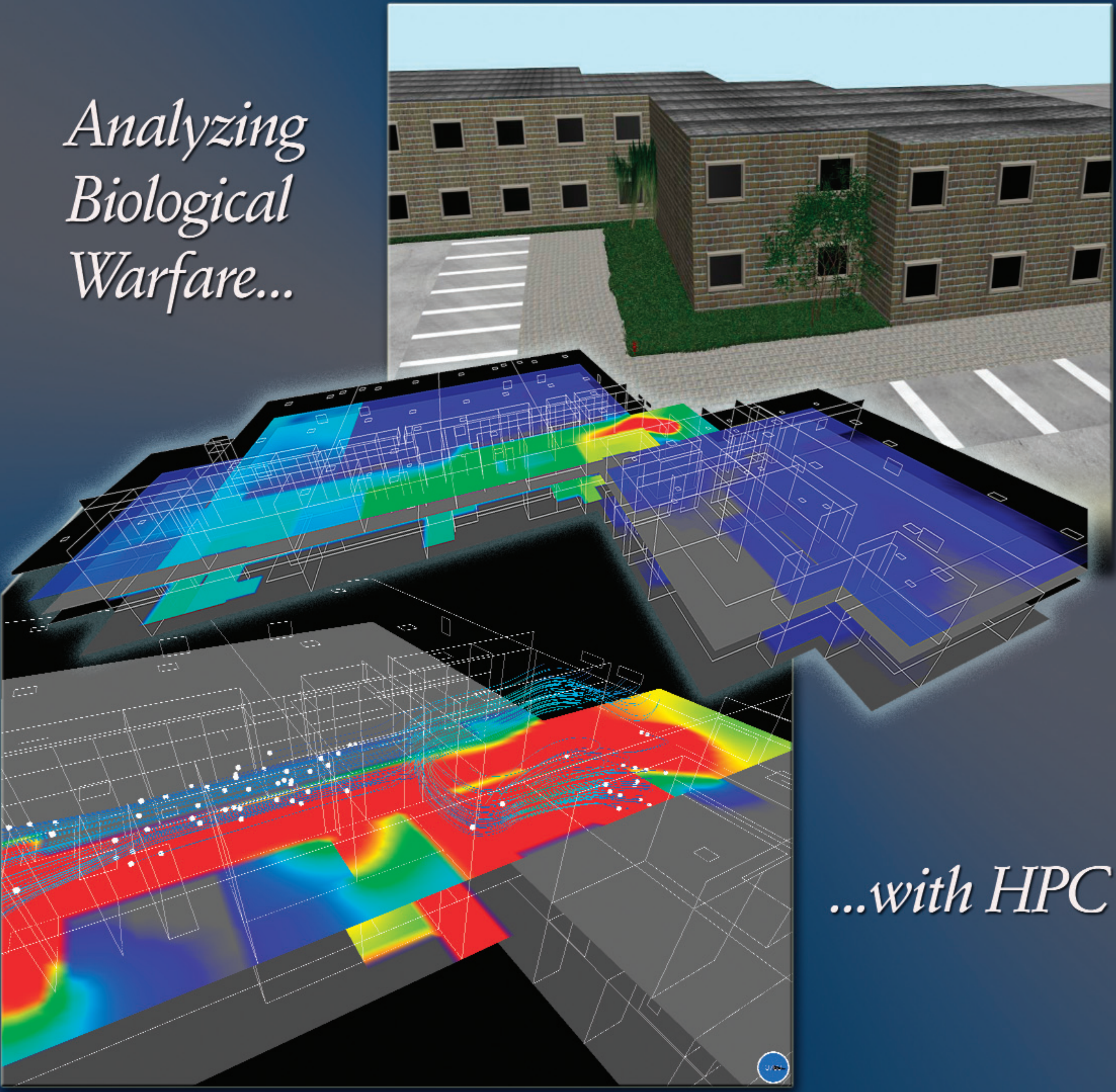
Navigator

NAVO MSRC FALL 2006

Analyzing Biological Warfare...



...with HPC

News and information from...
The Naval Oceanographic Office Major Shared Resource Center

Steve Adamec, NAVO MSRC Director

As this issue of the Navigator goes to press, we're fielding the most capable HPC systems ever installed at the NAVO MSRC.

Our two new IBM POWER5+ HPC systems, BABBAGE and PASCAL, have performed well in their factory predeployment testing and look to be outstanding performers in the future. As icing on this tasty cake, we're fielding them in a newly renovated, physically hardened facility with the necessary resilience to sustain 24 x 7 operations in Mother Nature's worst weather or even with major computer facility support equipment failure.

In addition, we are working with NASA to establish resilient, multi-gigabit, northbound communications service for Stennis Space Center which will further improve the operational availability of the NAVO MSRC for the nationwide DoD HPCMP user community.

We have also been actively participating in the planning for the consolidated customer assistance and visualization environments for the six centers within the HPCMP, all of which have received preliminary approval and endorsement by the HPCMP Corporate Board of Directors. As I mentioned in the last issue, please be assured that our primary goal throughout this budget adjustment and consolidation process will be the maintenance of a premier HPC environment with the support you have come to expect from all of the centers.

To this end, the NAVO MSRC has been asked by the HPCMP to lead a collaborative effort with the other centers and industry to completely reengineer and modernize the data storage environment and capabilities across the entire program. This will be a huge effort with the potential to dramatically improve the technical capabilities, cost-effectiveness, and usability of the HPC systems themselves and their associated data storage/management environments. I am extremely excited by this project and very pleased that we have been asked to participate in it.

New HPC Systems, More, to Serve You Better

As always, we solicit your constructive criticism of the NAVO MSRC and invite you to let us continue to assist you in bringing this cutting-edge capability to bear in support of your HPC needs.

The Naval Oceanographic Office (NAVO) Major Shared Resource Center (MSRC): Delivering Science to the Warfighter

The NAVO MSRC provides Department of Defense (DoD) scientists and engineers with high performance computing (HPC) resources, including leading edge computational systems, large-scale data storage and archiving, scientific visualization resources and training, and expertise in specific computational technology areas (CTAs). These CTAs include Computational Fluid Dynamics (CFD), Climate/Weather/Ocean Modeling and Simulation (CWO), Environmental Quality Modeling and Simulation (EQM), Computational Electromagnetic and Acoustics (CEA), and Signal/Image Processing (SIP).

NAVO MSRC
Code N7
1002 Balch Boulevard
Stennis Space Center, MS 39522
1-800-993-7677 or
msrchelp@navo.hpc.mil

NAVO MSRC Navigator
www.navo.hpc.mil/Navigator

NAVO MSRC Navigator is a biannual technical publication designed to inform users of the news, events, people, accomplishments, and activities of the Center. For a free subscription or to make address changes, contact NAVO MSRC at the above address.

EDITOR:
Gioia Furness Petro, petrogio@navo.hpc.mil

DESIGNERS:
Kerry Townson, ktownson@navo.hpc.mil
Lynn Yott, lynn@navo.hpc.mil

Contents

Hypersonic Scramjet Technology Enhancements for Long Range Interceptor Missile

S.M. Dash, A. Hosangadi, R.J. Ungewitter, J.D. Ott, and K.W. Brinckman, Combustion Research and Flow Technology, Inc., Pipersville, PA and K. Kennedy, Army Missile Research Development and Engineering Center, Redstone Arsenal, AL

INTRODUCTION

The Army has been involved in the development of a new hypersonic missile that is rocket boosted and scramjet propelled. Scramjet air breathing propulsive systems are highly integrated with the missile using aerodynamic surfaces to compress the captured airstream before it enters the combustor.

Hypersonic scramjets, operating at high Mach numbers, use hydrogen as the primary fuel, and there are many complex flowpath design issues that must be addressed that include transitional turbulence, shock/boundary layer interactions, fuel/air mixing in a highly compressible environment, and ignition/flameholding for operation at higher altitudes.

Computational Fluid Dynamics (CFD) has played a major role in overall design and scramjet component optimization performed in conjunction with full-scale experiments in the LENS shock tunnel facility. CFD has proven invaluable in the interpretation of scramjet data, in providing performance parameters not available from testing (i.e., combustion efficiency), and in shedding insight into the complex physics occurring along the propulsive flowpath, which is often highly nonlinear (i.e., small changes often produce large effects).

Via the use of the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC) massive parallel hardware platforms and CFD codes configured to perform efficiently on these platforms, end-to-end Reynolds Average Navier-Stokes (RANS) solutions with resolved grids can now be obtained in several days. This is extremely encouraging as the implementation of advanced thermo-chemical and turbulence/transitional models requires the integration of a very large system of coupled partial differential equations with widely disparate length/time-scales.

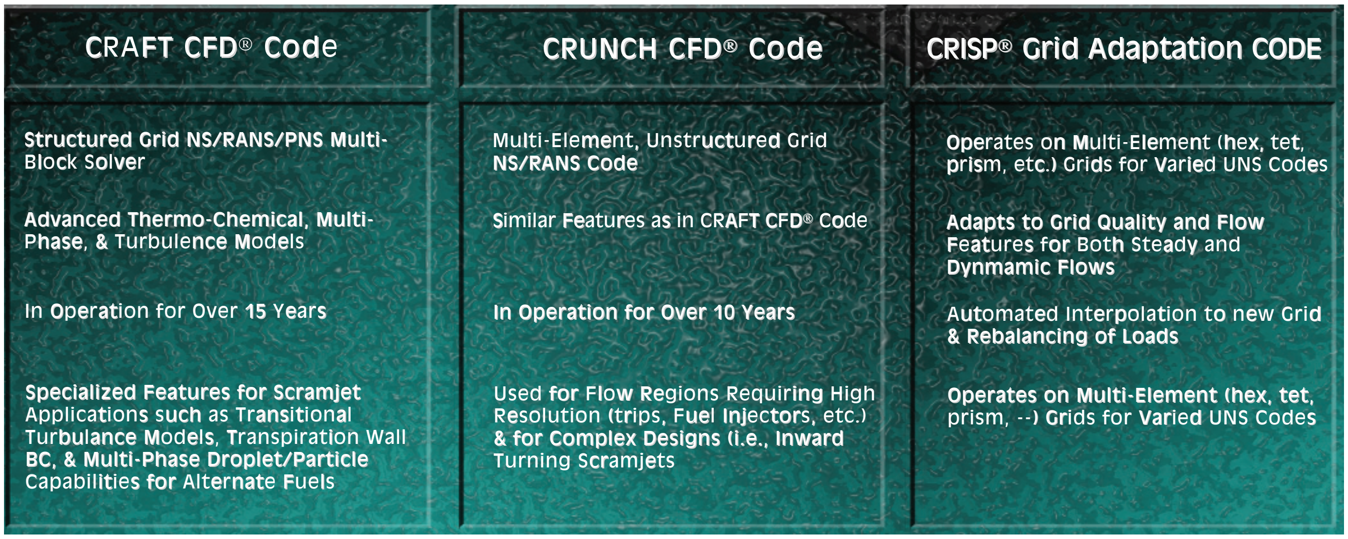
This paper first describes the CFD codes used for scramjet design and component optimization followed by a description of the modeling used in the codes–including an overview of how large eddy simulation is implemented to calibrate the advanced RANS turbulence models being used.

Some aspects of scramjet component evaluation is also discussed as well as the approach used for design optimization using an evolutionary algorithm. In performing optimization studies, a series of cases are performed concurrently at each design level, which for fuel injector optimization has required very substantive resources.

CODES UTILIZED

An overview of the CFD and grid adaptation codes utilized for scramjet calculations is provided in Table 1. For

Continued Next Page...



CRAFT CFD® Code	CRUNCH CFD® Code	CRISP® Grid Adaptation CODE
Structured Grid NS/RANS/PNS Multi-Block Solver	Multi-Element, Unstructured Grid NS/RANS Code	Operates on Multi-Element (hex, tet, prism, etc.) Grids for Varied UNS Codes
Advanced Thermo-Chemical, Multi-Phase, & Turbulence Models	Similar Features as in CRAFT CFD® Code	Adapts to Grid Quality and Flow Features for Both Steady and Dynmamic Flows
In Operation for Over 15 Years	In Operation for Over 10 Years	Automated Interpolation to new Grid & Rebalancing of Loads
Specialized Features for Scramjet Applications such as Transitional Turbulance Models, Transpiration Wall BC, & Multi-Phase Droplet/Particle Capabilities for Alternate Fuels	Used for Flow Regions Requiring High Resolution (trips, Fuel Injectors, etc.) & for Complex Designs (i.e., Inward Turning Scramjets	Operates on Multi-Element (hex, tet, prism, --) Grids for Varied UNS Codes

Table 1. Codes Utilized for Scramjet Applications.

conventional rectangular designs (e.g., NASP, Hyper-X), a hybrid approach is utilized, with the structured grid numerics in the CRAFT CFD® code applied for most of the flowfield, but with the Unstructured Navier-Stokes (UNS) numerics of the CRUNCH CFD® code required where fine-scale features must be resolved–such as in the vicinity of transition trips on the forebody and in the fuel injector regions of the combustor.

Several levels of grid adaptation are often required for which the CRISP CFD® code is utilized. Figure 1 shows product formation contours at several stations in the elliptical combustor of a generic inward-turning scramjet, where fuel is injected from the walls using flush, angled injectors.

The combustion efficiency (obtained with the original and adapted UNS grids) indicates that unless the grid is adapted, this efficiency is overestimated (due to numerical diffusion effects). Since there is only half-plane symmetry, the fuel/air mixing details for multiple injectors must be resolved.

Use of conventional, cell-splitting adaptation can lead to a large increase in the number of nodes (see the table in Figure 1–which shows an increase from 7 million (M) to 13.5M cells in Pass 1–which did provide a grid resolved solution since Pass 2 results were essentially identical). The authors have been working on hybrid cell splitting/cell stretching adaptation concepts, which remedy this issue but still require further development to resolve "tangling issues."

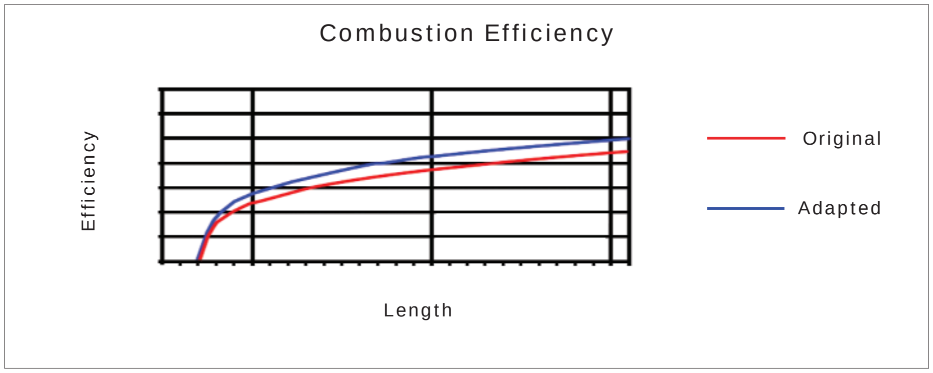
While conventional load balancing (same number of cells per domain) using domain decomposition with MPI is effective for aerodynamic problems, it is problematic for scramjet combustor problems where varied zones in the flow have differing work loads. Using iterative matrix-split chemical kinetic techniques, it is possible that 100 iterations per CFD time-step in fuel injection regions may be performed (where ignition reactions have small time-scales), and minimal iterations in other regions.

