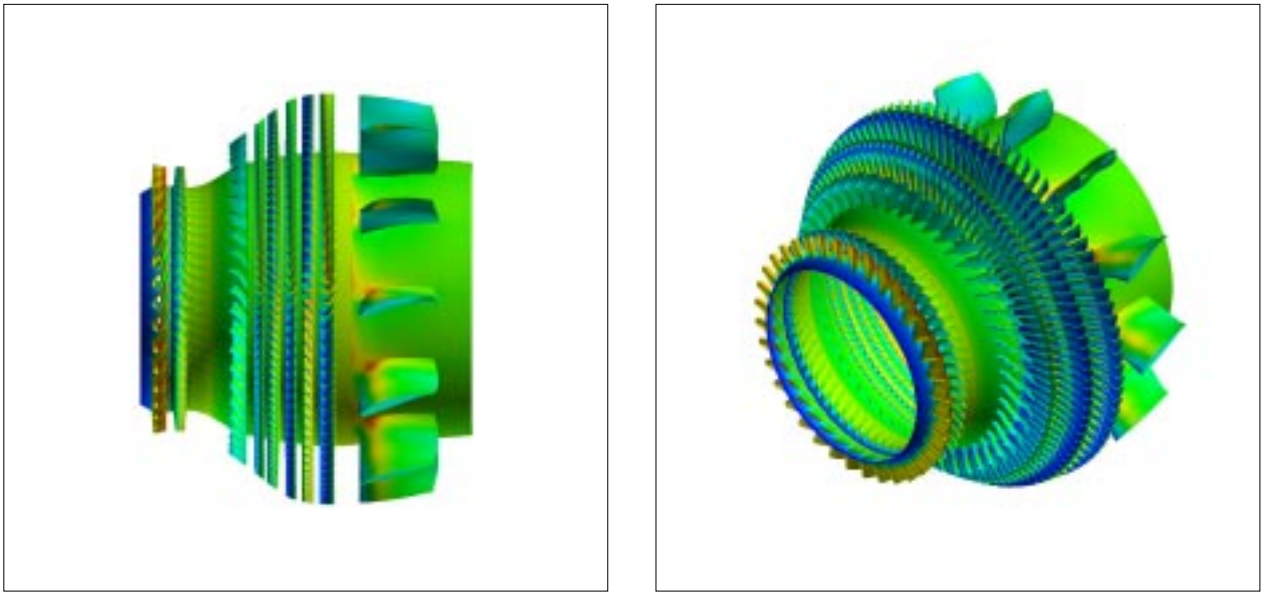


Figure 4.5: TFLO Predicted Surface Entropy Contours for the Pratt & Whitney Turbine

component has been simulated. As in the steady flow simulation, a total of nine blade rows were included. However, in order to arrive at a common circumferential pitch (domain) for all nine blade rows, the unsteady simulation required that $\frac{1}{6}$th of the engine circumference be included. This requirement greatly increased the size of the computation and therefore, the number of computer processors required. In addition, approximately 100 time-steps are required to resolve the temporal solution for a single vane/blade passing. This additional temporal requirement leads to long solution times in order to achieve time-periodic solutions. Table 4.2 provides the number of grid points and the number of computational blocks used for each blade row for this simulation. Approximately four blocks were placed on each of the 512 processors used on the Lawrence Livermore Blue Pacific IBM SP2. A total of 30 pseudo-time (inner) iterations were used for each global time-step for the implicit, dual-time step algorithm previously described. Approximately 3000 time-steps will be required to reach a time-periodic solution and time-average the flow. Each global time-step requires approximately 1.3 hours of wall-clock time (671 CPU hours).

Figure 4.6 shows the instantaneous contours of entropy at a cylindrical shell located at the mid-span of the turbine. The wakes from each blade row can be seen to pass through the various inter-blade row computational planes with little or no effect. As a result, viscous flow wake/blade interaction occurs in each row leading to unsteady pressure fields. At the aft-end of the turbine, where many wakes of various strengths are super-imposed, the resulting frequency content of the unsteady pressure field is expected to be high.

In addition to the wake/blade interaction, each blade row's pressure field is excited from the potential field of it's adjacent blade-rows. In the high-pressure turbine, the flows are transonic and the shocks from the trailing edge shock systems impact the downstream blade-rows leading to additional excitation. Figure 4.7 shows contours of the instantaneous pressure field of the turbine. The trailing edge shocks from the high-pressure turbine can be seen, especially from the blade where it propagates to the downstream first vane of the low-pressure turbine.

Once the current simulation has reached time-periodicity, the flow field will be time-averaged over the global cycle (period) and compared with the steady-flow simulation to quantify the effects of the flow unsteadiness on performance. In addition, the unsteady pressure field will be examined for each blade-row to determine the peak loads on the airfoils.

Blade-Row	**Points**	**Blocks**
Vane1	5,124,870	96
Blade1	12,235,050	240
Blade Tip	499,390	140
Vane2	8,901,576	224
Blade2	12,008,817	272
Vane3	14,825,187	304
Blade3	12,008,817	272
Vane4	12,484,368	256
Blade4	12,812,175	300
EGV	2,890,242	88
Total	93,790,392	2,192

Table 4.2: Computational Grid Dimensions For Unsteady Flow Simulation of Pratt & Whitney Turbine

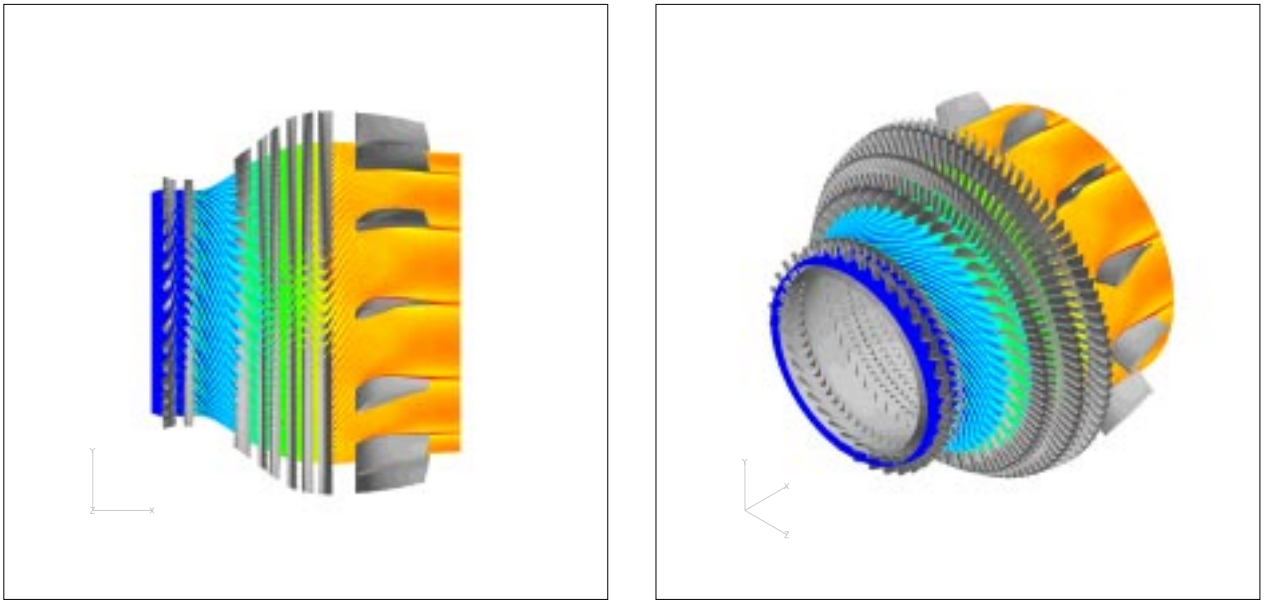


Figure 4.6: TFLO Predicted Instantaneous Entropy Contours for the Pratt & Whitney Turbine

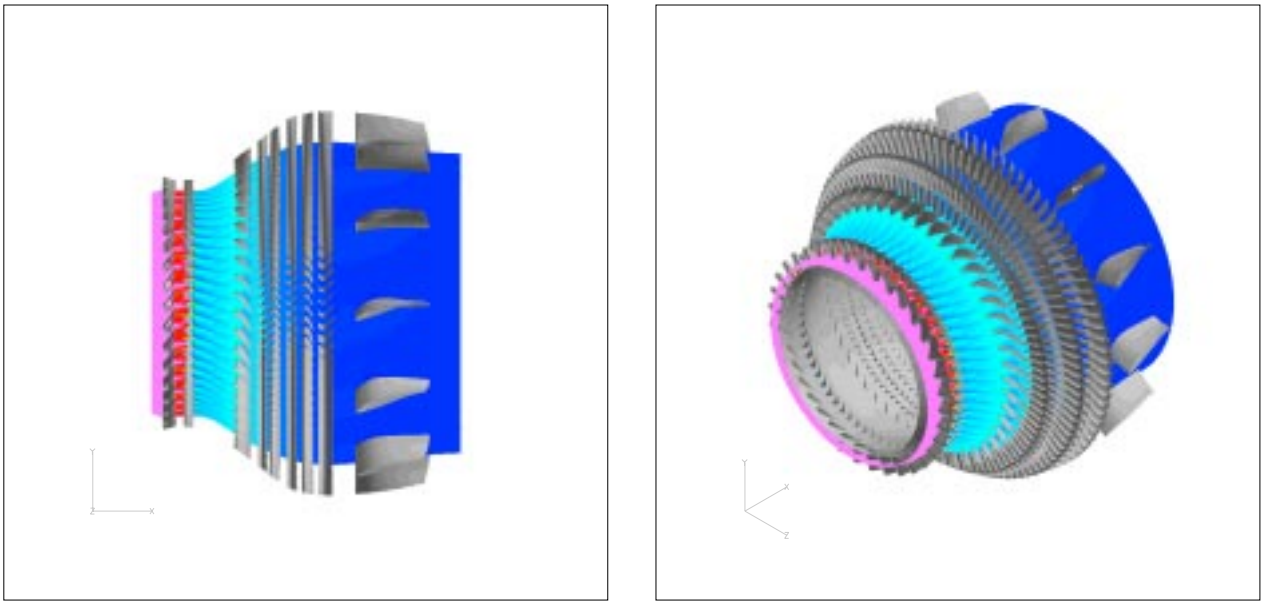


Figure 4.7: TFLO Predicted Instantaneous Pressure Contours for the Pratt & Whitney Turbine

4.3.3 Scalability of TFLO Code for Parallel Computing

The steady- and unsteady-flow simulations were carried out on the Lawrence Livermore Blue Pacific IBM SP2 using a wide range of processors to document the scalability of the TFLO code. The load balancing algorithm in TFLO associates complete blocks to a given processor. The domain decomposition of the grids in the various blade-rows was performed with the intent to create nearly uniform size blocks over the global domain in order to optimize parallel computing performance. Because of the varying block dimensions, exact load balancing is not always possible, leading to a slight degradation in performance that is unrelated to the parallel implementation of TFLO.

Figure 4.8 shows the non-dimensionalized solution times for both the steady- and unsteady-flow simulations as a function of the number of processors used on the Lawrence Livermore Blue Pacific IBM SP2. Non-dimensionalization of the wall-clock time by the number of iterations and number of grid points is required to present a continuous curve since the two simulations were very different in size. For the cases with larger numbers of processors, a much finer mesh was used. In a sense, the scalability results presented here are intended to follow the definition of the isoefficiency metric.

Figure 4.8 shows three actual TFLO timing curves in reference to the ideal curve (shown in red and assuming perfect scalability). The top (blue) curve shows the first benchmark of the TFLO code for the turbine configuration. This curve, which includes both the computational and input/output time to read and write restart files, shows that the TFLO code leveled off in performance as the number of processors increased to over 1,000. Investigation showed that the TFLO code was spending approximately 20% of its wall-clock time in reading and writing the restart files. This large time was a consequence of serial I/O procedures.

As a result, an effort was launched to incorporate IBM's version of MPIIO or Parallel I/O in which each processor writes it's portion of the restart file (solution domain) to what is perceived as one file. The use of the Parallel I/O software resulted in significant time savings in writing the restart files. The middle (pink) curve, shown in Fig. 4.8 shows the new benchmark TFLO solution times using the Parallel I/O routines.

Finally, the non-dimensional wall clock time for the TFLO simulations that only includes the computational time is also shown in this figure (green curve) to help quantify the time necessary to initialize the flow, read in the input files, and write out the restart files. The increase in time

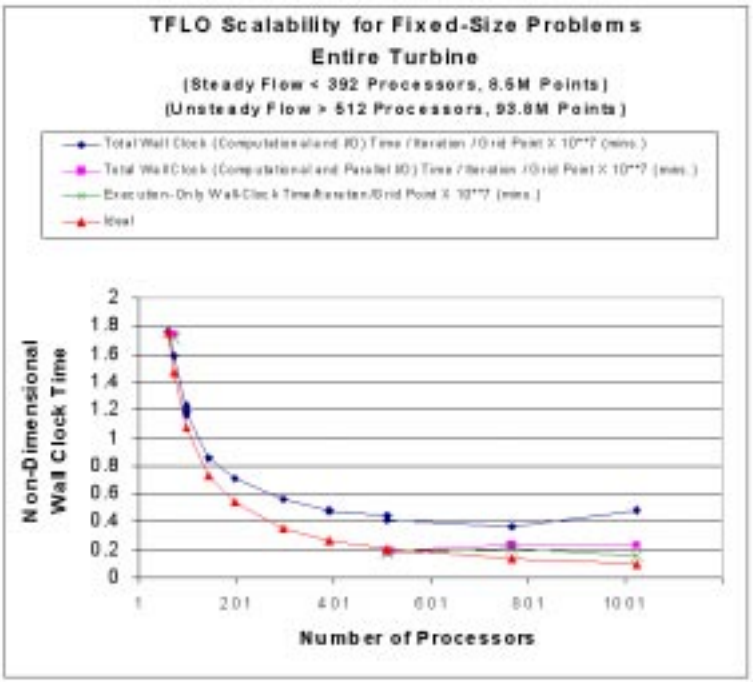


Figure 4.8: TFLO Scalability for Steady- and Unsteady-Flow Simulations of Pratt & Whitney Turbine

for the 768 processor simulation is believed to be due to less efficient load balancing compared to the 512 and 1024 processor simulations. The new TFLO benchmarks show that the code does a very good job of scaling up to 1000+ processors and, for the larger mesh size, only begins to lose efficiency with numbers of processors larger than 512 processors mainly due to shortcomings in I/O performance.

