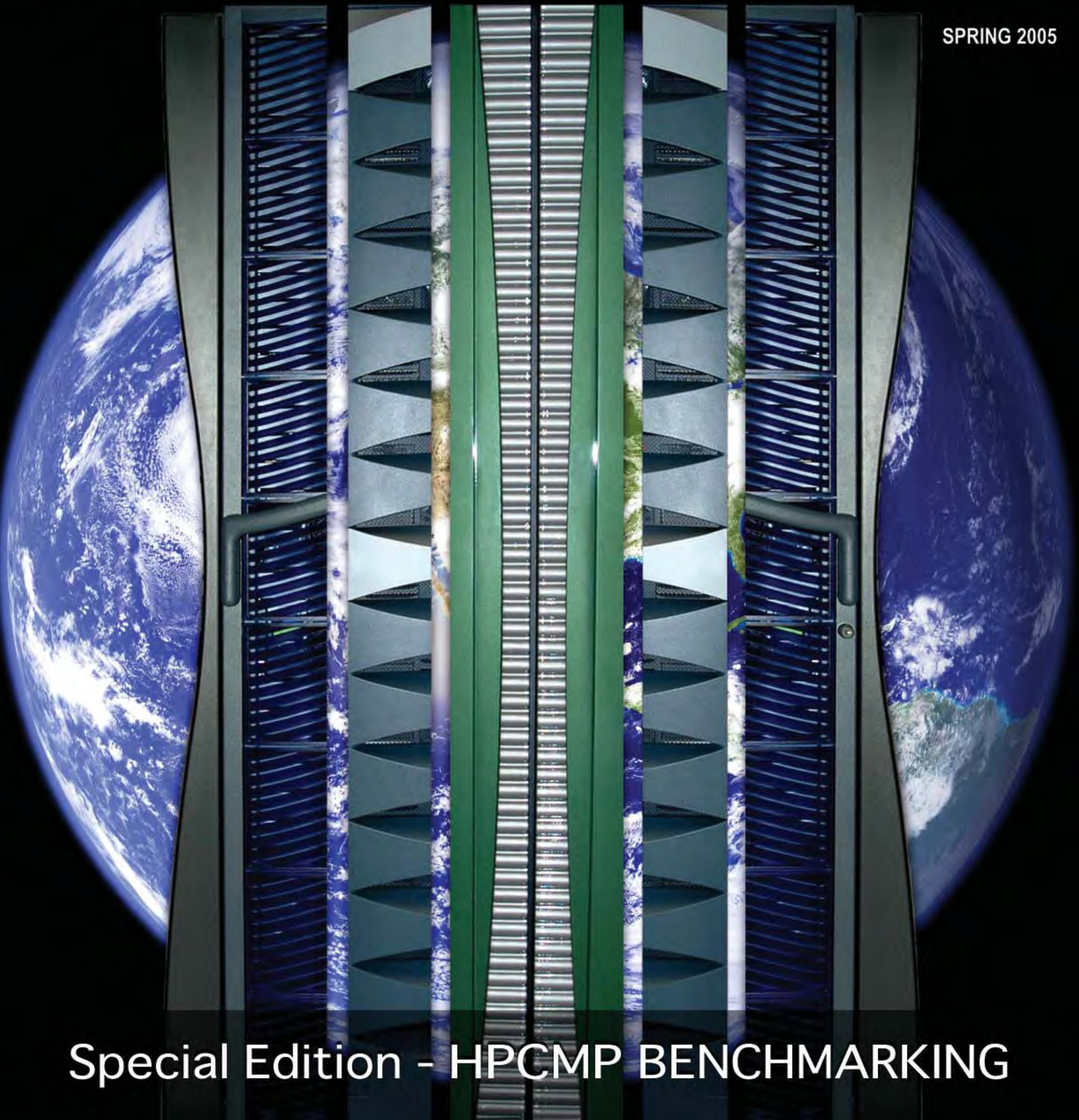
# ERDC MSRC

Major Shared Resource Center

# RESOURCE

SPRING 2005



Special Edition - HPCMP BENCHMARKING

**ERDC MSRC *Resource***
**Editorial Staff**

**Chief Editor/Technology Transfer Specialist**
Rose J. Dykes

**Interdisciplinary Coordinator**
Mary L. "Dean" Hampton

**Visual Information Specialist**
Betty Watson

## from the editor…

Many Government organizations have a substantive interest in the modernization of high performance computing. These organizations share a need for technical information.

In his introductory article, guest writer Dr. Larry Davis explains the importance of benchmarking as a source of information for decision makers. As Deputy Director for the Department of Defense High Performance Computing Modernization Program (HPCMP), Dr. Davis has engaged various Government organizations to develop and share benchmarking results.

This issue of the *Resource* represents a cooperative effort among the HPCMP, Oak Ridge National Laboratory, the Air Force Research Laboratory, Ames Laboratory, the National Aeronautics and Space Administration Langley Research Center, the National Center for Atmospheric Research, as well as the Army Research Laboratory and a commercial partner, Instrumental, Inc. The results of this cooperation include a better understanding of performance within the normal operating environment of today's high performance computing centers, where hundreds of jobs may contend for resources.

As John West, Director of the U.S. Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC), points out in his "from the director" article, the benefits of cooperative benchmarking extend well beyond the acquisition process.

ERDC MSRC HPC Service Center
Web site: www.erdc.hpc.mil
E-mail: msrchelp@erdc.hpc.mil
Telephone: 1-800-500-4722

The ERDC MSRC welcomes comments and suggestions regarding the *Resource* and invites article submissions. Please send submissions to the above e-mail address.

Design and layout provided by the Visual Production Center, Information Technology Laboratory, U.S. Army Engineer Research and Development Center.

## from the director...

Welcome to the latest issue of the *Resource*. We always strive to provide our readers and users with a broad view of what's going on in the Center in this publication. From research to new equipment deployment and policy updates, we try to fashion each issue of the *Resource* into a vehicle for strengthening your relationship to us and to improving your access to the finest people, hardware, and software available to supercomputing users anywhere.

In this issue we depart a little from our established formula to shed some light into an often neglected - but critical - corner of the ERDC MSRC's research efforts by taking an in-depth look at the benchmarking process used to acquire new hardware for our users. Acquiring systems to serve the very diverse needs of the High Performance Computing Modernization Program (HPCMP) user community is a challenging task. ERDC has played a leading role in the Program's benchmarking efforts for the yearly Technology Insertion since the first joint acquisition in 2001, and that leadership role continues today. Through assessing the performance of potential technical solutions via benchmarks - both real and synthetic applications - we are able to construct technical solutions to meet your computational needs and to provide a quantitative framework for measuring the effectiveness of that solution.

But ERDC's involvement in benchmarking has benefits for users that extend beyond the acquisition of the best machine for the job. Each year the ERDC Computational Science and Engineering group spends a tremendous amount of time analyzing, porting, building, and running the HPCMP user community's most important computational applications. This experience has given us an understanding of your applications, the way in which you do your work, and the science you are trying to accomplish that is one-of-a-kind in its depth and breadth. This is experience that we rely on every day as we improve our services to you in administration and configuration of the systems, in data analysis and visualization, in computational science, and in customer service.

I hope this special issue of the *Resource* will give you added insight into the tremendous value that the benchmarking effort has to the Program as a whole, and to you as an individual user.

As always, I want to hear from you! If you'd like to let me know how we're doing, share a success story, or make a suggestion for ways to improve our service, drop me a line at **john.e.west@erdc.usace.army.mil**.

John

***John E. West, Director***
*Major Shared Resource Center*
*U.S. Army Engineer Research and Development Center*
*Vicksburg, MS*

# Contents

# DoD HPCMP Requires Accurate Benchmarks to Choose HPC Systems

*By Dr. Larry P. Davis*

***Dr. Larry P. Davis***
Deputy Director
HPCMP

The Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) acquires over $50 million worth of new high performance computers each year. With a wide variety of system architectures from many vendors, how does it know which ones to choose? Along with very important usability criteria, performance of these systems is critically important. And how does the Program measure performance? A careful measurement of performance of offered systems on a set of benchmarks representative of the workload of the Program user community fulfills this task.

As discussed in detail in several of the following articles, the HPCMP uses a mix of synthetic benchmarks and application programs to represent its workload. A recent study by several Benchmark Team members of requirements stated by users and current usage on the Program's systems produced a set of eight application benchmarks for the Fiscal Year (FY) 2005 acquisition activity and nine application benchmarks for the FY 2006 activity. As a set, these application benchmarks directly represent approximately 40 percent of the Program's non-real-time workload (as run on the shared, allocated systems) and indirectly represent about 70 percent of the workload. A key set of synthetic benchmarks are used to supplement the application benchmarks and address any gaps in the representation of the complete Program workload. The entire set of Program benchmarks is run on all existing HPCMP systems each year and then provided to prospective hardware vendors to run on their offered systems, with timings to be used in the evaluation process each year.

Once the vendor-provided timings on the benchmark set are received, performance scoring of these timings for each benchmark on each system is done by a quantitative process that plots a power-law curve of the performance (reciprocal of the run time) vs. the number of processors and interpolates that curve at key system sizes that represent how the system would likely be used in production. These times at key system sizes are then compared with the run time measured on the standard DoD system for that year; for FY 2005, that standard system was Marcellus, the IBM Power4 at the Naval Oceanographic Office Major Shared Resource Center (NAVO MSRC), and for FY 2006, the standard system is the newer IBM Power4+ at NAVO. Thus, each system score on each benchmark is a relative performance value compared with the DoD standard system. In addition, a size-independent system score on each benchmark can be deduced from the curve by calculating the number of processors on the scored system that it would take to match the performance (run time) for the standard DoD system on that benchmark. This value (standard number of processors) becomes a basic performance score for each system.

Performance, however, is not the only consideration. The Program seeks to acquire not only the most powerful systems but also the most total performance across its entire acquisition on the set of application test cases that represent its workload. To accomplish this, the Program has developed an optimization technique that maximizes total overall performance across the benchmark suite for a fixed acquisition budget. This process effectively determines price/performance for a great number of possible solution sets, or sets of acquired systems. The optimizer and its use are described in a following article.

Finally, the Program is investigating benchmarking and scoring methods that will relieve some of the burden of running large benchmark sets on the vendor while providing a more consistent basis of scoring to make acquisition decisions. The basis of these investigations is a performance modeling and prediction methodology pioneered by Allan Snavely and Laura Carrington from the University of California at San Diego. This methodology relies on benchmark results from a few simple synthetic probes (already part of the HPCMP synthetic benchmark suite) to measure system performance in key basic system processes, such as memory operations. The performance on these probes is then convolved with a detailed trace of the operations that each application test case performs to produce a predicted performance of that application test case on the system for which the synthetic performance has been measured. The plan is to begin using results from these performance predictions in a limited way for the FY 2006 acquisitions.

The tremendous progress the benchmarking and performance modeling activities have made over the last few years has been directly attributable to the hard and productive work that all members of the Benchmark Team have performed. Key groups that have made major contributions include the Computational Science and Engineering (CS&E) group from the U.S. Army Engineer Research and Development Center (ERDC) MSRC (applications), Instrumental, Inc. (synthetics), the group from the University of California at San Diego (performance modeling), and the Army Research Laboratory (ARL) MSRC (scoring and optimizer development). In addition, support from the NAVO MSRC, the Aeronautical Systems Center (ASC) MSRC, the Arctic Region Supercomputing Center (ARSC), the Maui High Performance Computing Center (MHPCC), and the Army High Performance Computing Research Center (AHPCRC) has been very helpful. All look forward to significant progress over the next few years as performance modeling and prediction become a more integral part in the entire benchmarking process.

## Role of Application Benchmarks in the DoD HPC Acquisition Process

*By Dr. William A. Ward, Jr.*

**Dr. William A. Ward, Jr.**
CS&E Group Lead
ERDC MSRC

The DoD HPCMP uses a comprehensive performance evaluation process in its Technology Insertion (TI) acquisition process for high performance computer systems. When purchasing desktop, or even modest-size server class systems, the Government goes through no such elaborate process because the systems are typically commodities, the characteristics of the systems are well-known, and the risk to the Government of overpaying for an underperforming system is low. However, when acquiring a state-of-the-art, limited-production high performance computing (HPC) system costing millions of dollars, the risk is higher and so the DoD must be more thorough and cautious. Benchmarking is an integral part of this approach.

Previously presented in the Fall 2000 edition of *The Resource*, a spectrum of possible types of benchmark programs with respect to representativeness, maintainability, and scalability is illustrated in Figure 1. Synthetic benchmarks, compact artificial programs whose sole purpose is to measure some aspect of system performance (e.g., floating-point rate, memory bandwidth, input/output (I/O) capability), are less representative, but more maintainable and scalable. The Streams benchmark is an example of this type. Next, toy programs, such as Prime Sieve; package kernels, such as LINPACK; and application kernels, which are compute-intensive fragments extracted from actual programs, span the middle of the spectrum. Finally, complete application programs are the most representative, but because of their size and complexity, are less maintainable and (usually) less scalable. The DoD HPCMP uses benchmark components from the first and last of these categories. Each of the synthetic benchmarks, discussed in a subsequent article by Cal Kirchhof and Henry Newman, effectively delineates the edge of the "performance envelope" by stressing a single-system component, e.g., central processing unit (CPU), memory, I/O subsystem. These quickly identify any

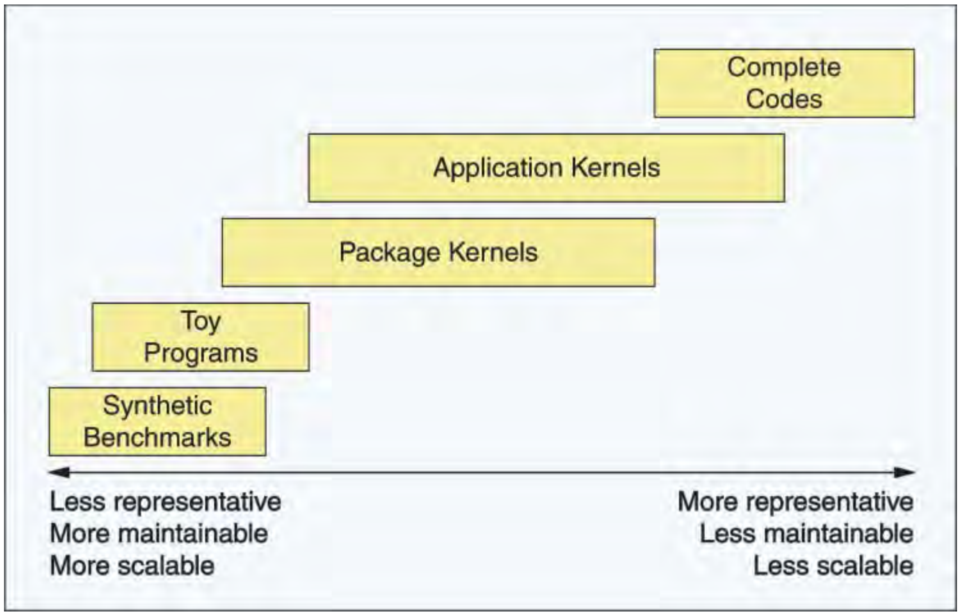


*Figure 1. Spectrum of possible types of benchmark programs*

obvious system performance shortcomings and gather data for performance modeling. The purpose of the application programs in the DoD HPC benchmark, on the other hand, is to stress the system with codes that represent DoD's actual workload in the various computational technology areas (CTAs) and to measure the performance a user would experience when running an actual application.

The application program portion of a typical DoD TI benchmark is called the application benchmark test package. The test package includes program source code (or directions for obtaining it), makefiles, sample batch submission scripts, and so forth. Each program (eight programs were used in Technology Insertion 2005 (TI-05)), typically has two sets of input data, standard and large; these are called test cases. The test package includes correct job output for at least one processor count for each test case plus a script for automatically verifying correctness at arbitrary processor counts. Members of the CS&E group at the ERDC MSRC validate the test package by running each of the test cases on the current DoD baseline system. For TI-05 this system was Marcellus, a 1,328-processor, 1.3-GHz IBM p690 (IBM Power4) at the NAVO MSRC; for TI-06, the system will be Kraken, the 2,832-processor, 1.7-GHz IBM p655 (IBM Power4+), also at NAVO. These runs are also used to obtain baseline times that provide performance expectations for prospective vendors. Benchmark program input data are adapted so that the baseline times for standard test cases, running on 64 processors, and for large test cases running on 384 processors, are roughly 1 hour (on the baseline system). The codes and test cases used for TI-05 and their performance on various Government-owned systems are described in several of the following articles.

After a prototype of the application benchmark test package is constructed, it is sent to the staff at the ASC MSRC for a dry run as a final quality control check. Based on feedback from ASC, the CS&E group makes any necessary modifications and resubmits the package to ASC, where it is burned onto DVDs for distribution to prospective vendors. Vendors are given several months to prepare their benchmark response. To participate in the TI process, a system vendor must, at a minimum, submit job times for at least one test case; realistically, however, most vendors submit complete or nearly complete sets of times, sometimes for multiple systems, to improve their competitiveness. For each test case, vendors are required to submit job wall times for no less than three processor counts. At least one of these times is required to be from an actual system identical to, or highly similar to, the offered system. Because the vendor's inventory may not include a full-scale system of the size necessary to run some of the tests, two of the three times may be estimates. However, the Government reserves the sole discretion for determining the acceptability of such estimated times and times from substitute systems, and the vendor must meet any estimates should the system be selected.