Using dynamic load balancing, based on work per node,[1] has led to effective load balancing in such situations and is being supplemented by use of tabulated procedures (In Situ Adaptive Tabularization (ISAT), Artificial Neural Network (ANN), etc.).

The fuel injection region of combustors can be very Central Processing Unit (CPU) intensive, particularly for problems where ignition kinetics is required, requiring 10-15M cells and solving 20 or more coupled Partial Differential Equations (pde's)(five gas dynamic, nine or more chemical species, and eight turbulent/transition equations–further discussed in the Models Utilized section). Analysis of this region takes about three-quarters of the overall CPU time (including grid adaptation), and full end-to-end runs on 256-512 processors are routinely performed in several days.

Models Utilized

The modeling utilized in performing scramjet simulations is summarized in Table 2. Basic models are used for



	Original	Adapted Pass 1	Adapted Pass 2
Number of Venticies	1,848,643	2,456,536	2,830,591
Number of Tetrahedral Cells	5,955,505	10,011,954	12,413,432
Number of Hexahedral Cells	603,801	603,801	603,801
Number of Prism Cells	406,866	406,866	406,866

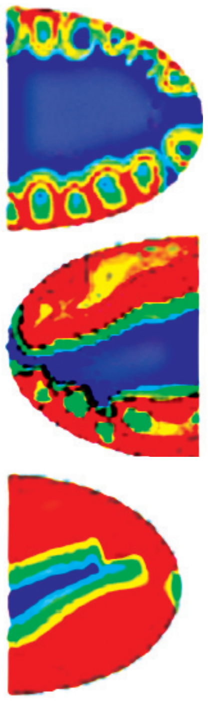


Figure 1. Grid adaption effects on combustion efficiency and grid size.

preliminary design and to expedite the early stages of design optimization.

Advanced models are used for interpreting experimental data and for refining designs. Transitional models, which solve pde's for onset location and for intermittency (i.e., three dimensional blending from laminar to turbulent flow), have been calibrated for hypersonic flows and compared with varied shock tunnel data sets.

These models take into account the fluctuation in acoustic noise levels in test facilities (prescribed as inflow condition) and are extremely useful in suggesting boundary layer trip location modifications in going from full-scale testing to a quieter flight environment. Analysis techniques for the transitional flow downstream of trips using the intermittency model is still in development and has required some "local" modifications to the source terms used in the low Reynolds Number turbulence models.

Scalar fluctuation modeling is playing a dominant role in obtaining accurate values for fuel/air mixing. The turbulent pde's implemented solve the temperature/energy and species variance and related dissipation rates and are used to obtain local values of turbulent Prandtl (Pr_t) and Schmidt (Sc_t) numbers that govern thermal and species turbulent diffusion. Most CFD codes require specification of constant values for these parameters, but their values in fuel injection regions can vary substantially as shown by the Sct contours in Figure 2.

Calibration and validation of these models suffer from lack of scalar fluctuation data in high speed flows. However, Large-Eddy Simulation (LES) solutions of unit fuel injection problems are providing supportive data. Figure 3 shows one such LES solution with contours of time-averaged mean flow (u, T, YH2) and corresponding Root Mean Squared (RMS) fluctuations exhibited.

This case represented Hydrogen angled fuel injection into a high speed airstream to emulate the environment in a high Mach combustor. RANS comparisons were quite reasonable, and values of Turbulent Pr_t Number and Sc_t and Lewis Number (Le) (= Pr_t/Sc_t) varied substantially in the injector region.

Studies Supporting System Design

Earlier studies have been performed for rectangular scramjet designs, with interdigitated flush and wedge injectors,[2] to assess the ability of the CFD codes shown in Table 1 to reproduce data obtained in varied Large Energy National Shock Tunnel

Continued Next Page...

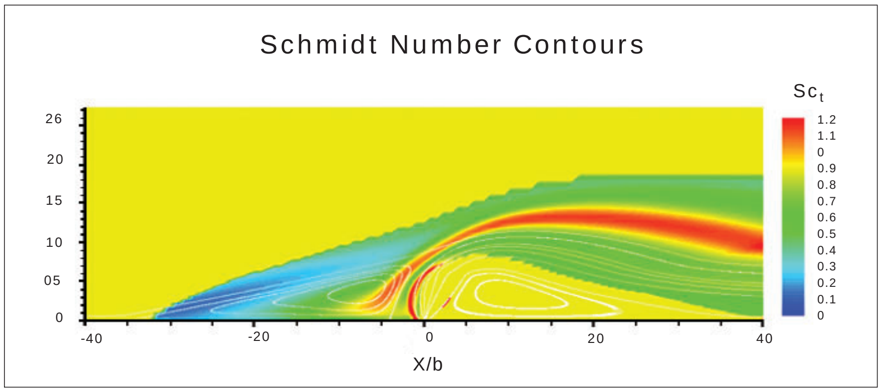
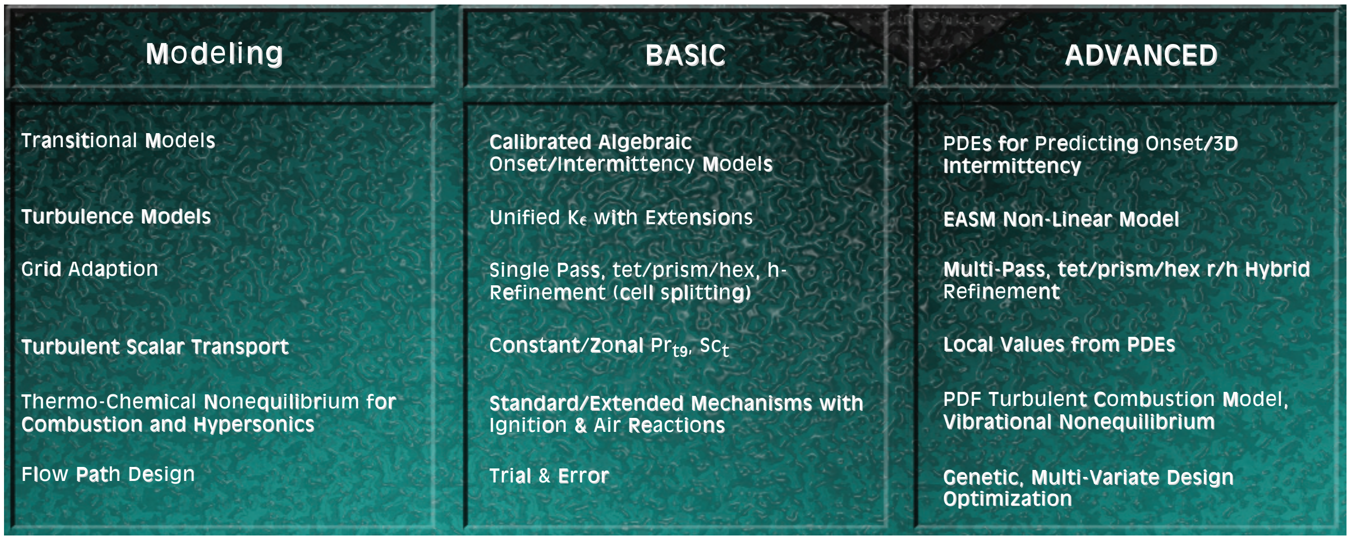


Figure 2. Schmidt number contours –fuel injection in high speed stream.



Modeling	BASIC	ADVANCED
Transitional Models	Calibrated Algebraic Onset/Intermittency Models	PDEs for Predicting Onset/3D Intermittency
Turbulence Models	Unified $K\epsilon$ with Extensions	EASM Non-Linear Model
Grid Adaption	Single Pass, tet/prism/hex, h-Refinement (cell splitting)	Multi-Pass, tet/prism/hex r/h Hybrid Refinement
Turbulent Scalar Transport	Constant/Zonal Pr_{t9}, Sc_t	Local Values from PDEs
Thermo-Chemical Nonequilibrium for Combustion and Hypersonics	Standard/Extended Mechanisms with Ignition & Air Reactions	PDF Turbulent Combustion Model, Vibrational Nonequilibrium
Flow Path Design	Trial & Error	Genetic, Multi-Variate Design Optimization

Table 2. Basic and advanced models used for scramjet applications.

(LENS) tests performed at full-scale and for duplicated flight conditions.

These studies highlighted the need to develop advanced transitional and scalar fluctuation models and to use grid adaptation in fuel injector regions.

The conclusion from these varied comparative studies was that CFD could provide very reasonable comparisons with data under conditions where strong burning occurred without ignition aids. NAVO MSRC resources were used to support a substantive number of end-to-end comparative studies and were key to the progress made in generating accurate solutions in a timely manner.

Of most recent interest in this Army/NAVO MSRC work are inward-turning concepts (utilizing a "sugar-scoop" inlet) that compress the flow in a shock-free manner into an elliptical combustor with flush, angled injectors. These concepts have been tested in the Calspan-University of Buffalo Research Center (CUBRC) LENS facility with an inlet section and numerical model as shown in Figure 4. Such propulsive flowpaths are now being integrated into hypersonic missile designs, and numerous CFD calculations have been performed in support of systems studies.

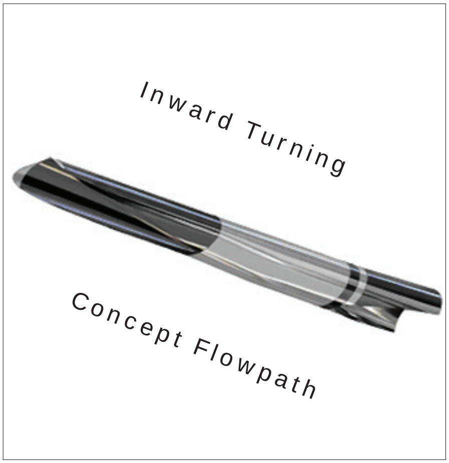


Figure 4. Inward turning model and flowpath.

This flowpath is very difficult to calculate, and UNS numerics are used for the entire end-to-end calculation. While on-design inlet performance is generally quite good, achieving good combustion efficiency in a reasonable length is a design concern.

A basic issue is that of fuel injector spacing with studies for fixed fuel/air stoichiometry and injector size as shown in Figure 5. Using a smaller number of injectors provides good penetration, but air passes between injectors, while using a larger number of injectors, results in poor penetration and unburnt air in the central core. In this preliminary study, ten injectors worked best, but the solution is far from optimal.

Fuel Injector Optimization

To optimize fuel injector patterns/conditions to yield the highest combustion efficiency for a fixed combustor length, multi-variate genetic-based optimization[3] is being used, with a schematic of the Graphical User Interface (GUI)-driven framework implemented shown in Figure 6.

The genetic optimization procedure

Continued Page 24...

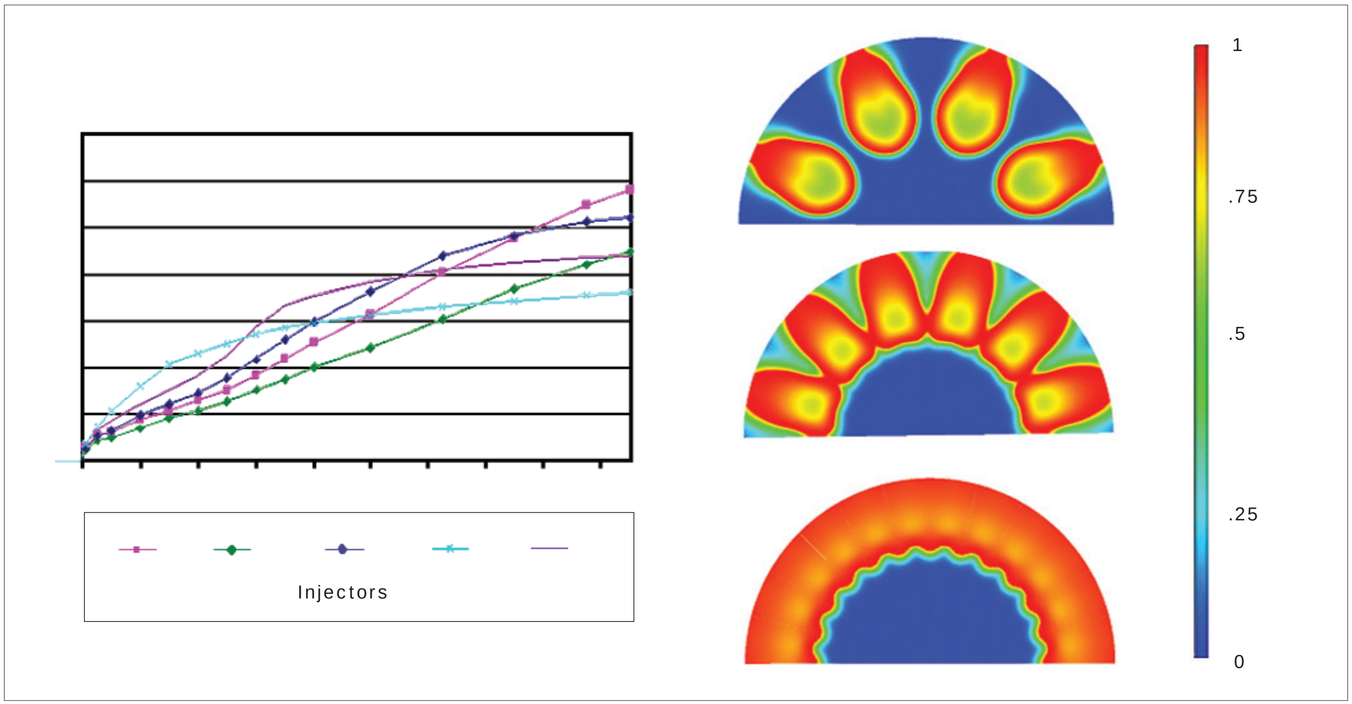


Figure 3. LES simulation of fuel injection flowfield.

High-Performance Computing Tools Enable Army Survivability and Lethality Technology Development

Stephen J. Schraml and Kent D. Kimsey, U.S. Army Research Laboratory
Sean Ziegeler, Visualization Software Engineer, NAVO MSRC VADIC

Physics-based simulations, when utilized on High Performance Computing (HPC) systems, can be used to develop and assess lethality and survivability technologies. Examples of these for Army applications include penetrators, warheads, explosives, passive and active armors, and armor materials for equipment and personnel.