4.4 Multiple-Code, Multi-Component Integrated Simulations

A general-purpose interface procedure has been developed to handle the data transfer between the NCC and TFLO codes. A description of this interface procedure can be found at the CITS web site.

In the simulation of both steady- and unsteady-flow, all primary variables are exchanged between adjacent blade rows. This is performed through the application of an overlaid-grid procedure in which ghost nodes beyond the true computational domain are used in each code. For the integrated combustor/turbine simulation, ghost nodes for the NCC code extend beyond the exit plane of the combustor into the inlet region of the turbine (within the grid domain of TFLO). All of the primary variables at these ghost nodes are interpolated from the TFLO grid to the NCC grid, transformed into NCC variables, and then used as part of the NCC computation. Likewise, all of the primary variables that exist at the ghost nodes of the TFLO code, that extend upstream of the turbine inlet into the combustor domain, are interpolated from the NCC grid to the TFLO grid, transformed into TFLO variables, and then used as part of the TFLO computation. This interface treatment is performed at every steady-flow iteration and unsteady-flow inner iteration in the dual time-step scheme. In conjunction with the communication of primary variables across the interface, circumferential mixing boundary condition treatments may be used for steady-flow simulations to enable the reduction of the circumferential domain of each component to its lowest level.

In order to verify that the interface treatment is working properly, the interface routines were

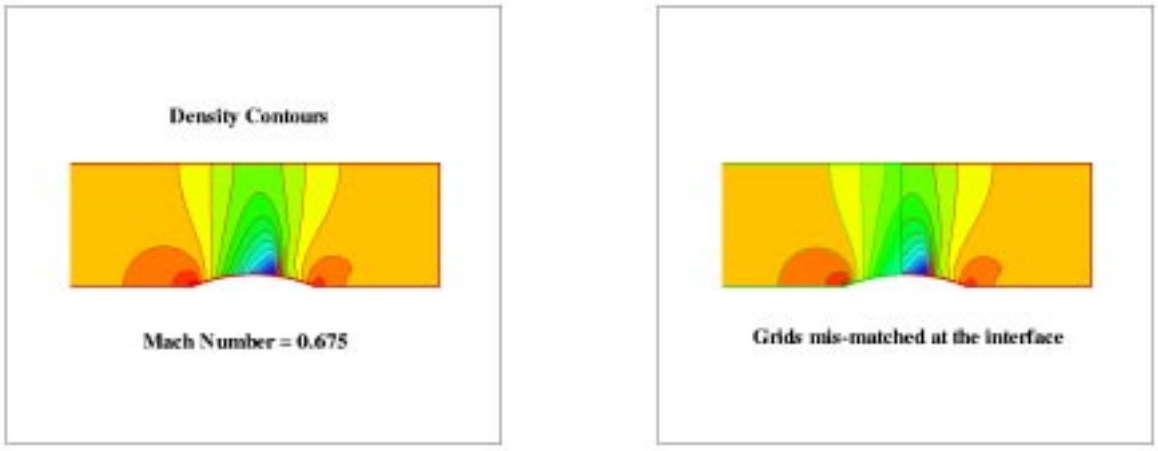


Figure 4.9: Density Contours from TFLO-TFLO Integration Via Multi-Code Interface for Inviscid Transonic Flow Over Bump (Interface Locate at Center of Airfoil)

used to first couple TFLO-to-TFLO and also NCC-to-NCC for a series of model problems with various types of inviscid and viscous phenomena passing through the interface. Figures 4.9-4.11 show the results from the TFLO-to-TFLO coupling using the new interface treatment for the inviscid, transonic flow over a circular-arc airfoil, the inviscid, supersonic flow over a ramp, and the viscous, subsonic flow over a circular arc airfoil. The solution using a single block (without the interface) are shown along with the solution using two blocks with the interface treatment to communicate between the blocks. The computational grids used in the 2-block simulations were purposely mismatched in the cross-flow directions in order to test the capability of the interface to handle disparate grids. These figures show that the TFLO solutions with the interface treatment are essentially the same as those TFLO solutions with a single block demonstrating that there is little or no error associated with using the new interface. At the moment, no effort has been made to make sure that this interface treatment is fully conservative. However, in the conception of the data exchange, the provision has been made to request additional information across the interface that may be used to devise coupling schemes that conserve mass, momentum, and energy.

Figures 4.12 and 4.13 show the results of coupling NCC-to-NCC using the new interface. Since the NCC is an unstructured-grid code, additional considerations were required to handle the numerical dissipation operators at the boundary of the interface. Figure 4.12 shows the pressure contours for the same inviscid, transonic flow circular-arc test case shown in Fig. 4.9. The solution using a single block (without interface treatment) is shown along with the solution using 2 blocks and the interface scheme to communicate between the blocks. This figure shows some minor discontinuities at the interface between the 2 blocks that have been attributed to the way in which the two-block solutions were averaged for plotting. Examination of the fluxes and primary variables at the interface have shown that there is essentially no difference in the solutions of the two blocks at the interface. Figure 4.13 shows the velocity contours for this same inviscid, transonic flow circular-arc test case with and without the interface treatment. The next step in the integration effort is to couple the TFLO code to the NCC directly and to demonstrate that these same model problems can be computed with no loss in accuracy. After this step, the TFLO and NCC codes will be used to couple the Pratt & Whitney combustor and turbine in a steady-flow simulation. A comparison will then be made with the isolated component simulations to quantify the predicted performance effects and required solution resources due to coupling.

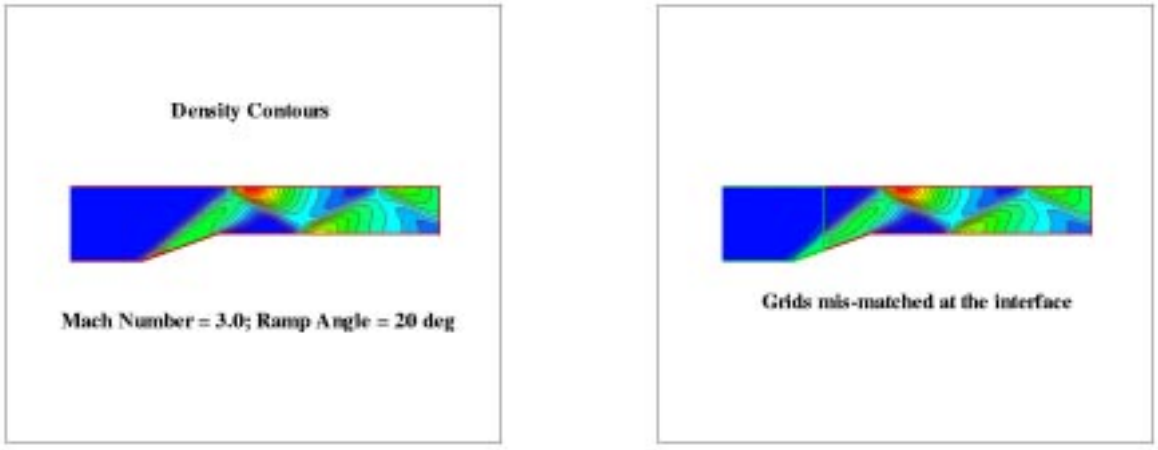


Figure 4.10: Density Contours from TFLO-TFLO Integration Via Multi-Code Interface for Inviscid Supersonic Flow Over Ramp (Interface Located at Center of Ramp)

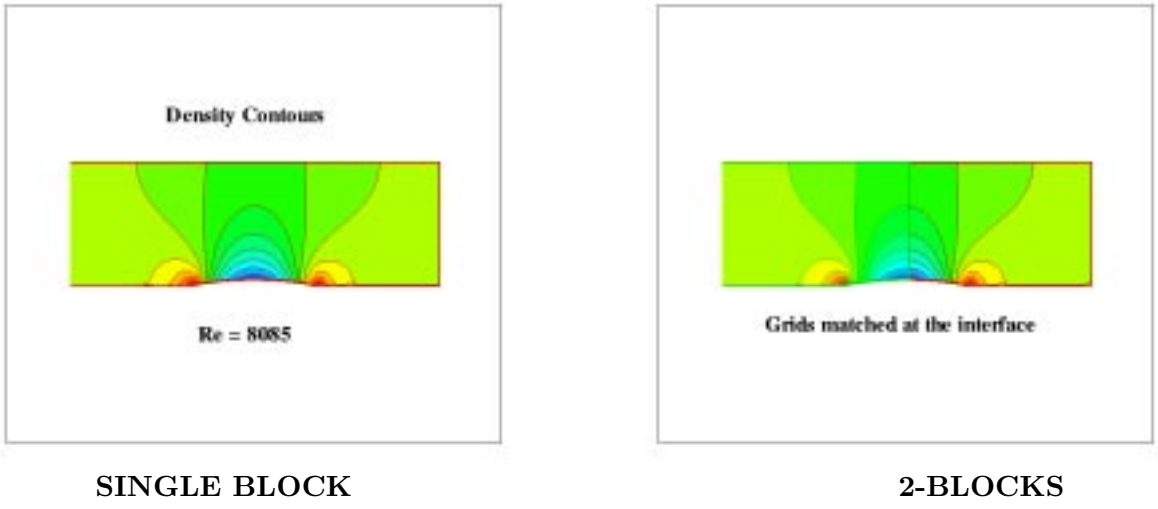
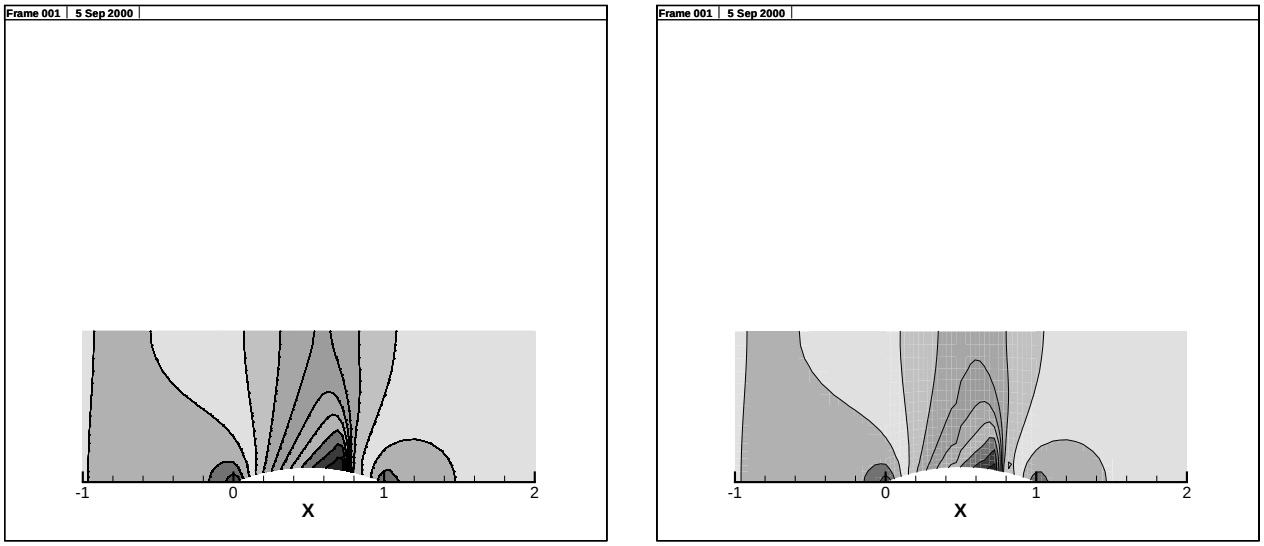
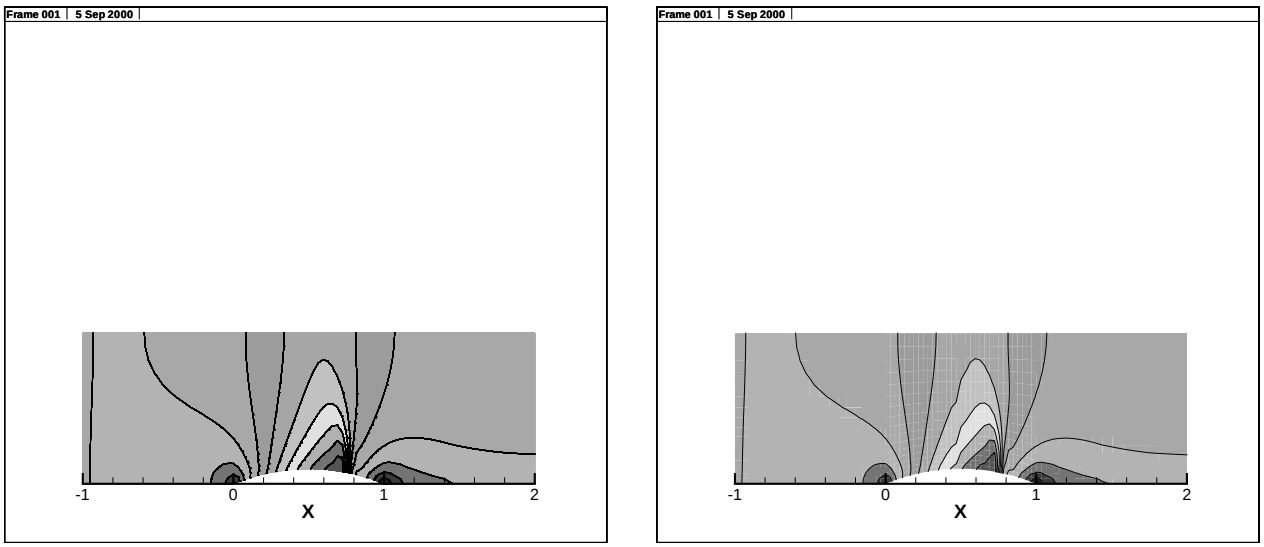


Figure 4.11: Density Contours from TFLO-TFLO Integration Via Multi-Code Interface for Viscous Subsonic Flow Over Bump (Interface Located at Center of Airfoil)



SINGLE BLOCK **2-BLOCKS**

Figure 4.12: Pressure Contours from NCC-NCC Integration Via Multi-Code Interface for Inviscid Transonic Flow Over Bump (Interface Located at Center of Airfoil)



SINGLE BLOCK **2-BLOCKS**

Figure 4.13: Velocity Contours from NCC-NCC Integration Via Multi-Code Interface for Inviscid Transonic Flow Over Bump (Interface Located at Center of Airfoil)

Chapter 5

Concurrent VLSI Architecture Group

The Concurrent VLSI architecture group is developing technology to enable the construction of large, scalable parallel computer systems. For the past year, this work has focused on three areas: high-speed signaling, router architecture, and construction of a prototype M-Machine parallel computer.