For TI-05, while the vendors were running the tests on their systems, the CS&E group along with staff members from the other three MSRCs and several other DoD HPC sites, ran the tests on most DoD HPC systems. For TI-06, the CS&E group has the sole responsibility for running all of the application test cases on all unclassified (and a few classified) systems. The purpose of gathering these times is to accurately quantify DoD's HPC capability and capacity in each CTA. Then, one of the criteria for selecting a new system is how well it complements DoD's already installed computational resources. This part of the process is described in following articles on scoring by Dr. Alvaro Fernández and this author, and in another article on system selection by Dr. Roy L. Campbell, Jr.

As presented in a General Services Administration (GSA) handbook (*Use and Specifications of Remote Terminal Emulation in ADP System Acquisitions*), the success of benchmarking in system selection is judged by how well it meets several goals. The first two goals are to minimize time and cost of acquisition and to maximize competitiveness; the TI process does this by providing a well-documented benchmark framework that is consistent from year to year. The next two goals, to maximize quality of system sizing and to maximize benchmark representativeness, are accomplished by selecting test cases that impose a significant computational load on candidate systems and by using application programs to represent the DoD CTAs that use the most CPU hours on DoD HPC systems. The three final goals, to minimize benchmark discrepancies, to maximize benchmark uniformity, and to maximize benchmark repeatability, are ensured by a validation process that runs the test package on many systems, by providing comprehensive guidance to vendors on running the test suite, and finally by carefully checking vendor benchmark responses and requesting new runs if necessary to remedy deficiencies. TI-06 will be the sixth year of implementing the application test package with these goals in mind, and the result has been increased vendor participation and improved quality of vendor submissions.

# Benchmarking AERO, an Aeroelastic CFD/CSM Application

*By Drs. Raymond E. Gordnier and Thomas C. Oppe*

AERO is the informal name given to an aeroelastic computational fluid dynamics/computational structural mechanics (CFD/CSM) code developed in the Computational Sciences Branch in the Air Vehicles Directorate of the Air Force Research Laboratory to investigate basic nonlinear fluid/structure interactions. The code evolved from the FDL3DI flow solver and is used primarily in aerospace research applications.

The system of partial differential equations modeled by AERO is mixed elliptic/hyperbolic, and the overall solution method is implicit. The physical model used in the code is integrated multiphysics over a physical domain. The fluid dynamics part of the code is a finite difference formulation that solves the full Navier-Stokes equations. The structures part of the code is a finite element formulation for solving the von Karman plate equations. Implicit coupling with the finite element structural solver is accomplished via a subiteration procedure. AERO uses an Eulerian coordinate framework with the mesh being dynamic, structured, and anisotropic. To perform the fluid/structure interaction computations, the structure deflects in response to the imposed loads. The fluid grid is then moved to accommodate the new position of the structure. This remeshing is accomplished by simple transfinite interpolation techniques.

The AERO code is written entirely in FORTRAN 90 and, thus, should be portable to any platform that has a FORTRAN 90 compliant compiler. AERO also contains calls to several BLAS and LAPACK routines, in particular a banded matrix solution routine from LAPACK. Fortran equivalents of all the needed BLAS and LAPACK routines were supplied in the TI-05 AERO distribution in the event that vendor-optimized libraries were not available. AERO contains approximately 15,000 lines of code.

AERO is unique among the components of the TI-05 application benchmark test suite in that it has no explicit parallelism. It was originally developed as a vectorized code appropriate for running serially on the Cray C90 and Cray SV1 platforms. Thus, the source code contains no message passing interface (MPI) calls or OpenMP directives. Implicit parallelism in a shared-memory environment must be achieved through autoparallelizing compilers or by linking to threaded BLAS and LAPACK libraries. Alternatively, explicit parallelism can be achieved by the manual insertion of OpenMP directives. In any case, the parallel scalability is limited to the numbers of CPUs on a shared-memory node. AERO cannot run in distributed-memory mode.

*Dr. Raymond E. Gordnier*
Senior Research
Aerospace Engineer
Air Force Research
Laboratory

*Dr. Thomas C. Oppe*
Computational Scientist
CS&E Group
ERDC MSRC

Because AERO had its roots as a vector code, the execution of AERO on a microprocessor-based platform is typically limited by the bandwidth to memory. Many loops involve non-unit stride memory accesses that incurred no penalty on the traditional Cray PVP (parallel vector processor) platforms, but which were detrimental to efficient execution on HPC platforms employing microprocessors with caches. Several HPC vendors participating in TI-05 attempted to improve the speed of execution by a variety of strategies, such as (1) interchanging the order of loop indices in a series of nested loops to achieve better cache utilization, (2) inlining calls to frequently called subroutines, (3) padding or rearranging arrays in COMMON blocks to better align the arrays with cache line boundaries, (4) manually inserting OpenMP compiler directives to achieve loop parallelization, (5) linking in optimized intrinsic mathematics (e.g., SIN, EXP) or BLAS and LAPACK libraries, and (6) compiling the source code with an autoparallelizing compiler option or preprocessor.

Only one input data set was supplied for AERO in TI-05. This particular problem computed the supersonic flutter of a flexible panel. (Figures 1 and 2 show visualizations of the flexible panel.) A large 240-MByte binary restart input file was read at the start of execution, and another large binary file of the same size was written upon program termination. Program execution required approximately 1.1 GBytes of memory to run.

In an experiment, the unoptimized AERO code was compiled with the autoparallelizing compiler option and run on an SGI Origin 3900 at the ERDC MSRC using 700-MHz MIPS chips and an IBM p655 at the NAVO MSRC using 1.7-GHz Power4+ chips. In both cases,

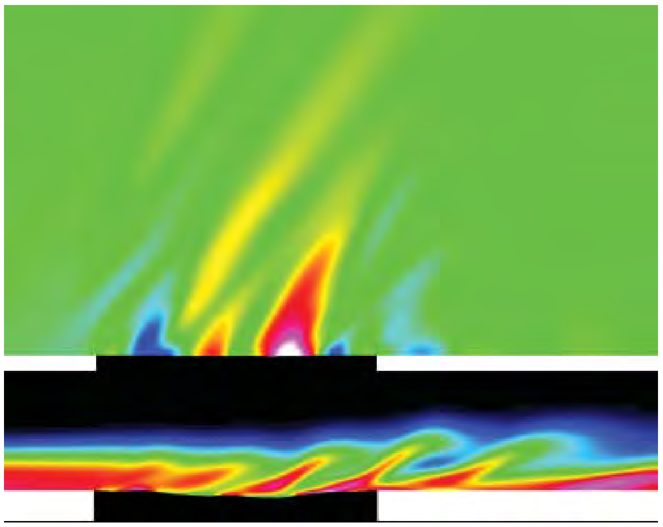


*Figure 1. Pressure and vorticity magnitude contours showing the interaction of the surface boundary layer with a three-dimensional (3-D) flexible panel*

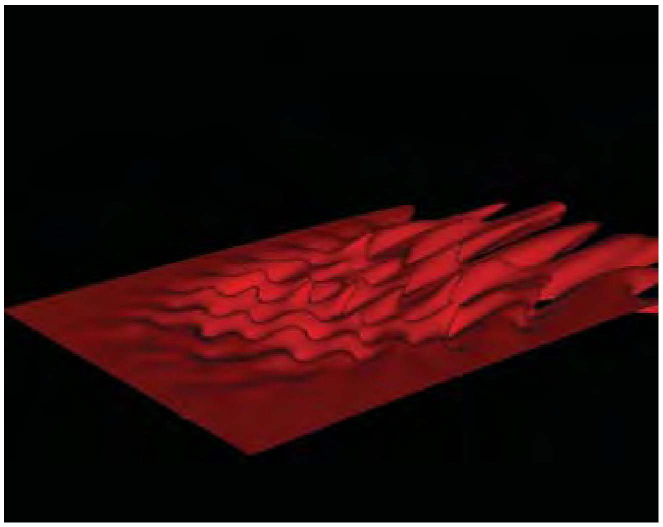


*Figure 2. Isosurface of vorticity magnitude showing 3-D, finger-like vortical structures over a 3-D flexible panel*

the executable was linked with the threaded versions of the BLAS and LAPACK libraries (i.e., "-lscs_mp" for SGI and "-lesslsmp" for IBM). The autoparallelizing compiler options for SGI and IBM were "-apo -mp" and "-qsmp", respectively. The IBM compilation required a FORTRAN version of the LAPACK routine since this routine was not in the IBM Engineering and Scientific Subroutine Library (ESSL). The resulting times in seconds as a function of the number of threads are given in Figure 3. A parallel speedup of only 2.5 was attained using the autoparallelizing option for both machines.

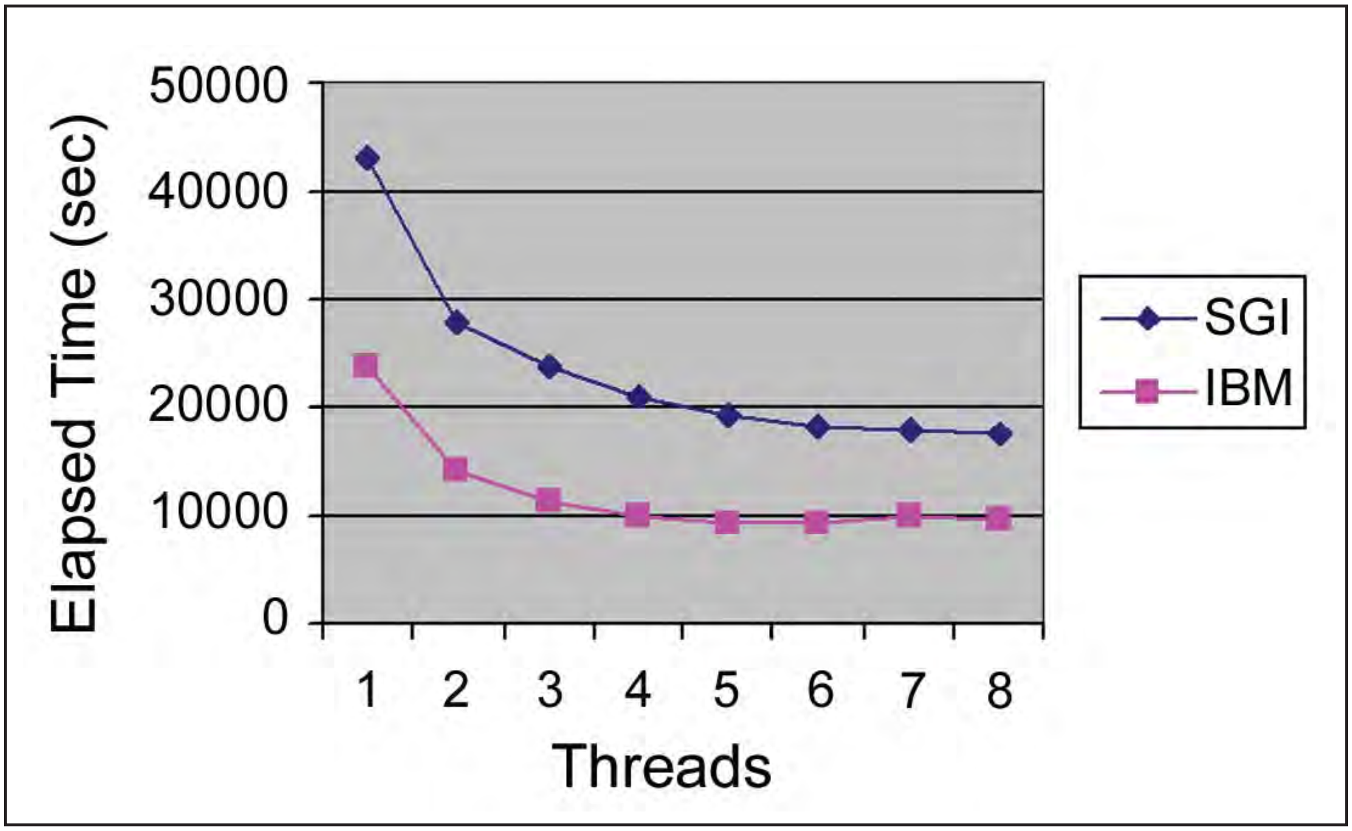


*Figure 3. AERO performance obtained using autoparallelization*

# Benchmarking AVUS, an Aerospace CFD Solver for Unstructured Grids

*Drs. Victor S. Burnley, Matthew J. Grismer, and Thomas C. Oppe*

The Air Vehicles Unstructured Solver (AVUS), formerly called Cobalt$_{60}$, is a parallel, implicit CFD code that solves the compressible Euler and Navier-Stokes equations subject to the ideal gas equation of state. Two-dimensional (2-D), 3-D, and axisymmetric spaces can be modeled. Unstructured grids with arbitrary cell types are permitted. The developers' goal was to make AVUS as general, flexible, robust, accurate, and easy to use as possible. The fundamental algorithm of AVUS is conceptually based on the exact Riemann solver of Godunov, a finite-volume, cell-centered method that is first-order accurate in both space and time. However, in practice, Godunov's exact Riemann solver is very expensive, so the more efficient method of Gottlieb and Groth is employed. Second-order accuracy in space is patterned after van Leer's monotone upwind scheme for scalar conservation laws (MUSCL), where the flow state is assumed to vary linearly within each cell. The linear variations (gradients) are constructed by a central-difference, least-squares method that, in turn, is solved by QR factorization. In cells requiring limiting, the gradients are corrected to give a one-sided least-squares scheme. First- and second-order temporal accuracy is achieved using the unconditionally stable point-implicit scheme as implemented by Tomaro, Strang, and Sankar.

Second-order accurate viscous terms, loosely patterned after the work of MacCormack, are added to the above inviscid algorithm to yield a Navier-Stokes solver. The temporal accuracy of the viscous terms is equivalent to that of the inviscid terms. Several turbulence models, including the Spalart-Allmaras model, Wilcox's k-w model, and Mentor's SST model, are available to model the fine-scale effects of turbulence. Detached eddy simulations (DES) can also be performed with AVUS. Lastly, much effort was devoted to boundary conditions to achieve high accuracy with robustness and flexibility.

The model grid may be composed of cells of arbitrary type (tetrahedra, quadrilaterals, pyramids, triangles, etc.); different cell types are permitted within the same grid. The set of boundaries forming each cell, called faces, may also be arbitrary (triangles, pentagons, lines, etc.), though each cell boundary face should be convex. The grid is decomposed into subdomains called zones, permitting parallel processing in which computations in each zone are done by a separate processor. The domain decomposition is accomplished using ParMETIS, the MPI-based parallel grid-partitioning library from the University of Minnesota that performs both static and dynamic graph partitioning. ParMETIS is highly portable and easy to install on a variety of platforms. AVUS uses ParMETIS to partition a multidimensional grid among a set of processors, producing roughly equally sized zones so that the computational load among processors is well balanced. In addition, each zone has a minimized surface area, thus reducing the amount of communication required between zones. Consequently, AVUS's excellent scalability may be attributed to two characteristics: (1) good load balancing with minimal communications overhead, attributable to ParMETIS, and (2) high computational intensity requiring little communication.

Development of AVUS began in 1990 at the Air Force Research Laboratory (AFRL), Aeronautical Sciences Division, Computational Sciences Branch. The original code, Cobalt$_{60}$, was developed under the CFD CTA of the HPCMP's Software Applications Support (formerly known as the Common High Performance Computing Software Support Initiative (CHSSI)) and can be distributed only to U.S. citizens.

AVUS is written entirely in standard FORTRAN 90 with the exception of one utility routine and the ParMETIS library, which are written in C. MPI is used to

*Dr. Victor S. Burnley*
Research Aerospace Engineer
Air Force Research Laboratory

*Dr. Matthew J. Grismer*
Senior Research Aerospace Engineer
Air Force Research Laboratory

*Dr. Thomas C. Oppe*
Computational Scientist
CS&E Group
ERDC MSRC

achieve parallelism. Hence, AVUS is quite portable and has been successfully run on IBM SPs, SGI Origins, Compaq SC40/45s, the Cray X1, Linux clusters, and Mac OS X clusters.

AVUS is commonly used to model fluid flow and turbulence around complex objects. For example, Figure 1 depicts the surface-pressure distribution on an F/A-18C, as computed by AVUS. The number of cells in a model typically ranges from several hundred thousand to tens of millions, while the number of processors used ranges from tens to thousands, depending on the size of the grid.