Large-scale numerical simulations of complex weapon-target interactions help guide experiments, illustrate physical processes, ascertain performance limits, extract transient response characteristics, and augment experimental databases. This article provides an overview of the computational terminal ballistics methods that are used on HPC resources at the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC) by researchers at the U.S. Army Research Laboratory (ARL).

The evolution of scalable HPC systems, along with recent advances in continuum mechanics codes, has enabled large-scale simulations to play a paramount role in the development of advanced survivability and lethality technologies. Possibly the most notable use of numerical simulations in this development process is parametric analysis. Once a numerical model of a particular weapon-target interaction is developed and the bounds on the parameters of interest are defined, numerous simulations can be performed to determine how each parameter's variation influences the performance of the system under study.

Computational Methodology

The computational methodology that is commonly used in modeling complex weapon-target interactions is the shock physics code CTH.[1] CTH is an Eulerian finite-volume code for modeling solid dynamics problems that involve shock wave propagation, multiple materials, and large deformations in one, two, and three dimensions.

CTH is widely used across the defense research and development community to model problems in shock wave propagation resulting in tens of millions of processor hours of usage each year on HPC resources deployed under the DoD High Performance Computing Modernization Program (HPCMP).

CTH uses a two-step solution scheme –a Lagrangian step followed by a remap (or advection) step. The conservation equations are replaced by the explicit finite-volume equations solved in the Lagrangian step. The remap step uses operator-splitting techniques to replace multidimensional equations with a set of one-dimensional equations.[2]

High-resolution material interface trackers are available to minimize material dispersion. Analytical and tabular equations of state are available to model hydrodynamic behavior or materials. A variety of constitutive models are available to treat elastic-plastic material behavior. Models for explosive detonation are also available to handle the reactions of energetic materials.

Continued Next Page...

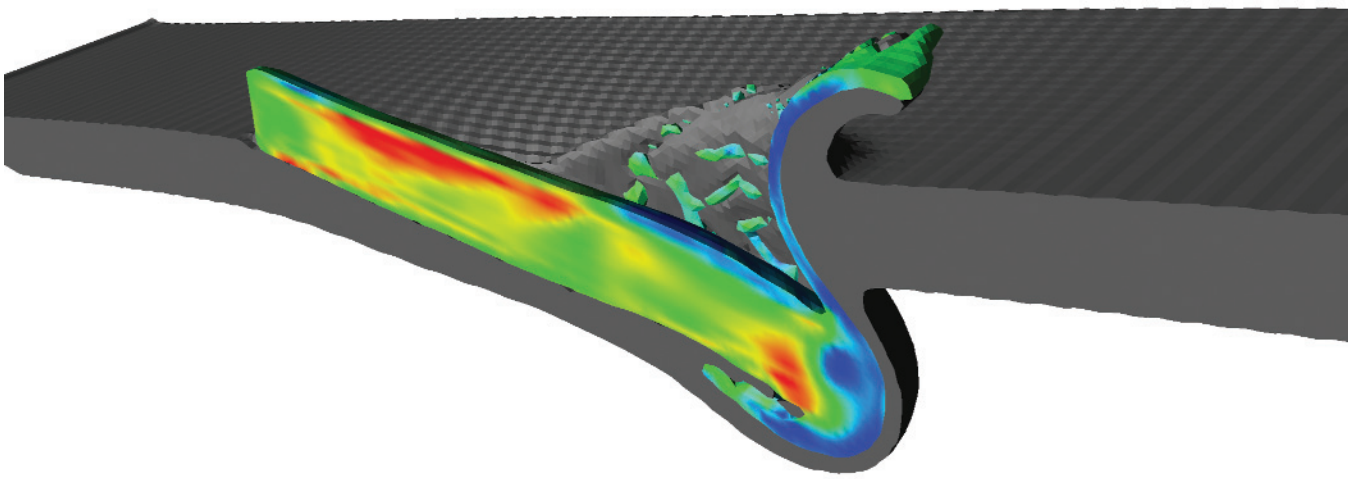


Figure 1. CTH mesh decomposition with explicit message passing.

Fixed and Adaptive Meshes

Traditionally, the computational domain of a CTH simulation would be a fixed, structured, orthogonal mesh. In such a case, the mesh must be sufficiently resolved to capture the activity of interest that may occur anywhere in the computational domain during the simulation. This approach results in the unavoidable use of fine mesh where there is no activity of interest at any given time during the simulation.

An Adaptive Mesh Refinement (AMR) capability has been added to CTH that allows the definition of the mesh to change during the simulation based on the evolving characteristics of the simulation. The adaptation of the AMR mesh is based on user-defined indicators, such as the value, gradient, or difference, of a variable in the solution (pressure, density, velocity, stress, etc.). This technique results in simulations in which the most highly resolved mesh “follows” the activity of interest to the analyst while using lower mesh resolution in other regions of the computational domain. This allows the analyst to configure highly resolved simulations that have fewer total computational cells than a comparable fixed mesh simulation having the same minimum cell size.

The AMR implementation in CTH is a block-based scheme in which each block consists of an orthogonal mesh with a fixed number of cells in the X, Y, and Z directions. The blocks are connected in a hierarchal manner with adjacent blocks having either exactly the same cell/block size or exactly a 2:1 ratio in cell/block size.

Refinement or un-refinement of the mesh is accomplished through a series of transitions of adjacent blocks with a difference in mesh density ratio of 2:1. All mesh blocks at a given mesh density are at the same refinement level. The finest mesh resolution that can exist in the computational domain is controlled by defining the maximum number of mesh refinement levels.

Computational Requirements

The increasing complexity of emerging survivability-lethality technologies and the requirement to evaluate increasingly sophisticated threat technologies mandate higher fidelity in numerical models to capture the geometry and material behavior under high strain-rate loading.

Increasing model fidelity requires larger computational domains (meters in each coordinate direction), finer grid resolutions (sub-millimeter), and longer simulation times (from microseconds to milliseconds). The number of Eulerian cells, and therefore a simulation's total memory requirement, scales with the cube of the cell size for Three Dimensional (3D) problems. As the cell size decreases to capture finer details in geometry and response, the

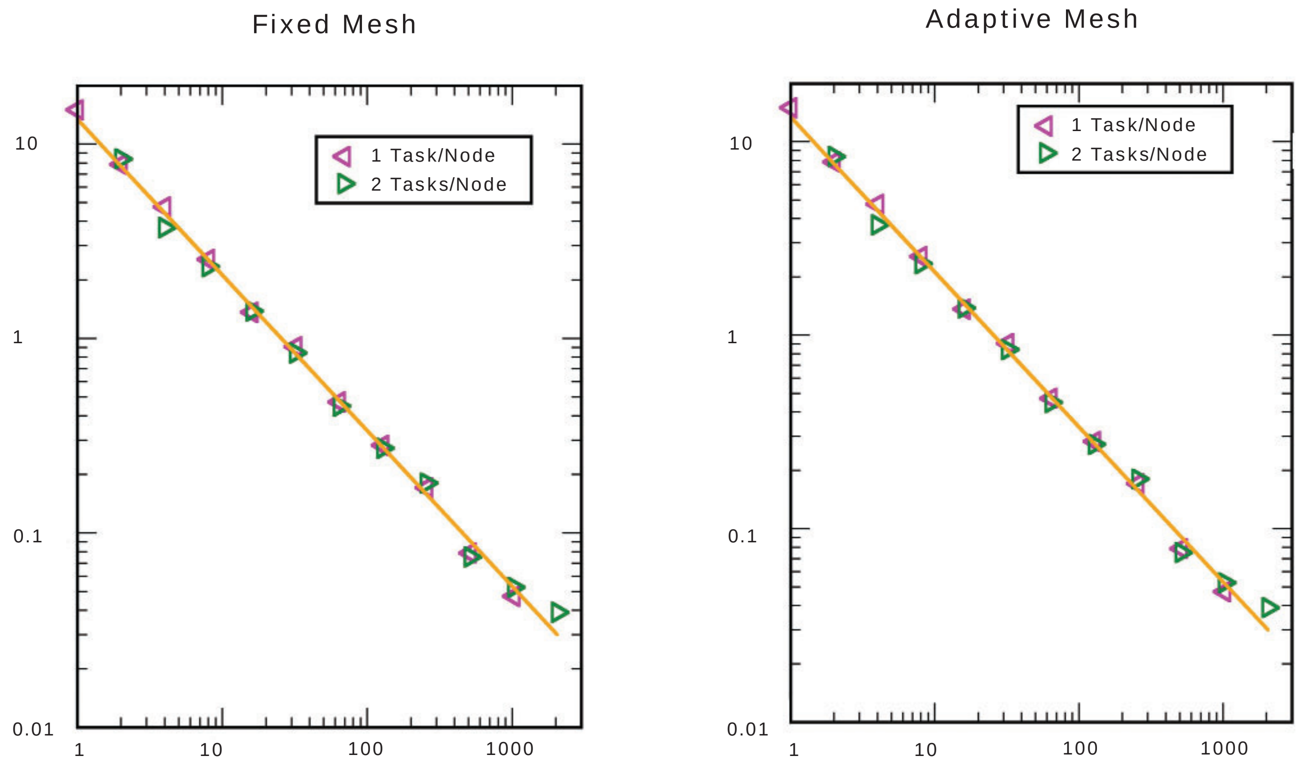


Figure 2. CTH scalability on IBM SP Power 3 and SGI Origin 3800 systems.

time step also decreases to satisfy the stability criteria of the explicit time integration scheme. This reduction in the time step results in the need for additional time integration cycles in order to complete a simulation.

Consequently, the total computational workload of a 3D CTH simulation scales with the fourth power of the cell size (three powers for the increase in the number of cells and one power for the increase in the number of time integration cycles). The combination of these factors provides a strong motivation for the exploitation of scalable HPC architectures.

Scalable Paradigm

The Eulerian finite-volume method's structured mesh is well-suited for scalable paradigms that use message passing between computational sub-domains. To enable the exploitation of scalable HPC architectures for munitions technology applications, CTH has been adapted to a scalable computing paradigm to simulate large-scale problems in shock physics.

A Single Program, Multiple Data (SPMD) paradigm was employed in the adaptation of CTH to scalable HPC architectures. [3,4] Under the SPMD paradigm, the same executable code runs on each computational node/processor, but each executable code works on a different set of data. Algorithms that depend on a logically connected mesh are readily adapted to the SPMD paradigm.

Ghost cells are a common technique for applying boundary conditions to finite-difference and finite-volume schemes, which makes the internal differencing computations independent of edges and corners in the Eulerian mesh. To adapt CTH to the SPMD paradigm, these ghost cells are used for passing messages between adjacent sub-domains. Such explicit message passing between sub-domains gives each sub-domain access to its neighboring sub-domains' boundary cell data.

For fixed-mesh simulations, domain decomposition is accomplished at problem initialization and results in the subdivision of the computational domain into smaller sub-domains, with one sub-domain assigned to each computational task.

The fixed-mesh domain decomposition scheme is designed to make each sub-domain as computationally "cubic" as possible, maximizing the ratio of interior computational cells to bordering ghost cells. This approach maximizes the ratio of computation-to-communication for maximum performance on scalable HPC architectures.

For AMR simulations, each mesh block is treated as a computational sub-domain. Groups of AMR blocks are mapped onto individual processors to distribute the workload. Because the AMR mesh changes during the simulation, a dynamic load balancing scheme is used to update the mapping of AMR blocks as the simulation progresses.

Scalable Performance

Since the mid-1990s, the parallel performance of CTH has been evaluated on many systems deployed under the auspices of the HPCMP. [5, 6, 7] The benchmark simulation used in these scalability studies involved the impact of a yawed long rod penetrator on a thin steel plate as illustrated in Figure 1.

The scalability of CTH on a Linux Opteron cluster was recently determined through a series of simulations that employed both fixed and adaptive meshes. The fixed-mesh scalability simulations were conducted with a nearly constant workload.

This was done to keep the computation-to-communication ratio as close to constant as possible for simulations involving different numbers of processors. As the number of processors was increased, the fixed mesh was incrementally refined by uniformly decreasing the characteristic cell size in each coordinate direction by the nearest integer factor of $2^{1/3}$.

This approach approximately doubles the total number of Eulerian cells with each successive mesh refinement. The fixed-mesh scalability benchmark simulation was designed such that each sub-domain contained approximately 387,000 Eulerian cells.

Thus, for a 2,048-processor simulation, the computational domain of the benchmark simulation contained approximately 800 million Eulerian cells.

The AMR CTH benchmark used in the scalability study was configured to be physically identical to the fixed-mesh simulation. The only difference between the fixed-mesh simulation and the AMR simulation was the definition of the mesh. The size of the mesh in the AMR simulation was scaled with the number of processors in a manner similar to the fixed-mesh study.

However, it is not possible to precisely scale the total number of cells in the AMR simulation since the refinement and un-refinement indicators are based on the physics, not the topology, of the computational domain. Thus, to scale the size of the simulation in a controlled manner, the maximum refinement level was increased by one for every factor of eight increase in the number of processors. The 2:1 ratio of cell size between refinement levels results in a factor of approximately eight in the total number of cells in the 3-D simulation.

The fixed-mesh and AMR scalability results on the Linux Opteron cluster are presented in Figure 2, which illustrates a linear speedup of CTH as the problem size is increased with the number of processors. This linear scalability of CTH enables the continued growth of shock physics problems on HPC architectures.

Currently, production CTH simulations at the NAVO MSRC occur on MARCELLUS, the IBM Cluster 1600, with Power4 processors. These simulations are typically run on 256 to 1000 processors, achieving linear scalability trends similar to those illustrated here.

Continued Next Page...

Scientific Visualization Methods

The linear scalability demonstrated by CTH on modern HPC architectures enables continued growth in both problem size and the number of required processors used. As a result, the amount of data output is large and challenging to post-process, visualize, and analyze. This problem is addressed by scaling the analysis processing proportionally with the original computational processing.

In many cases, visualization and analysis are performed on a specialized graphics cluster to leverage hardware graphics acceleration. However, in the case of large shock physics simulations, much of the post-processing consists of the creation of isovolumes, isosurfaces, cutting planes, etc., that are more compute-intensive than graphics-intensive.

A system like the same computational system used for the CTH simulation itself is more beneficial to these post-processing tasks given the larger number of Central Processing Units (CPU's) available for processing.

Three scalable visualization software packages are utilized: the Interdisciplinary Computing Environment (ICE),[8] EnSight, and ParaView. ICE is used for batch processing of the data to create isosurfaces, cutting planes, etc. EnSight and ParaView are used to interactively view computational results and to leverage their data analysis capabilities. Both software packages are well-suited to the task since they offer interactive, client-server remote visualization features.