5.1 High-Speed Signaling

We are developing technology that increases the bandwidth per pin of CMOS chips by two orders of magnitude. We have developed several prototype signaling systems. The latest system demonstrates 4Gb/s signaling over a pair of I/O pads in a standard 0.25um CMOS technology. Equalization enables these signaling rates to be used over cables up to 10m in length. Our current design is power efficient, requiring only 90mW per link, and requires only $0.01mm^2$ of chip area. This low-power and area make it possible to put hundreds of these high-speed pads on a conventional CMOS chip enabling single-chip multicomputer routers with bandwidth approaching 1Tb/s.

5.1.1 Approach

We are using the capability of modern VLSI technology to overcome the bandwidth bottleneck it has created. Our 4Gb/s I/O pads overcome the limitations of conventional signaling systems through a combination of current-mode signaling, incident-wave switching, precision timing circuits, small aperture receivers, and high-speed equalization.

We have concentrated our efforts in two directions. First, we have focused on reducing the power and area of our 4Gb/s pads to make it feasible to put hundreds of these pads on a single chip. In parallel with this effort we have developed methods to operate at the highest possible bit rates over the longest possible distances.

Our drivers use differential, current-mode signaling. This results in power dissipation of as little as 20mW (for 5mA drive using a 2.5V supply) as compared to over 100mW for conventional systems. Self-adjusting parallel terminations at both ends of the line absorb all of the signal energy, giving incident wave signaling and eliminating most inter-symbol interference. By canceling, rather than overpowering most noise sources, this system gives excellent noise immunity with just a 250mV signal swing (500mV differential).

Operation with 250ps bit cells requires precision timing circuits. Our first prototype used very stable source-coupled delay lines to generate a 20-phase 400MHz clock with 125ps phase spacing. These clocks are used as timing references to sequence the bits onto and off of the line. The source-coupled circuits have very little timing sensitivity to supply and substrate noise and very little

fixed pattern jitter due to stage-to-stage mismatches. Our more recent prototypes use a regulated inverter delay line that retains the timing precision of the source-coupled circuits but with about one third the power dissipation.

Transmitter equalization compensates for the frequency-dependent attenuation due to the skin-effect in the transmission line and due to package parasitics. Our first prototype employed a 5-tap equalizing digital filter in the transmitter and used an output-multiplexed architecture with a 10:1 multiplexing ratio. The filters and digital-to-analog converters operate at 400MHz and are sequenced onto the line by a 10-phase 400MHz clock. While this architecture was effective, it was quite large, requiring about $0.5mm^2$ of chip area and power hungry, less than 30% efficient.

Our most recent prototype equalizes the line using a two-tap discrete-time analog filter and employs input multiplexing with a 4:1 multiplexing ratio. The input multiplexer operates at 1GHz and the remainder of the transmitter operates at the full 4GHz rate employing a novel active clamp circuit to boost bandwidth. This architecture results in a transmitter that is very power efficient and requires less than $0.01mm^2$ of chip area making it feasible to place hundreds of such transmitters on a single chip.

5.1.2 Recent Accomplishments

Over the past year we have completed testing of our first 4Gb/s prototype incorporating an input-multiplexed transmitter architecture and have completed the design of a new 4Gb/s prototype incorporating complete clock and data recovery.

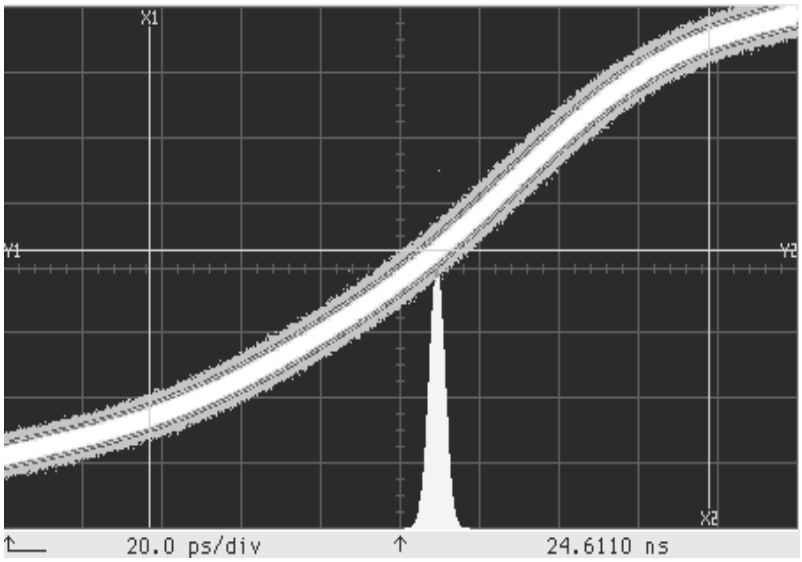


Figure 5.1: Jitter measurement of prototype transceiver shows 16.4ps peak-to-peak jitter.

Our input-multiplexed 4Gb/s transceiver worked well in the lab. We were able to demonstrate transmitter operation up to 5.3Gb/s. However, a speed-path in the on-chip PRBS checker limited receiver operation to 4Gb/s. As illustrated in Figure 5.1, transmitter jitter was measured at 16.4ps peak-to-peak with a clean power supply. As shown in Figure 5.2, the receiver jitter tolerance was measured to be better than 0.85UI, and reliable receiver operation was demonstrated with input signal swings down to 8mV.

The low jitter of this prototype demonstrated the feasibility of our low-power, low-jitter timing circuits based on CMOS inverter chains with regulated power supplies. These circuits give better jitter performance than conventional source-coupled delay line circuits at a fraction of the power.

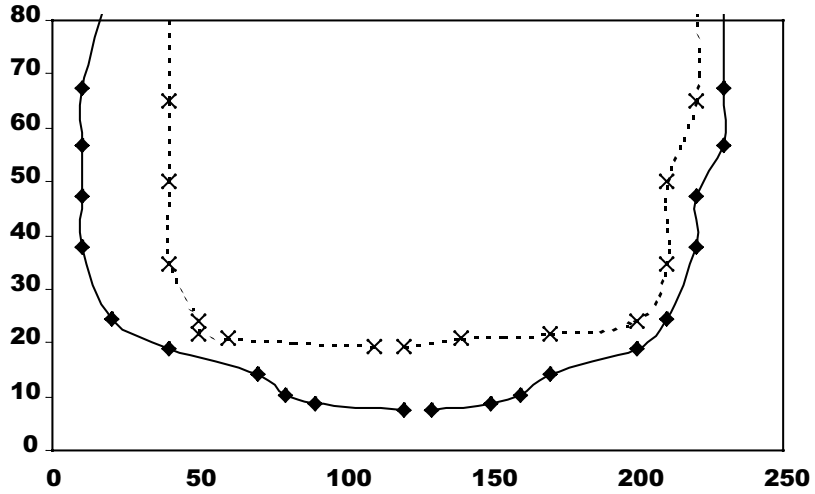


Figure 5.2: Measured receiver characteristics show jitter tolerance of better than 0.85UI and operation with input signal swing less than 8mV.

The receiver operation at very-low voltage swings demonstrates the effectiveness of our self-calibrating receiver design. This receiver uses a low-capacitance input stage that has an inherently high offset voltage and actively cancels this offset voltage using a self-calibration procedure.

The prototype incorporates transmitter preemphasis to equalize the frequency-dependent attenuation of typical transmission lines. This permits operation at 4Gb/s over 1m of printed-circuit board or up to 10m of twisted-pair cable. The eye diagrams of Figure 3 illustrate equalized operation. Without equalization the transmitter produces a clean eye (Figure 5.3a), but the frequency-dependent attenuation of the line completely closes the eye (Figure 5.3b). With equalization enabled, the transmitter preemphasizes the high-frequency components of the signal (Figure 5.3c). This compensates for the frequency-dependent attenuation of the line resulting in a good eye-opening at the far end (Figure 5.3d).

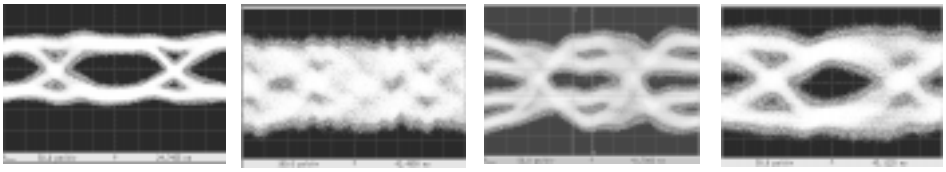


Figure 5.3: Eye diagrams of prototype transmitter (a) at transmit end of line without equalization, (b) at receive end of 1m PCB trace without equalization, (c) at transmit end of line with equalization, and (d) at receive end of line with equalization.

To build on the success of the input-multiplexed prototype transceiver, we have designed and laid out a new transceiver prototype. The new transceiver was submitted to fabrication in July and we expect to receive parts for testing in mid-September. The new transceiver incorporates all of the features of the input multiplexed transceiver described above, and adds a low-power clock recovery circuit. Details of our transceiver circuits are described in [6] and [7].

5.2 Router Microarchitecture

5.2.1 Flit-Reservation Flow-Control

Interconnection networks connect processors to memories in shared-memory multiprocessors and connect nodes to each other in message-passing multicomputers. Given a particular topology and routing strategy, the efficiency of an interconnection network is largely determined by its flow-control: the method used to allocate resources (buffer space and channel bandwidth) to the flits (flow-control digits) of packets traversing the network. The shape of the characteristic latency-throughput curve of a network is determined by flow control. A good flow control strategy enables a network to operate at 80% or more of bisection bandwidth with low latency. A poor flow-control strategy, on the other hand, results in a network that saturates at less than 30% of capacity.

We have recently developed flit-reservation flow control in which control flits traverse the network ahead of data flits reserving buffers and channel bandwidth. When the data flits arrive, they are forwarded or buffered according to the reservation. This advance scheduling makes very efficient use of buffers, allowing them to be reused immediately following the departure of a flit. In contrast, the on-the-fly scheduling of existing flow-control methods idles each buffer for a considerable period after each flit departure.

Reserving network resources ahead of the arrival of data flits yields two performance improvements. First, data latency is reduced (by 15.6% for an example network) because routing and arbitration are performed in advance of data arrival. Second, for a fixed amount of buffer space, saturation throughput is increased because of the more efficient buffer scheduling. For an 8x8 mesh network with eight flit buffers per input, virtual-channel flow control [3] saturates at 63% of bisection bandwidth while flit-reservation flow control extends performance to 77% of available bandwidth, a 20% improvement with equal storage and bandwidth overheads.

The packet format for a network using flit-reservation flow control is shown in Figure 5.4. This packet consists of five data flits which are scheduled by two control flits. The data flits contain no overhead information, only data payload. The head control flit contains a virtual channel identifier (VCID), the packet destination, and the arrival time td0 for the first data flit. Subsequent control flits carry the arrival times for four data flits each.

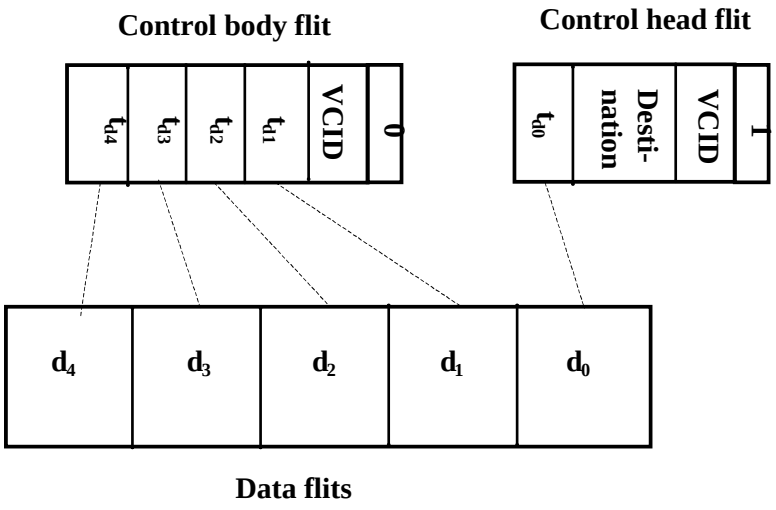


Figure 5.4: With flit-reservation flow control, control flits traverse the network ahead of data flits making buffer and link reservations for the data flits. The figure shows a packet with five data flits that are scheduled by two control flits. The head control flit schedules the first data flit and the second control flit schedules the remaining four data flits.