For TI-05, two test cases were supplied. The first test case, denoted the standard input set, was a wind tunnel model of a wing with a flap and endplates. It used 7,287,723 cells and ran for 100 time-steps. The second test case, denoted the large input set, modeled a generic pilotless aircraft, or UAV (uninhabited air vehicle), using 24,040,002 cells and running for 150 time-steps. Both of these cases were 3-D and modeled turbulent viscous flow over complicated geometries. Each test case was defined by two input files: a small ASCII (American Standard Code for Information Interchange) file defining the boundary conditions and a large IEEE (Institute of Electrical and Electronics Engineers) binary file defining the grid. Figures 2 and 3 contain timing data from running the standard and large data sets, respectively, on various MSRC platforms. Superlinear speedups occur for several processor counts because of improved cache performance as the sizes of the subdomains decrease.

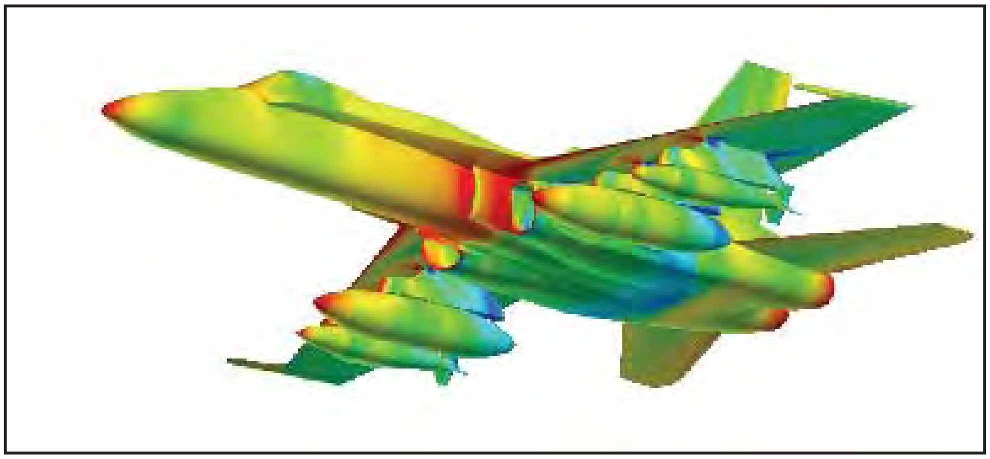


*Figure 1. Surface-pressure distribution on an F/A-18C*

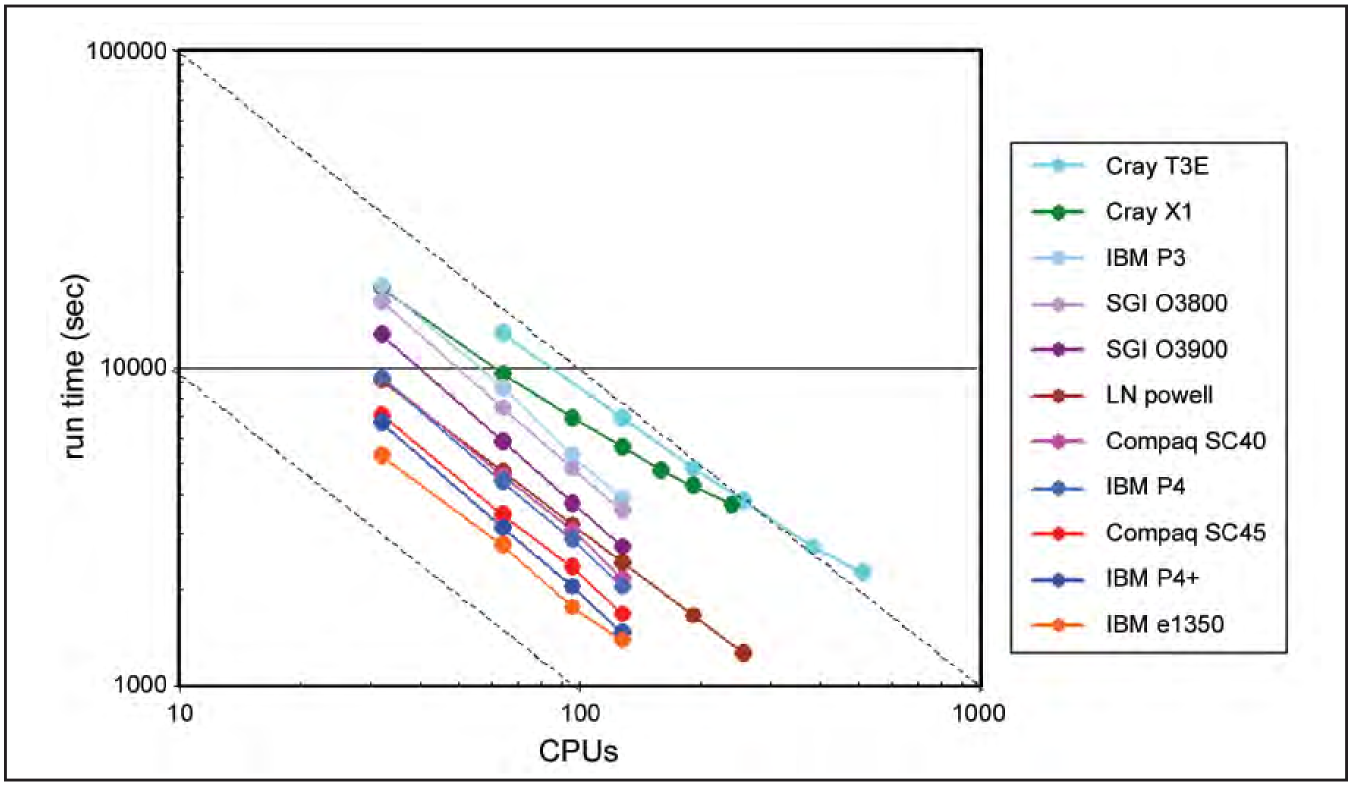


*Figure 2. AVUS performance for the standard test case on selected DoD HPC platforms*

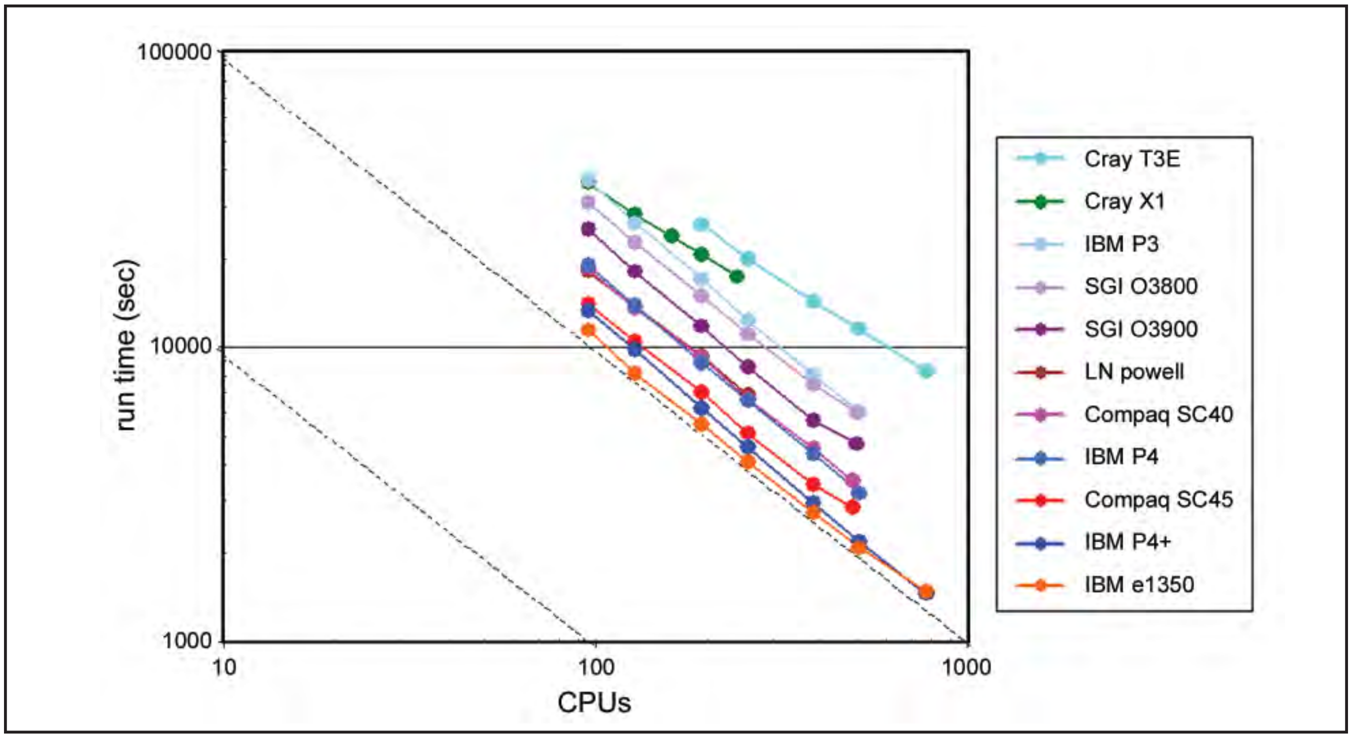


*Figure 3. AVUS performance for the large test case on selected DoD HPC platforms*

## Benchmarking GAMESS, a Quantum Chemistry Code

*By Drs. Paul M. Bennett and Mark S. Gordon*

The General Atomic and Molecular Electronic Structure System (GAMESS) is a code for computational quantum chemistry, available at http://www.msg.ameslab.gov/GAMESS/GAMESS.html. It is an *ab initio* code in that material properties may be calculated from first principles, although semi-empirical methods are available as well. GAMESS evolved from an early version of HONDO, which was developed using funding from the National Science Foundation (NSF), the Department of Energy (DOE), and IBM. Development continues at Iowa State University with sponsorship from the Air Force Office of Scientific Research and, more recently, the DOE. Many individuals and several research organizations have contributed to the development of GAMESS. The contributions can be found in the user's guide, which accompanies the source code distribution. The developers would especially like to acknowledge two Software Applications Support (formerly CHSSI) projects funded by the DoD HPCMP that were used to develop significant portions of GAMESS.

GAMESS has an extensive set of capabilities. Following calculation of the molecular energy, GAMESS users may direct the code to calculate analytic and numerical gradients, analytic and numerical Hessians, and other properties. GAMESS also provides a variety of wave functions to use in the computations, including restricted Hartree-Fock, unrestricted Hartree-Fock, restricted open-shell Hartree-Fock, multiconfigurational self-consistent field, generalized valence bond, configuration interaction, second-order perturbation theory, and coupled cluster. A complete description of the program's capabilities and the input language interface may be found in the user's guide.

***Dr. Paul M. Bennett***
Computational Scientist
CS&E Group
ERDC MSRC

***Dr. Mark S. Gordon***
Head of Gordon Research Group
Ames Laboratory
Iowa State University

GAMESS does not use numerical grids for its computations. Instead, integrals modeling the energies of the electron shells are computed using analytic expressions and recursion formulae. More details can be found in the user's guide.

Figure 1 depicts a POSS (polyhedral oligomeric silsesquioxane) adsorbed on a cluster that represents the Si(100) surface. POSS are very important species, as they are resistant to extreme environments and can function as coatings as well as viscosity modifiers. Considerable controversy has arisen regarding the two possible mechanisms (shown in the figure) for adsorption of this POSS onto Si(100). Using the ERDC MSRC Cray T3E system, Tejerina and Gordon were able to use very high levels of theory in GAMESS to provide a consistent and accurate comparison of the two mechanisms.

The second data set, or large test case, was intended to require about 1 hour on 384 CPUs and between 0.5 and 1.0 GBytes of memory on 256 CPUs of the reference HPC system. Perfect scaling of parallel computation is indicated by the dotted black lines. The Compaq SC45, the IBM e1350, and the IBM Power4+ all outperformed the reference system for this test case. All of the other systems took longer. Only the SGI Origin 3900 demonstrated superscaling. GAMESS scaled well on all of the systems, with the IBM Power 4+ again giving the worst scalability.

Figure 2 presents benchmark data obtained during the TI-05 hardware acquisition process. The benchmark data consist of two series of runs performed on the reference HPC system, the IBM Power4 p690 at the NAVO MSRC, and several other DoD HPC systems. The first series of runs, called the standard test case, used a data set intended to require about 1 hour on 64 CPUs on the reference system. Perfect scaling of parallel computation is indicated by the dotted black lines. Most systems outperformed the reference system on this benchmark, with the best times produced by the Cray X1, the IBM e1350, and the IBM Power4+. Only the Cray X1 demonstrated superscaling. All of the other systems scaled less efficiently, with the IBM Power 4+ losing efficiency most rapidly.