EnSight is a commercial software package with many features and functions for data analysis, manipulation, and visualization. ParaView is an open-source visualization application for visualizing large data sets and enables customization. It also utilizes the Message Passing Interface (MPI) for parallel processing, which allows it to scale well to visualize very large data sets.

The analysis and visualization tasks currently performed on MARCELLUS typically require at least 32 processors. As CTH problem sizes increase with the growth in available resources, the post-processing requirements will also increase.

Conclusion

Advances in the development of scalable HPC systems and continuum mechanics for modeling shock wave propagation in materials coupled with the evolution of scalable visualization technologies have maximized the design utility of large-scale terminal ballistics simulations.

Today, large-scale simulations are critical to concept evolution as well as the research and development of emerging lethality and survivability technologies.

Numerical simulations provide valuable insight into complex weapon-target interactions that cannot be obtained from experiments alone. The integration of large-scale physics-based simulations into the early stages of the development cycle allows costly experiments to be focused in areas of greatest possible benefit.

Today, large-scale simulations are an integral component of research and development programs for emerging lethality and survivability technologies.

References

1. McGlaun, J.M. and S.L. Thompson, “CTH: A Three-Dimensional Shock Wave Physics Code,” International Journal of Impact Engineering, Volume 10, Numbers 1-4, 1990, pp. 351-360.
2. Van Leer, B., “Towards the Ultimate Conservative Difference Scheme IV: A New Approach to Numerical Convection,” Journal of Computational Physics, Volume 23, 1977, pp. 276-299.
3. Robinson, A.C., et al., “Massively Parallel Computing, C++, and Hydrocode Algorithms,” Proceedings of the 8th Conference on Computing in Civil Engineering, 1992.
4. Kimsey, K.D., S.J. Schraml, and E.S. Hertel, "Scalable Computations in Penetration Mechanics,” International Journal of Advances in Engineering Software Including Computing Systems in Engineering, Volume 29, July 1998, pp. 209-215.
5. Schraml, S.J. and K.D. Kimsey, “Scalability of the CTH Hydrodynamics Code on the Sun HPC 10000 Architecture,” ARL-TR-2173, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, February 2000.
6. Schraml, S.J., K.D. Kimsey, and T.M. Kendall, “Scalable Simulations of Penetration Mechanics on the SGI Origin 3800 and the IBM SP Power3 Computer Systems,” ARL-TR-2537, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, June 2001.
7. Schraml, S.J., H.W. Meyer, D.S. Kleponis, and K.D. Kimsey, “Simulating the Formation and Evolution of Behind Armor Debris Fields,” 2005 Department of Defense (DoD) High Performance Computing (HPC) Users Group Conference, Nashville, TN, June 2005.
8. Clarke, J.A., C.E. Schmitt, and J.J. Hare, “Developing a Full Featured Application from an Existing Code Using the Distributed Interactive Computing Environment,” Proceedings of the Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) Users Group Conference, U.S. Department of Defense, Washington, D.C., 1998.

Validation and Installation of LSF on Large IBM AIX Clusters

T.J. Lee, NAVO MSRC IBM System Administration, Jeff Gosciniak, NAVO MSRC User Services

The Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) is committed to establishing consistent, and where possible, common computing environments across the HPCMP.

This commitment is demonstrated by the efforts of the HPC Modernization Office (HPCMO) Meta Computing Working Group and the Baseline Configuration Team. The goal of these efforts is to create a highly common operating environment. This environment would enable users to utilize any resource in the HPCMP with a minimal effort.

The Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC) has firmly embraced this vision and actively participates on the various commonality initiatives.

A key component of this common operating environment is the queuing system. To be effective, this common queuing system must run efficiently on all HPCMP architectures. In Fiscal Year 2004, the HPCMO selected Platform Computing's Load Sharing Facility (LSF) to be the common queuing system.

This article describes the efforts to validate, enhance, and verify LSF on the IBM AIX Cluster architecture.

MIGRATING FROM LOADLEVELER

Prior to the selection of LSF as the HPCMO's common queuing system, the NAVO MSRC used LoadLeveler (LL), the IBM native scheduler, on its IBM resources.

Since it is an IBM product, LL is tightly integrated into a number of additional IBM utilities, such as Workload Manager (WLM). LL uses WLM to give greater control over Central Processing Unit (CPU) and real memory resource allocation.

WLM monitors system resources and regulates its allocation to processes running on AIX, which prevents jobs from interfering with each other when they have conflicting resource requirements. WLM achieves this control by creating different classes of service and allowing attributes to be specified for those classes.

LL dynamically generates WLM classes with specific resource entitlements. This is done for each node that a job step is assigned to execute on. LL creates and assigns each job step to its own WLM class (based on a job step's resource requirements). LL then defines “resource shares” for those WLM classes. These resource shares represent the job's resource usage in relation to the amount of resources available on the machine.

Since AIX resources are only allocated to a WLM class with active processes, WLM resource percentages are calculated based on the total number of shares requested by all active WLM classes.

In other words, WLM creates a desired resource entitlement for processes within each WLM class by assigning a dynamic percentage equal to the resource shares of that class divided by the total shares of all active WLM classes. It is important to note that AIX Workload Manager will only enforce these percentages when the resources are under contention.

CHALLENGES

Historically, LSF was used to run serial and parallel code on systems smaller than those at NAVO MSRC. In addition, LSF was not deployed on resources using AIX, IBM's UNIX operating system.

Therefore, Platform and NAVO MSRC knew that significant test time was required to validate LSF on this challenging architecture. There are many capabilities required for a high performance computing queuing system. Some of the most critical capabilities are:

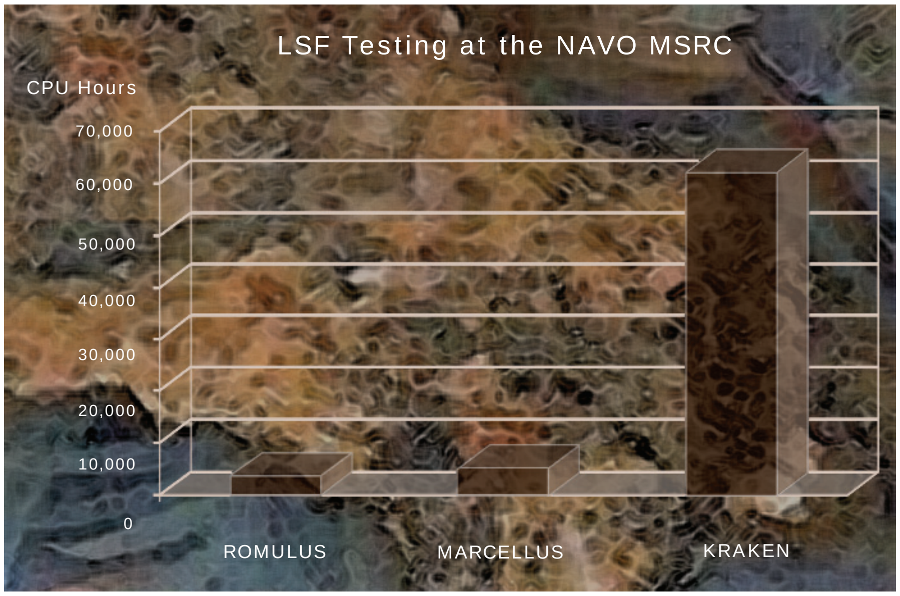


Figure 1. CPU hours used during LSF testing on NAVO MSRC HPC resources.

Continued Next Page...

- Scalability. The system must scale up to the number of available processors without any degradation of service. Failure to scale across the entire machine effectively reduces the available resources of the machine, making a large cluster rather small.
- Efficiency. The queuing system must quickly dispatch and kill jobs, even those across large number of processors. A queuing system that requires significant time for dispatching and killing jobs will greatly reduce the utilization of a resource; a significant amount of time will be consumed in a non-productive manner.
- Reusability. The queuing system must ensure that all resources are released when a job successfully completes or abends. By incompletely releasing resources, the queuing system will incrementally shrink the machine until there are no longer any available resources.
- Memory Management. The queuing system must effectively ensure that running jobs do not oversubscribe memory, causing nodes to hang or fail.

During the initial test period, NAVO MSRC identified scalability and resource management (adapter window management issues and memory management) issues on the IBM p4 system, MARCELLUS, and the p4+ systems, ROMULUS and KRAKEN.

On MARCELLUS which has a Colony switch, jobs could not scale beyond 176 processors; MARCELLUS has 1,328 processors available for computation. On KRAKEN and ROMULUS, which have the newer Federation switch, jobs could not scale beyond 80 processors, but needed to scale at least to 2,832 on KRAKEN.

In addition, the job dispatch and kill processes were prohibitively slow as a result of an adapter window allocation and release issues the required time increased as job size increased. Also, adaptor windows did not always release following job completion (without regard to jobs completing successfully, exiting due to error, or manual deletion of the job), leaving adapter windows unavailable for future allocation and causing follow-on jobs to fail immediately. Lastly, memory management did not work, and there was no means to prevent jobs from oversubscribing memory and hanging or killing nodes.

Since these issues are critical and pose daunting technical challenges, the NAVO MSRC teamed with Platform and IBM to resolve these issues. The NAVO MSRC worked directly with Platform to develop additional test regimens and incremental testing of LSF enhancements. IBM provided programming support and access to their small test systems to assist in the resolution of the LSF problems on the Federation switch architectures. The NAVO MSRC dedicated time on their production machines to assist in the scalability and resource release problems.

The NAVO MSRC, Platform, and IBM team dedicated a tremendous amount of resources and time to resolve these issues.

The NAVO MSRC alone devoted 12 man-months and 71,328 CPU hours to this effort. Figure 1 displays a breakdown of the testing time per NAVO MSRC resource. The testing spanned from October 2004 through August 2006. Although there were many trials and tribulations, this dedicated team achieved success. LSF was effectively tested on MARCELLUS in March 2005.

Due to the complexity of the Federation switch and the large size of KRAKEN, the team completed its testing in August 2006. Once LSF was successfully tested on the IBM Clusters, the migration to the LSF queuing system could begin.

MIGRATING TO LSF

MARCELLUS became fully operational with LSF in April 2005. LSF v6.1 has been operating efficiently with minimal problems. It will continue to be the queuing system on MARCELLUS until the system's decommission, estimated to be the end of calendar year 2006.

In May 2006 LSF v6.1, with Multi-Cluster, was installed on a four-node partition on ROMULUS. In this configuration 60 ROMULUS nodes were allocated to LL for scheduling batch jobs, and 4 nodes allocated as computational nodes for LSF. In July 2006 three additional nodes were added to the LSF compute pool to make a total of seven LSF server nodes on the system.

Multi-Cluster enables an organization to form cooperating clusters of computers so that load sharing happens not only within clusters, but also among them. The NAVO MSRC formed a multi-cluster with Fleet Numerical Meteorology and Oceanography Center (FNMOC).

In this multi-cluster arrangement the FNMOC cluster is a submitting or client cluster only, while the NAVO MSRC cluster (ROMULUS) is an execution only or server cluster (i.e., users at FNMOC submit jobs from their LSF cluster that are sent to, and run on, the NAVO MSRC cluster).

In October 2006 LSF will be installed on KRAKEN in preparation for the new fiscal year (2007). Also, the two new IBM Power5+ Systems, BABBAGE and PASCAL, will have LSF v6.2 when they become production resources.

Validating and verifying LSF on large IBM AIX Clusters was a daunting technical challenge. However, the NAVO MSRC, Platform, and IBM team were able to identify, correct, and verify the critical operational functions required of a HPC queuing system. This dedicated effort will come to fruition in Fiscal Year 2007 as the NAVO MSRC installs LSF on the computational resources, KRAKEN, BABBAGE, and PASCAL.

The Remote Visualization Resource Manager

Sean Ziegeler, Naval Oceanographic Office, Major Shared Resource Center

The Remote Visualization Resource Manager (RVRM) was created by the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC) to allow users of a UNIX-based visualization system to remotely access its graphics hardware rendering capability.

While several software programs and tools (e.g., ParaView, EnSight, and Chromium) provide remote visualization options via some sort of client/server architecture, other tools (like VirtualGL) allow nearly any graphics program to be run remotely and with good performance. However, none of these programs and tools solve a fundamental problem: how to securely access graphics hardware remotely.

The problem arises because, in order to communicate with a Graphics Processing Unit (GPU) on a UNIX-based system, a visualization program must, either directly or indirectly, issue OpenGL commands that are encapsulated as X11[1] GLX[2] commands.

X11 (also known as “X-Windows” or just “X”) is a UNIX standard for computer graphics. GLX is an extension of X to support Three Dimensional (3D) graphics, usually with a GPU. (See Figure 1 for a simplified illustration of this process.) This encapsulation requires that the program have access to the X server; the server that has exclusive access to the GPU.

Given the design of the X security model, this means that the remote user's program must be able to open and manipulate graphical windows on the remote system.

This is currently possible at a cost: security. An X server typically belongs exclusively to a single user; that is, the user who is physically present and logged in at the console or, if a user is not logged in, the System Administrator.

By opening the X server to anyone (via the “xhost +” command[3] or the “-ac”[4] command line option), the remote user will have the necessary access, but, then again, so will everyone else.

An additional motivation for the development of the RVRM was to allow multiple users, local and remote, to share multi-GPU remote visualization resources. For example, several users may only require a few GPUs of a graphics cluster and so could efficiently share the cluster.

RVRM allows users to reserve multiple GPUs, which enables sharing without the users inadvertently interfering with each other and maintains rudimentary load balancing for the system.

Solution

Because an X server is so closely tied to a given GPU, the RVRM allows a remote user to reserve an entire X server just as if the user had physically logged in at the console.

When the remote user uses RVRM after logging in, the X server currently running is shut down and a new one is started. The remote user alone is given the “X-Authority” credentials to access this new X server.

A simple example for making such a reservation for a single system follows:

```
rvreserve :0
```

The parameter that follows rvreserve is the name of the “display” that corresponds with the X server. By default this is usually “:0”. In systems with multiple GPUs there will be multiple X server displays, each with their own display names like “:1” or “:2”.

If properly configured, the rvreserve command may also be used to reserve displays on multiple nodes of a graphics cluster. The host name of a single node must be given followed by, without spaces, the display name. Successive nodes and display names may follow, separated by a space. For example:

```
rvreserve node1:0 node2:0 node3:0
```

Replacing the Display Manager

The most important part of granting a reservation to a user is starting a new X server and giving the user access to it. To make this possible and secure,

Continued Page 27...