The operation of a flit-reservation router is illustrated in the datapath schematic of Figure 5.5. When a control flit arrives at a router it proceeds through the steps of routing, output scheduling, and input scheduling. First, the routing logic block uses the destination field of the control head flit to determine the output port to which this packet should be forwarded. This output port selection is stored in a table indexed by virtual channel identifier (VCID). Subsequent control body flits look up their output port in this table using their VCIDs. In parallel with determining the output port, the arrival times of each data flit associated with a control flit are recorded in the input reservation table. After completing the routing step, the control flit, annotated with the input port number is forwarded to the output scheduler for the selected output port.

The output scheduler schedules the departures of each data flit associated with the control flit. Its goal is to schedule the data flits to leave as soon as possible for the next hop. This scheduling is done using an output reservation table that records the status of each output channel (busy or not) and the number of free flit buffers at the far end of the channel. This status is kept for each clock cycle within a window of time from the present to a scheduling horizon. Once all of the data flits associated with a control flit have been scheduled, the control flit, updated with these new times, is forwarded over the link to the next node along the route. The control flit is also passed back to the input scheduler to complete the scheduling process for the current node.

While the output scheduler reserves the departure times of data flits, the input scheduler orchestrates the movement of a data flit through the router at the scheduled time. When the input scheduler receives reservation signals from the output scheduler, it updates the input reservation table to reflect the flit departure and sends a credit back to the previous node to update the buffer availability in that node's output reservation table. After a data flit has been scheduled, all its movements are tracked in the input reservation table. Each cycle, the table directs which buffer is to be written with data from the input link, and which buffers are to be driven onto which output links. There are no decisions to be made as all of the work has been done ahead of time by the control flits.

In the absence of contention, a data flit can easily depart the router the cycle after it arrives. This is because all of the decision logic: the routing and arbitration, has been performed in advance by the control flit. This pre-arranged control is responsible for the low data latency of flit-reservation flow control.

The performance advantages of flit-reservation flow-control are illustrated in Figure 5.6. The figure shows latency in clock cycles (on the vertical axis) as a function of offered traffic, fraction of bisection bandwidth, (on the horizontal axis). This latency-throughput relationship is plotted for three flow control cases: virtual-channel flow control with 8 flit buffers (VC8), virtual-channel flow control with 16 flit buffers (VC16), and flit-reservation flow control with 6 flit buffers (FR6). Because data flits are able to propagate through the router with no routing or arbitration delay, FR6 shows lower latency at low loads than either of the virtual-channel routers. The saturation throughput of FR6 is also greater than VC8 which has comparable storage requirements. This is because flit-reservation flow control makes more efficient use of buffer space, allowing buffers to be reused the instant they are vacated, while virtual channel flow control idles buffers during a round-trip delay over the link. More details on flit-reservation flow control can be found in [9].

5.2.2 Pipeline Model of Router Architectures

The performance of interconnection networks and hence of the systems in which they are employed depends critically on the performance of the routers from which these networks are constructed. Accurate performance models are needed to enable the architecture of these networks to be optimized. Existing router models include a number of assumptions that do not match current design practice and hence are poor predictors of router performance.

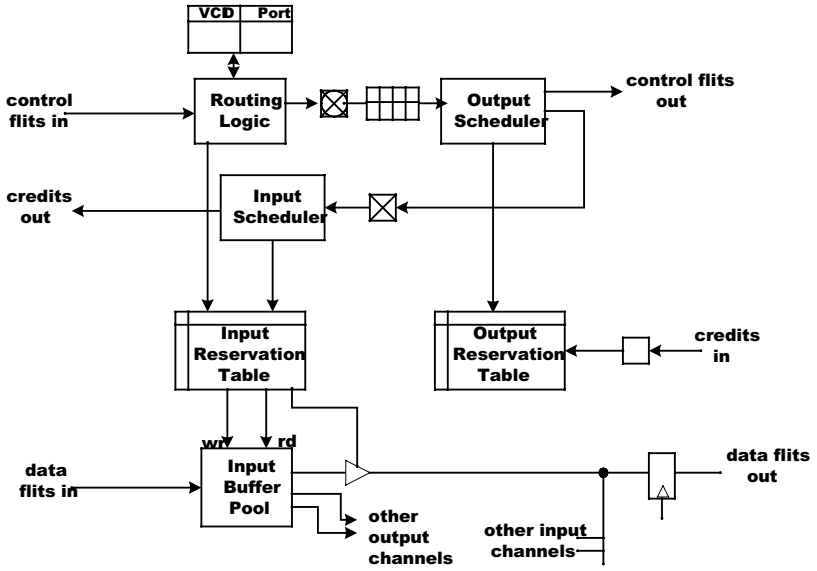


Figure 5.5: Datapath for a flit-reservation router. Control flits proceed through a path from the routing logic to the output scheduler and finally to the input scheduler making reservations for data flits. When data flits arrive, they are optionally buffered in the input buffer and then forwarded directly to the output.

To enable network designers to accurately predict performance and make tradeoffs early in the design process, we have developed an accurate pipelined router model. This model accounts for the pipelined nature of contemporary routers, the specific flow-control method employed, the delay of the flow-control credit path, and the sharing of crossbar ports across virtual channels. Our model uses technology-independent parametric equations for delay that are derived from detailed gate-level designs and analyses.

Motivated by our model, we have also developed a microarchitecture for a speculative virtual-channel router. In a conventional virtual-channel router, an arriving packet must first arbitrate for an output virtual channel (VC) before arbitrating for switch bandwidth. This serialization of resource arbitration significantly increases latency compared to a wormhole router. A speculative virtual-channel router arbitrates for an output VC and switch bandwidth in parallel, speculating that it will be allocated a VC.

We have developed models for wormhole routers, virtual-channel routers, and speculative virtual-channel routers. For each of these pipelines, our gate-level models are used to provide accurate estimates of delay and to determine pipeline depth as a function of cycle time. Using these models we have compared wormhole and virtual-channel flow control. Our results show that a speculative virtual-channel router can achieve the same zero-load latency as a wormhole router with 40% higher throughput.

As illustrated in Figure 5.7, our model gives substantially more accurate performance predictions than conventional 'unit-latency' or 'single-cycle' models. The figure shows our model shows substantial performance differences of 56% in zero-load latency, and 40% in throughput. This establishes the need to account for pipeline delays and credit latency in performance models.

5.3 M-Machine Prototype

The M-Machine is an experimental multi-computer demonstrating processor mechanisms optimized for systems of 1K-4K nodes, and scalable to 64K nodes.The Multi-ALU processor (MAP) is the

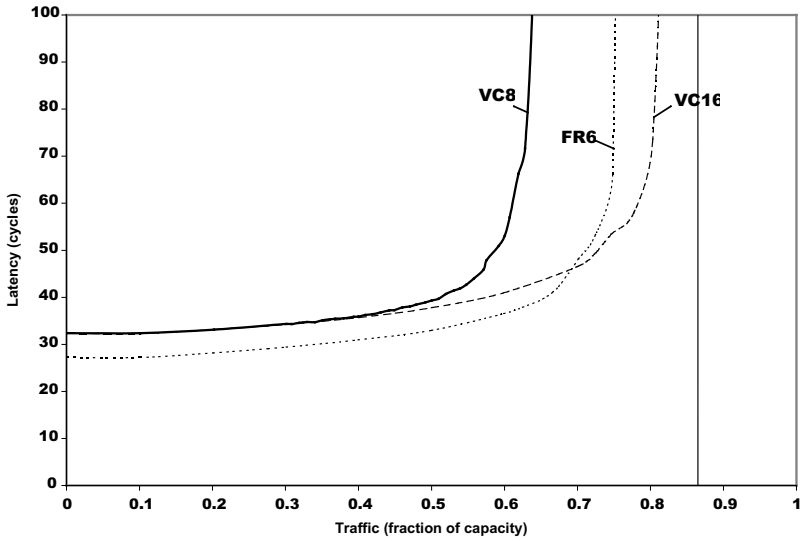


Figure 5.6: Performance of flit-reservation flow control. This figure compares the latency-throughput curve for flit-reservation flow control with 6 flit buffers per channel (FR6) with virtual-channel flow control with 8 (VC8) and 16 (VC16) buffers per channel. The figure shows that flit reservation flow control reduces latency at low loads and increases the saturation throughput for a given amount of buffering.

core computing element in each node of the M-Machine and is designed to maximize on-chip performance by simultaneously exploiting available parallelism at a wide range of granularities - from instruction-level, through loop-level to thread-level. It combines and integrates four key mechanisms: zero-time hardware multithreading to hide latency, including single-cycle function unit latencies; guarded pointers to provide fine-grain memory protection down to the individual word level; concurrent event handling to improve application performance in the face of frequent exceptions, events and faults; and an integrated low latency, high bandwidth networking system built around a register-based messaging system to enable seamless communication between nodes.

The actual MAP chip is a 5 Million transistor custom 64-bit microprocessor composed of 7 ALU's (6 integer and 1 floating-point), 32KByte unified cache, two global communication switches, a network interface with 2D mesh router and an I/O controller. The chip was designed in a 0.7um drawn, 0.5um effective 5-level CMOS process and fabrication was completed in 1999.

The M-Machine makes five key contributions to parallel computer architecture:

1. Efficient inter-processor communication and interaction mechanisms overcome the scaling limitations of superscalar architectures and enable single CPU performance beyond 8-issue [5].

2. Demonstration of a practical implementation of zero-time multithreading without sacrificing single-threaded performance [1].

3. Fast (no-lookup) capability-based addressing via guarded pointers enables parallelism without sacrificing protection and security [2].

4. User-level messaging 100x faster than existing 'fast' messaging systems [8].

5. Concurrent event handling through multithreading [4].

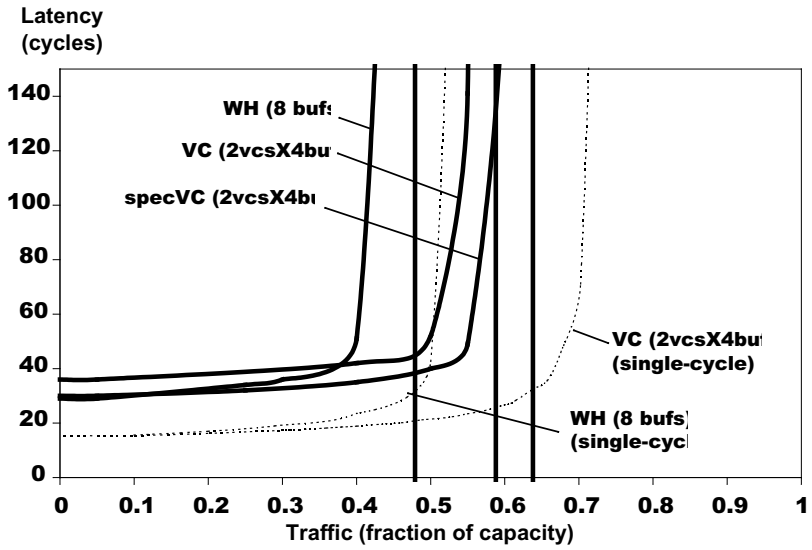


Figure 5.7: Latency-throughput curves for wormhole and virtual-channel routers using our model (solid lines) and a conventional single-cycle model (dashed line). The figure shows that ignoring the pipeline structure in a router model results in optimistic figures for both zero-load latency and saturation throughput.

During the past year we have worked toward the construction of an M-Machine prototype. The implementation effort involves three parallel steps: chip packaging, system logic design and system-level board design. In the chip packaging effort, with the assistance of the Microelectronics and Computer Technology Corporation (MCC), we completed the design and implementation of four prototype packages based on MCC's Flexible Manufacturing of Multichip Modules (FMM) process. Unfortunately, while the FMM process was successful in building the packages, the complexity of the required die attach step was beyond MCC's capabilities. Subsequently, we explored C4 die-attach options with National Semiconductor corporation and with Promex to no avail. We are continuing to seek vendors with the C4 die-attach expertise, however, the M-Machine prototype is currently stalled on this issue.

In the system logic design effort, we have completed the verilog design and associated structural netlist for the M-Machine host interface and all required system logic. In the system-level board design, we have completed the specification and purchase of all the components required to build the prototype system. However, the detailed design of the printed circuit boards are on hold pending the resolution of the aforementioned die-attach issue. Once we are able to solve the die-attach issue we can proceed to complete the M-Machine system prototype.

Bibliography

[1] Andrew Chang, William J. Dally, Stephen W. Keckler, Nicholas P. Carter, and Whay Sing Lee, "The Effects of Explictly Parallel Mechanisms on the Multi-ALU Processor Cluster Pipeline", 1998 International Conference on Computer Design, Austin, TX, October 1998, pp. 474-481.

[2] Nicholas P. Carter, Stephen W. Keckler, and William J. Dally, "Hardware Support for Fast Capability-based Addressing," 6th International Conference on Archiectural Support for Programming Languages and Operating Systems (ASPLOS VI), San Jose, CA. 1994.