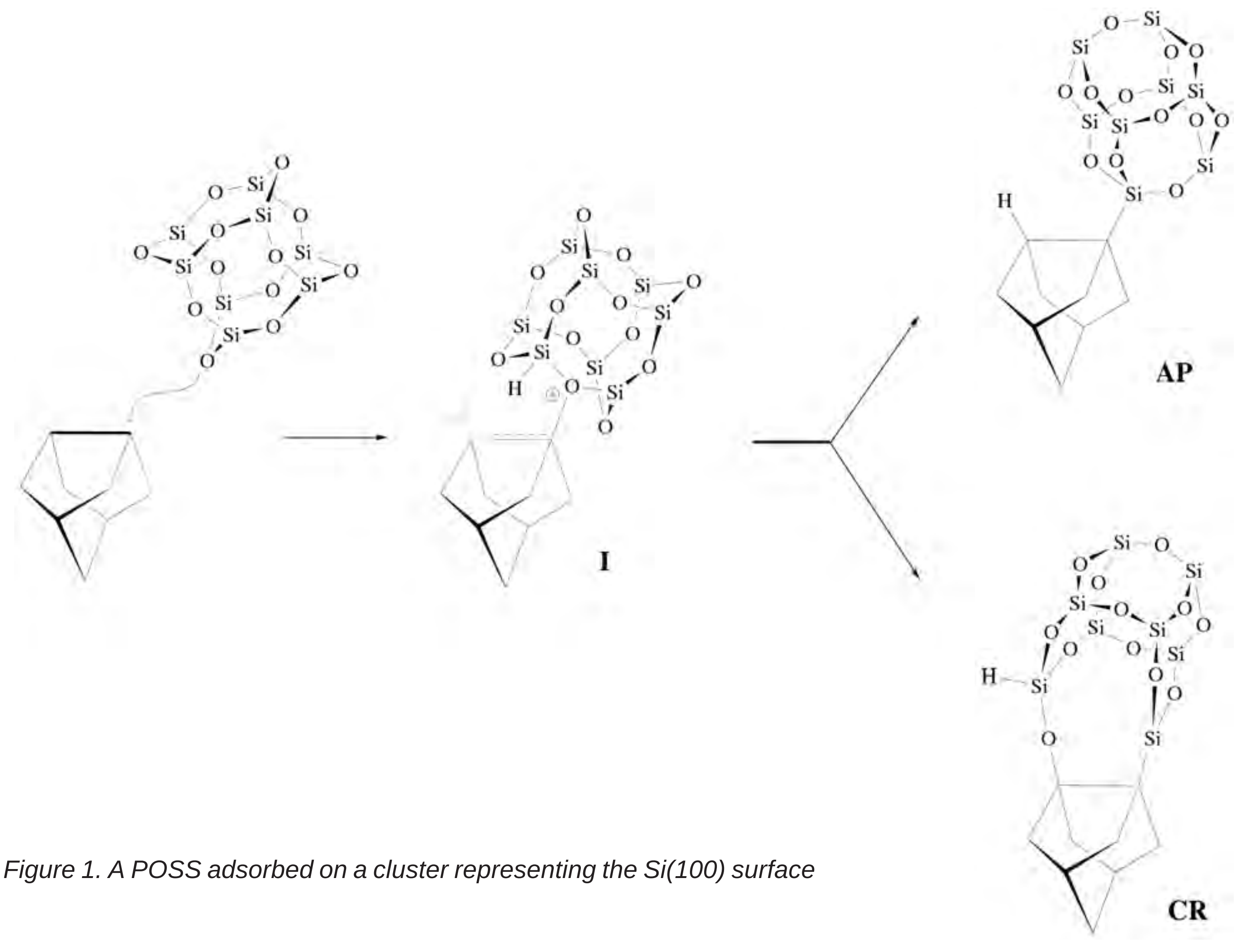


*Figure 1. A POSS adsorbed on a cluster representing the Si(100) surface*

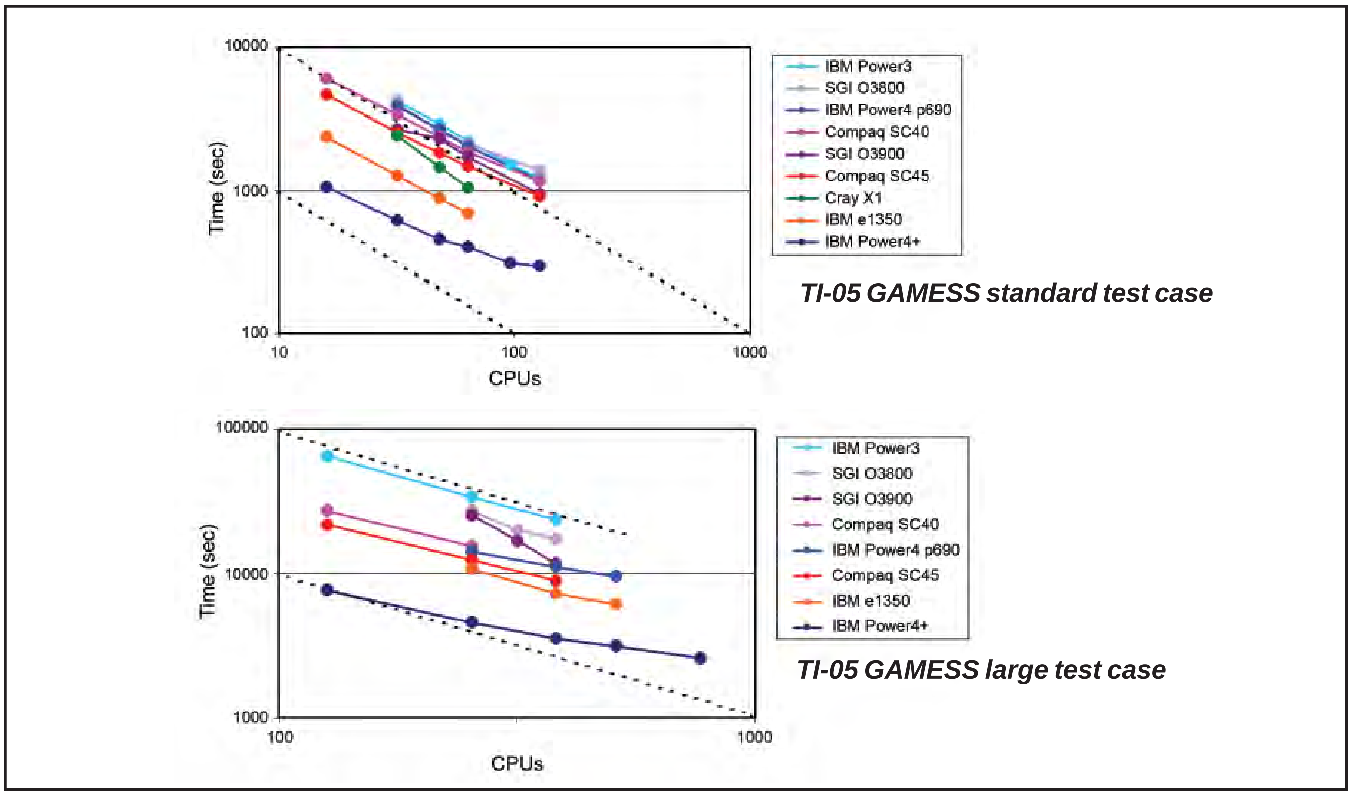


*Figure 2. Benchmark data obtained for TI-05 acquisition process*

## Benchmarking HYCOM, a Structural Grid Ocean Model

*By Carrie L Leach*

The HYbrid Coordinate Ocean Model (HYCOM), like its predecessors, the Naval Research Laboratory (NRL) Layered Ocean Model (NLOM) and the Miami Isopycnic-Coordinate Ocean Model (MICOM), is isopycnal (uses traditional isopycnic vertical coordinates) in the open, stratified ocean. However, HYCOM utilizes a hybrid (generalized) vertical coordinate that allows an arbitrary partitioning between density coordinates (i.e., isopycnals) and depth coordinates on a time-step by time-step basis, and so smoothly transitions to a terrain-following coordinate in shallow coastal regions and to z-level coordinates in the mixed layer and unstratified waters. HYCOM is an open-source structured grid ocean model, consisting of about 31,000 lines of FORTRAN 90 code, downloadable from http://hycom.rsmas.miami.edu.

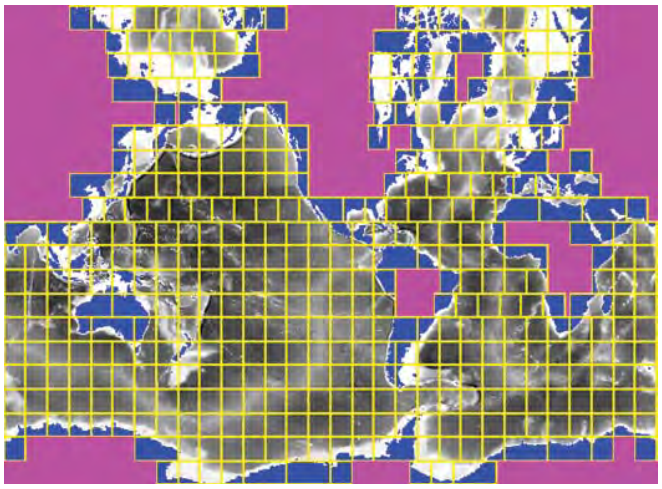


*Figure 1. Tiling for a global domain*

*Carrie L. Leach*
Computational Scientist
CS&E Group
ERDC MSRC

HYCOM's basic parallelization strategy is domain decomposition, i.e., the region is divided into smaller subdomains, or tiles, and each processor owns one tile with off-tile communication via either MPI or Cray's SHMEM library. Figure 1 shows one such tiling for a global domain, consisting of 600 (30 by 20) approximately equal-sized tiles, but 174 all-land tiles are discarded leaving 426 MPI tasks, each owning a single tile. This is the large TI-05 case, and each tile contains about 125 by 175 by 26 grid points. A halo is added around each tile to allow communications to be completely separated from the computational kernel. Rather than the conventional one- or two-element-wide halo, HYCOM's halo is six elements wide and is consumed over several operations to minimize communication. HYCOM also allows parallel-ization via loop-level OpenMP directives alone or via both domain decomposition and OpenMP.

In the TI-05 benchmarking suite, the two fully global HYCOM test cases are standard and large. These are configured in exactly the same way as normal production runs, other than shortening the general 30 model-day extent and generating the initial state from representative ocean profiles instead of using a restart file. The standard test case runs one model-day with 1/4° (equatorial) resolution, while the large test case runs half a model-day with 1/12° resolution. In each case a representative amount of I/O is included, and the reported time is wall time in seconds for the entire run.

Figures 2 and 3 show the test cases' time performance on the following DoD HPCMP machines: Stryker, a 2,304-CPU IBM Cluster e1350 at ARL; Kraken, a 2,800-CPU IBM POWER4+ at NAVO; Marcellus, a 1,344-CPU IBM POWER4 at NAVO; Habu, a 976-CPU IBM POWER3 at NAVO; Emerald, a 488-CPU Compaq SC45 at ERDC; Opal, a 488-CPU Compaq SC40 at ERDC; Silicon or Sand, identical 504-CPU SGI Origin 3900s at ERDC; and Diamond, a 60-MSP Cray X1 at ERDC. The X1 times were obtained by running Cray's optimized version of HYCOM on the standard case. The large case needs a minimum of 64 CPUs (too many to run on Diamond) and is configured for a maximum of more than 1,000 CPUs.

Performance is measured in run time, which is the time in seconds that TI-05 HYCOM runs at the given processor count. (Lower run time means better performance). When a code's run time decreases as the number of processors increases, the code is said to "scale." Perfect scaling is designated by the dashed black lines. Based on this time performance, a consistent ranking of the various platforms may be seen in Figures 2 and 3 except the IBM P3 and the SGI Origin 3900 cross paths. Both test cases scale well at the given processor counts.

This author is pleased to acknowledge Dr. Alan J. Wallcraft, NRL, Stennis Space Center, who provided the HYCOM test cases, Figure 1, and much of the HYCOM background information, and also graciously offered suggestions and revisions.

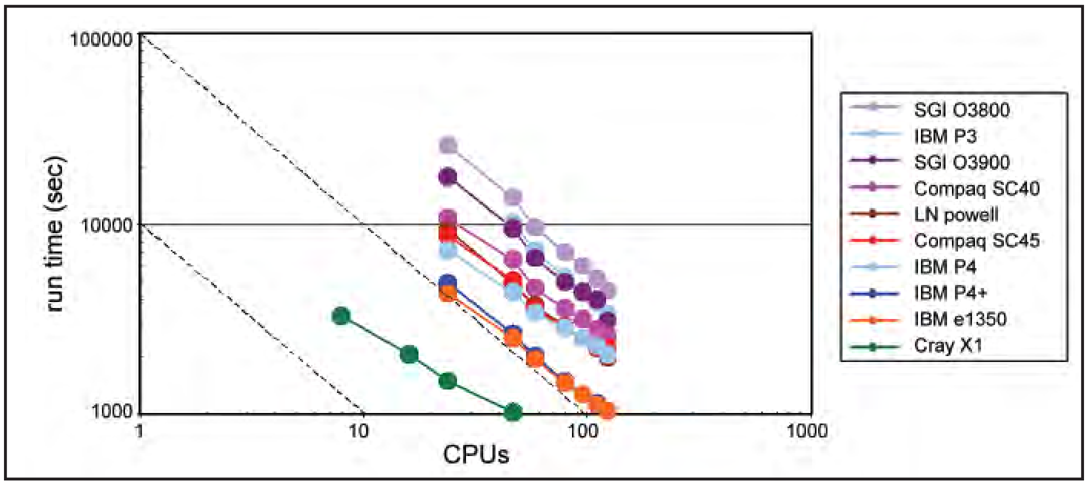


*Figure 2. TI-05 standard HYCOM test case*

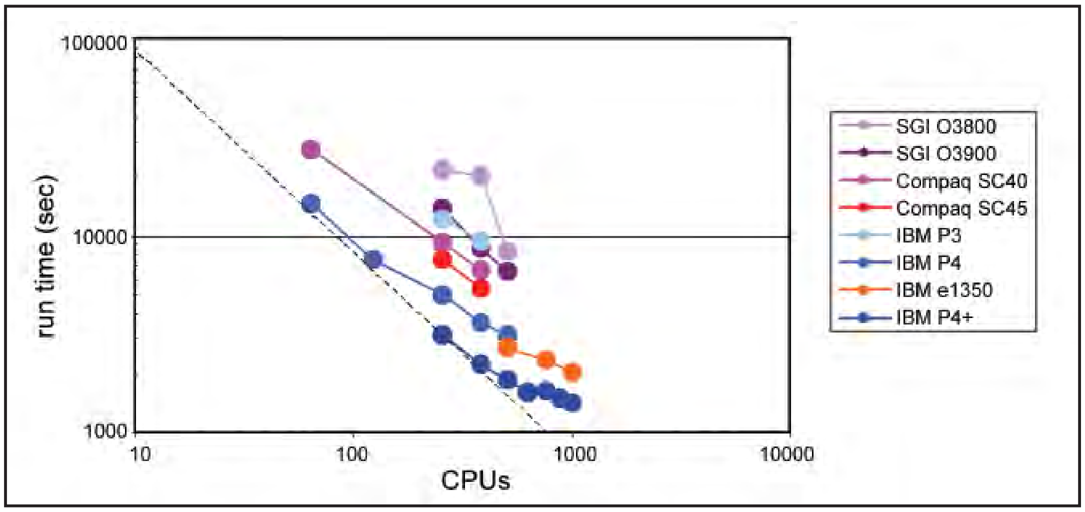


*Figure 3. TI-05 large HYCOM test case*

# Benchmarking OOCORE, an Out-of-Core Matrix Solver

*By Drs. Samuel B. Cable and Eduardo D'Azevedo*

OOCORE is the name given by the CS&E benchmarking group to the out-of-core solver developed by the SCALAPACK group at The University of Tennessee at Knoxville. OOCORE has been included in the HPCMP TI benchmark suite since 2004. Its importance in the TI suite is twofold. Firstly, OOCORE can be configured to perform by far the most disk I/O of any application code in the suite. Therefore, it is the best available application test of a platform's disk I/O capabilities. Secondly, it serves as the kernel of the SWITCH code from Northrop-Grumman Corporation, which is run by DoD HPC users investigating electromagnetic signatures. Including SWITCH in the TI benchmarks was desirable but impossible because SWITCH in its entirety is a classified code. However, the bulk of the cycles taken up by a typical SWITCH run are used by OOCORE, so OOCORE can act as a useful benchmarking surrogate for SWITCH.

OOCORE is an out-of-core solver, meaning that it solves matrix equations too large for the core memory of a set of CPUs to contain. In lieu of core memory, it stores the matrix data in temporary files on the machine's disk, hence its large amount of disk I/O. A single OOCORE input parameter allows the user to artificially restrict the amount of memory that can be used in storing the matrix. Thus, large disk I/O can be ensured when needed for testing purposes such as benchmarking. OOCORE can solve matrix equations by LU, QR, and Cholesky factorization. LU factorization is used in SWITCH, so it has been chosen for the TI suites as well. The TI-05 suite contained two runs of OOCORE: a standard test case run that solved a matrix equation with 40,000 double complex unknowns and a large test case run with 69,000 double complex unknowns. In both cases, the machines were restricted to storing in memory a relatively small maximum of 1.8 by $10^6$ matrix elements per processor.

Figure 1 shows the performance of the most common DoD HPC platforms running the TI-05 OOCORE large test case (69,000 double complex unknowns). The particular DoD machines representing the various platforms here are listed in Table 1. OOCORE was indeed run on all the DoD MSRC HPC machines. In general, different machines of the same type behaved very similarly, as expected. Figure 1 clearly shows that OOCORE scales very well on most platforms. That is, increasing the number of processors by some factor cuts the run time by approximately that same factor. Scaling on the standard test case, not shown here, was even better. The only obvious exceptions to its good scaling appear in the Compaq machines, and, at very high processor numbers, the SGI Origin 3800 and the IBM POWER4+. The SGI Origin 3800 and the IBM POWER4+ show diminishing returns from added processors at their highest processor numbers. For the SGI Origin 3800, a 1.5 increase in processor number from 256 CPUs to 384 cuts the run time by a factor of only 1.2. For the IBM POWER4+, a twofold increase from 1,024 processors to 2,048 cuts the run time by a factor of 1.25. One should note that DoD HPC users currently run their codes only very rarely above a few hundred processors.

In the Compaq SC40 data, the overall scaling is relatively flat compared with the ideal case. The SC45 competes well with the other DoD platforms on most of the TI-05 codes. Its performance on OOCORE is, therefore, less than expected. The Compaq SC45 typically achieved about 330 MFLOPS/proc on the 40,000 unknowns case and about 425 MFLOPS/proc with 69,000 unknowns. These FLOP (floating-point operations per second) rates are about 16 and 25 percent, respectively, of its theoretical peak rate. Several other platforms achieved rates of 25-35 percent of peak. When running the 40,000 unknowns case at low processor counts, the Compaq SC45 was slower

***Dr. Samuel B. Cable***
Computational Scientist
CS&E Group
ERDC MSRC

***Dr. Eduardo D'Azevedo***
Computational Mathematics Group Lead
Oak Ridge National Laboratory

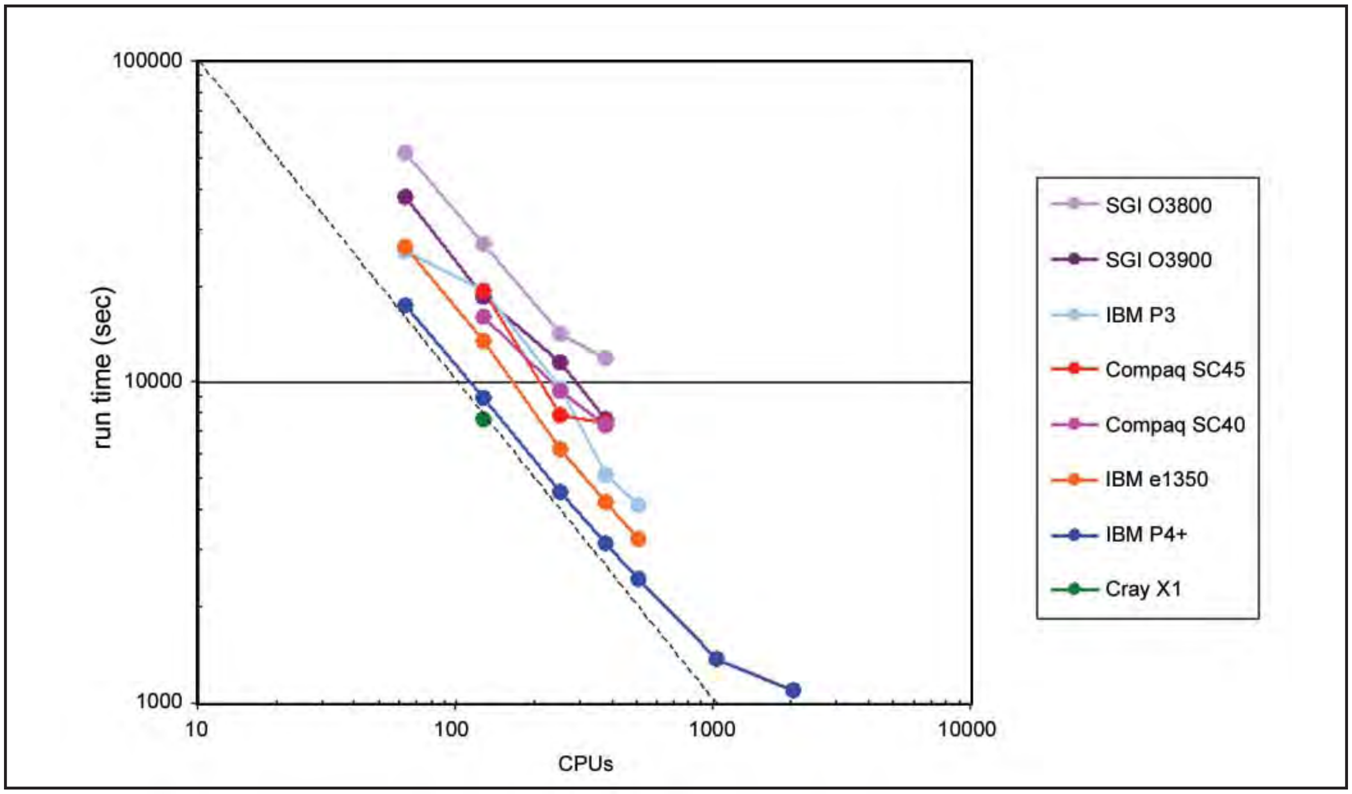


*Figure 1. OOCORE performance for TI-05. The dashed line is a guide to the eye, indicating perfect scaling of run time with number of processors. Specifically, perfect scaling is exhibited in any curve running parallel to the dashed line*

