ERDC/ITL TR-04-4

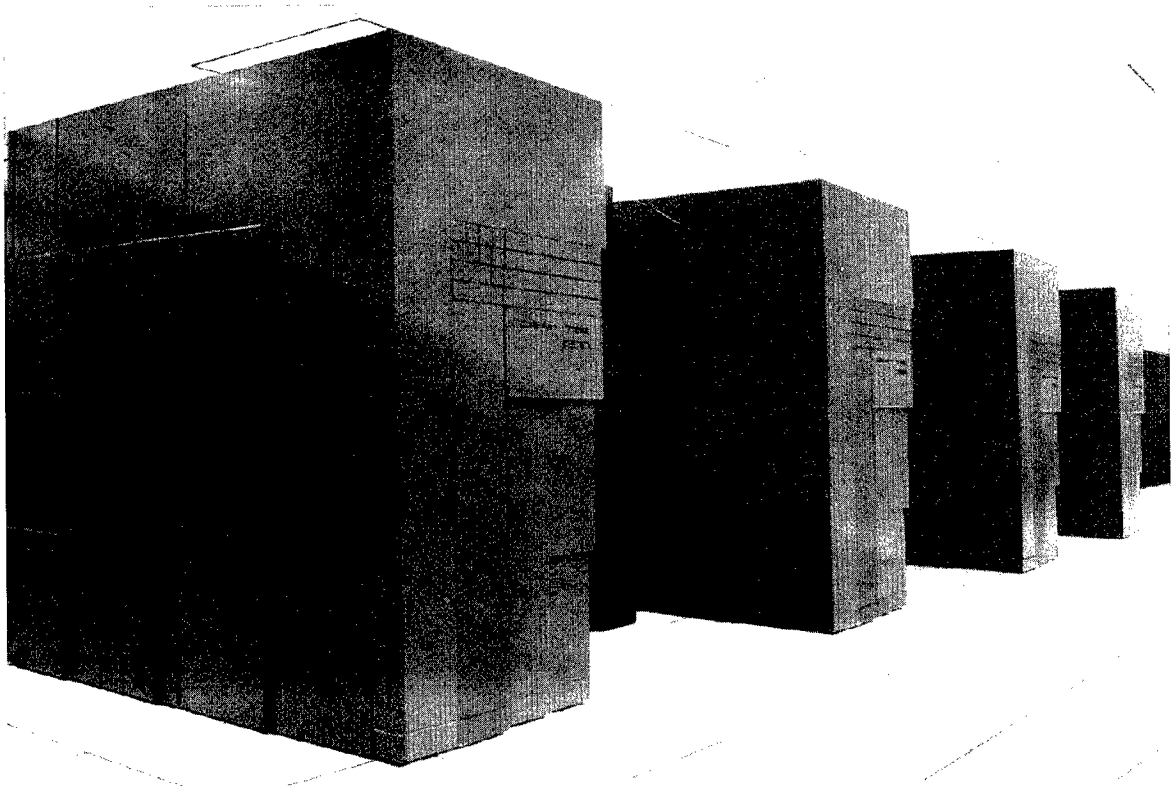
Information Technology Laboratory

US Army Corps
of Engineers®
Engineer Research and
Development Center

Development of the TI-01 Application Benchmark

Robert W. Alter, Daniel Q. Duffy, Mark R. Fahey,
Rebecca A. Fahey, Jeffrey L. Hensley, Thomas C. Oppe,
and William A. Ward, Jr.

August 2004



ERDC/ITL TR-04-4
August 2004

Development of the TI-01 Application Benchmark

Robert W. Alter, Daniel Q. Duffy, Mark R. Fahey, Rebecca A. Fahey,
Jeffrey L. Hensley, and William A. Ward, Jr.