[3] W. J. Dally, "Virtual-Channel Flow Control", IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 2, pp. 194-205, March 1992.

[4] Stephen W. Keckler, Andrew Chang, Whay Sing Lee, Sandeep Chatterjee, and William J. Dally, "Concurrent Event Handling Through Multithreading," IEEE Transactions on Computers, September 1999, pp 903-916.

[5] Stephen W. Keckler, William J. Dally, Daniel Maskit, Nicholas P. Carter, Andrew Chang, and Whay Sing Lee, "Exploiting Fine-Grain Thread Level Parallelism on the MIT Multi-ALU Processor", 25th International Symposium on Computer Architecture, Barcelona, Spain, July 1998. pp. 306-317.

[6] Ming-Ju Edward Lee, William J. Dally, and Patrick Chiang, "A 90mW 4Gb/s Equalized I/O Circuit with Input Offset Cancellation," IEEE International Solid State Circuits Conference, Digest of Technical Papers, San Francisco, CA, February 2000, pp. 252-253.

[7] Ming-Ju Edward Lee, William J. Dally, and Patrick Chiang, "Low-Power, Area-Efficient, High-Speed I/O Circuit Techniques," IEEE Journal ofl Solid State Circuits, to appear, November 2000.

[8] Whay Sing Lee, William J. Dally, Stephen W. Keckler, Nicholas P. Carter, and Andrew Chang, "Efficient Protected Message Interface in the MIT M-Machine", IEEE Computer, November 1998. pp 69-75.

[9] Li-Shiuan Peh and William J. Dally, "Flit-Reservation Flow Control", In Proceedings of 6th International Symposium on High-Performance Computer Architecture, Toulouse, January 2000.

Chapter 6

The SUIF Parallelizing Compiler

6.1 Introduction

In the last few years, we have developed a new theory to program transformation called affine partitioning[15] that unifies a large class of program transforms, including unimodular transformations, fusion, fission, reindexing, scaling, and statement reordering. In this model, instances of a statement are identified by the loop index values of their surrounding loops, and affine expression are used to map these loop index values to a partition number. Affine partitioning is applicable to programs with arbitrarily nested loops and affine array accesses. Constructs such as conditionals and non-affine accesses are generally handled by treating them conservatively.

Our previous focus in this work was on maximizing parallelism and minimizing synchronization. We have developed an algorithm that can find the best affine partitioning that can provably minimize the degree of parallelism while minimizing the degree of synchronization. Experimentation with applications from the ASCI research project revealed that the performance of the overall system can be improved greatly by enhancing the data locality of the programs on the individual processors. This year, we focused on using the affine partitioning theory to address the data locality optimization problems. The advantages of the affine partitioning framework are that it is more general and powerful. Thus, we are able to devise an algorithm that optimizes across entire routines rather than previous approaches that can only optimize a perfect loop nest at a time. In particular, we applied the concept of affine partitioning to two ideas identified to be significant in improving data locality: (1) blocking and other loop transformations to improve data locality in programs and (2) array contraction to reduce the dimensionality of the arrays in a program. We have prototyped the locality optimization in the SUIF parallelizing compiler and found that the generality of these algorithms is effective in improving the programs we experimented with significantly.

6.1.1 Distributive Blocking

The concept of blocking is well known among numeric analysts. The idea is that decomposing a matrix computation into submatrix calculations instead of row and column operations often improves data locality and enhances the memory subsystem performance. Blocking can be applied recursively to increase the reuse of data at each level of the memory hierarchy: physical memory, caches, registers as well as the translation lookaside buffer.

Automatic techniques that use a combination of unimodular transformations and blocking techniques have been developed. These algorithms have been shown to improve the performance of programs, such as matrix multiplications, by as much as a factor of three on uniprocessors. By reducing the memory bandwidth requirement on individual processors, blocking also enables the performance of a multiprocessor to scale.

However, as unimodular transforms are defined only for perfectly nested loops, these algorithms have a severe limitation on its applicability. For example, blocking is not applicable to the following simple program:

```
For i = 0 to n
   For j = 0 to n
     stmt1;
   For k = 0 to n
     stmt2;
```

The only perfectly nested loops in this program are the two innermost loops, and blocking has no effect when applied to one-deep loops. Blocking can be very important when it is applicable; however, it only applies to a small subset of programs. Techniques like software prefetching [16] are considered to be more effective for most programs. However, unlike blocking, prefetching does not reduce but in fact increases the memory bandwidth requirement. Prefetching is therefore not as effective in multiprocessor environments.

Previous attempts in blocking non-perfectly nested loops would first use some heuristics to transform the code into one large perfect loop nest. Predicates guard the execution of the statements in the loop to ensure that the original sequential semantics is maintained. Then, apply the traditional blocking technique to the perfectly nested loop. This approach is very limited because many routines cannot be put into this form.

We have developed a new approach we call *distributive blocking* as an alternative approach to generalizing blocking. By formulating the problem from first principles, we can get the full effect of blocking on whole routines relatively easily, once we have a general program transformation framework like affine partitioning.

6.1.2 Array Contraction

Typically, array contraction refers to the conversion of an array into a scalar variable[7, 13, 18]; here we generalize the notion to mapping an array onto a lower-dimensional array. The primary motivation behind the invention of array contraction is to handle array constructs in languages such as Fortran 90 and HPF effectively. A Fortran 90 or HPF front end would translate each array operation into a loop and use temporary arrays to hold partial terms in array expressions. Direct execution of such programs would perform poorly on architectures with caches. Not only are there more memory operations, the working set of the program is increased thus reducing the effectiveness of the cache. A very useful optimization for such programs is to fuse these loops together and to replace the temporary arrays with scalar variables.

For example, suppose the source is an array expression

```
A = B + C + D
```

where each array is an n-element array. The front end would generate:

```
For i = 0 to n-1
  T[i] = B[i] + C[i]
For i = 0 to n-1
  A[i] = T[i] + D[i]
```

The two loops can be fused and the temporary array contracted as below:

```
For i = 0 to n-1
   T = B[i] + C[i]
   A[i] = T + D[i]
```

Once we have converted the array T into a scalar variable, it can now be register allocated. This is an important performance optimization as the processor executes fewer memory operations which may be very expensive due to cache misses. In addition, as the working set becomes smaller, the rest of the memory accesses may also miss less in the cache.

Array contraction is important also for optimizing legacy scientific codes. Many scientific programs in use were developed and optimized for vector computers. To aid vectorization, programmers often deliberately expand a scalar variable so that consecutive iterations in the innermost loop operate on consecutive memory locations. Whereas simple applications of loop fusion might be adequate for array constructs in languages, sophisticated program transforms can expose many more opportunities for array contraction in legacy codes.

6.1.3 Blocking and Array Contraction

Like vector processors, uniprocessors and multiprocessors can sometimes benefit from array expansion. It is sometimes necessary to expand an array to make blocking possible. That is, the goal of increasing data locality through blocking is sometimes at odds with the goal of array contraction.

Consider the following example:

```
For i = 0 to n-1
  For j = 0 to n-1
     t = t + A[j] * B[i]
  C[i] = t
```

Notice that in the above program, the same matrix A[j] is read over and over again in the inner loop. We can block the computation to improve its data locality. Blocking the computation means that multiple t terms will be operated on at the same time, requiring the array be expanded by the size of the block. The result of blocking and array expansion is shown below; for the sake of simplicity, we assume below that n is divisible by b.

```
For ii = 0 to n-1 by b
  For j = 0 to n-1
     FOR i = 0 to b-1
        t[i] = t[i] + A[j] * B[ii+i]
  FOR i = 0 to b-1
     C[ii+i] = t[i]
```

Thus, blocking can conflict with the goal of array contraction; array contraction and blocking must be considered together in a controlled manner.

6.1.4 Overview of the work

We have developed a data locality algorithm that achieves a combination of affine transformations, array contraction and blocking. Whereas previous approaches use just loop fusion to support array contraction, we support array contraction with the full generality of affine partitioning. Whereas previous approaches apply blocking only to perfectly nested loops, our optimizations work on entire routines with arbitrary loop nesting. The generality of our algorithm makes it more robust, enabling it to produce the desirable effect across a large class of programs independent of the original source order.

Our algorithm has two main steps. The first step untangles the dependences in the original source program by partitioning the computation into as many independent threads as possible. This step collects together in one thread all the related operations in the entire routine. Executing the threads sequentially would achieve two important effects. (1) The operations in each thread

are related and therefore exhibit locality. (2) Because all the unrelated operations are separated, the dynamic live ranges (duration between the first write and last read) of all the modified data are minimized. This would naturally expose all the opportunities of array contraction. The independent threads also serve as a canonical program form for the next step. By capturing the dependences that must be obeyed in each thread, we can formulate the space of possible code schedules as an interleaving of threads. In this step, after we have found as many independent threads as possible using affine partitioning, we apply array contraction whenever applicable.

The second step is to exploit data locality between the independent threads. Independent threads often share common read-only data or modified data belonging to the same cache line. This step uses an algorithm based on *distributive blocking* that allows different statements be blocked at different levels according to the data access pattern in the program. It expands the array by a chosen constant factor where necessary, thus achieving both high locality and a small working set.

Affine partitioning also offers a solution technique for handling programs that do not contain any independent threads at the outermost level. Using affine partitioning, we can transform an arbitrarily structured program to expose the largest outermost fully permutable loop nest. Blocking is immediately applicable as fully permutable loops are all blockable. If a program has neither independent threads nor pipelined parallelism, the same routines are applied recursively to the inner loops.

We have implemented the proposed algorithm in the Stanford SUIF compiler, and applied the algorithm to several kernels as well as a full application. We were pleasantly surprised to find that the algorithm succeeds in restructuring the code globally and finds many opportunities for array contraction and distributive blocking. The experimental results speak to the significance of the technique.

6.2 Array Contraction

Our array contraction algorithm has two components:

- The first is to reorder the computation using affine partitioning to expose the opportunity for array contraction. We find an affine partitioning that separates the computation into independent threads.
- The second is to determine arrays that can be contracted and modify the code. This involves determining that the regions accessed in the different iterations do not overlap, the regions have a lower dimensionality of the array, and the live ranges are confined to that of an iteration. This involves finding array liveness information.

6.2.1 Finding Independent Partitions

To find the independent threads of computation from programs with arbitrary loop nestings, we apply our affine space-partition algorithm previously developed for parallelism[15]. Given a program, the algorithm first constructs a set of space-partition constraints which ensures that data dependent operations are placed in the same partition. These necessary and sufficient constraints are formulated without the use of imprecise abstractions such as dependence vectors, allowing us to find more independent partitions. To find partition mappings that satisfy the constraints, we first convert the constraints into systems of linear inequalities using the affine form of the Farkas lemma[8]. Then the basis vectors of the null space of the linear constraints would naturally form the maximal independent partitions.

Intuitively, our algorithm[15] collects together all the operations that work on a disjoint subset of the variables, and the analysis is strong enough to treat the elements of an array individually.

The algorithm is insensitive to how the programmer happens to express the control flow of the program; it performs the equivalent of unimodular transforms, fusion, fission, reindexing, scaling and statement reordering to create the independent threads.

As an example, we describe how our algorithm transforms the program btrix, a vectorized block tri-diagonal solver in the SPEC92 NASA7 benchmark. An excerpt of the 160-line routine is shown in Figure 6.1. Like many other programs, we see that the nesting depth varies from loop to loop, and none of the other automatic algorithms that we are aware of can handle this program well, especially with the presence of conditional statements. As affine partitioning can handle code with arbitrary nesting well, it has no problem with this code and picks out all the independent threads in this program easily.

Specifically, the algorithm puts into the same partition all those iterations whose loop index variable L carry the same value. The fact that the programmer uses the same loop index variable L for all the different loops makes it easy to explain the partitioning results, but it does not affect the algorithm in any way. From a loop transformation perspective, we can think of it as permuting all the loops with index L to the outermost level, across enclosing conditional if necessary, and finally fusing them altogether. Our algorithm finds the answer by constructing the necessary and sufficient constraint for independent partitions, and solving for the solutions using a set of mathematical routines. The transformed code, with an outermost loop that visits each of the partitions sequentially, is shown in Figure 6.2.

6.2.2 Contractable Arrays

An array used in a loop is contractable if

- the region of data accessed in each iteration has a lower dimensionality than that of the original array.
- the live range of each array element is confined to a single iteration.

For the transformed btrix code shown in Figure 6.2, there are a large number of arrays that satisfy the conditions for contraction, such as array L11. After replacing the contractable arrays with scalar variables, we get the transformed code in Figure 6.3.

The first condition can be determined easily by examining the array index expressions. The second condition for array contraction is also a condition that guarantees an array to be privatizable, which is an important technique used in parallelization. However, the technique that has been commonly used for privatization is not adequate here. We have developed and implemented a region-based array liveness analysis to address this issue.

To determine whether an array element is confined to an iteration, we need to determine if there are any subsequent uses of the same array in the rest of the computation. Thus, unlike the upwards-exposed read question, array liveness cannot be found by summarizing the effect of an iteration. That is, it cannot be found by a single bottom-up pass in an elimination-style algorithm, but rather a full-blown bottom-up and a top-down pass are needed. This analysis is complicated by language constructs such as common blocks in Fortran programs, where data of different dimensionalities and types can share the same storage.