**Table 1**
**Particular DoD HPC Machines Representing Common Platforms**

| Machine Make (single-processor peak performance) | Representative Machine | MSRC |
|---|---|---|
| Cray X1 (3.2 GFLOPS) | Diamond | ERDC |
| Compaq SC45 (2.0 GFLOPS) | Emerald | ERDC |
| Compaq SC40 (1.67GFLOPS) | Opal | ERDC |
| IBM e1350 (4.4 GFLOPS) | Stryker | ARL |
| IBM P4+ (7.1 GFLOPS) | Kraken | NAVO |
| IBM P3 (1.5 GFLOPS) | Habu | NAVO |
| SGI O3900 (1.4 GFLOPS) | Sand/Silicon complex | ERDC |
| SGI O3800 (0.8 GFLOPS) | Sard | ERDC |

than the Compaq SC40, which has a very similar architecture but a slower clock. The Compaq SC45 data shown here were taken from ERDC's Emerald machine. ASC's SC45 hpc10 ran even slower, taking 91,242 seconds to factor the matrix for the 69,000 unknowns case on 64 processors (150 MFLOPS/proc) and 52,105 seconds on 128 processors (137 MFLOPS/proc). As is well-known, the Compaq SCs are relatively limited in disk I/O speed compared with some other platforms. These results, then, probably reflect OOCORE's intensity in disk I/O operations. Why OOCORE stresses the Compaq SC45 so much more than the Compaq SC40 is unknown but is possibly related to the Emerald machine's disk configuration.

Author Dr. Eduardo D'Azevedo, Oak Ridge National Laboratory, is a primary author of the OOCORE code. Author Dr. Samuel B. Cable ran the OOCORE benchmarks for TI-05.

# Benchmarking OVERFLOW 2, a Structured CFD Code

*Drs. Pieter G. Buning and Samuel B. Cable*

Complex, unsteady flow problems abound in aerospace applications. From winged flight to propulsion systems to internal flows, more and more fluid dynamics problems are being tackled using computational simulations. OVERFLOW 2 is a flow solver for modeling compressible, viscous flow, with a moving body capability. It is the product of merging the National Aeronautics and Space Administration (NASA)-developed OVERFLOW code with the 6-degree-of-freedom moving body simulation techniques incorporated in OVERFLOW-D, developed by Dr. Robert Meakin and colleagues at the U.S. Army Aeroflightdynamics Directorate at NASA Ames Research Center in California.

Begun in 1990, OVERFLOW development was funded by the Space Shuttle Program Office following the Challenger accident. The code was an outgrowth of existing CFD codes and methods in use at NASA Ames and was used primarily to simulate aerodynamic loading on the shuttle launch configuration during ascent, including Orbiter, External Tank, and Solid Rocket Boosters. Further applications were pursued in a number of areas, including powered lift, advanced subsonic and supersonic transport design, and launch vehicle aerodynamics. The moving body capability (which became OVERFLOW-D) was also funded by the shuttle project, but included collaboration with the Air Force Arnold Engineering Development Center and later the Army Ballistic Research Laboratory. Applications of this capability included store separation, submunition dispense, and rotorcraft dynamics. OVERFLOW 2 is currently being used for a variety of civil- and defense-related applications, including space shuttle return-to-flight, capsule unsteady aerodynamics, rotorcraft and wind turbine aerodynamics, business jet and antenna fairing design, and a novel dart-dispense problem by the Navy.

The OVERFLOW 2 code simulates the Reynolds-averaged Navier-Stokes equations of fluid motion using finite differences and implicit approximate factorization methods. This involves the solution of a set of penta-diagonal matrices for every step as the equations are iterated to a steady-state result, or every subiteration for a time-accurate process. The airflow is simulated on a set of structured computational grids representing the volume about a vehicle. The use of structured grids allows efficient memory access and matrix solution procedures to be employed. However, these grids can be overlapped in a variety of ways to more conveniently represent complex shapes and to allow relative motion between bodies or components, such as the rotating blades and tilting nacelles of the V-22 Tiltrotor (Figure 1). Communication between the component grids is handled through automated hole cutting and interpolation of boundary information.

Parallel computing in OVERFLOW 2 is accomplished using a hybrid approach. At the coarse-grained level, component grids are subdivided (if necessary) and collected into groups. Each group is assigned to a processor or group of processors, with communication between groups handled using MPI. The solution procedure for each component grid includes fine-grained parallelism using OpenMP, with each slice of the grid processed in parallel. With these two techniques, efficient use can be made of both shared- and distributed-memory architecture platforms.

In TI-05, OVERFLOW 2 was run with two test cases, a standard test case with about 30 million grid points (30MGP) and a large test case with 120 million grid points (120MGP). Figure 2 illustrates the performance of the more common DoD HPC architectures in running the 120MGP case of OVERFLOW 2. Overall performance on the code is as expected. The Cray X1 is the fastest machine, followed by the relatively fast IBM e1350, the IBM P4+, and the Compaq SC45. The SGI Origin 3800, a somewhat older platform, is the slowest. The P4+ performance is significantly better than the old P4 performance (not shown), which performed similarly to the Compaq SC45.

Codes are run on multiple processors to cut

*Dr. Pieter G. Buning*
Aerospace Engineer
NASA Langley Research Center

*Dr. Samuel B. Cable*
Computational Scientist
CS&E Group
ERDC MSRC

run time by distributing the workload among the processors. A code "scales well" if increasing the number of processors by some factor cuts the run time by approximately that same factor. Scaling in the standard test case was quite good on most platforms. Figure 2 shows that the OVERFLOW 2 large test case scaled well on the three IBM machines. (Insufficient data were available to determine the scaling on the X1, but it is showing good scaling in TI-06.) Scaling on the SC45 was marginal, while scaling on the SC40 and the SGI machines was problematic. The SGI Origin 3800 behaves rather erratically in the 120MGP case, sharply increasing run time from 128 to 256 CPUs, and then decreasing again from 256 to 384 CPUs. The SGI Origin 3900 and the Compaq SC40 both run the 120MGP case more slowly at 384 CPUs than at 256 CPUs.

In TI-06, OVERFLOW 2 is scaling much better on these previously problematic platforms. Run times of all cases run so far are monotonically decreasing across the entire range of processors. This improvement is due in part to interactions between the OVERFLOW 2 developers and the TI-05 application benchmark team. In TI-05, the SGI machines were determined to be taking inordinately long times to read the input data. Subsequent improvements in OVERFLOW 2 remedied this problem.

Author Dr. Pieter Buning is the primary author of the OVERFLOW 2 code. Author Dr. Samuel B. Cable ran the OVERFLOW 2 benchmarks for TI-05.

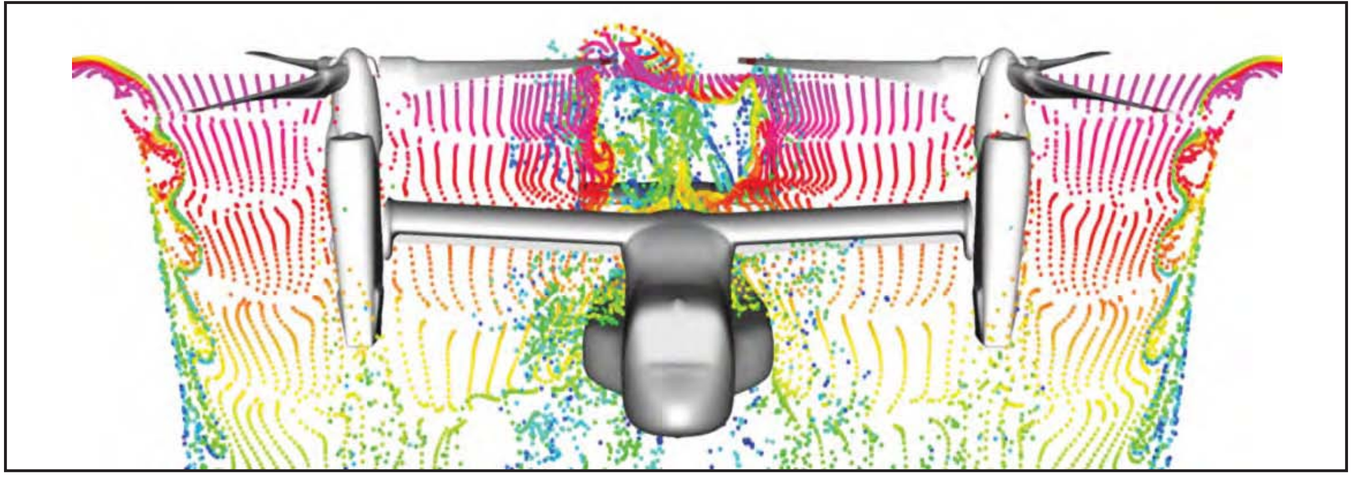


*Figure 1. Airflow about the V-22 Tiltrotor as calculated by OVERFLOW 2 (Graphic courtesy of Mark Potsdam, U.S. Army Aeroflightdynamics Directorate, NASA Ames Research Center, CA)*

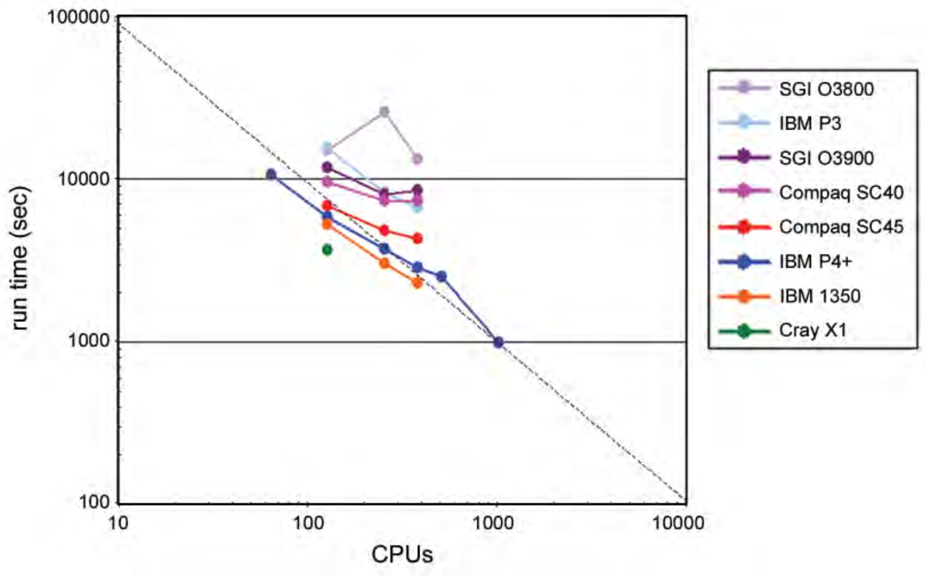


*Figure 2. Run times of OVERFLOW 2 on common DoD HPC platforms. The dashed line is a guide to the eye, indicating how run times would scale on perfect multiprocessor machines. Specifically, perfect scaling is exhibited in any curve running parallel to the dashed line*

# Benchmarking RF-CTH, a Shock Physics Code

*By Robert W. Alter*

*Robert W. Alter*
Computational Scientist
CS&E Group
ERDC MSRC

RF-CTH (Reduced Functionality CTH) was synthesized from the 1999 export controlled version of CTH, to provide a nonexport controlled benchmark code for the DoD TI program. CTH is a family of codes developed by Sandia National Laboratories for modeling complex multidimensional, multimaterial problems that are characterized by large deformation and strong shocks. CTH is widely used in the DOE and has become one of the most heavily used applications in the DoD research community. Since the beginning of the TI process, either CTH or RF-CTH has been one of the benchmark codes. RF-CTH has limited capabilities to make it nonexport controlled as compared with the full version of CTH, but it was designed to exhibit the same benchmark performance as CTH. Development of RF-CTH was done in an effort to facilitate the TI process, whereby the HPC vendors do not have to sign export control agreement to get a copy of the source code.

CTH is an acronym of an acronym; it stands for “CSQ to the Three-Halves.” CSQ stands for “CHARTD Squared,” where CHARTD stands for Computational Hydrodynamics and Radiative Thermal Diffusion. CHARTD, CSQ, and CTH are 1-, 2-, and 3-D codes, respectively. CTH can be compiled as a serial or as a parallel MPI execution. The CTH software family consists of CTHGEN, which sets the initial configuration of the problem; CTHREZ, which rezones the problem; CTHPLT, which produces graphics at specified intervals; and CTH, which computes the physics models (Hertel et al.). CTH is a time-domain, structured-grid, Eulerian code that can run in 1-, 2-, or 3-D and commonly used Cartesian or cylindrical coordinates. It is capable of using adaptive mesh refinement (AMR) to partition the grid or use CTHGEN to partition the grid onto each MPI process. CTH is capable of modeling material strength, factures, porous materials, and high-explosive detonations (Hertel et al.), where multiple materials and voids can occupy a single computational cell.

RF-CTH was developed from the 1999 version of CTH by stripping the code, physical modes and material properties that make CTH export controlled. The basic shock hydrodynamics were retained, but all of the advanced material equations, facture models, and models of explosive behavior were removed. A Simple Line Interface Construction (SLIC) model was retained for the material interface reconstruction modeling. Finally, all FORTRAN comment lines were removed to make it virtually impossible to restore any lost capability. To date the TI test cases have modeled a rod penetration into a thick plate. For example, the TI-05 standard test case used an adaptive mesh (AMR mesh) modeling a 7.67-cm-long rod made of 10 materials penetrating a plate made of 8 materials at a oblique angle of 73.5 degrees and having an initial velocity of 1,210 meters per second. Each MPI process could have a maximum of 520 blocks, where each block contained 8 by 8 cells, with a maximum of five mesh refinement levels. The TI-05 large test case used a fixed grid requiring 240 GBytes of memory divided up by the MPI process. The grid had 200 million cells modeling the same rod and plate as in the standard test case.

The performance of some of the existing MSRC hardware for the standard TI-05 test cases is presented in Figure 1. The benchmark performance was obtained on

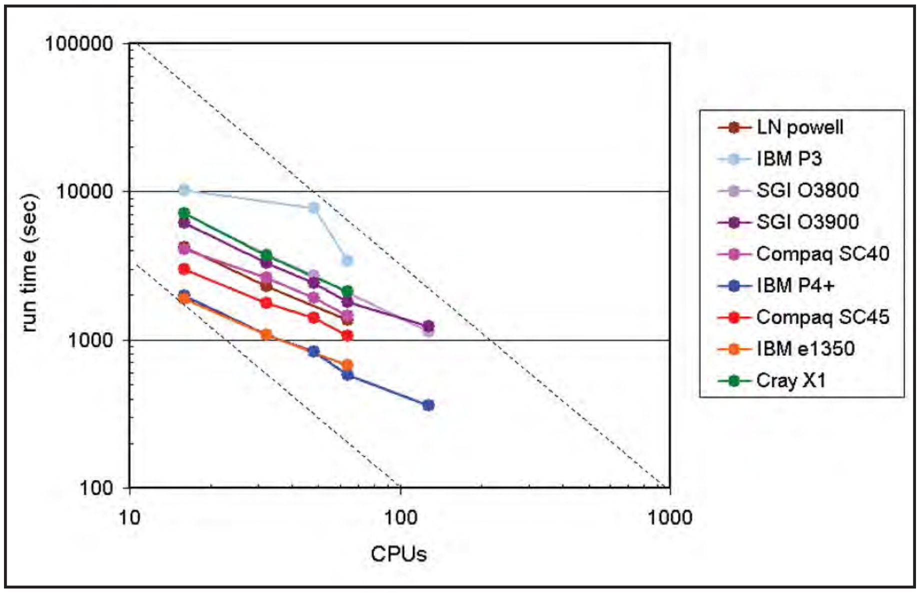


*Figure 1. RF-CTH performance for TI-05*

Hertel, E. S., Jr.; Bell, R. L.; Elrick, M. G.; Farnsworth, A. V.; Kerley, G. I.; McGlaun, J. M.; Petney, S. V.; Silling, S. A.; Taylor, P. A.; and Yarrington, L., *CTH: A Software Family for Multi-Dimensional Shock Physics Analysis,* Proceedings of the 19th International Symposium on Shock Waves, Volume 1, pages 377-382, Marseilles, France, 26-30 July 1993.

the following HPC systems: the 1.3-GHz IBM POWER4 (Marcellus) at the NAVO MSRC, the 700-MHz SGI Origin 3900s (Sand/Silicon) at the ERDC MSRC, the 1-GHz Compaq SC45 (Emerald) at the ERDC MSRC, the 1.7-GHz IBM POWER4+ (Kraken) at the NAVO MSRC, and the 3.06-GHz LNX Xeon cluster (Powell) at the ARL MSRC. The standard TI-05 test case scales up to 64 processing elements (PEs), with the IBM POWER4+ (Kraken) demonstrating the best performance with a next-best performance on the Compaq SC45 and IBM POWER4 (Emerald and Marcellus).