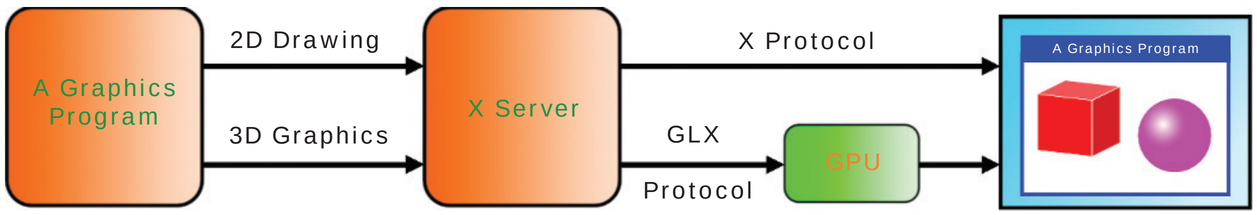


Figure 1. An illustration of a graphics program using 2D and 3D graphics via the X server.

CFD Simulation of Flow and Dispersion in Complex Buildings

Phu V. Luong, Mario J. Sanchez, and Robert S. Bernard
ERDC Coastal and Hydraulics Laboratory

William J. Croisant, David M. Bailey, Dale L. Herron,
Dahtzen Chu, David T. McKay, David M. Schwenk,
Chang W. Sohn, and David M. Underwood
ERDC Construction Engineering Research Laboratory

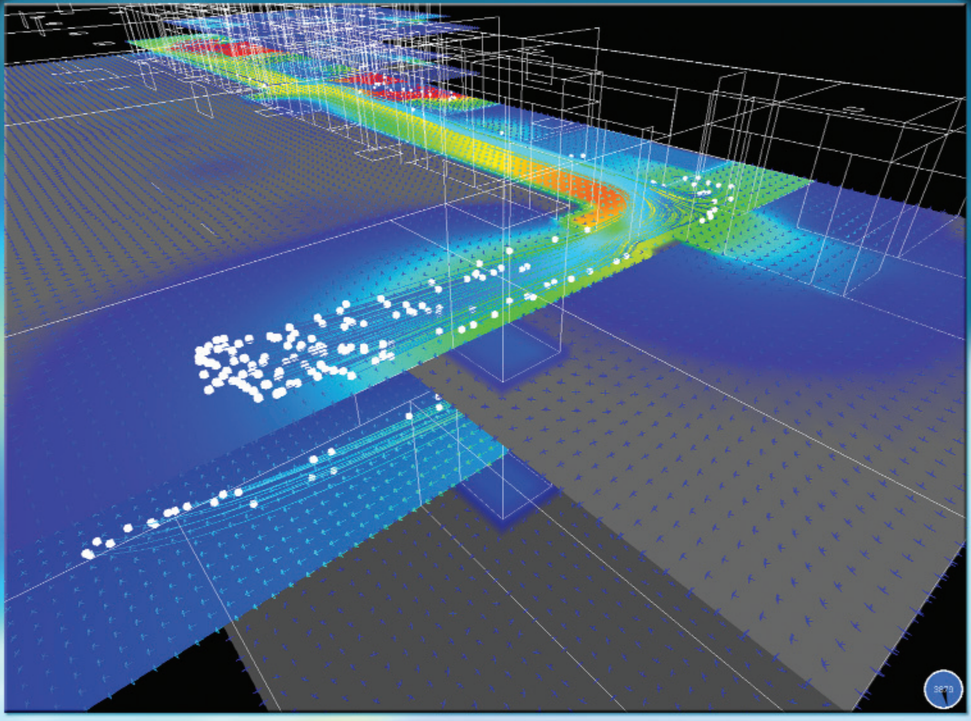
Sean Ziegeler
Visual Analysis and Data Interpretation Center, NAVOCEANO MSRC

Overview

Computational Fluid Dynamics (CFD) is now being used to simulate air flow and contaminant dispersion inside complex buildings. A building's air flow may be produced by a combination of forced air, direct ventilation, and internal/external air leakage. With sufficient time, the flow may transport airborne contaminants from their point of release to remote locations in the building. The term "contaminant" refers here to any material, whether harmful or beneficial, that is carried aloft and dispersed by the moving air. Moreover, the contaminants may consist of solid particles, liquid droplets, or superfluous gases.

Whole-building CFD simulations may be used to help design or improve Heating, Ventilation, and Air Conditioning (HVAC) systems. For an existing or proposed HVAC system, simulations may also be used to predict the fate of toxic Chemical and Biological (CB) agents delivered by an internal or external source. When presented in appropriate visual form, the output from these simulations allows building designers and security officials to assess threats posed by various modes of CB release and to develop effective countermeasures and emergency response plans.

For applications of this kind, prospective CFD codes should be able to accommodate multi-level buildings with many rooms and vents on each level and return-air spaces (plenums) separating the room ceilings from the floor (or roof) above. CFD is a fairly mature branch of computational science, and there are many publicly and privately developed codes now available with the required capabilities. The CFD code employed for the work discussed here was PAR3D,[1,2] which was developed by the Coastal and Hydraulics Laboratory (CHL) of


Particles traced several minutes into the simulation show a bifurcation in the flow at the end of a hallway, with some particles turning into a room and others turning to descend a stairwell.

the Engineering Research and Development Center (ERDC), U.S. Army Corps of Engineers, in Vicksburg, Mississippi. The related flow simulations were executed on the KRAKEN IBM Power4+ system at the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC).

PAR3D Code

The PAR3D code is a general-purpose numerical model that uses multi-component grids and parallel processors for computing three-dimensional incompressible flow.

With its parallel-processing capability, it represents an extension of the earlier MAC3D code,[3] which employed multi-component grids but was limited to single (vector) processors. PAR3D uses finite-volume discretization with curvilinear Marker-And-Cell (MAC) grids to solve the Reynolds-Averaged Navier-Stokes (RANS) equations and the associated mass-transport equations for dispersion. The MAC designation[4] refers to the volume-averaged evaluation of scalar quantities inside the grid cells and the surface-averaged evaluation of normal vector components on the cell faces. Eddy viscosity/diffusivity for the RANS and mass-transport equations is supplied by the standard κ-ϵ turbulence model, originally developed by Launder and Spalding.[5]

The computational grids are constructed such that, regardless of their shape in physical (x, y, z) space, they map directly into rectangular grids in computational (i, j, k) space. To improve local resolution and geometric flexibility (and to facilitate implementation on parallel processors), the grids are subdivided into structured components. The term "structured" implies that each grid cell is a hexahedron and that neighboring cells in each grid component are identified with sequential integer (i, j, k) indices. Each grid component occupies a separate parallel processor, and the standard Message Passing Interface (MPI) library[6] is used to transfer shared information between processors.

Model Building

To demonstrate modeling capability for complex buildings, a grid was developed for an existing two-story building with two corridors, three stairwells, one elevator, and more than 60 rooms. Both levels of the building had 9-foot ceilings, each of which was separated from the floor (or roof) above by a 4-foot plenum. Figure 1 shows a plan view for each level of the grid, which had a uniform spacing of 1 foot in all three (x, y, z) directions.

In the case reported here, a single air handler provided air to all the supply vents with a fresh-air content of 74 percent. Return air was routed through the plenums, into a transfer duct, and back to the main air handler. The elevator shaft was assumed to be completely closed, and this portion of the grid was excluded from the computed flow.

The airflow boundary conditions included room inflows through 69 supply vents, plenum inflows and outflows through 22 Variable Air Volume (VAV) terminal units, plenum return flow to the main air handler, first-floor plenum outflow through a dedicated exhaust air handler, and room outflows through 13 exhaust vents.

Air leakage to the outside was imposed at the joints where the walls met the floors and ceilings. The leakage boundary conditions were established from fan pressurization tests, which employed blower door assemblies and followed the procedure described in the Annual Book of ASTM Standards.[7]

Using combinations of single zone and guarded zone tests as discussed by Feustel,[8] air leakage curves were deduced for individual zones.

Envelope leakage parameters were then established based on these curves, along with pressure differentials that were measured across the envelope boundaries. The building HVAC system was operating in full cooling mode, with all exterior doors and windows closed, and all interior doors fully opened.

Contaminant Release

The model building was subjected to two simulated modes of release. In both cases the contaminant source occupied a single grid cell (with a

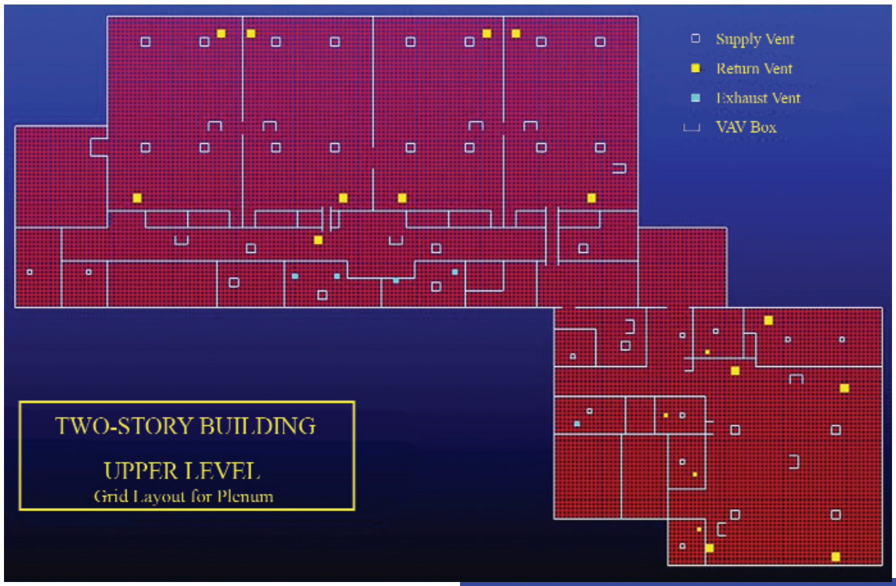


Figure 1a. (Above) Overhead view of plenum for upper level.

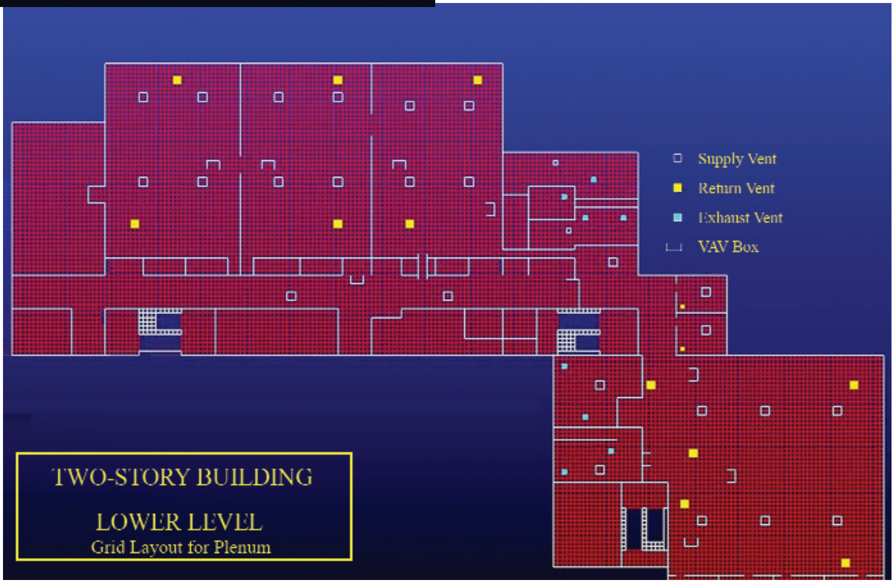


Figure 1b. (Right) Overhead view of plenum for lower level.

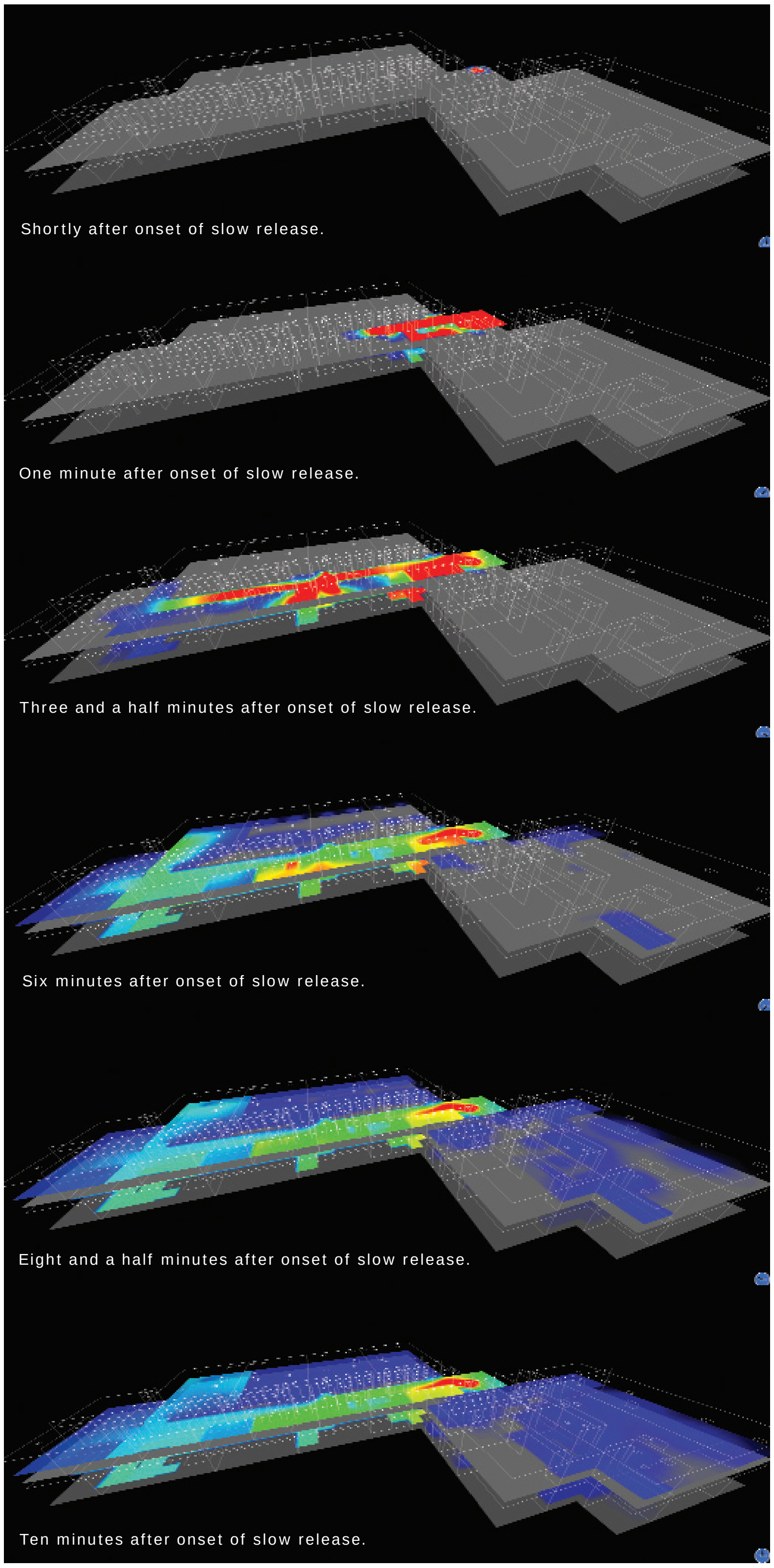

Shortly after onset of slow release.
One minute after onset of slow release.
Three and a half minutes after onset of slow release.
Six minutes after onset of slow release.
Eight and a half minutes after onset of slow release.
Ten minutes after onset of slow release.