Although array liveness can be used in array privatization, it is not computed in many array privatization algorithms, such as the one implemented in the previous SUIF parallelizing compiler. Instead, it simply tests if every iteration must write to exactly the same region of the array. If so, only the values produced in the last iteration can be live after the loop, so the processor assigned to execute the last iteration of the loop uses the original data array as its own private copy. This technique is not applicable to array contraction because arrays can be contracted when the different iterations write to different locations.

```
   DO 100 J = JS,JE
     IF(J.EQ.JS) GO TO 4
     DO 3 M = 1,5
       DO 3 N = 1,5
         DO 3 L = LS,LE
           B(M,N,J,L) = B(M,N,J,L) - A(M,1,J,L)*B(1,N,J-1,L) - ...
  $             - A(M,4,J,L)*B(4,N,J-1,L) - A(M,5,J,L)*B(5,N,J-1,L)
 3   CONTINUE
 4   CONTINUE
     DO 20 L = LS,LE
       L11(L)  = 1.D0 / B(1,1,J,L)
       U12(L)  = B(1,2,J,L)*L11(L)
       U15(L)  = B(1,5,J,L)*L11(L)
20   CONTINUE
     DO 25 L = LS,LE
       L51(L)  = B(5,1,J,L)
       L52(L)  = B(5,2,J,L) - L51(L)*U12(L)
25   CONTINUE
     IF(J.EQ.JS) GO TO 34
     DO 33 M = 1,5
       DO 33 L = LS,LE
         S(J,K,L,M) = S(J,K,L,M) - A(M,1,J,L)*S(J-1,K,L,1) - ...
             - A(M,4,J,L)*S(J-1,K,L,4) - A(M,5,J,L)*S(J-1,K,L,5)
33   CONTINUE
34   CONTINUE
     DO 35 L = LS,LE
       D1 = S(J,K,L,1)*L11(L)
       D5 = (S(J,K,L,5) - L51(L)*D1 - ...
       S(J,K,L,1)  = D1 - U12(L)*S(J,K,L,2) - ... - U15(L)*D5
35   CONTINUE
     IF(J.EQ.JE) GO TO 100
     DO 40 N = 1,5
       DO 40 L = LS,LE
       C1 = C(1,N,J,L)*L11(L)
       C5 = (C(5,N,J,L) - L51(L)*C1 - ...
       B(1,N,J,L)  = C1 - U12(L)*B(2,N,J,L) - ... - U15(L)*C5
40   CONTINUE
100  CONTINUE
     DO 200 J = JE-1,JS,-1
       DO 200 M = 1,5
         DO 200 L = LS,LE
           S(J,K,L,M) = S(J,K,L,M) - B(M,1,J,L)*S(J+1,K,L,1) - ... -
  $           - B(M,4,J,L)*S(J+1,K,L,4) - B(M,5,J,L)*S(J+1,K,L,5)
200  CONTINUE
```

Figure 6.1: Example: original btrix code from SPEC92 NASA7.

```
      DO 300 L = LS,LE
        DO 100 J = JS,JE
          IF(J.EQ.JS) GO TO 4
          DO 3 M = 1,5
            DO 3 N = 1,5
              B(M,N,J,L) = B(M,N,J,L) - A(M,1,J,L)*B(1,N,J-1,L) - ...
     $             - A(M,4,J,L)*B(4,N,J-1,L) - A(M,5,J,L)*B(5,N,J-1,L)
    3     CONTINUE
    4     CONTINUE
          L11(L)  = 1.D0 / B(1,1,J,L)
          U12(L)  = B(1,2,J,L)*L11(L)
          U15(L)  = B(1,5,J,L)*L11(L)
          L51(L)  = B(5,1,J,L)
          L52(L)  = B(5,2,J,L) - L51(L)*U12(L)
          IF(J.EQ.JS) GO TO 34
          DO 33 M = 1,5
            S(J,K,L,M) = S(J,K,L,M) - A(M,1,J,L)*S(J-1,K,L,1) - ...
                - A(M,4,J,L)*S(J-1,K,L,4) - A(M,5,J,L)*S(J-1,K,L,5)
   33     CONTINUE
   34     CONTINUE
          D1 = S(J,K,L,1)*L11(L)
          D5 = (S(J,K,L,5) - L51(L)*D1 - ...
          S(J,K,L,1)  = D1 - U12(L)*S(J,K,L,2) - ... - U15(L)*D5
          IF(J.EQ.JE) GO TO 100
          DO 40 N = 1,5
            C1 = C(1,N,J,L)*L11(L)
            C5 = (C(5,N,J,L) - L51(L)*C1 - ...
            B(1,N,J,L)  = C1 - U12(L)*B(2,N,J,L) - ... - U15(L)*C5
   40     CONTINUE
  100   CONTINUE
        DO 200 J = JE-1,JS,-1
          DO 200 M = 1,5
            S(J,K,L,M) = S(J,K,L,M) - B(M,1,J,L)*S(J+1,K,L,1) - ... -
     $           - B(M,4,J,L)*S(J+1,K,L,4) - B(M,5,J,L)*S(J+1,K,L,5)
  200   CONTINUE
  300 CONTINUE
```

Figure 6.2: Example: transformed btrix code

Our array liveness algorithm is an extension of the array summary analysis implemented in the SUIF compiler[9]. It is a context- and flow-sensitive interprocedural analysis that finds the solution efficiently by using a region-based approach. Our extension to a full-blown bottom-up and top-down framework is efficient in our experiment.

A *region graph* is a hierarchical program representation where every procedure, loop, and loop body in the program are represented as a region. The edges connect a region to its subregions, i.e. from callers to callees, and from code representing an outer scope to that of an inner scope. Our algorithm currently does not handle recursion, thus the region graph is simply a DAG (directed acyclic graph). Recursion can be handled by applying a fixed-point calculation to each of the strongly connected components in the call graph in both the bottom-up and top-down phases.

The array liveness analysis has two phases. The first bottom-up phase summarizes the array accesses of each region starting with the leaf regions. It uses a flow-sensitive analysis to compute a summary of the array access information at the end of each subregion to the end of the parent region. The algorithm keeps track of the sections of the array read, the sections read that are upwards exposed, the sections that *must* be written into, and the sections that *may* have been written into. The summary for the entire region is simply the summary information from the start

```
      DO 300 L = LS,LE
        DO 100 J = JS,JE
          IF(J.EQ.JS) GO TO 4
          DO 3 M = 1,5
            DO 3 N = 1,5
              B(M,N,J,L) = B(M,N,J,L) - A(M,1,J,L)*B(1,N,J-1,L) - ...
     $             - A(M,4,J,L)*B(4,N,J-1,L) - A(M,5,J,L)*B(5,N,J-1,L)
    3     CONTINUE
    4     CONTINUE
          L11  = 1.D0 / B(1,1,J,L)
          U12  = B(1,2,J,L)*L11
          U15  = B(1,5,J,L)*L11
          L51  = B(5,1,J,L)
          L52  = B(5,2,J,L) - L51*U12
          IF(J.EQ.JS) GO TO 34
          DO 33 M = 1,5
            S(J,K,L,M) = S(J,K,L,M) - A(M,1,J,L)*S(J-1,K,L,1) - ...
                - A(M,4,J,L)*S(J-1,K,L,4) - A(M,5,J,L)*S(J-1,K,L,5)
   33     CONTINUE
   34     CONTINUE
          D1 = S(J,K,L,1)*L11
          D5 = (S(J,K,L,5) - L51*D1 - ...
          S(J,K,L,1)  = D1 - U12*S(J,K,L,2) - ... - U15*D5
          IF(J.EQ.JE) GO TO 100
          DO 40 N = 1,5
            C1 = C(1,N,J,L)*L11
            C5 = (C(5,N,J,L) - L51*C1 - ...
            B(1,N,J,L)  = C1 - U12*B(2,N,J,L) - ... - U15*C5
   40     CONTINUE
  100   CONTINUE
        DO 200 J = JE-1,JS,-1
          DO 200 M = 1,5
            S(J,K,L,M) = S(J,K,L,M) - B(M,1,J,L)*S(J+1,K,L,1) - ... -
     $           - B(M,4,J,L)*S(J+1,K,L,4) - B(M,5,J,L)*S(J+1,K,L,5)
  200   CONTINUE
  300 CONTINUE
```

Figure 6.3: Example: transformed btrix code with array contraction

of the starting node to the exit node in the control flow graph.

The top-down phase of the analysis calculates for each region, the summary of the upwards-exposed accesses from the end of the region to the end of the program. We initialize the exposed accesses on program exit as empty set. The desired summary is a composition of the summary from the end of its parent region to the end of the program, and the summary from the end of the region to the end of its parent region. Finally, the array sections written by the region that are live afterwards are simply the intersection of the exposed array sections at the end of the region and the array sections that may or must have been written in the region.

Each array section is represented by a set of systems of integer linear inequalities. The intersection operator we use on array sections is not precise but is conservative to avoid expensive calculations that produce excessively large results[2]. The inequalities representing a data access are derived from the loop bounds and the array indices in the program. Since only linear relationships with respect to loop indices and symbolic variables in the loop are accurately represented, our algorithm applies an interprocedural symbolic analysis to find linear relations between scalar variables before the array analysis, so as to increase the precision.

6.3 Distributive Blocking

Executing identical operations, independent partitions often share common read-only data. In addition, adjacent partitions often operate on adjacent data found in the same cache line. The locality of the threads can be exploited by interleaving the execution of the independent threads so that operations on common data or cache lines are executed close to each other.

6.3.1 Interleaving Independent Threads

One advantage of finding independent threads in a program first is that they are in a form that is easy to work with. Since all data dependences are captured in each thread, we are free to interleave execution of the threads in any way. The key is that the original ordering between operations from the same thread must be maintained. For example, consider the following non-perfectly nested loop:

```
Example 1:
                For I = 1 to M
                  A(I,1) = B(1,I)              [SS1]

                  For J = 2 to N               [LS2]
                    A(I,J) = f(A(I,J-1))            [SS2]

                  For K = 1 to N-1             [LS3]
                    A(I,N-K) = g(A(I,N-K+1))        [SS3]
```

In the following, we assume that each iteration of the outermost loop I represents an independent thread. Each thread executes statement SS1, followed by two loop statements LS2 and LS3. Suppose M=8 and N=5, Figure 6.4(A) depicts the execution order of this program, where the independent threads are executed sequentially. The figure shows the iteration space of the three simple statements (SS1, SS2, SS3): Each iteration is represented by a node in the figure; an n-deep loop is represented by an n-dimensional space; every pair of consecutively executed nodes is connected by an arrow. Given that all the iterations at the outermost loop are independent, any schedule that executes each row of operations in order from left to right is legal. Figure 6.4 shows four other legal interleavings.

The concept of thread interleaving allows us to handle non-perfect nesting simply. Given the simplified syntax below, we look at the possible structure of each thread of computation and describe formally the different choices of interleaving.

Thread	T	::=	SL
Statement List	SL	::=	S — S;SL
Statement	S	::=	SS — L
Simple Statement	SS	::=	*assignment statement* — $\cdots$
Loop Statement	LS	::=	*For index = lower, upper, step* SL

There are two simple structural interleaving primitives. Combinations of these primitives generate all the interleavings of interest.

1. *Interleave a sequence of statements.* Consider the case where each independent thread executes a sequence of statement S1 and statement S2. We can choose to execute the threads sequentially, or to interleave the sequence by executing statement S1 from all the threads, then S2 from all the threads. This effect can be produced by distributing the outer parallelizable loop around each statement.

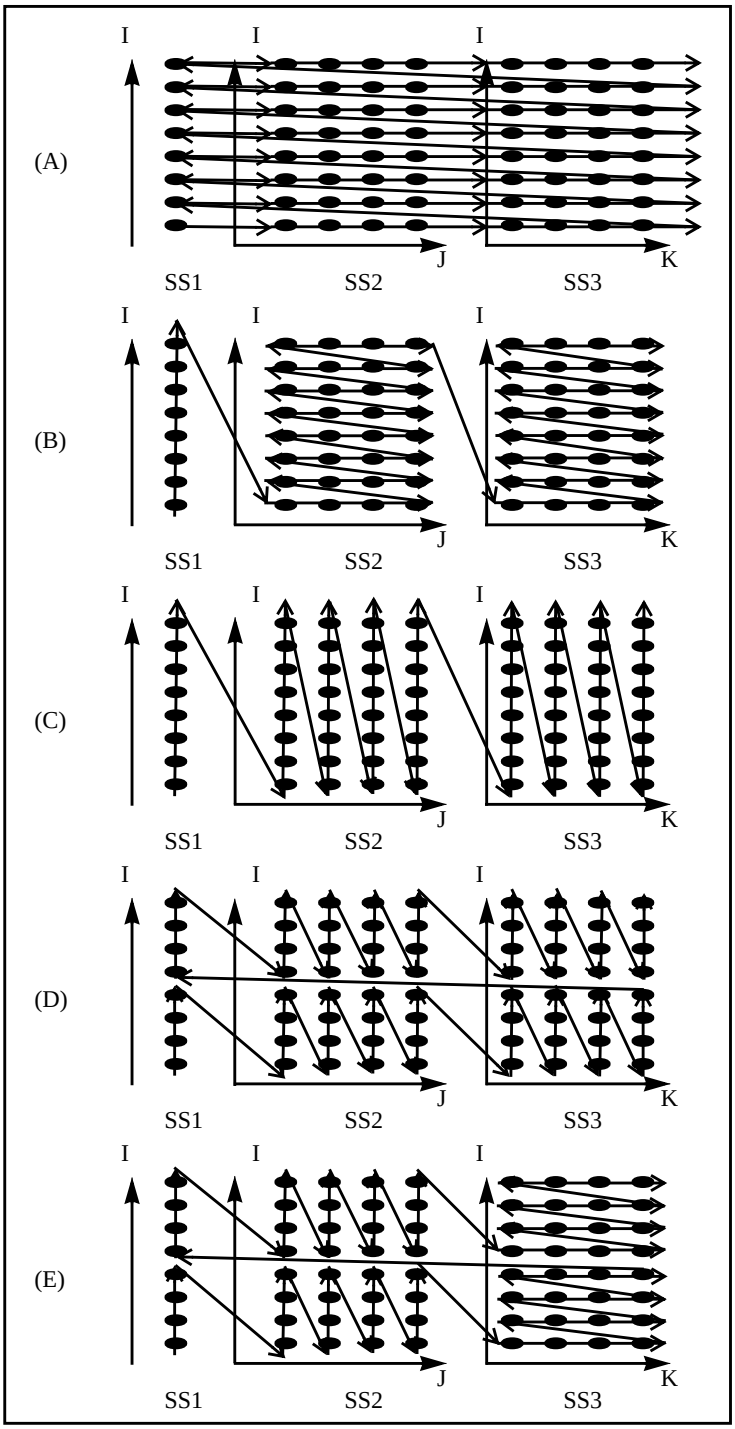


Figure 6.4: Legal Interleaving for Example 1. (M=8, N=5). Each row of operations forms an independent thread; arrows connecting the operations show their execution order. (A): Execute independent threads sequentially. (B): Execute a statement from each thread sequentially before moving to the next. (C): Execute an operation from each thread before moving to the next. (D): Execute an operation from each block of threads (4 iterations) before moving to the next. (E): Execute a statement from each block of threads before moving to the next. Execute threads for SS1 and SS2 sequentially, but for SS3, execute an iteration of J from all threads before moving to the next iteration.