## Benchmarking WRF, a Next-Generation Weather Research and Forecasting Model

*By John G. Michalakes and Dr. Thomas C. Oppe*

The Weather Research and Forecast (WRF) project is a large, multi-institution effort to develop a next-generation mesoscale forecast model and assimilation system designed to advance both the understanding and the prediction of mesoscale precipitation systems as well as promote closer ties between the atmospheric research and operational forecasting communities. WRF can be applied to problems in storm-scale research and prediction, air-quality modeling, wildfire simulation, hurricane and tropical storm prediction, regional climate, and operational numerical weather prediction (NWP) (Figure 1).

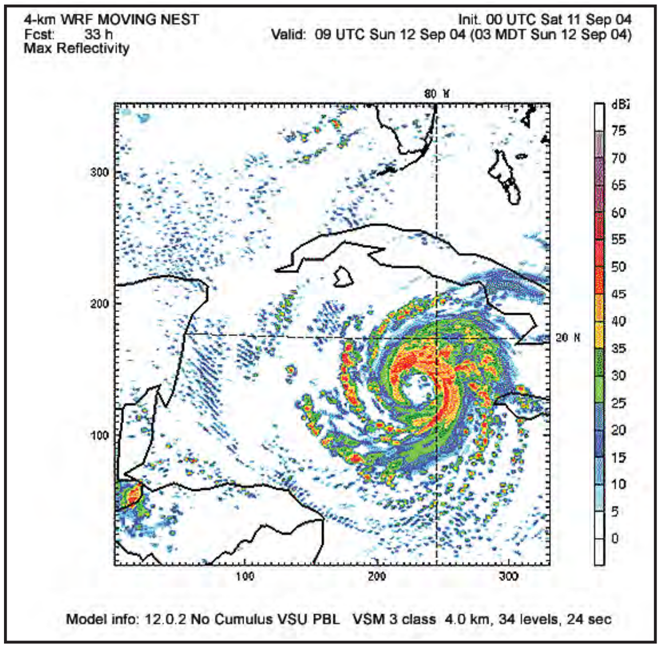

4-km WRF MOVING NEST
Fcst: 33 h
Max Reflectivity
Init. 00 UTC Sat 11 Sep 04
Valid: 09 UTC Sun 12 Sep 04 (03 MDT Sun 12 Sep 04)
80 W
20 N
dBZ
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
300
200
100
100
200
300
Model info: 12.0.2 No Cumulus VSU PBL VSM 3 class 4.0 km, 34 levels, 24 sec

*Figure 1. Still image from a WRF simulation of Hurricane Ivan in September 2004. (For the WRF animation of the hurricane, see www.mmm.ucar.edu/wrf/WG2/wrf_moving_nest.gif)*

WRF is similar to MM5, the PSU/NCAR Mesoscale Model, in many respects, and both codes have a common institutional heritage in the NWP community. The development of WRF started in 1998, but the code

***John G. Michalakes***
Senior Software Engineer
Mesoscale and
Microscale Meteorology
NCAR

***Dr. Thomas C. Oppe***
Computational
Scientist
CS&E Group
ERDC MSRC

was in planning before that time. The first alpha community release of the software occurred near the end of 2000. A major beta release of the software occurred in May 2003 as WRF 1.3. A full research community release occurred in May 2004 as WRF 2.0. Operational implementation of WRF is underway at the National Oceanic and Atmospheric Administration (NOAA) National Centers for Environmental Prediction (NCEP) and at the U.S. Air Force Weather Agency (AFWA). A joint NOAA/National Center for Atmospheric Research (NCAR)/DoD Developmental Testbed Center has been formed to facilitate the ongoing testing, evaluation, and transition of new developments from the research community into operations at NCEP, AFWA, and at the U.S. Navy. WRF was introduced into the DoD HPCMP application benchmark test suite as a climate/weather/ocean modeling and simulation (CWO) component with TI-05. The version used was WRF 2.0.2.

The WRF Model contains numerous options for physical parameterizations and two choices for dynamical core, providing maximum flexibility across institutions and applications. The NCAR-developed Advanced Research WRF (ARW) used for this benchmark employs a time-split high-order Runge-Kutta method to integrate a conservative formulation of the compressible nonhydrostatic equations for mass and momentum. ARW is supported to the research community as WRF Version 2 and is undergoing operational implementation at the AFWA. NOAA/NCEP operational implementation of WRF is using dynamics adapted to the WRF Advanced Software Framework (ASF) from the Nonhydrostatic Mesoscale Model (NMM). The Naval Research Laboratory Marine Meteorology Division is adapting the COAMPS (Coupled Ocean/Atmospheric Mesoscale Prediction System) model so that both COAMPS and WRF will use unified physics through a common interface within the WRF framework; in addition, COAMPS and WRF will be interoperable with respect to data conventions and formats.

One of the key objectives in the WRF project has been development of a software framework for mesoscale modeling that is efficient, portable, maintainable, and extensible, thus allowing incremental and rapid development of new algorithms while maintaining overall consistency and adherence to architecture and interface requirements. The WRF 2.0 release supports the full range of functionality envisioned for the model, including efficient scalable performance on a range of HPC platforms, multiple dynamic cores and physics options, low-overhead two-way interactive nesting, moving nests, support for model coupling, and interoperability with other common model infrastructure efforts such as the Earth System Modeling Framework (ESMF). The WRF ASF features a modular, hierarchical organization of the software that insulates scientific code from parallelism and other architecture-, implementation-, and installation-specific concerns. This design has also been crucial for managing the complexity of a single-source-code model for a range of users, applications, and platforms. A flexible approach for parallelism is achieved through a two-level decomposition in which the model domain may be subdivided into patches that are assigned to distributed-memory nodes and then may be further subdivided into tiles that are allocated to shared-memory processors within a node. Model layer subroutines are required to be tile callable, that is, thread-safe and callable for an arbitrarily sized and shaped subdomain. All data must be passed through the argument list (state data) or defined locally within the subroutine. No COMMON or USE-associated state array with a decomposed dimension is allowed. Domain, memory, and run dimensions for the subroutine are passed separately and unambiguously. This approach addresses all current models for parallelism (single-processor, shared-memory, distributed-memory, and hybrid) and is flexible with respect to processor type: tiles may be sized and shaped for cache blocking or to preserve maximum vector length.

WRF is written in FORTRAN 90 and C. It uses MPI calls and OpenMP compiler directives to achieve parallelism. WRF currently runs on the following systems:

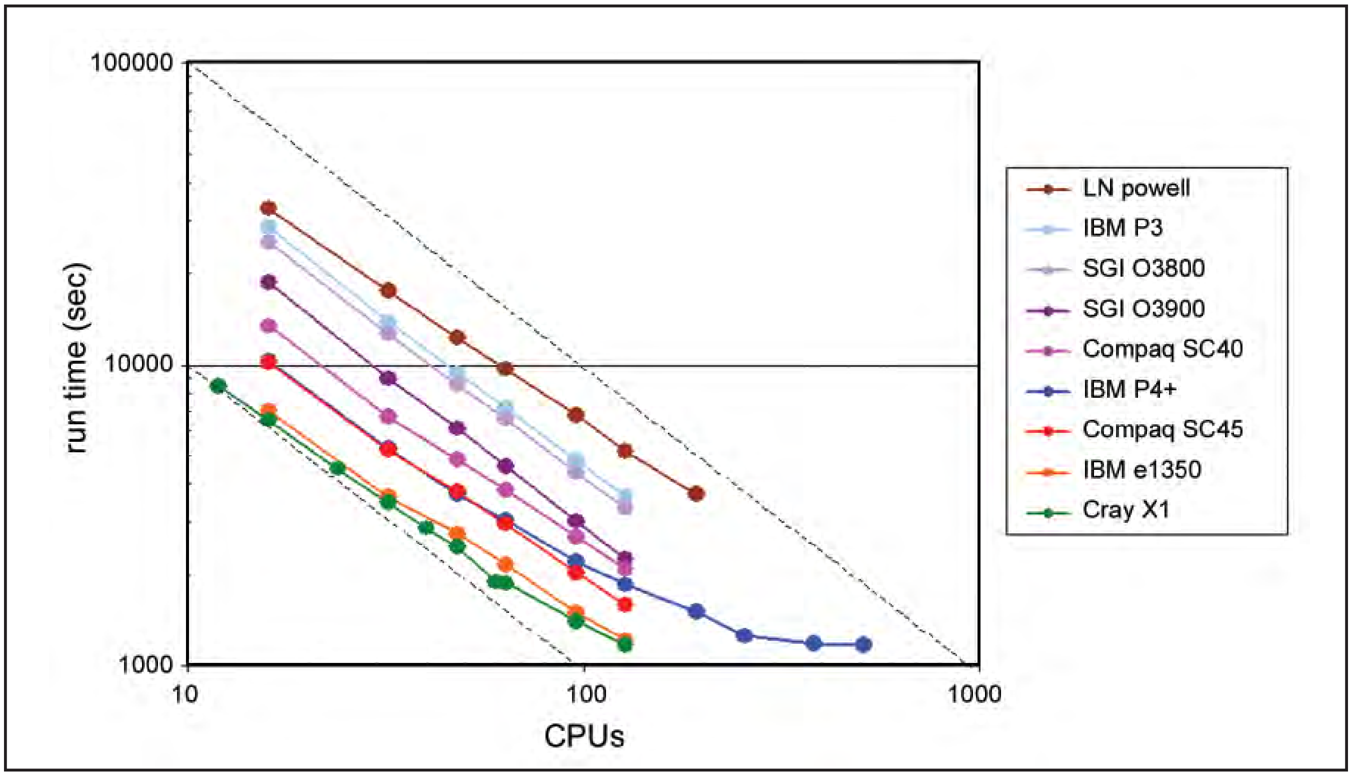


*Figure 2. WRF performance on selected DoD HPC platforms*

IBM SP, SGI Origin and Altix platforms, Compaq SC45 and Alpha True64 platforms, Cray X1, and Linux clusters using Alpha, Pentium, Xeon, Itanium, or Opteron processors and Intel or Portland Group compilers.

The WRF TI-05 benchmark consisted of one case, called the standard case. This case was a 48-hour Continental U.S. (CONUS) simulation using a 12-km spatial resolution in the horizontal dimensions. The horizontal grid was 425 by 300 cells and also had 35 vertical levels. Input and output for the TI-05 benchmark test case was generally done using NetCDF, although two large big-endian IEEE binary files and several small ASCII files containing input data to initialize particular physics packages were also required. NetCDF was used for the creation of the main output file containing the state field variables. The model was run in distributed-memory parallel mode, for which WRF can scale in the 60- to 80-percent parallel efficiency range to approximately 256 processors. Figure 2 gives the run times for WRF on selected HPC platforms in the DoD MSRCs.

## Synthetic Benchmarks for TI-05

*By Cal Kirchhof and Henry Newman*

***Cal Kirchhof***
BPA Program Manager
Instrumental, Inc.

***Henry Newman***
Chief Technical Officer
Instrumental, Inc.

Synthetic benchmarks used in TI-05 are a group of tests developed by Instrumental, Inc., for the DoD HPCMP. The various tests measure the performance of all the main subsystems of HPC systems and provide a picture of the best features the machine has to offer, as well as where bottlenecks or design flaws can negatively impact the system as a whole.

The goal of the synthetics is to understand the "Whys?" of machine performance. While the application benchmarks might show that two machines proposed from two vendors have different run times, the synthetics can point to the reason of the performance difference, which can be something as simple as cluster interconnect latency. The Synthetics Benchmark Suite comprises five groups of tests, CPUBench, MEMBench, NETBench, OSBench, and StreamingIO.

CPUBench contains a series of tests that measure single CPU computational performance of common computational kernels. These tests are performed in a cache-friendly and cache-unfriendly way with multiple operand sizes (32- and 64-bit words) and with both C and FORTRAN compilers. The results of these nearly 150 tests can be plotted to show a 3-D surface of the profile of each system's computational properties with respect to cache, word size, and compiler choice. CPUBench also has multi-CPU tests using various linear algebra libraries (BLAS, SCALAPACK, and a parallel conjugate gradient solver) that measure the scalability of the system for parallel applications.

Since memory subsystems typically operate at clock-cycle rates many times slower than CPU-instruction execution times, memory performance significantly affects overall HPC system performance. MEMBench tests memory performance by measuring a variety of memory operations in varying patterns to show how memory read-ahead performs, how memory stride patterns affect overall memory bandwidth, and how memory operations perform in a parallel environment. More complex memory-access operations are measured with several Fast Fourier Transform (FFT) tests at various matrix sizes, and using both MPI and OpenMP parallel-programming libraries.

NETBench measures the performance of the network of interconnected CPUs in a cluster or other multiprocessor system. In these systems, a CPU may be required to perform a significant amount of memory accesses from remote nodes. Poor performance on these operations can have severe impacts on general system performance. NETBench measures performance of blocking and nonblocking transfers from one node to another, collective broadcasts (one node sending to all other nodes), and the collective Allreduce test (one node receiving transfers from all other nodes). Additionally, NETBench measures the ability of the interconnect to respond to rapidly changing back-and-forth transfers (ping-pong operations) that are both blocking and nonblocking as well as bi-directional, blocking and nonblocking ping-pong operations with blocks up to 16 MBytes.

OSBench measures the operating system's performance on various system calls. It measures read and write performance for both external storage and pipes. OSBench measures the ability of the operating system to scale transmission-control-protocol (TCP) performance over a range of CPU counts. It also measures the operating system's performance on IPv4 and IPv6 networking stacks with test strategies similar to ping-pong tests in NETBench.

Over the years where the synthetics benchmarks have been used, IOBench has included a number of intense I/O tests sometimes requiring many hours to run. In TI-05, IOBench was scaled back and replaced with StreamingIO to test simple, but large, I/O operations involving sequential writes, sequential reads, and full duplex read/writes (i.e., simultaneous reading and writing).

The benchmarking process begins with the vendors running the application and synthetic benchmarks on systems they hope to sell to the U.S. Government. The vendors pick systems and configurations they feel will meet or exceed the Government's needs and run the benchmarks on those systems. After the U.S. Government procures a system, the benchmarks are run again to ensure that the systems are delivering the performance as promised. Each year as a part of the current TI process, the benchmarks are run again. This provides an updated look at how the system is performing based on its configuration at that point; and combined with the results of all the systems in the HPCMP, it provides a baseline for what the Government will require from vendors in the upcoming TI cycle.

Application and synthetic benchmarks are used together in measuring and evaluating systems. The application benchmarks provide reliable measures of how those representative applications can be expected to perform on systems under evaluation, as well as good indications of the performance of similar codes. When applications have unexpected benchmark results with respect to a particular type of architecture or configuration, the synthetic benchmark results can be used to focus on particular subsystems or interactions among subsystems to track down the cause of unexpected results, good or bad.

Synthetic benchmarks were introduced in FY 2001 and have been continually upgraded and enhanced since that time. Many of the changes increased the coverage of various aspects of the systems and subsystems tested. Extensive changes have been made to some of the variables measured, as has been the case with IOBench. For TI-06, a number of the individual tests have been combined into a smaller set of tests, and some have been dropped either because they are now outdated or have been found to be redundant. The goal for future versions of the Benchmark Suites is to reduce the time required for the vendors to prepare and execute them. This may be accomplished by cutting back on the number of tests or by increasing the level of automation, while still maintaining complete system-measurement coverage. As technology changes and new features are added to HPC systems, the benchmark tests will change to ensure that those features are properly measured and evaluated.

## Benchmarking the Locality Space with HPCC

*By Dr. Paul M. Bennett*

***Dr. Paul M. Bennett***
Computational Scientist
CS&E Group
ERDC MSRC

The High Performance Computing Challenge benchmark, or HPCC, complements the LINPACK benchmark, which is used to rank HPC systems in the Top 500 list according to their performance. As such, HPCC is a synthetic benchmarking program. It includes the LINPACK benchmark to measure the floating-point rate to solve a dense system of linear equations. However, HPCC performs six additional benchmark tests to determine the overall capabilities of the systems running it.