Figure 2. Visualizations tracking concentration and dispersal of slow-release contaminants from source through hallways, to open rooms, and via the ventilation system. After ten minutes concentration remains highest in open rooms/hallways.

volume of 1 cubic foot) centered on the floor of the lobby in the upper level of the building. In the first simulation, contaminant was released slowly, at a rate of 1 gram per second, for 10 minutes. Figure 2 shows the contaminant dispersion for the slow-release mode as it unfolded over time. The varying colors indicate the local (instantaneous) concentration of the contaminant, with blue for the lowest values, followed by green, yellow, and finally red for the highest values.

In the second simulation, 600 grams of contaminant were released instantaneously. The amount released was equal to the entire amount released over 10 minutes in the first simulation. Figure 3 shows the contaminant dispersion for the quick-release mode as it unfolded over time. The same range of colors is used to indicate the concentration, so that corresponding figures for quick and slow release may be compared directly.

Visualization

Visualizing output from simulations of this kind is important, because it allows modelers first to see qualitative flow and dispersion patterns at a glance, and then to study the quantitative details in depth.

The ToxTrack visualization software can interactively display both the vector and scalar variables that are output by the CFD code. In this case, the scalar is contaminant concentration, which is portrayed by color maps on horizontal cutting planes that can be selected at different heights in the grid, as exemplified in Figures 2 and 3.

Conversely, the flow velocity is a vector field, which can be represented at any instant by three-dimensional arrows like those in Figure 4.

To observe the transportive nature of the flow directly, one may insert tracer particles at arbitrary locations in the flow field, and then watch as the particles are

Continued Next Page...

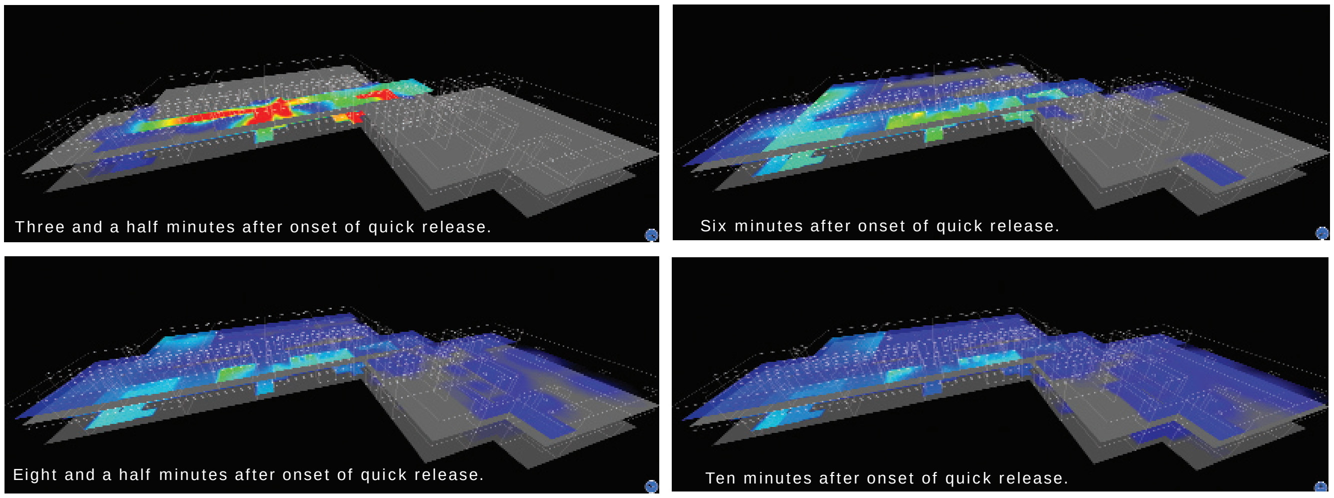

Three and a half minutes after onset of quick release.
Six minutes after onset of quick release.
Eight and a half minutes after onset of quick release.
Ten minutes after onset of quick release.

Figure 3. The departure from the slow-release mode is readily apparent. As in the slow-release mode, the contaminant entered the ventilation system and spread through the supply vents, but the highest concentrations are noticeably smaller. The contaminant spread to all rooms with supply vents, but the highest concentrations decline over time because contaminated air is continually refreshed.

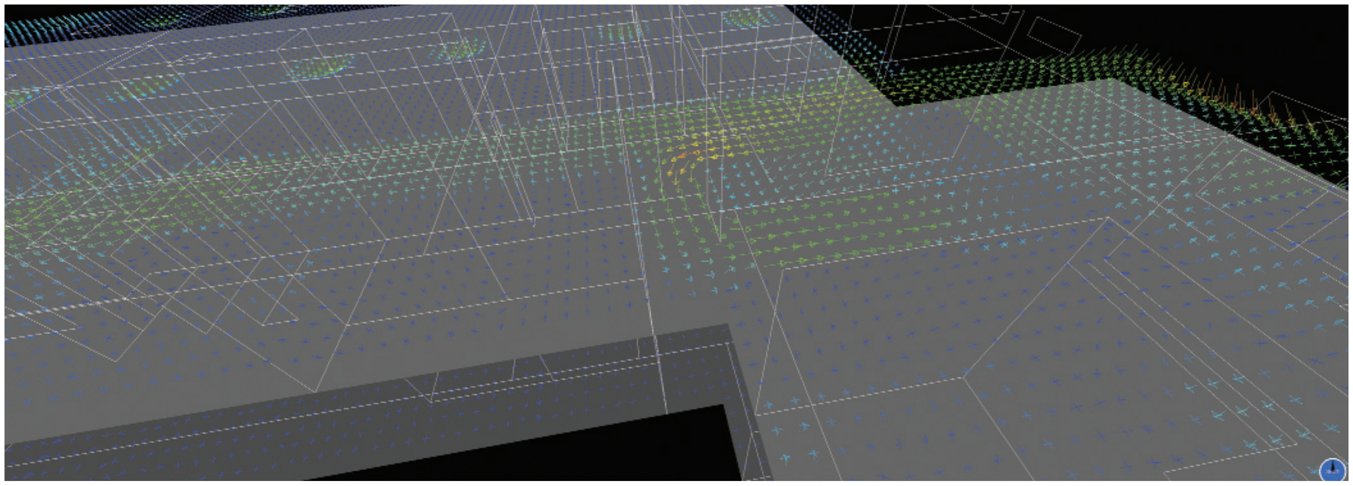


Figure 4. Velocity vectors for the flow at a height of 4 feet above the floor.

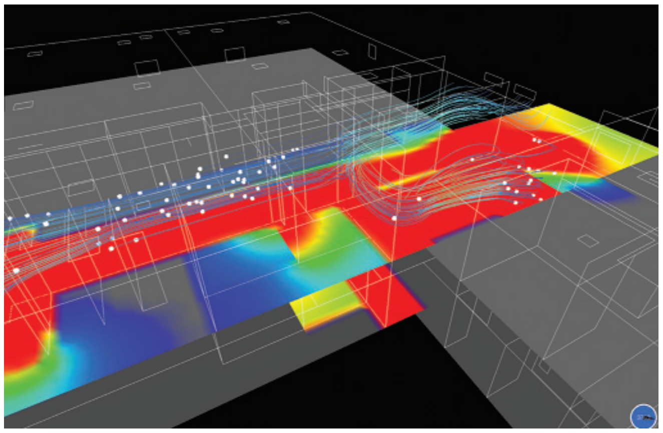


Figure 5. Particles traced from the initial release site show a bifurcation in the flow, with some particles continuing down the hallway and others turning to descend a stairwell.

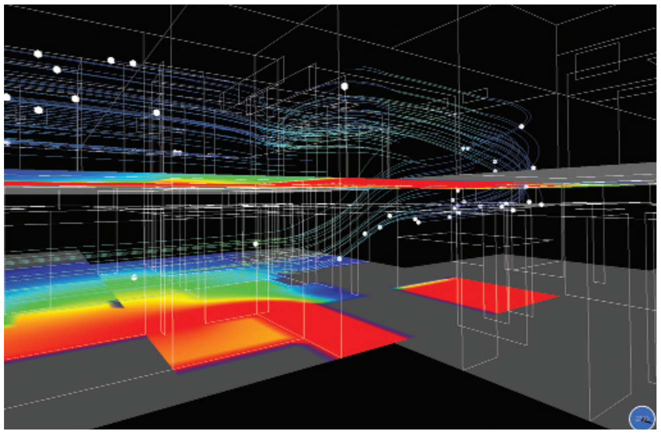


Figure 6. Particles that entered the stairwell begin traveling down to the first level. This corroborates the trend in the color map of concentration, which indicates that the contaminant quickly spreads downward.

carried downstream in real time. Figures 5 and 6 show, for example, consecutive snapshots of tracer particles and their instantaneous streak lines before and after their paths diverge at a stairwell.

Conclusion

The most daunting task in modeling flow and dispersion in complex buildings is the lengthy process of preparing data for input to the CFD code. This includes the placement of walls, doors, and supply/return/exhaust vents; the characterization of internal and external air leakage; and the determination of flow rates through all the supply and exhaust vents. Grid construction, CFD code execution, and data post-processing (including visualization) currently require much less time than does input preparation.

As this type of effort becomes more common, however, automated systems will be developed that will eliminate most of the preliminary drudgery.

Eventually the total time needed for the building characterization, the CFD simulation, and the analysis of computed results may be reduced from several months to a few days or a few hours, even for very complex buildings.

And, adopting the likely viewpoint of prospective building occupants, one might say the quicker the better.

References

1. Bernard, R.S., "User Manual for the PAR3D Numerical Flow Model," Unpublished Document (available upon request from the author), ERDC Coastal and Hydraulics Laboratory, Vicksburg, MS (2004).

2. Bernard, R.S., "Technical Manual for the PAR3D Numerical Flow Model," Unpublished Document (available upon request from the author), ERDC Coastal and Hydraulics Laboratory, Vicksburg, MS (2004).

3. Bernard, R.S., "MAC3D: Numerical Model for Reservoir Hydrodynamics with Application to Bubble Diffusers," Technical Report CHL-98-23, U.S. Army Engineer Research and Development Center (ERDC) Coastal and Hydraulics Laboratory, Vicksburg, MS (1998).

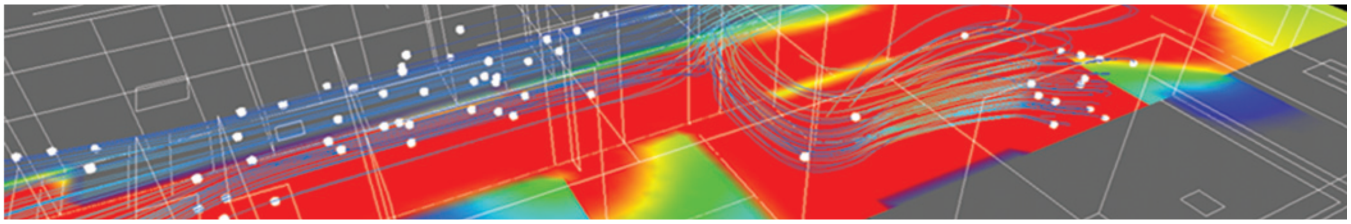
4. Bernard, R.S., and H. Kapitza, "How to Discretize the Pressure Gradient for Curvilinear MAC Grids," Journal of Computational Physics, Vol. 99, No. 2, pp. 288-298, 1992.

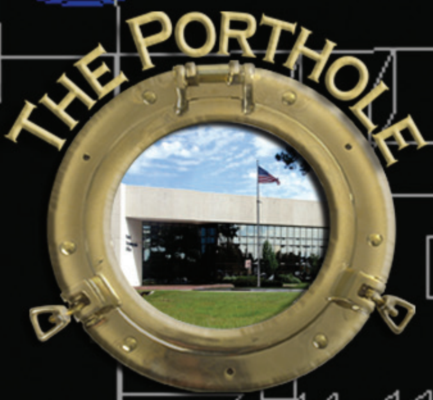
5. Launder, B.E., and D.B. Spalding, "The Numerical Calculation of Turbulent Flows," Computer Methods in Applied Mechanics and Engineering, Volume 3, pp. 269-289, 1974.

6. Gropp, W., E. Lusk, and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface," MIT Press, Cambridge, MA (1994).

7. American Society of Testing and Materials (ASTM), "Standard Test Method for Determining Air Leakage Rate by Fan Pressurization," ASTM E779-99, Annual Book of ASTM Standards, Vol. 04.11, 1999.

8. Feustel, Helmut E., "Measurement of Air Permeability in Multizone Buildings," Energy and Buildings, Volume 14, Issue 2, 1990.





(L-R) Charles Martinek, NAVOCEANO Technical Director, Stephen Burich, Commander, Undersea Surveillance (CUS), Steve Adamec, Director, NAVO MSRC.

Representatives of GOOGLE Earth visit the NAVO MSRC.

(L-R) Dr. Walter Dixon, Enterprise Architect, Office of Naval Research (ONR), Mike McConnell, contractor, Steve Adamec, Director, NAVO MSRC, and John A. Lever, Director, Information Architecture Governance, Naval Meteorology and Oceanography Command.

Steve Adamec, Director, NAVO MSRC with representatives of the Government Printing Office (GPO).

(L-R) Chief of Naval Operations, ADM Michael Mullen and Commander, Naval Meteorology and Oceanography Command, RDML Timothy McGee.

NAVOCEANO Boat students visit the NAVO MSRC.