Computer Sciences Corporation
Major Shared Resource Center
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Thomas C. Oppe

Data Management Consultants
Major Shared Resource Center
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Prepared for High Performance Computing Modernization Program Office
1010 North Glebe Road, Arlington, VA 22201

Under Contract No. DAHC 94-96-C0002
Millennia Contract GS00T99ALD0203

Monitored by Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road, Vicksburg, MS 39180-6199

ABSTRACT:

This report documents the design, construction, and validation of the Technology Insertion 2001 application benchmark test package for the Department of Defense (DoD) High Performance Computing Modernization Program Office (HPCMPO). This test suite contained 12 application programs chosen to be representative of the DoD's HPC workload. The report justifies this claim of representativeness. On a given HPC system, each application program was typically tested using at least two sets of input data, and each program/input data test case was always tested at multiple numbers of processors. These tests were done in dedicated mode with no other applications on the system, so as to obtain a presumably "best possible'" performance. This collection of tests was repeated on five different HPC systems: a Cray T3E, an IBM POWER2-based SP, an IBM POWER3-based SP, an SGI Origin 2000, and an SGI Origin 3800. Additionally, two throughput tests constructed to represent HPCMPO workloads were conducted. Again, the methodology used to obtain representativeness is described.

Contents

List of Figures

List of Tables

Preface

The production of this report was sponsored by the Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) and funded through the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC), Information Technology Laboratory (ITL), under Contract Number DAHC 94-96-C0002 and Millennia Contract GS00T99ALD0203, both with Computer Sciences Corporation (CSC). The first contract was managed by Mr. Douglas D. Walker, former Program Manager, CSC, and was monitored by Mr. Bradley M. Comes, former Director, ERDC MSRC, while the second contract was managed by Mr. John D. Mauldin, Task Area Two Manager, CSC, and was monitored by Mr. John E. West, Director, ERDC MSRC. At the time of publication of this report, Mr. Cray Henry was Director of the HPCMP and Dr. Jeffery P. Holland was Director of ITL.

This report was prepared by the following members of the Computational Science and Engineering Group (CS&E) of the ERDC MSRC, formerly known as the Computational Migration Group: Mr. Robert W. Alter, Dr. Daniel Q. Duffy, Dr. Mark R. Fahey, Ms. Rebecca A. Fahey, Dr. Jeffrey L. Hensley, and Dr. William A. Ward, Jr., CSC, ERDC MSRC, and Dr. Thomas C. Oppe, Data Management Consultants, ERDC MSRC; these members also assembled the software, performed the tests, and gathered the data on which this report is based.

The following scientists, by means of personal communications, provided technical information and advice essential to the study: 1LT Joseph D. Blahovec, Jr., Drs. Richard Linderman and Michael White, and Mr. Jerry A. Boatz, Air Force Research Laboratory (AFRL); Dr. Margaret Hurley and Messrs. Tom Crimmins, Harris L. Edge, Kent D. Kimsey, and Steve Schraml, Army Research Laboratory (ARL); Dr. Ronald L. Hinrichsen, Aeronautical Systems Center (ASC); Dr. Joseph Werne, Colorado Research Associates; Mr. James M. Brock, Jr., Eglin Air Force Base (AFB); Drs. Steve Akers, Byron Armstrong, Tommy Bevins, Kent Danielson, Zeki Demirbilek, Jeffery P. Holland, and Fred T. Tracy, ERDC; Dr. Suresh Menon, Georgia Insitute of Technology; Dr. Mark S. Gordon, Iowa State University; Messrs. Wilbur P. Brown and Timothy J. Madden, Kirtland AFB; Mr. Charles A. Rendleman, Lawrence Berkeley National Laboratory; Dr. Brian D. Goble, Lockheed Martin Aeronautics Company; Mr. Alan Lampson, Logicon; Drs. Ramesh Pankajakshan and Nathan Prewitt, Mississippi State

University; Dr. Alan J. Wallcraft, Naval Oceanographic Office (NAVO); Drs. Noam Bernstein and George Heburn and Messrs. Gopal Patnaik and Theodore Young, Naval Research Laboratory (NRL); Dr. Tom Rosmond, NRL Monterey; Mr. Robert Robins, Northwest Research Associates; Dr. Roy Benedek, Northwestern University; Dr. Charles W. Manry, Space and Naval Warfare (SPAWAR) System Center; Dr. Joseph J. Gorski, Naval Surface Warfare Center (NSWC); Dr. Stephen Wornom, Ohio State University; Mr. Gerald H. Lushington, Ohio Supercomputing Center; Dr. Howard Gibeling, Pennsylvania State University; Drs. Greg Sjaardema and Paul Taylor and Ms. Sharon Healy, Sandia National Laboratory (SNL); Dr. C.J. Suchyta, Silicon Graphics, Inc. (SGI); Mr. Kurt Glaesemann, University of Utah; and Dr. Christopher F. Woodward and Mr. Kenneth E. Wurtzler, Wright Patterson AFB.