The first of these is a double-precision matrix-matrix multiplication, performed by the Basic Linear Algebra Subprograms (BLAS) algorithm DGEMM, to determine the performance rate for double-precision floating-point execution. The second benchmark measures the sustainable-memory bandwidth and corresponding computation rate for a simple vector kernel. After that, HPCC performs a parallel matrix transposition to determine the total communication capacity of the system's network. HPCC then measures the rate of integer updates to random memory locations, after which it measures the computation rates to perform a double-precision complex FFT. HPCC finishes its work with a set of tests that measure the latency and bandwidth of several simultaneous communication patterns.

The benchmark routines performed by HPCC can be used to bound the performance of many typical HPC applications in terms of spatial and temporal locality, four of which are presented in Figure 1. Each of HPCC's benchmarking routines is characterized by a certain degree of spatial and temporal locality in data access patterns, according to which they are plotted in the figure. Each routine is

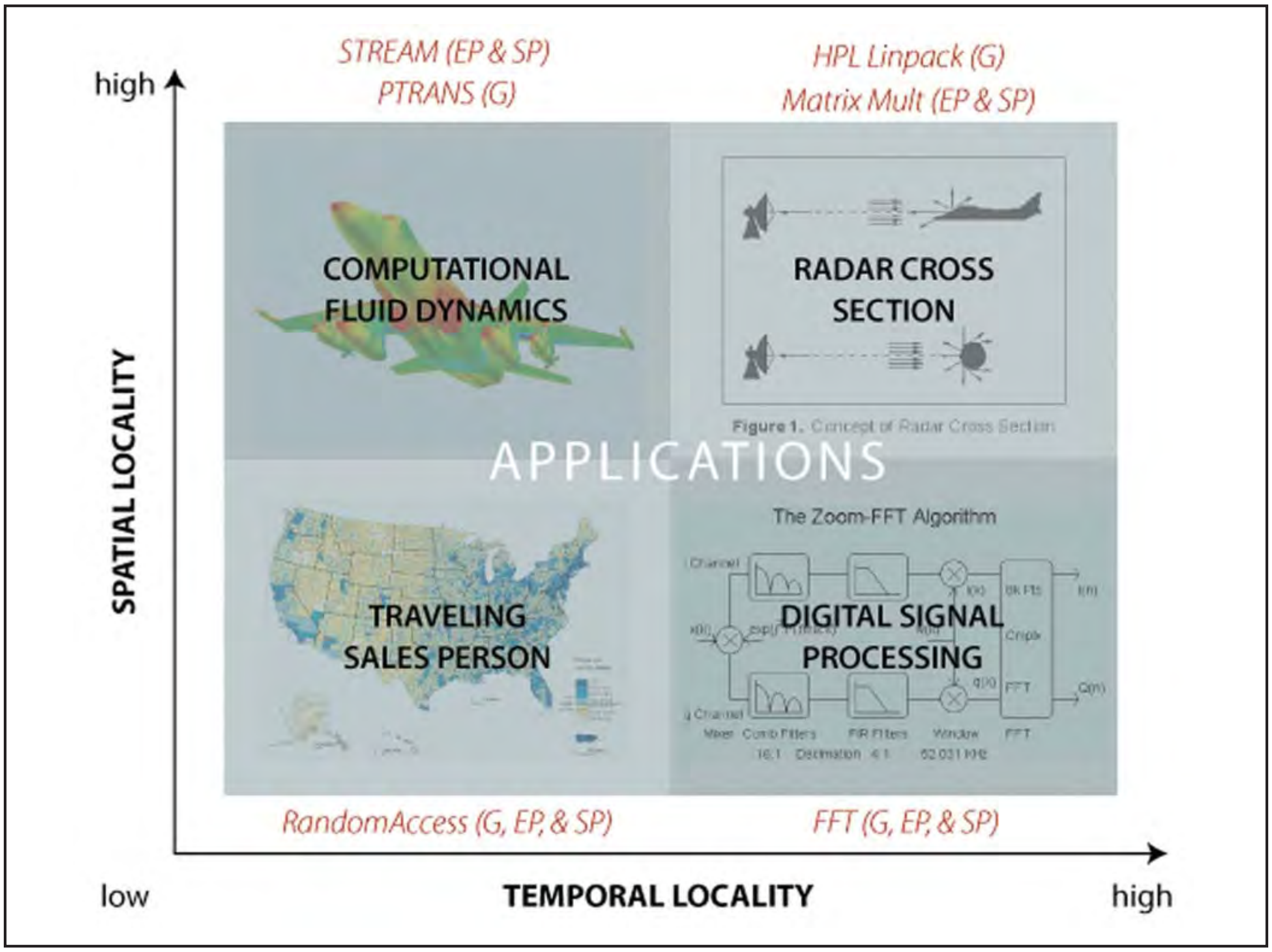


*Figure 1. Bounding the performance of four typical HPC applications in terms of spatial and temporal locality*

used in global tests, embarrassingly parallel tests, or single-process tests, as indicated by the letters *G*, *EP*, and *SP*.

HPCC produces a text file that the user can submit to the Web site http://icl.cs.utk.edu/hpcc by following the link to upload data. At the time of writing, benchmark results for 54 systems can be found at that site by following the link to view the results.

The on-line results can be displayed according to any one of several different criteria, including speed of the CPU, manufacturer of the CPU, manufacturer of the HPC system, and so on. At the time of writing, each system has five or eight benchmark times of various categories, which together describe its overall performance. Notably, no one system outperforms all others in every category. For example, while the ERDC MSRC's Cray T3E outperformed all other listed systems with the best random access performance, it had only 9.4 percent of the top performance on the LINPACK, or HPL, benchmark. The listed system with the best High Performance LINPACK (HPL) performance is the Cray X1 at Oak Ridge National Laboratory.

The HPCC benchmark was produced at the Innovative Computing Laboratory (ICL) at the University of Tennessee, Knoxville, through a project sponsored by the National Science Foundation (NSF), DOE, and Defense Advanced Research Projects Agency (DARPA) HPC systems. The DoD HPCMP was asked to assist in Fall 2003 by performing HPCC benchmarks on HPC systems at each MSRC. Several updates have been made to HPCC since that time, and the ERDC MSRC continues to provide assistance by maintaining current HPCC benchmarks. More information about HPCC can be found at the above-mentioned Web site.

The author would like to acknowledge Dr. Piotr Luszczek, ICL, at the University of Tennessee, for his suggestions and assistance with this article and the ICL team for the Figure 1 graphic.

# Stat.pl: Automating the Scoring of Benchmark Systems

*By Drs. Alvaro Fernández and William A. Ward, Jr.*

This article describes the rationale and methodology used to assign benchmark scores to HPC machines in the HPCMP. It also shows the input data needed and some sample output.

Certain terms and concepts are key to understanding the rationale behind the scoring scheme that has been developed. Of these, two of the most important terms are the baseline system (BLS) and the system under test, or SUT.

The BLS is a particular system, chosen anew every year from systems in the HPCMP. During TI-05, for example, the machine chosen was Marcellus, an IBM Power4 located at the NAVO MSRC. The criteria for choosing a BLS are speed, stability, and maturity. Thus, while the BLS should not be an old machine, neither should it be the newest in the Program. Fast and dependable are both key in the selection of the BLS.

In general terms, the scoring procedure seeks to measure the performance of each SUT relative to the chosen BLS. In benchmarking, the performance of any system is typically measured by recording the wall time[1] for the execution of certain specific codes (benchmark codes) for given numbers of processors. The expectation is that the wall time needed to execute a given code will decrease as more processors are brought to bear on the program being run. When this decrease is both significant and steady, the benchmark code is said to scale well. In this vein, the scoring scheme described here rewards good scaling behavior with a higher score and conversely punishes bad scaling with a lower score. This rationale should be kept in mind as the scoring procedure is described.

***Dr. Alvaro Fernández***
Computational Scientist
CS&E Group
ERDC MSRC

***Dr. William A. Ward, Jr.***
CS&E Group Lead
ERDC MSRC

To begin, each benchmark code is run on the BLS to generate at least two (and preferably three or more) wall times $T_1, T_2 T_3, \ldots$ at increasing processor counts $n_1, n_2 n_3, \ldots$ . One of these processor counts is predetermined and is called the baseline number of processors. The time needed to run the given code on the SUT using this baseline number of processors is called the baseline time, or BLT.

These times are then inverted to obtain a new quantity, called performance $P = {}^{1}\!/_{T}$. For each SUT, the resulting data, composed of processor/performance pairs $(n_i, P_i)$, are fit to a power law $P(n) = an^b$ by performing a linear least-squares fit.

With BLT, $a$ and $b$ computed, a new quantity, *cpus_std*, can be computed as

$$cpus\_std = \left( \frac{1}{a(\mathrm{BLT})} \right)^{\frac{1}{b}} .$$

This formula takes the BLT, recorded from the BLS, and uses it to compute the number of processors of the SUT that would be needed to match the baseline time of the BLS. Then, an alternate quantity is computed:

$$cpus\ \ alt = \frac{(cpus@max\ t)(max\ t)}{\mathrm{BLT}} .$$

Both *cpus_std* and *cpus_alt* are an attempt to express the performance of the SUT in units of BLS processors. However, while both quantities use the BLT, *cpus_std* uses the power law and its coefficients, while *cpus_alt* effectively performs a linear interpolation based on the number of processors used when the maximum wall time for this test case was measured (*cpus@max_t*).

[1] The time an observer, looking at a clock on the office wall, would measure from the start of a program to its successful end.

Some additional quantities needed to compute the overall score are as follows:

1. Compute_cpus: the number of processors the SUT has available for processing
2. Machine_fraction: the typical fraction of the total machine this benchmark codes uses (e.g., 1/16)

Once all the preceding quantities are computed, the complete calculation of the score essentially employs the curve fit to project the performance of the SUT at the required number of CPUs for each test case. Once that is accomplished, the overall score is computed taking into account the scaling of the code (the exponent "*b*") and the number of CPUs the code typically would use (compute_cpus and machine_fraction).

## . . .implementation of scoring algorithm

Processing of this information is automated by Stat.pl, 2,000-line Perl script. The input data are organized into hierarchical objects, or blocks. The types of blocks are system blocks, test case blocks, and score blocks. System blocks describe the attributes of the computer system, such as its name, interconnect, etc. Each system block may contain one or more test case blocks, which contain actual run information, such as processor counts and wall times. Score blocks contain information relating to all systems and test cases being processed, e.g., the BLS's name, how many CPUs it has, the machine fractions. There is one system block per SUT and one overall score block.

Stat.pl reads in these blocks from properly formatted text files and generates one spreadsheet in comma separated value (CSV) format for each test case. These spreadsheets summarize the statistics for each SUT in the given test case.

Stat.pl also generates multiple PDF files (one for every test case) containing performance plots, a typical example of which is shown in Figure 1. The SUT for this example is Marcellus, the BLS. The benchmark code AVUS is being run as a large test case. One notes the reported parameters for the curve fit (*a*, *b*, and *rsq*) are reported in the plot. The exponent *b* is slightly more than one, indicating better than linear scaling; *rsq* is also one, indicating a good fit. The BLP and n_BLP are both plotted on the performance curve.

Future work will include embedding the scoring procedure and its data structures in a database, as well as providing a smoother interface to the price/performance/workload allocation optimizer.

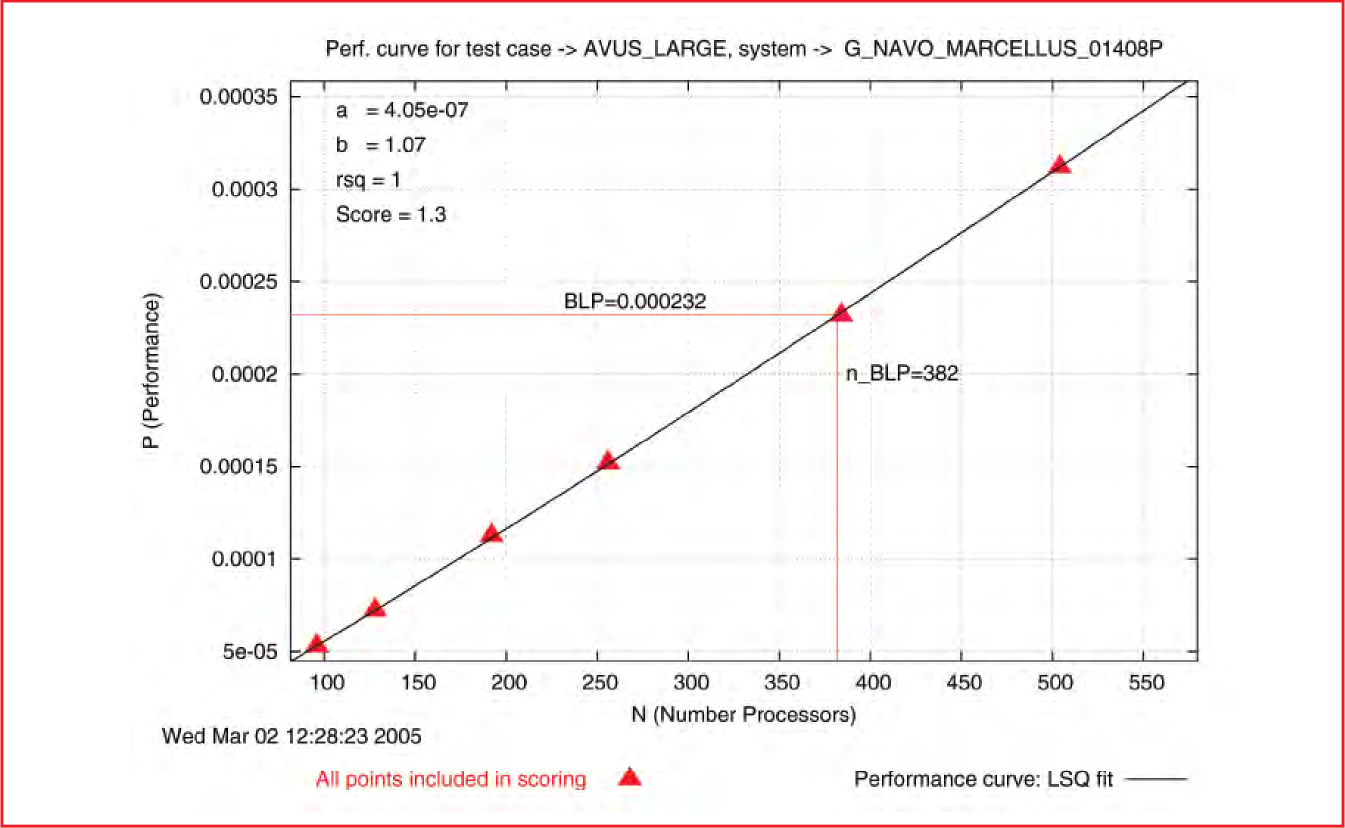


*Figure 1. A representative plot generated by Stat.pl for the AVUS benchmark code running on Marcellus. One can note the power law exponents reported. Scaling is excellent*

# Analyzing Price per Performance

*By Dr. Roy L. Campbell, Jr.*

*Dr. Roy L. Campbell, Jr.*
HPCMP Performance Team Vice-Chair, ARL

Analysis of price-per-performance, one of several important aspects considered during an HPCMP TI, involves hundreds of parameters ranging from individual benchmarking results to offered system pricing; therefore, a reliable tool that ties all of these quantities together can prove essential when conducting an HPC acquisition worth tens of millions of dollars. The HPCMP optimizer is a near-real-time, task-parallelized, linear optimization solver that processes pricing, benchmarking results, acquisition budget limitations, and a cryptic description of the target workload into two outputs – the most critical of which is the ideal quantity of each offered system that should be purchased and the lesser being the ideal allocation of systems (both elements being gauged solely in terms of price-per-performance). Put simply, to the extent that the application test cases constructed by the performance team are representative of the true workload, the HPCMP optimizer determines what systems to buy and how work should be distributed among the selected and already owned systems in an optimal sense when considering only bang-per-buck.

It would seem that the concepts behind this sort of insightful analysis would have been exhaustively explored by now. In many regards, they have. Optimization gained its momentum in World War II as the number of logistical and tactical variables mushroomed, sowing the seed for a discipline commonly referred to as operations research. The end products of optimization can now be seen in many somewhat hidden, but very powerful, ways ranging from advertisement strategies aimed at cost-effective penetration of the consuming public to the bent wingtips typically seen on Boeing 737s.

So why is so much enthusiasm being placed behind a technique that is ancient by technological standards? Most of the energy can be attributed to the tool's ability to rapidly solve the quantitative TI-XX problem. Some debate exists regarding whether using a linear programming technique rather than an integer programming technique is a wise use of resources, since the former requires, in many cases, 32 or more processors to yield a solution in near-real-time (i.e., within an hour), while the latter (for the same problem) requires a single processor producing a solution within a few seconds. The strategic selection of the former stems from the HPCMP's desire to uncover the top 10,000 acquisition possibilities to allow exhaustive analysis of the price-per-performance space. In such a case, the time-to-solution for the linear programming technique is dilated by a few percent, while that for the integer programming technique balloons to days or weeks. As an example of how price-per-performance results might be presented to decision makers, Figure 1 provides a comparison of the best possibility versus the average of the top 10,000 possibilities for TI-05.