Scramjet..continued

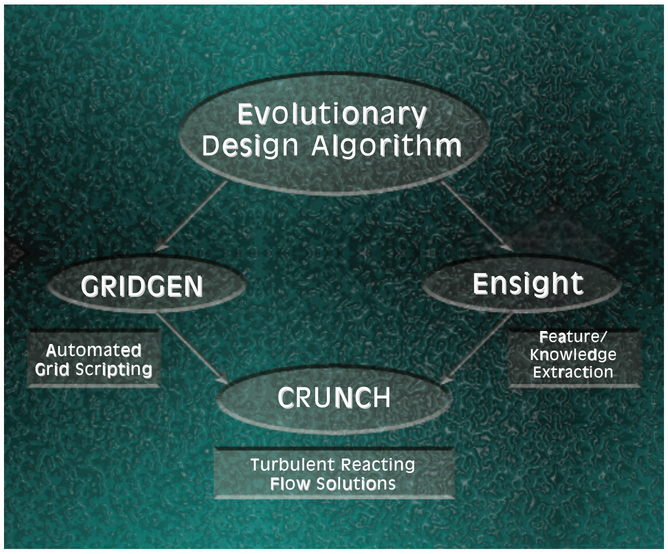

Evolutionary Design Algorithm
GRIDGEN
Ensight
Automated Grid Scripting
Feature/ Knowledge Extraction
CRUNCH
Turbulent Reacting Flow Solutions

Figure 6. GUI driven–automated optimization of CFD problems with several variables, constrains, and objectives with complex design landscapes.

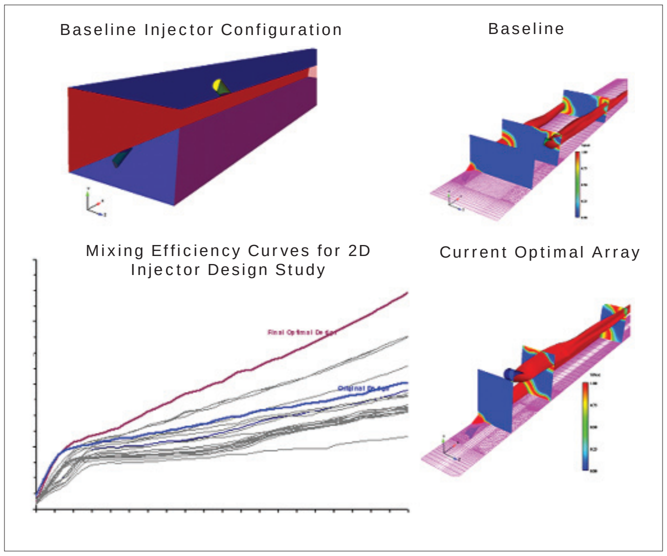


Figure 7. Fuel injector optimization study.

has been coupled to the CRUNCH CFD® code and the grid generation package, GRIDGEN.

This methodology was chosen because the search procedure is inherently parallel and has worked well in a number of earlier multi-variant design applications. Figure 7 shows the fuel injector optimization performed for a rectangular combustor with upper and lower interdigitated, flush fuel injectors. Parameter space included spacing, cant angle, and off-set, with the fuel/air ratio (=1.2) and injector diameter kept constant. Each case was performed on 64 processors, with 5 cases running simultaneously and 6 design levels performed to date yielding a close-to-optimal solution.

Summary

This article has discussed the application of RANS CFD codes to the evaluation and design of hypersonic scramjet components in support of a next-generation missile system. Advanced models for turbulence and use of multi-element UNS grid adaptation have improved overall accuracy, while use of dynamic domain decomposition and other efficiency enhancements have led to rapid solution throughput using available NAVO MSRC resources. Design optimization for fuel injector patterns and injection conditions has reached a practical state making use of evolutionary algorithms and GUI's to automate the process at each design level.

Acknowledgements

This work was supported by the U.S. Army Aviation and Missile Research, Development, and Engineering Center (AMRDEC) under several programs performed under the direction of Dr. Billy Walker. NAVO MSRC resources were made available to us under a multi-year Challenge Grant and varied platforms were used during the course of this program.

References

1. Calhoon, W.H., Jr., K. Brinckman, D.C. Kenzakowski, N.H. Sinha, and S.M. Dash, "Progress In Turbulent Combustion Modeling For Rocket Plumes," 28th Exhaust Plume Technology (EPTS) Joint Army, Navy, NASA, Air Force (JANNAF) Meeting, San Diego, CA, Nov. 1-5, 2004.
2. Joint Army, Navy, NASA, Air Force (JANNAF) Interagency Propulsion Committee, Combustion Science Meeting papers presented between 2000-2006.
3. Ahuja, V., and A. Hosangadi, "Design Optimization of Complex Flowfields Using Evolutionary Algorithms and Hybrid Unstructured CFD," AIAA-2005-4985, 17th Computational Fluid Dynamics Conference, Toronto, CA, June 2005.

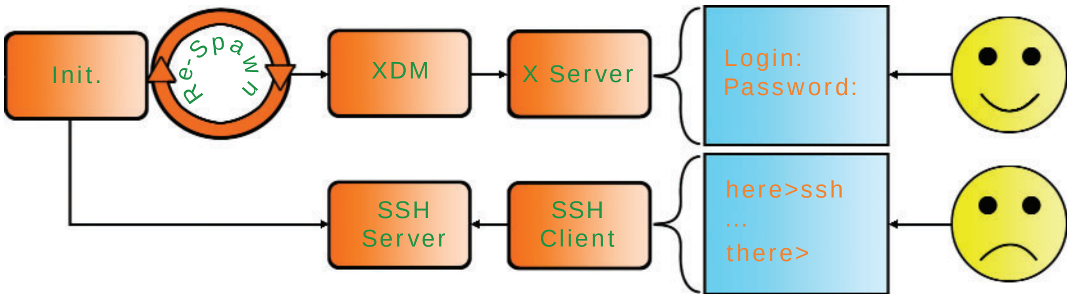


Figure 2. A simplified diagram of the initialization procedure for UNIX systems with respect to the X server and remote login access.

Remote Visualization...continued

some changes must be made to the system configuration. Figure 2 illustrates the default system configuration for most UNIX-based systems. Everything starts from the "init"[6] configuration file, which starts and maintains server processes.

A system configured to run X will usually start a process known as the X Display Manager (XDM)[7] or some variation thereof.[8,9] The XDM, in turn, executes an X server and asks a local user for authentication to log in. Notice that the initialization of the XDM process is configured to "respawn." Whenever a local user logs out, the X server and XDM exit and are re-executed by init. The next local user will get a fresh, new X server.

Completely separate processes are responsible for allowing remote users to log in. One example of many is the Secure Shell (SSH) server. These provide a text-only, command-line interface to a remote user. Access to an X server is not included by default.

When a remote user needs an X server, a new X server must be restarted without XDM. This new X-server can then be handed over to the remote user. This creation and transfer of the new X server is the purpose of a key component of RVRM: the Remote Visualization Display Manager (RVDM). Figure 3 illustrates how the RVDM becomes the new "init respawn" process. Two additional components are added for security (rvxlock) and convenience (display access).

RVRM and the components of RVRM in Figure 3 interact as follows:

1. Init will execute (i.e., spawn) RVDM which, without any reservations, RVDM executes XDM as usual, allowing a local user to log in if desired.
2. If a remote user logs in (e.g., via SSH) and executes the rvreserve command, rvreserve grants the reservation (assuming no other user has one) and "kills" the current RVDM process, allowing init to re-spawn a new RVDM.
3. The new RVDM detects the previously granted a reservation and starts a new X server.

Continued Next Page...

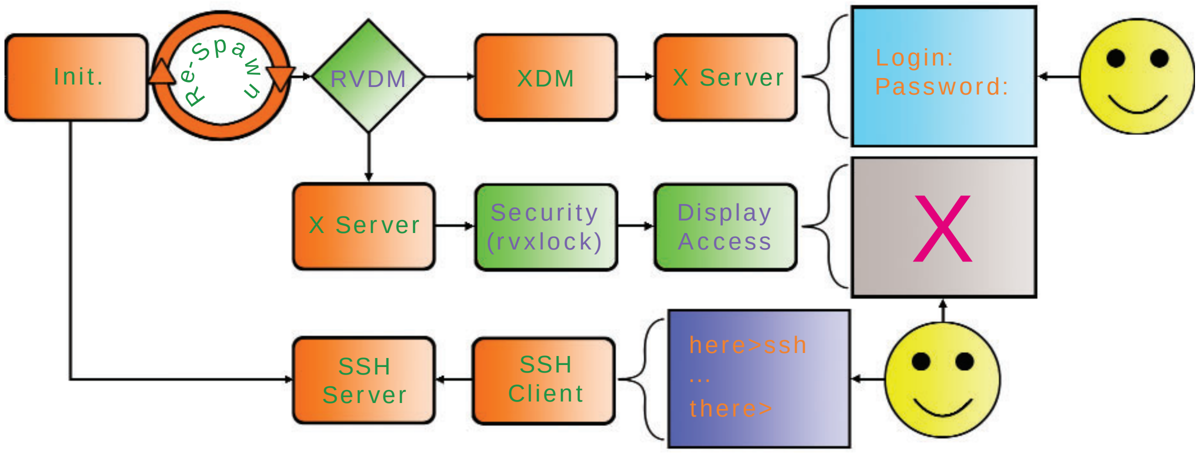


Figure 3. A diagram of the modified initialization procedure using RVRM.

4. RVDM activates a security measure (see the Security section) as a root process.
5. RVDM grants X server access to the remote user (see the Display Access section).
6. When finished, the remote user executes the rvrelease command, which clears the reservation and kills the current RVDM.
7. The previous step triggers the re-spawn of a new RVDM, and so the process returns to Step 1.

SECURITY

Security issues arise when a remote user is allowed to create and manage windows on an X display. The remote user must be protected from local users and vice versa.

Without such protection, local users can sit at the console and manipulate the remote user's windows, and remote users can post a fake log-in screen and capture authentication information from local users attempting to log in.

Unfortunately, the X security model does not protect against such situations by itself. Moreover, disabling the mouse and keyboard typically requires modification of the X server code; and an approach that does not require maintenance of a separate X server code base is preferred.

X does provide system calls that "grab" the mouse[10] and keyboard.[11] Xlock is a commonly used and mature screen locking program that uses the X system calls to grab the keyboard and mouse. However, Xlock in its standard Central Processing Unit (CPU) and GPU hogging configuration is not the ideal solution.

Rather than using a retrofitted Xlock, RVRM provides a program, rvxlock, which consists of only the code necessary to perform the mouse and keyboard grabbing and one screen-saver mode that runs in the background (behind all other windows). The screen-saver mode simply prints a message that informs local users that the display is reserved. The result is effective and uses a small executable that requires a negligible amount of CPU or GPU time.

Rvxlock runs as a root process to prevent user tampering. Running in the root window is also an extra security precaution. It is difficult (if not impossible) for a user with access to the X server to "close or kill" the root window. But just in case, RVDM executes rvxlock in a loop so the lock is re-acquired immediately. If rvxlock cannot regain its keyboard and mouse grabs, then it assumes that something nefarious is happening and aborts the entire session.

DISPLAY ACCESS

When a reservation is created for a remote user, the credentials for the X server, known as X-Authority[12] data, are also created. For security reasons, these are stored in a file on a local disk (usually /tmp). A user will typically have existing X-Authority data in a separate file. If multiple reservations are made on a cluster, then the reservations on each node will be in separate files on separate disks.

The rvreserve command automatically aggregates all X-Authority data and distributes them to every reserved display. In addition, if the remote user has an existing default X-Authority file, the data are added to that file. Thus, the remote user is granted access to every reserved X server display.

CONCLUSIONS

RVRM condenses the process of obtaining and maintaining access to hardware-accelerated graphics on remote visualization systems into a few simple commands. This allows local and remote users to share multiple graphics resources, like those on a graphics cluster, without interfering with each other. Most importantly, RVRM was developed with security in mind so that remote visualization is possible, both safely as well as easily.

References

1. UNIX Manual Pages: X (1x).
2. UNIX Manual Pages: glXIntro (3x).
3. UNIX Manual Pages: xhost (1x).
4. UNIX Manual Pages: Xserver (1x).
5. UNIX Manual Pages: init (8).
6. UNIX Manual Pages: xdm (1x).
7. Buddenhagen, Oswald, "The KDM Handbook," KDE Documentation, http://docs.kde.org/development/en/kdebase/kdm/.
8. Cameron, Brian, "The GNOME Display Manager," GNOME Projects Listing, http://www.gnome.org/projects/gdm/.
9. UNIX Manual Pages: XGrabPointer (3x).
10. UNIX Manual Pages: XGrabKeyboard (3x).
11. Ayers, Larry. "Xlockmore," Linux Gazette, June 1997, Issue 18, http://linuxgazette.net/issue18/xlock.html.
12. UNIX Manual Pages: xauth (1x).

Running X Windows With Parallel Codes on KRAKEN Under Platform Computing's Load Sharing Facility

John Skinner, NAVO MSRC User Outreach, Sheila Carbonette, NAVO MSRC User Support

This article describes a simple method for running multi-node, multiprocessor programs “interactively” on the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC) IBM resources, KRAKEN, a Power 4+. KRAKEN has 2,832 Central Processing Units (CPUs) under Load Sharing Facility (LSF), Platform Computing's batch scheduling system. We make use of the X Window system and Kerberized Secure Shell (SSH) to securely set up an X connection from the internal network used by KRAKEN's compute nodes back to a user's local computer running X Windows.

Parallel jobs on KRAKEN, such as Message Passing Interface (MPI) codes and OpenMP codes running under IBM's Parallel Operating Environment (POE), have to run on compute nodes and also have to be submitted as batch jobs via LSF. Since KRAKEN's internal network is locally configured to only talk to KRAKEN's two interactive login nodes and the NAVO MSRC Archive Servers, we cannot set our DISPLAY back to our local computer (in the standard X Windows manner) and send the xterm window or other X-based program from an LSF run back to our home computer. Instead we must make use of SSH to handle the required setup.

This method allows us to send xterm windows from a running LSF job back to our home computer for use like normal login windows. We can then run Unix commands, codes that display graphics, manually debug programs, or run the Totalview debugger. We can also start multiple processes in background and rerun our code with mpirun.lsf, The LSF parallel job starter, from the xterm window started within our LSF batch job.

The following procedure outlines the required steps to use whether you have ssh on your home system or not, and an example LSF script is also provided.

PROCEDURE

1. Connect to KRAKEN interactively via one of the Kerberized login commands Kerberized ssh, ktelnet, krlogin, or krsh (if possible, use ssh for its ease of use with the X Windows setup):

```
mycomputer% ssh -l skinman kraken.navo.hpc.mil

Last login: Thu Sep 14 13:11:58 GMT 2006 on
/dev/pts/23 from saturn.navo.hpc.mil

k19n15e%
```

If using ssh from your local system to logon directly to KRAKEN, continue to Step 3.

2. Manually set up and test a valid X Window connection and DISPLAY after logging in, as noted in sub-steps A through F below:

A) Run “xauth list” on your local workstation to list the MIT “magic cookie” string that your X Server uses to authenticate X Clients that connect to your computer.