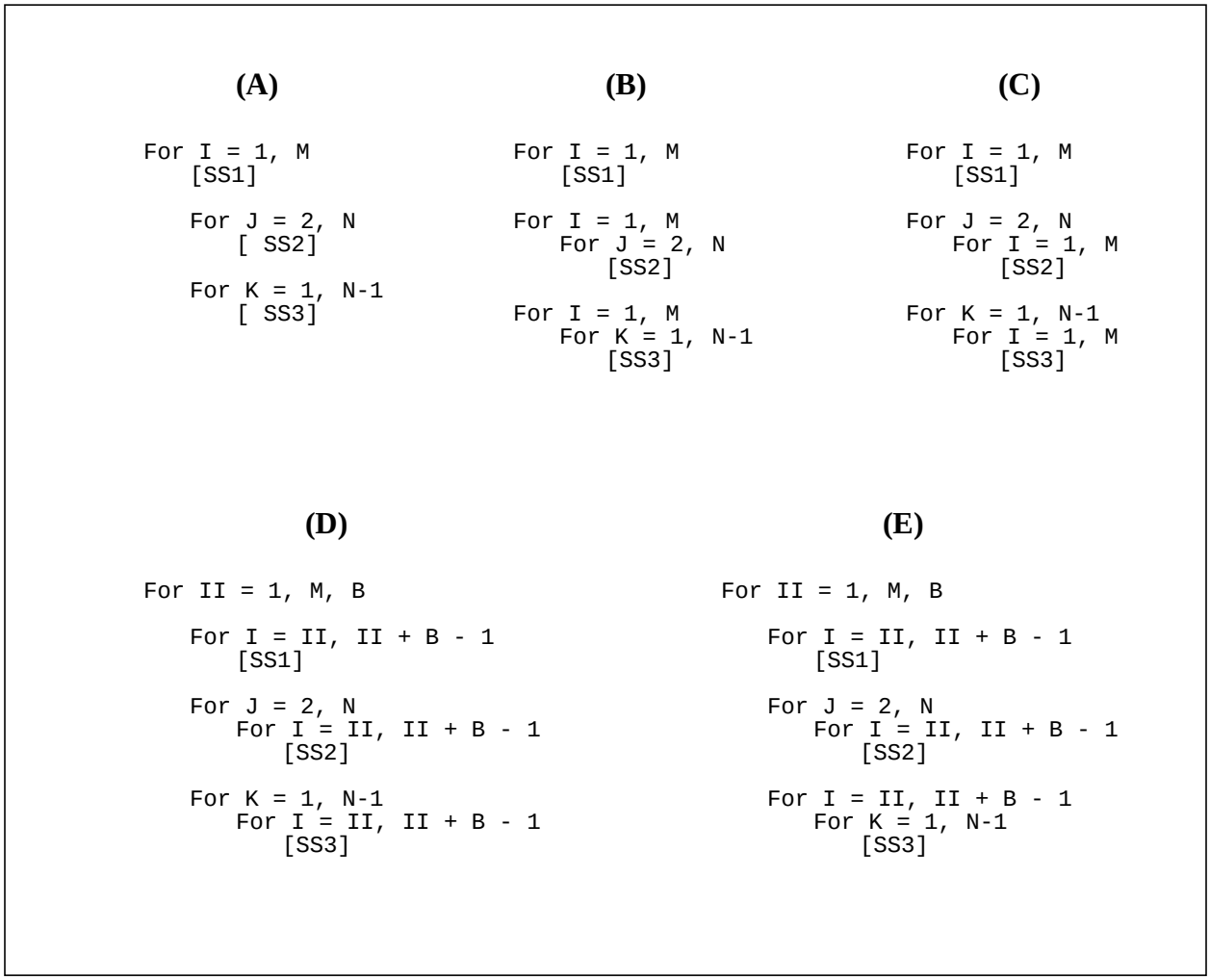


Figure 6.5: Transformed codes for the different interleaving of independent threads shown in Figure 3.

For example, interleaving the sequence of statements SS1, LS2, and LS3 in the program above produces the execution ordering shown in Figure 6.4(B), and the code specifying such a schedule is shown in Figure 6.5(B).

2. *Interleave iterations of a loop statement.* Instead of executing all the iterations of a loop statement from a thread before moving to the next thread, we can interleave the sequential iterations in each thread. That is, execute the first iteration from all the threads, then the second iteration from all the threads, and so on. This effect can be produced by simply interchanging the outer parallelizable loop with the inner loop.

   For the above example, if we apply sequence interleaving then iteration interleaving to both inner loops, we get the ordering in Figure 6.4(C). The code specifying such a schedule is shown in Figure 6.5(C). Clearly we can choose to apply iteration interleaving to only one of the two loops without violating the data dependences in the program.

It is easy to see why the two primitives are correct, as they preserve the ordering of operations in each thread. Once the affine partitioning algorithm identifies the outermost parallelizable loop, we can apply the interleaving primitives recursively up to different depths in different parts of the code.

6.3.2 Distributive Blocking for Non-Perfectly Nested Loops

The essence of blocking is to create a set of inner loops (also known as a tile), whose number of iterations is carefully controlled to ensure that all the data reused in the tile fit into the memory hierarchy in question. Our distributive blocking algorithm for non-perfectly nested loops consists of the following components. First, the algorithm uses affine partitioning to put the algorithm into canonical form, where all the independent partitions are identified. Second, we use array reuse analysis[21] to determine if these parallelizable loops carry reuse. Third, strip-mine all parallel loops that carry any array reuse for any statement. Fourth, the blocked loop is distributed into its components recursively until it is either part of the innermost tile, or none of its components carry any of the reuse.

Suppose applying reuse analysis to Example 1 indicates that it is desirable to tile all the loops because there is reuse in both loop dimensions. Wolf and Lam's algorithm cannot block this program because the program is not perfectly nested. In contrast, we can easily strip-mine the outer loop and apply our interleaving primitives to move the block of iterations to the innermost position for each loop nest. The new execution ordering and transformed code are shown in Figure 6.4(D) and Figure 6.5(D), respectively. (For simplicity, M is assumed to be divisible by B.)

Now suppose the results of the reuse analysis are that it is desirable to tile both loops (I and J) for statement SS2, and only loop K for SS3. Our algorithm will only apply the interleaving primitive to LS2, resulting in the execution ordering shown in Figure 6.4(E).

6.4 Fully Permutable Loop Nest

If independent partitions are not available in a program, it is desirable to restructure the program to create fully permutable loop nests. A loop nest is fully permutable if all the loops can be arbitrarily permuted while preserving the original program semantics. First of all, a nest of parallel loops is trivially also a fully permutable loop nest, and the converse is not true. Besides the fact that an n-deep fully permutable loop nest can also be pipelined to create $n-1$ degrees of parallelism, a fully permutable loop nest also has some very important properties that are relevant to locality.

- A fully permutable loop nest is blockable. Therefore, we can exploit all the reuse along multiple dimensions in a fully permutable loop nest by blocking.
- A sequential loop is a degenerate one-dimensional fully permutable loop. When we succeed in combining operations from different separate loop nests together, we have successfully created one common fused loop. This allows statements that share the same data to execute close together, thus enhancing data locality. Affine partitioning enables a very powerful form of fusion. It can collect together operations from arbitrarily nested loops into a common sequential loop.

The affine time-partition component of our parallelization algorithm[15] can be used to find the biggest nest of fully permutable loops whenever it is possible through a combination of unimodular transform, reindexing, scaling, statement reordering and loop fusion. The time-partition algorithm divides operations into partitions that can legally be executed in sequence. A set of necessary and sufficient time-partition constraints is constructed to capture the legality constraint of preserving the data dependences between different partitions.

Using the time-partition algorithm, we first try to create an outermost fully permutable loop nest out of all the loops in the program. If the algorithm fails in finding any solution, we use a greedy approach to combine as many loops with reuse as possible. If the algorithm finds one legal time partition, we have successfully found one fused loop. If there are n independent legal partitions, we have found an n-deep fully permutable loop nest that can be blocked. There are usually many

choices as to how the iterations can be assembled to create the largest fully permutable loop nest. The algorithm we have implemented will try to line up statements that share data reuse as much as possible.

Because affine partitioning works across arbitrary loop nestings, it can find fully permutable loops that had not been possible before. Below, we show two simple examples to illustrate how this algorithm create a fully permutable nest out of multiple loop nests and out of a non-perfectly nested loop.

6.4.1 Making a Sequence of Loops Fully Permutable

As an example of a code with multiple loop nests, consider the simple ADI (alternate direction integration) algorithm.

```
      DO 20 I = 1,N-1
         DO 10 J = 0,N-1
            A(I,J) = f(A(I,J), A(I-1,J))
10       CONTINUE
20    CONTINUE
      DO 40 I = 0,N-1
         DO 30 J = 1,N-1
            A(I,J) = g(A(I,J), A(I,J-1))
30       CONTINUE
40    CONTINUE
```

Because of the dependence along the J dimension in one case and I dimension in the other, it is not possible to partition the program into multiple independent threads. Our algorithm manages to create a two-dimensional fully permutable loop nest out of this code. It satisfy all the data dependences by putting into the same loop body iteration (I,J) of the first statement and iteration (I-1,J) of the second statement. An unoptimized version of the transformed code is shown below. Index set splitting can be applied to eliminate the predicate calculations in the innermost loops[3].

```
      DO 20 J = 0,N-1
         DO 10 I = 1,N
            IF (I .LT. N) THEN
               A(I,J) = f(A(I,J), A(I-1,J))
            ENDIF
            IF (J .GT. 0) THEN
               A(I-1,J) = g(A(I-1,J), A(I-1,J-1))
            ENDIF
10       CONTINUE
20    CONTINUE
```

This fully permutable loop can then be blocked to exploit reuse across all the dimensions of the loop nest. Previous blocking algorithms would individually block the two loops, causing the same array to be fetched from memory twice if the array is larger than the cache.

6.4.2 Making an Imperfect Loop Nest Fully Permutable

We now illustrate how our algorithm can block a non-perfectly nested loop for uniprocessor locality. We will use the example of cholesky decomposition (row version, aka ddot cholesky) as shown below.

```
      DO 50 I = 1,N
         DO 20 J = 1,I-1
            DO 10 K = 1,J-1
               A(I,J) = A(I,J) - A(I,K) * A(J,K)
10          CONTINUE
            A(I,J) = A(I,J) / A(J,J)
```

```
 20      CONTINUE
         DO 30 L = 1,I-1
            A(I,I) = A(I,I) - A(I,L) * A(I,L)
 30      CONTINUE
         A(I,I) = SQRT (A(I,I))
 50   CONTINUE
```

This non-perfectly nested loop consists of 4 statements, each in a different nesting level. Our affine time-partition algorithm is able to put all the statements into a 3-deep fully permutable loop nest, as shown in the code below. The algorithm places all the statements, including those that were originally nested 1- or 2-deep into the innermost loop of the new code. The affine partitioning simply comes up with an affine mapping from the original iteration space to the new iteration space for each statement that satisfies the dependence constraints. The affine mappings found are included in the output code example. The code generation routine guards the execution of the operations with the original loop bound to ensure that the new programs execute only operations that are in the original code.

```
      DO 50 I2 = 1,N
         DO 50 J2 = 1,I2
            DO 50 K2 = 1,I2
C
C              // FP Mapping: I2 = I, J2 = J, K2 = K
               IF (J2 .LT. I2 .AND. K2 .LT. J2) THEN
                  A(I2,J2) = A(I2,J2) - A(I2,K2) * A(J2,K2)
               ENDIF
C
C              // FP Mapping: I2 = I, J2 = J, K2 = J
               IF (J2 .EQ. K2 .AND. J2 .LT. I2) THEN
                  A(I2,J2) = A(I2,J2) / A(J2,J2)
               ENDIF
C
C              // FP Mapping: I2 = I, J2 = I, K2 = L
               IF (I2 .EQ. J2 .AND. K2 .LT. I2) THEN
                  A(I2,I2) = A(I2,I2) - A(I2,K2) * A(I2,K2)
               ENDIF
C
C              // FP Mapping: I2 = I, J2 = I, K2 = I
               IF (I2 .EQ. J2 .AND. J2 .EQ. K2) THEN
                  A(K2,K2) = SQRT (A(K2,K2))
               ENDIF
 50   CONTINUE
```

Again blocking can be applied to the code to improve the data locality of the code, and index-set splitting can be applied at the end to eliminate the conditional guard calculations from the loops.