The HPCMP optimizer was developed in early 2003 by time and resources provided by the Army Research Laboratory (ARL) in conjunction with the HPCMP. This tool provided the core price-per-performance analysis for both the 2004 and 2005 HPCMP technology insertions. Further information can be found in "The 5 Myths of the HPCMP Optimizer" published in this summer's edition of the ARL *Link*.

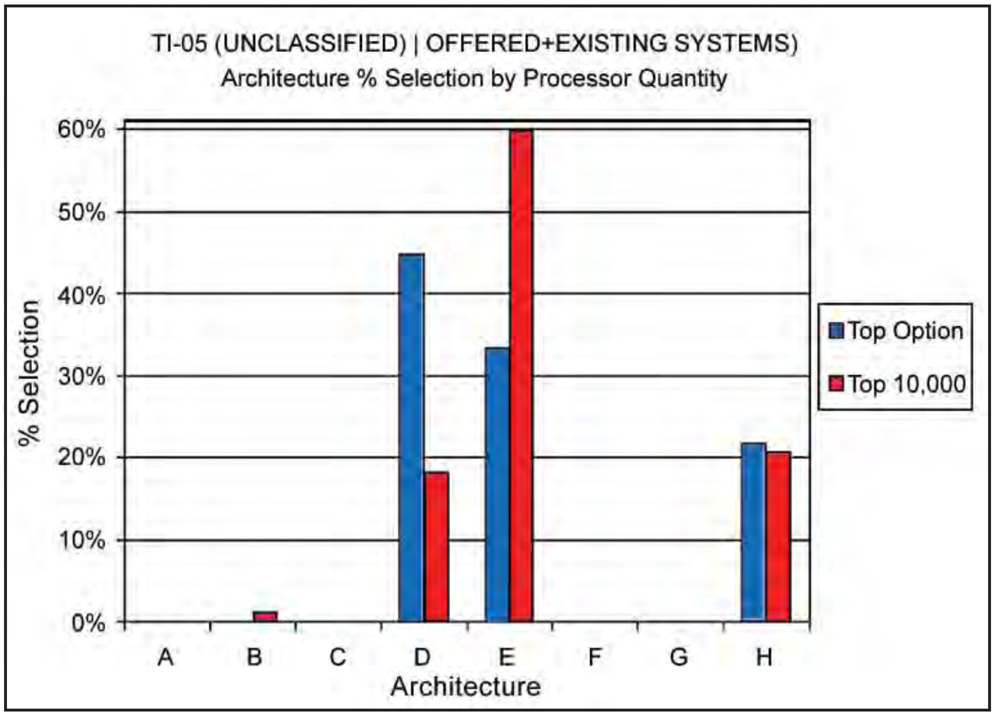


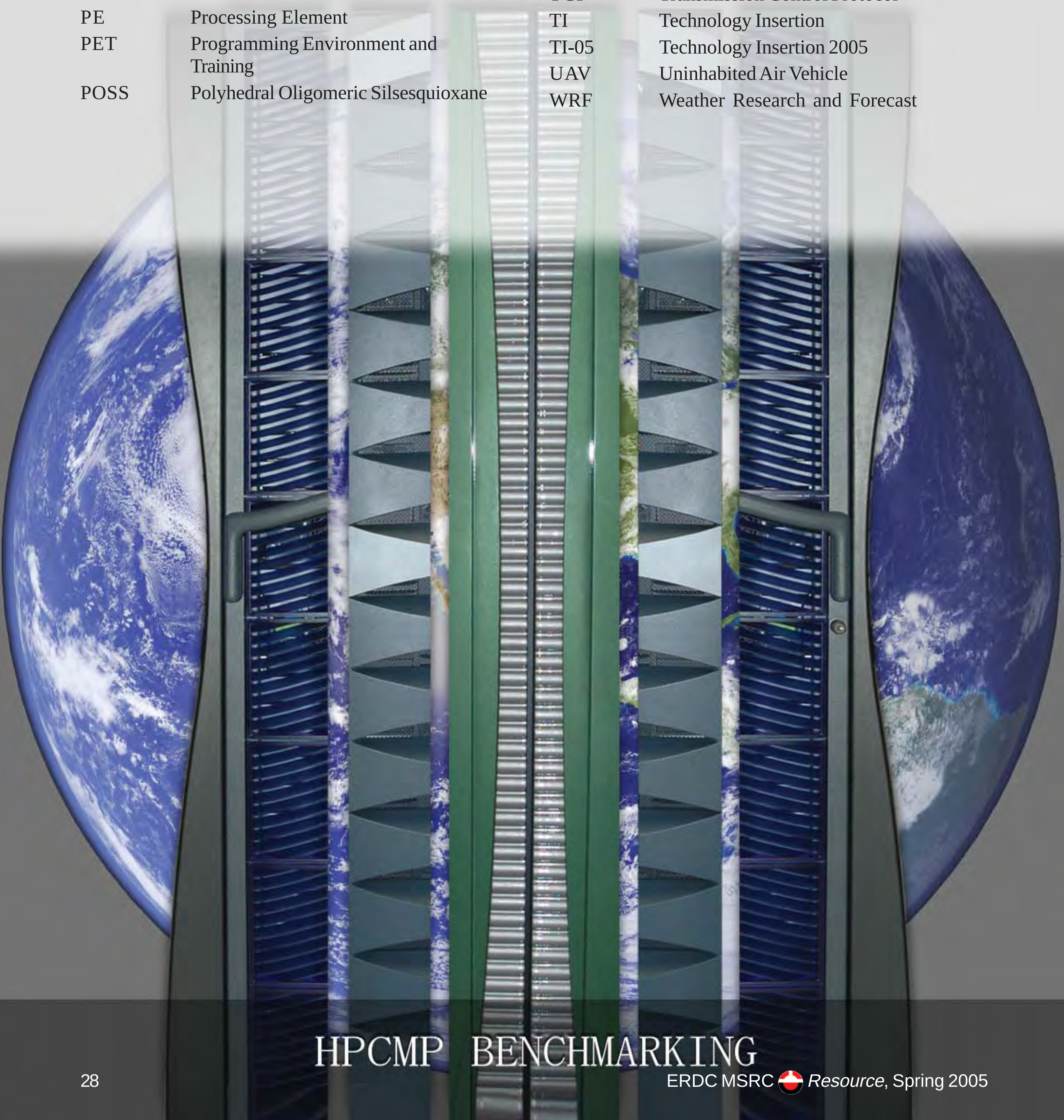
*Figure 1. TI-05 price per performance results*

Below is a list of acronyms commonly used among the DoD HPC community. These acronyms are used throughout the articles in this newsletter.

| | |
|---|---|
| AERO | Aeroelastic CFD/CSM |
| AFRL | Air Force Research Laboratory |
| AFWA | U.S. Air Force Weather Agency |
| AHPCRC | Army High Performance Computing Research Center |
| AMR | Adaptive Mesh Refinement |
| ARL | Army Research Laboratory |
| ARSC | Arctic Region Supercomputing Center |
| ARW | Advanced Research WRF |
| ASC | Aeronautical Systems Center |
| ASCII | American Standard Code for Information Interchange |
| ASF | Advanced Software Framework |
| AVUS | Air Vehicles Unstructured Solver |
| BLAS | Basic Linear Algebra Subprograms |
| BLP | Baseline Performance (1/BLT) |
| BLS | Baseline System |
| BLT | Baseline Time |
| CFD | Computational Fluid Dynamics |
| CHARTD | Computational Hydrodynamics and Radiative Thermal Diffusion |
| CHSSI | Common High Performance Computing Software Support Initiative |
| COAMPS | Coupled Ocean/Atmospheric Mesoscale Prediction System |
| CONUS | Continental United States |
| CPU | Central Processing Unit |
| CS&E | Computational Science and Engineering |
| CSM | Computational Structural Mechanics |
| CSQ | CHARTD Squared |
| CSV | Comma Separated Value |
| CTA | Computational Technology Area |
| CTH | CSQ to the Three-Halves |
| CWO | Climate/Weather/Ocean Modeling and Simulation |
| DARPA | Defense Advanced Research Projects Agency |
| DES | Detached Eddy Simulations |
| DGEMM | Double Precision General Matrix Multiply |
| DoD | Department of Defense |
| DOE | Department of Energy |
| EPS | Encapsulated Postscript |
| ERDC | U.S. Army Engineer Research and Development Center |
| ESMF | Earth System Modeling Framework |
| ESSL | Engineering and Scientific Subroutine Library |
| FFT | Fast Fourier Transform |
| FLOPS | Floating-Point Operations per Second |
| FY 2005 | Fiscal Year 2005 |
| GAMESS | General Atomic and Molecular Electronic Structure System |
| GSA | General Services Administration |
| HPC | High Performance Computing |
| HPCC | High Performance Computing Challenge Benchmark |
| HPCMP | High Performance Computing Modernization Program |
| HPCMPO | HPCMP Office |
| HPCS | High Performance Computing Systems |
| HPL | High Performance LINPACK |
| HYCOM | Hybrid Coordinate Ocean Model |
| ICL | Innovative Computing Laboratory |
| IEEE | Institute of Electrical and Electronics Engineers |
| I/O | Input/Output |
| LSQ | Load Store Queue |
| MHPCC | Maui High Performance Computing Center |
| MICOM | Miami Isopycnic-Coordinate Ocean Model |
| MM5 | Mesoscale Model 5 |
| MPI | Message Passing Interface |
| MSRC | Major Shared Resource Center |
| MUSCL | Monotone Upwind Scheme for Scalar Conservation Laws |
| NASA | National Aeronautics and Space Administration |
| NAVO | Naval Oceanographic Office |
| NCAR | National Center for Atmospheric Research |
| NCEP | National Centers for Environmental Prediction |
| NetCDF | Network Common Data Form |
| NLOM | Naval Research Laboratory Layered Ocean Model |

## acronyms

| | |
|---|---|
| NMM | Nonhydrostatic Mesoscale Model |
| NOAA | National Oceanic and Atmospheric Administration |
| NRL | Naval Research Laboratory |
| NSF | National Science Foundation |
| NWP | Numerical Weather Prediction |
| OOCORE | Out of Core |
| PE | Processing Element |
| PET | Programming Environment and Training |
| POSS | Polyhedral Oligomeric Silsesquioxane |
| PSU | Pennsylvania State University |
| PVP | Parallel Vector Processor |
| RF-CTH | Reduced Functionality CTH |
| SHMEM | SHared MEMory |
| SLIC | Simple Line Interface Construction |
| SUT | System Under Test |
| TCP | Transmission Control Protocol |
| TI | Technology Insertion |
| TI-05 | Technology Insertion 2005 |
| UAV | Uninhabited Air Vehicle |
| WRF | Weather Research and Forecast |

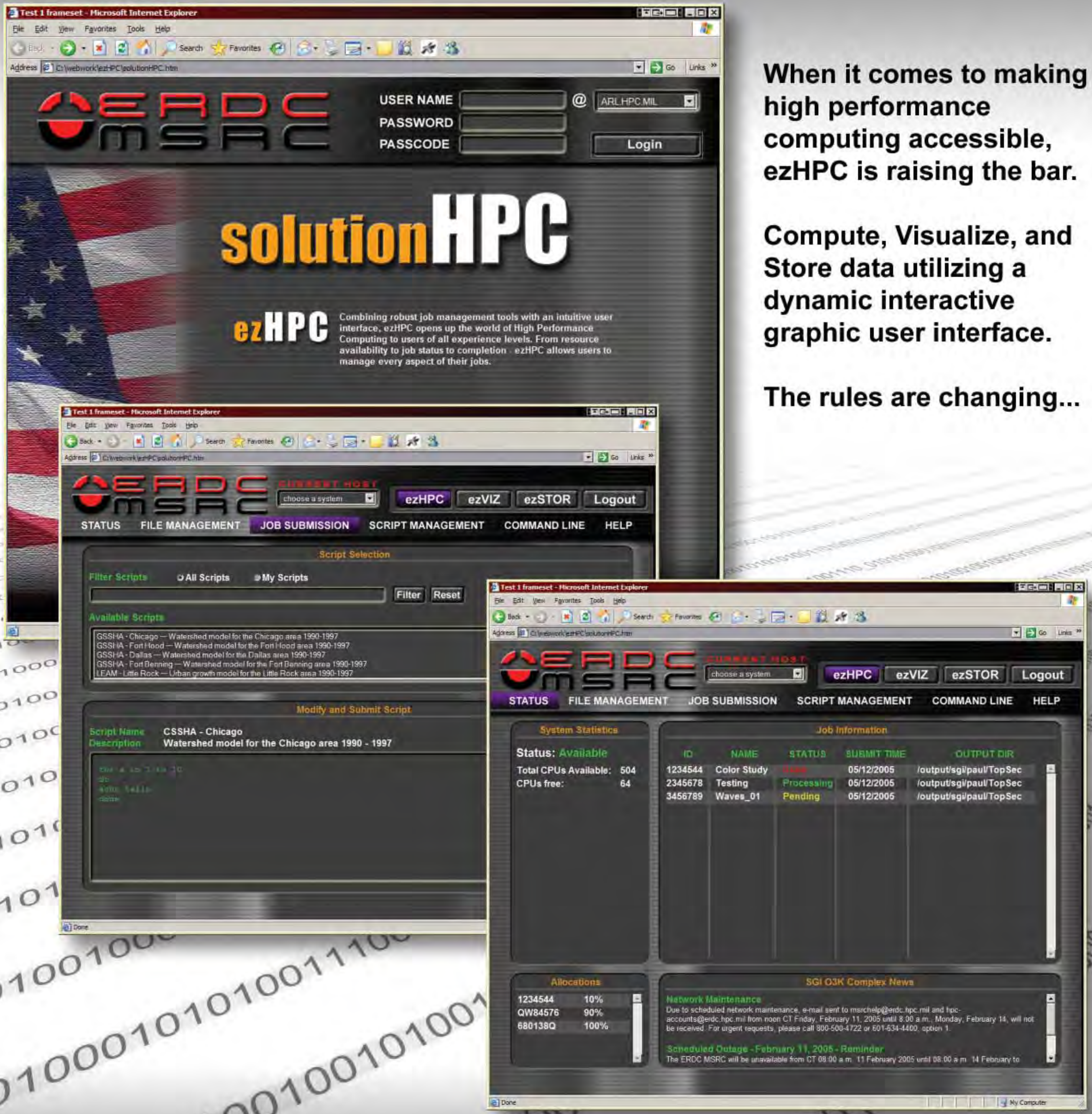


For the latest on training and on-line registration, one can go to the Programming Environment and Training (PET) On-line Knowledge Center Web site:

https://okc.erdc.hpc.mil

Questions and comments may be directed to PET at (601) 634-3131, (601) 634-4024, or PET-Training@erdc.usace.army.mil

# Making High Performance Computing Easy for even the novice user. . .



Combining robust job management tools with an intuitive user interface, ezHPC opens up the world of High Performance Computing to users of all experience levels. From resource availability to job status to completion . ezHPC allows users to manage every aspect of their jobs.

**Script Selection**

Filter Scripts ○ All Scripts ● My Scripts [Filter] [Reset]

Available Scripts

GSSHA - Chicago — Watershed model for the Chicago area 1990-1997
GSSHA - Fort Hood — Watershed model for the Fort Hood area 1990-1997
GSSHA - Dallas — Watershed model for the Dallas area 1990-1997
GSSHA - Fort Benning — Watershed model for the Fort Benning area 1990-1997
LEAM - Little Rock — Urban growth model for the Little Rock area 1990-1997

**Modify and Submit Script**

Script Name CSSHA - Chicago
Description Watershed model for the Chicago area 1990 - 1997

**System Statistics**

Status: Available
Total CPUs Available: 504
CPUs free: 64

**Job Information**

| ID | NAME | STATUS | SUBMIT TIME | OUTPUT DIR |
|---|---|---|---|---|
| 1234544 | Color Study | [illegible] | 05/12/2005 | /output/sgi/paul/TopSec |
| 2345678 | Testing | Processing | 05/12/2005 | /output/sgi/paul/TopSec |
| 3456789 | Waves_01 | Pending | 05/12/2005 | /output/sgi/paul/TopSec |

**Allocations**

| | |
|---|---|
| 1234544 | 10% |
| QW84576 | 90% |
| 680138Q | 100% |

**SGI O3K Complex News**

Network Maintenance
Due to scheduled network maintenance, e-mail sent to msrchelp@erdc.hpc.mil and hpc-accounts@erdc.hpc.mil from noon CT Friday, February 11, 2005 until 8:00 a.m., Monday, February 14, will not be received. For urgent requests, please call 800-500-4722 or 601-634-4400, option 1.

Scheduled Outage - February 11, 2005 - Reminder
The ERDC MSRC will be unavailable from CT 08:00 a.m. 11 February 2005 until 08:00 a.m. 14 February to

**When it comes to making high performance computing accessible, ezHPC is raising the bar.**

**Compute, Visualize, and Store data utilizing a dynamic interactive graphic user interface.**

**The rules are changing...**

**U.S. ARMY ENGINEER RESEARCH AND DEVELOPMENT CENTER INFORMATION TECHNOLOGY LABORATORY**