An xterm window or any other X-based programs you want to run from within your LSF job will be the X Clients that need access to this encrypted "cookie" so they can display on your screen:

```
mycomputer% xauth list

mycomputer:0 MIT-MAGIC-COOKIE-1
057834173e477d52401161623779276a
```

B) Use xauth with the “add” option to cut and paste the appropriate line from your local X setup into your .Xauthority file on KRAKEN. Be sure to use either your local system's IP number or full domain name when adding this to your .Xauthority file on KRAKEN:

```
k19n15e% xauth add mycomputer.at.my.domain:0
MIT-MAGIC-COOKIE
1057834173e477d52401161623779276a
```

NOTE: It should be necessary to execute steps 2.A and 2.B only once, since the information will be saved and valid as long as you don't log off your current interactive login session on KRAKEN.

C) On KRAKEN, set your DISPLAY environment variable to either the IP number or the fully qualified domain name for your local workstation (don't forget the “0.0” part):

```
k19n15e% setenv DISPLAY
mycomputer.at.mydomain:0.0
```

or

```
k19n15e% setenv DISPLAY 204.222.179.104:0.0
```

D) Test the connection by starting an X client such as xclock or xterm from your KRAKEN login window and verifying that it displays on your workstation screen:

```
k19n15e% /usr/bin/X11/xterm
```

E) From the same login window on the IBM, now ssh locally from KRAKEN to KRAKEN itself:

```
k19n15e% ssh kraken.navo.hpc.mil
Last login: Thu Sep 14 20:39:05 GMT 2006 on
/dev/pts/26 from saturn.navo.hpc.mil
k19n15e%
```

F) List the new DISPLAY setting created by this ssh login so that you can hardcode it into your LSF batch script.

```
k19n15e% env | grep DISPLAY
DISPLAY=k19n15e.navo.hpc.mil:11.0
```

Notice that SSH has set your DISPLAY variable in this new login session to a new value, which happens to be one of the two IBM interactive login nodes with an additional ssh identifier at the end of the string.

The sshcommand automatically takes care of setting the DISPLAY and the xauth “magic cookie” setup needed for this new login and connects it back to your existing KRAKEN login, where your DISPLAY is still set to “204.222.177.65:0.0” (or “mycomputer.at.my .domain:0.0”).

The connection to the X11 DISPLAY is automatically forwarded by ssh in such a way that X programs started from a KRAKEN shell will now go through the encrypted channel, and the connection will be made from KRAKEN to the X Server running on your computer.

Once you close the interactive ssh login session, the new DISPLAY setting that ssh just set for you is no longer valid and can't be used again. For this reason, do not abort this ssh connection until you are through with your debugging work or any other X-based programs that you want to run from within your LSF script on the IBM compute nodes (which is where LSF jobs run).

3. Add the DISPLAY information created from Step 2 to your LSF script and bsub your job. Be sure to set the DISPLAY variable within your script before you execute any X commands such as xterm or aixterm within the LSF script itself. You can then use your current interactive login window to run bjobs to monitor your job until it begins execution and sends the new xterm window to your workstation.

SCRIPT

What follows is a sample LSF script that sets up the correct DISPLAY and starts an interactive xterm session from which you can run any KRAKEN commands, start a parallel code with the mpirun.lsf command, or run a debugging session and send the information back to your computer. You can run in this manner until you reach the wallclock limit set for your batch job. Then you can bsub a new ob, again using the same DISPLAY setting.

Remember that if you kill the original ssh-based login session that first set up the DISPLAY, you will have to re-login with ssh and modify your LSF script accordingly to point to the new DISPLAY setting.

```
#!/bin/csh
#BSUB -J my_debug_job
#BSUB -o %J.out
#BSUB -e %J.error
#BSUB -a poe
#BSUB -P NAVOSLMA
#BSUB -W 1:00
#BSUB -q standard
#BSUB -n 32
#BSUB -R "span[ptile=8]"

# delete setenv WORKDIR /scr/skinman/my_test

# delete if (! -e ${WORKDIR}) then mkdir -p
$WORKDIR
# delete endif

cd $WORKDIR

# Copy needed files to $WORKDIR space

cp $HOME/mpi.exe $WORKDIR
cp $HOME/mpi-src/*.f90 $WORKDIR
cp $HOME/input.dat $WORKDIR
```

Set DISPLAY to value set by your ssh login to KRAKEN or, if logged in via ktelnet/krsh/krlogin to KRAKEN, set to the value created by the ssh from KRAKEN to itself.

```
setenv DISPLAY k19n15e.navo.hpc.mil:11.0
```

Start an xterm from this script and send it to your local workstation and use it like a normal login session.

```
echo "trying xterm connection to $DISPLAY..."
xterm
```

You can also run the xterm in background, start up another xterm, and use both windows before quitting the parent LSF batch job.

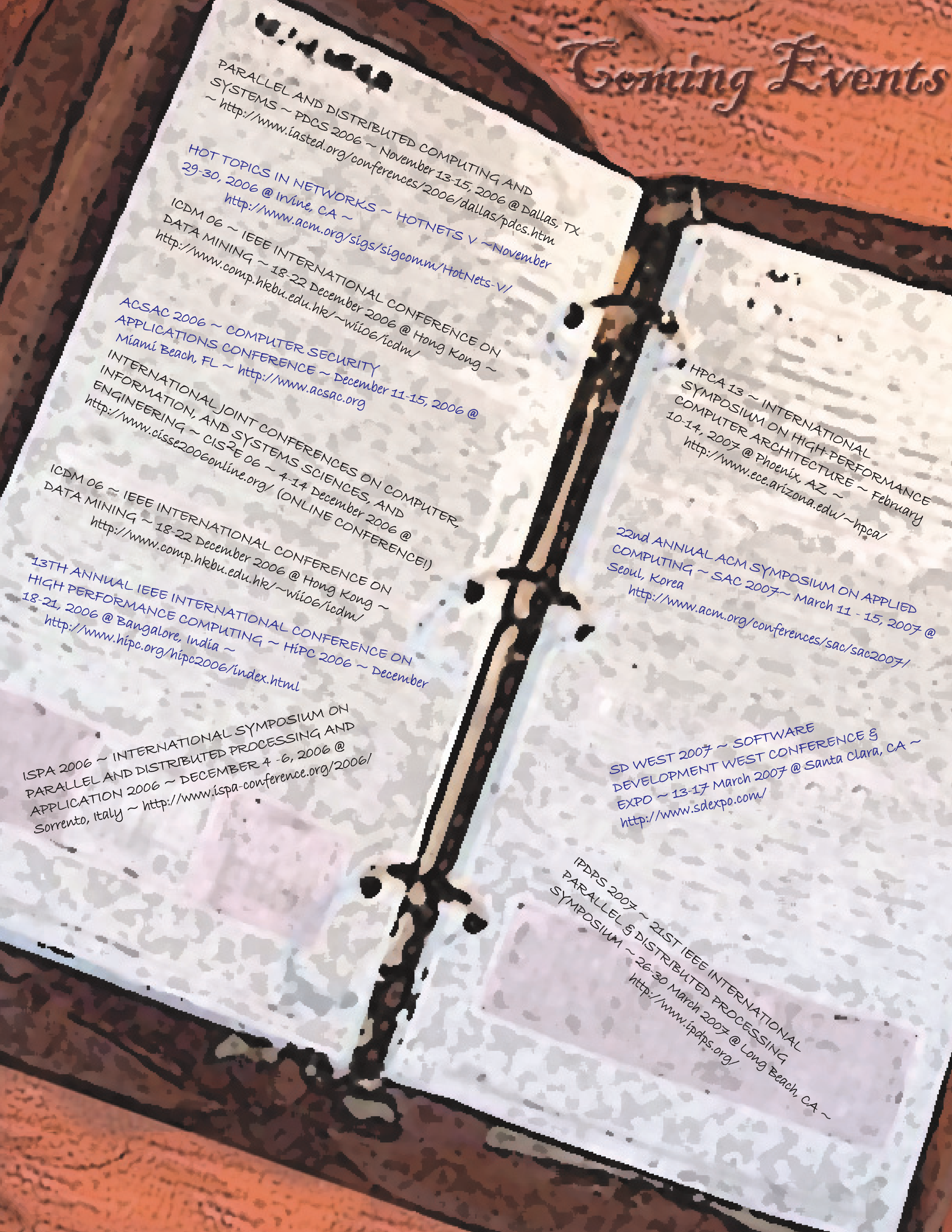
```
xterm&; xterm
```

You can now run your parallel code interactively from within either xterm window and step through your program while it is executing on multiple processors. You can rerun the code under LSF's mpirun.lsf multiple times and also run other commands, but you are limited to no more than the 32 processors this LSF script requests when it is queued with the bsub command.

```
mpirun.lsf ./mpi.exe -procs 32 -labelio yes
```

End of sample LSF script

Coming Events



PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS ~ PDCS 2006 ~ November 13-15, 2006 @ Dallas, TX ~ http://www.iasted.org/conferences/2006/dallas/pdcs.htm

HOT TOPICS IN NETWORKS ~ HOTNETS V ~November 29-30, 2006 @ Irvine, CA ~ http://www.acm.org/sigs/sigcomm/HotNets-V/

ICDM 06 ~ IEEE INTERNATIONAL CONFERENCE ON DATA MINING ~ 18-22 December 2006 @ Hong Kong ~ http://www.comp.hkbu.edu.hk/~wii06/icdm/

ACSAC 2006 ~ COMPUTER SECURITY APPLICATIONS CONFERENCE ~ December 11-15, 2006 @ Miami Beach, FL ~ http://www.acsac.org

INTERNATIONAL JOINT CONFERENCES ON COMPUTER, INFORMATION, AND SYSTEMS SCIENCES, AND ENGINEERING ~ CIS^2E 06 ~ 4-14 December 2006 @ http://www.cisse2006online.org/ (ONLINE CONFERENCE!)

ICDM 06 ~ IEEE INTERNATIONAL CONFERENCE ON DATA MINING ~ 18-22 December 2006 @ Hong Kong ~ http://www.comp.hkbu.edu.hk/~wii06/icdm/

13TH ANNUAL IEEE INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING ~ HiPC 2006 ~ December 18-21, 2006 @ Bangalore, India ~ http://www.hipc.org/hipc2006/index.html

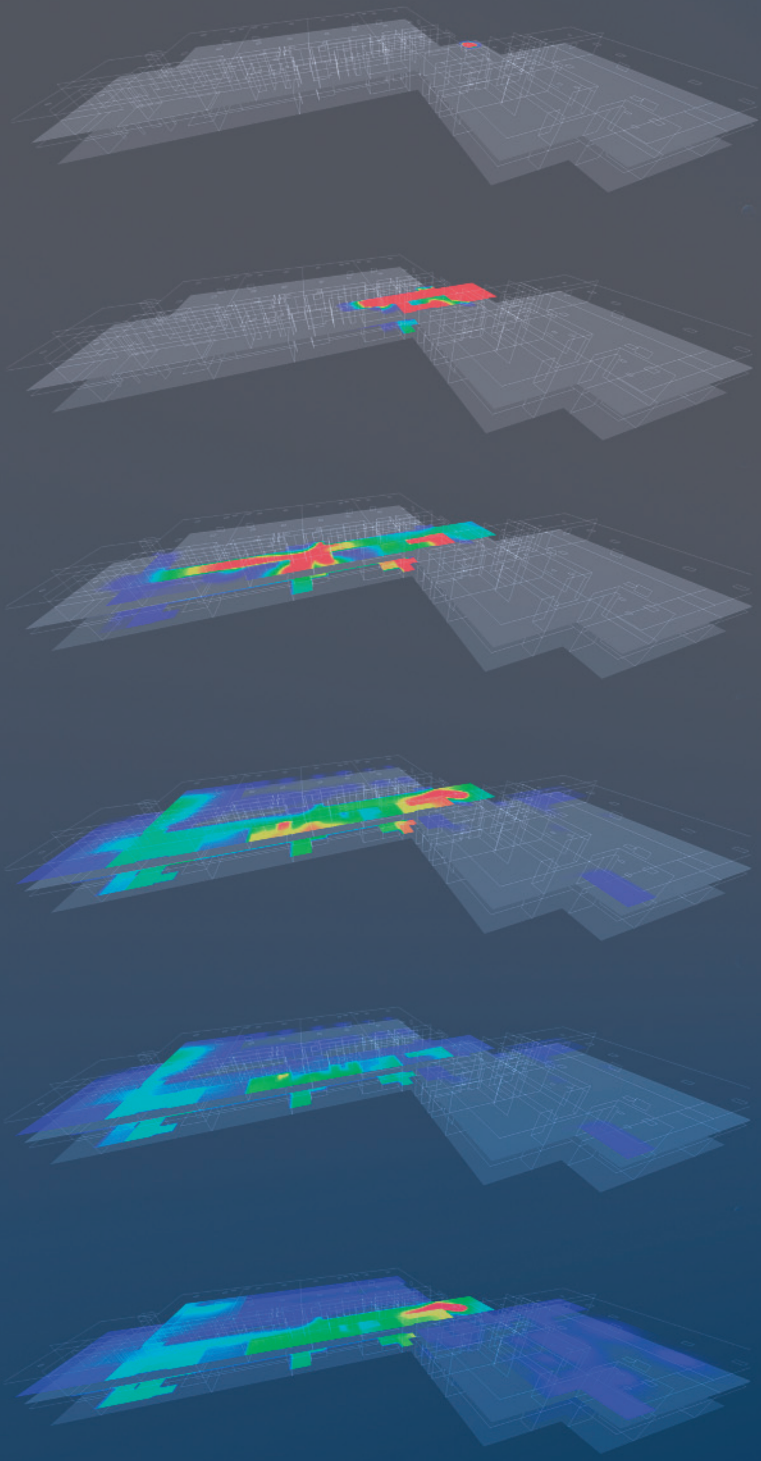
ISPA 2006 ~ INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING AND APPLICATION 2006 ~ DECEMBER 4 -6, 2006 @ Sorrento, Italy ~ http://www.ispa-conference.org/2006/

HPCA 13 ~ INTERNATIONAL SYMPOSIUM ON HIGH-PERFORMANCE COMPUTER ARCHITECTURE ~ February 10-14, 2007 @ Phoenix, AZ ~ http://www.ece.arizona.edu/~hpca/

22nd ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING ~ SAC 2007~ March 11 - 15, 2007 @ Seoul, Korea http://www.acm.org/conferences/sac/sac2007/

SD WEST 2007 ~ SOFTWARE DEVELOPMENT WEST CONFERENCE & EXPO ~ 13-17 March 2007 @ Santa Clara, CA ~ http://www.sdexpo.com/

IPDPS 2007 ~ 21ST IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM ~ 26-30 March 2007 @ Long Beach, CA ~ http://www.ipdps.org/



Naval Oceanographic Office Major Shared Resouce Center
1002 Balch Boulevard, Stennis Space Center, Mississippi 39522