6.5 The Data Locality Algorithm

So far, we have described how our algorithm would handle programs that have only one nesting of parallelizable or fully permutable loops. All the ideas can be put together to create an algorithm that can handle the general case where iterations of a parallelizable or fully permutable loop nest may contain a sequence of statements which can themselves be made parallelizable or fully permutable.

Algorithm 6.5.1 Optimize data locality using affine partitioning, distributive blocking, and array contraction.

1. Repeat the following steps, applying the algorithm first to the whole program, then to the computation inside the parallelizable and fully permutable loops found in the last iteration:

   - Put statements that access related data into the same equivalence class. This partitions the computation into unrelated sets of statements that can be optimized separately. Apply the following steps to each set of statements.
   - Extract the independent partitions in the computation using our affine space partitioning algorithm. Reorder the partitions to be executed sequentially as iterations of an outermost parallelizable loop nest. In the case where there are multiple loops, loops that carry read-only data reuse would be placed as the inner loops of the loop nest. This minimizes the working set size, thus enhances temporal locality of modified data. If the algorithm is successful in finding independent partitions, skip the next step.
   - Create an outermost fully permutable loop nest that groups together statements operating on the same data using our affine time partitioning algorithm. If it is not legal (or beneficial) to group all the related statements together, the algorithm would first divide the program dependence graph into strongly connected components. Then a greedy approach is used to group together as many components as possible, ordering the attempt based on the potential improvement on group reuse.

2. Apply distributive blocking to improve spatial locality and locality of read-only data. We place into the innermost tile the strip-mined portions of all the outer parallelizable loops and all the loops in the innermost fully permutable nest that carry reuse.* Blocking an outer fully permutable nest is advisable to exploit higher levels of memory hierarchies, such as the third-level cache or the physical memory.

3. Identify array contraction based on the interprocedural array liveness analysis, then contract the arrays to minimize the footprint of the working set.

The data locality optimization algorithm can also be used to optimize programs for multiprocessors.

Algorithm 6.5.2 Optimize parallelism and data locality.

1. Find parallelism in a sequential program using our affine partitioning parallelization algorithm[14].

2. Apply the locality optimization algorithm to the SPMD program produced by the parallelization step.

3. If any parallel loop carry reuse, distributive blocking is applied to the loop, so that we can get both parallelism and locality benefits.

Note that pipelined partitions found in the parallelization step would already be tiled by the parallelization algorithm to reduce interprocessor communications.

6.6 Experimental Results

We experimented with a set of real applications to study the generality and effectiveness of our integrated locality optimization algorithm. Table 6.1 lists the Fortran programs used in our experiments. The algorithms have been implemented in the Stanford SUIF compiler; our compiler uses the Omega Code Generator[20] to produce the blocked and/or parallelized C code.

*Distributive blocking can also be used to optimize locality for different levels of memory hierarchies.

Benchmark	*Subroutine*	*Program Description*	*No. of lines*	*Problem Size*
cholesky		Cholesky decomposition (row version)	25	1000 x 1000
mgrid	interp	A trilinear interpolator in a multigrid solver	55	130 x 130 x 130
NASA7	btrix	A vectorized block tri-diagonal solver	160	30 x 30 x 30 x 5
NASA7	chotst	Cholesky decomposition in parallel	155	250 x 4 x 40
flo88		Wing-body analysis solving transonic flow	7379	256 x 32 x 48

Table 6.1: Benchmark Programs

	Our Integrated Algorithm				*Unimodular Transform & Tiling*			
	1 proc	2 proc	4 proc	8 proc	1 proc	2 proc	4 proc	8 proc
cholesky	3.1	5.2	9.8	17.5	1.0	1.0	1.0	1.0
interp (mgrid)	1.4	2.7	5.0	8.1	1.0	2.0	3.9	6.5
btrix (NASA7)	1.5	3.1	5.9	10.6	0.8	0.4	0.4	0.4
chotst (NASA7)	1.1	1.6	3.2	6.3	1.0	1.2	1.9	1.8

Table 6.2: Speedups on a Digital Turbolaser with 8 300-Mhz 21164 processors.

The experiments were run on a Digital Turbolaser with 8 300-Mhz 21164 processors, except experiments for the Flo88 program which were run on a 32-processor SGI Origin. (We could not get the performance of Flo88 on the Digital Turbolaser due to a bug in Digital's system software). For each of the programs, we use the Stanford SUIF compiler to generate both the unoptimized and optimized, sequential and parallel codes. The native compiler on the multiprocessors is then used to generate the binaries for performance timing.

We measure performance as speedup over the sequential version without any code transformation for locality or parallelism. We also compare our results with an algorithm that is based on unimodular transform and tiling[12]. Table 6.2 and 6.3 present the speedups on our benchmarks. Our locality algorithm delivers 1.1 to 3.1 times speedup on a uniprocessor; in contrast, the unimodular approach cannot find any speed up because it can only optimize perfectly nested loops.

For btrix and cholesky, Sections 6.2 and 6.4 have already described in detail how our algorithm improves data locality. The 3.1-times speedup for cholesky results from the algorithm's ability to tile non-perfectly nested loops, and the 1.5-times speedup for btrix comes from the algorithm's strength in finding independent partitions and contracting arrays for programs with arbitrary nestings. When we combine our data locality and parallelization algorithm, we get superlinear speedups (17.5 and 10.6) on 8 processors. On the other hand, the unimodular approach does not improve the performance of these programs; as a matter of fact, synchronization overheads cause slow down in these programs.

Our Integrated Algorithm					
1 proc	2 proc	4 proc	8 proc	16 proc	32 proc
1.1	1.9	3.6	6.4	12.3	19.6
Unimodular Transform & Tiling					
1 proc	2 proc	4 proc	8 proc	16 proc	32 proc
0.9	1.7	3.1	5.5	6.6	6.3

Table 6.3: Speedups on a 32-processor SGI Origin for Flo88.

The sequential code for NASA7 chotst is already near optimal for data locality. Our algorithm was effective in extracting an outermost parallel loop while simultaneously exploiting the spatial locality by applying distributive blocking to the iterations of the parallelized loops (Section 6.3).

The interp subroutine in mgrid consists of 8 instructions in one non-perfectly nested loop. The instructions are data independent. Our algorithm would have partitioned them into 8 separate loop nests, if not for the group reuse among the read-only data. To exploit the read-only data reuse, the instructions are placed together in one 3-deep loop nest and subsequently tiled. Due to cache conflicts between arrays accessed in the loop, we tile only 2 out of the 3 loops and get a speedup of 1.4 on a uniprocessor.

The Flo88 program is a real-world application in use at the Center for Integrated Turbulence Simulations at Stanford. It consists of 421 loops in 7379 lines of code and a large number of intermediate arrays. This code, when parallelized by a commercial compiler, shows little speedup. We currently modified the source program slightly to improve the result of our interprocedural scalar symbolic analysis. This modified source program is used as the base version in our experiment. As shown in the bottom half of Table 6.3, the code delivers a speedup of only 6.3 on 32 processors. When we apply our data locality algorithm to the program, it was able to apply array contraction to 20 multi-dimensional arrays, 9 out of which are 3-dimensional and the rest are 2-dimensional. The transformed code runs 10% faster, and because fewer memory references are needed, the contracted version is more scalable. With our optimization, the performance improves from a 6.3 times speedup to a 19.6 times speedup on 32 processors on the Origin.

6.7 Related Work

In the area of data locality optimization, a number of algorithms have been proposed in the past. However, many of these algorithms are applicable only to perfectly nested loops where unimodular transforms and tiling are applied to attempt to create the largest innermost tiles of loops that carry reuse[22, 5]. There have also been other proposals that would try to convert non-perfectly nested loops to perfectly nested loop, either by heuristically applying transformations such as fusion and fission, or by conversion technique such as code sinking[7, 10, 23].

Our integrated locality algorithm is based on affine partitioning that unifies a large number of loop transformations, including unimodular transformations, reindexing, fusion, fission, scaling, and statement reordering. It is not sensitive to the artificial program structure, and is more effective in improving data locality compared to previous approaches that use heuristics or a pre-defined sequences of transforms to handle non-perfectly nested loops and sequence of multiple loops. Due to the aggressiveness of our memory contraction and affine partitioning algorithm, our algorithm can succeed in contracting more arrays than other schemes[4, 13].

Kodukula *et al.* proposed the idea of data shackling whereby a compiler uses the owner-compute rule to distribute a program with arbitrary loop nests and sequences into blocks of computation (data shackles)[11]. Their algorithm would require user's input and does not automatically determine how to legally distribute the data. In contrast, our algorithm does not require any user input, and would automatically find a legal desirable reordering of the operations to enhance locality. Also, our optimization would not be constrained unnecessarily by the owner-compute rule.

Ahmed *et al.* proposes an algorithm to tile non-perfectly nested loop nests[1]. Their algorithm tries to embed each statement in a common product space, whose dimensionality is the sum of the loop depth of each statement. The goal of their technique is to is tile the product space. Briefly, it tries out different combinations of loop permutation, reversal and skewing until a valid embedding function can be determined. In comparison, our algorithm does not need to embed the statements in a product space that can be quite large, and it can automatically determine the affine transformation necessary to tile all non-perfectly nested loops that can be tiled in Ahmed *et al.*'s

algorithm.

Rosser and Pugh apply iteration space slicing technique to optimize data locality[17]. Roughly, given an array and a way to partition it, their algorithm reorders the slice of operations contributing to each array partition. A major drawback is that the authors do not have a method to estimate profitability of various locality slices, and a compiler would need to employ heuristics to choose what arrays to optimize. Our approach, on the other hand, considers the locality of the entire program and automatically determines the necessary reordering transformations and array contraction that is beneficial.

To increase opportunity for array contraction, it is desirable to have a precise yet efficient liveness analysis. While the previous SUIF compiler only computes liveness information on scalars[9], our implementation in the current SUIF compiler performs array liveness analysis. The Polaris compiler[6] has an intraprocedural array liveness analysis for detecting privatizable variables that need not be finalized[19]. They formulated the liveness problem as an iterative data flow problem, and summarize the effect of each loop with array sections representing the upwards exposed uses of the loop. Since their algorithm does not make use of the write sections to reduce the exposed reads, the result is therefore less precise and degenerates to a largely flow-insensitive analysis. Our interprocedural array liveness algorithm is more precise without incurring large overheads in practice.

Bibliography

[1] N. Ahmed, N. Mateev, and K. Pingali. Tiling imperfectly-nested loop nests (revised). Technical Report TR2000-1782, Cornell University, January 2000.

[2] S. P. Amarasinghe. *Parallelizing Compiler Techniques Based on Linear Inequalities*. PhD thesis, Stanford University, January 1997.

[3] S. P. Amarasinghe and M. S. Lam. Communication optimization and code generation for distributed memory machines. In *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, pages 126–138, June 1993.

[4] D. Bacon, S. Graham, and O. Sharp. Compiler transformations for high-performance computing. *Computing Surveys*, 26(4):345–420, December 1994.

[5] U. Banerjee. Unimodular transformations of double loops. In *Proceedings of the Third Workshop on Languages and Compilers for Parallel Computing*, pages 192–219, August 1990.

[6] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu. Parallel Programming with Polaris. *IEEE Computer*, pages 78–82, December 1996.

[7] S. Carr, K. S. Mckinley, and C.-W. Tseng. Compiler optimizations for improving data locality. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 252–262, October 1994.

[8] P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one dimensional time. *International Journal of Parallel Processing*, 21(5):313–348, October 1992.

[9] M. W. Hall, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, and M. Lam. Detecting coarse-grain parallelism using an interprocedural parallelizing compiler. In *Proceedings of Supercomputing '95*, November 1995.

[10] K. Kennedy and K. S. McKinley. Optimizing for parallelism and data locality. In *Proceedings of the 1992 ACM International Conference on Supercomputing*, pages 323–334, July 1992.

[11] I. Kodukula, N. Ahmed, and K. Pingali. Data-centric multi-level blocking. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation*, pages 346–357, June 1997.

[12] M. S. Lam and M. E. Wolf. Compilation techniques to achieve parallelism and locality. In *Proceedings of the DARPA Software Technology Conference*, pages 150–158, April 1992.

[13] E. C. Lewis, C. Lin, and L. Snyder. The implementation and evaluation of fusion and array contraction in array languages. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, pages 50–59, June 1998.

[14] A. W. Lim, G. I. Cheong, and M. S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. In *Proceedings of the 13th ACM SIGARCH International Conference on Supercomputing*, Rhodes, Greece, June 1999.

[15] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, France, January 1997.

[16] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992.

[17] W. Pugh and E. Rosser. Iteration space slicing for locality. In *Proceedings of the Twelfth Workshop on Languages and Compilers for Parallel Computing*. Springer-Verlag, August 1999.

[18] V. Sarkar and G. R. Gao. Optimization of array accesses by collective loop transformations. In *Proceedings of the 1991 ACM International Conference on Supercomputing*, pages 194–204, June 1991.

[19] P. Tu. *Array Privatization and Demand-Driven Symbolic Analysis*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[20] University of Maryland. *The Omega Library Version 1.1.0 Interface Guide*, November 1996.

[21] M. E. Wolf. *Improving Locality and Parallelism in Nested Loops*. PhD thesis, Stanford University, August 1992. Published as CSL-TR-92-538.

[22] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 30–44, June 1991.

[23] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, 1995.