The project reported here could not have been completed without assistance from the ERDC MSRC staff. System administrators Messrs. Marty Bullock and George Moncrief, SGI, and Messrs. Tim Dunaway, Jim Moody, and Paul Szepietowski, IBM, labored tirelessly to provide dedicated time on ERDC HPC systems. Messrs. Jay Cliburn, Mike Gough, and Owen LaGarde, CSC, provided usage data as well as user names and e-mail addresses. Messrs. Paul Adams and J.D. Becker, CSC, gave invaluable advice on specific applications, particularly **CTH**. Other individuals who made valuable contributions were Mmes. Robin Phillips and Jeanie McDonald and Messrs. David Longmire, Jay Sykes, and John Eberle, CSC, and Ms. Dean Hampton, ERDC. A special acknowledgment is also due to the system staff at all four MSRCs for providing usage data and to the system staffs at the ASC, ERDC, and NAVO MSRCs for dedicated time on their HPC systems.

The HPCMP Benchmark Working Group (BWG) provided technical advice and guidance throughout this project. This group included Mr. Terry Blanchard, BWG Chairman, NAVO; Dr. Tim Campbell, Logicon; Mr. Tom Crimmins, ARL; Dr. Larry Davis, HPCMP Deputy Director; Mr. Robb Graham, Instrumental; Mr. Henry Newman, Instrumental; Mr. David Potts, ASC; Mr. Steve Thompson, Raytheon, and Dr. William A. Ward, Jr., CSC. The authors particularly note that the success of this project is due in large part to the administrative and technical support of Mr. Blanchard, and to his professional and collegial leadership of the BWG.

COL James R. Rowan, EN, was Commander and Executive Director of ERDC. Dr. James R. Houston was Director.

Summary

The Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP), like any other large corporate user of computer systems, periodically requires a series of benchmark tests to evaluate the performance of proposed competing systems. Previous efforts in this area were not firmly grounded in HPCMP system usage data and were not formally documented. This test package, designed to aid decision makers in the HPCMP Technology Insertion 2001 (TI-01) competitive system procurement, aims to remedy those shortcomings and provide a starting point for future DoD HPC benchmark activities.

The quality of any benchmark test package depends on the presence of several important characteristics. These characteristics serve as goals for the test package constructor and guide the constructor in the selection of test package components (e.g., synthetic tests and application codes). The first and most important characteristic is that the benchmark must be *representative* of both current and projected system workloads. The types, patterns, and rates of computation, communication, and input/output of programs in the test package must match those of programs actually in use to as great a degree as possible. Furthermore, the programs in the test package must be imposed on the system under test (SUT) in a manner similar to that in practice. Second, demonstrating that these conditions have been met for a range of operating environments and problem sizes establishes that the benchmark is *valid*. Third, and this is particularly important for any HPC benchmark, the test package must be *scalable*; it must be possible to vary the number of processors to be used and the size of the test problem to be solved. Fourth, the test package must be *maintainable*; the size of the test package must be kept to a minimum, and the test package must be constructed in a modular, easy-to-modify fashion. Finally, the combination of the above four features, representativeness, validity, scalability, and maintainability, produces test package *durability*.

Historically, benchmark test packages have contained codes chosen from one or more of the following types: (a) relatively short *synthetic programs*, such as Whetstone and Streams; (b) *toy benchmarks*, such as Quicksort and Prime Sieve; (c) widely used off-the-shelf codes, or *package kernels*, such as Linpack and ScaLAPACK; (d) *application kernels*, that is, sections of code, extracted from

actual application programs, that perform a significant fraction of work; and (e) *complete applications*. Originally, this test package was to have emphasized application kernels in order to avoid problems associated with export control and code portability. However, to provide a timely response to the HPCMP, this test package consists of complete applications supplemented by synthetic programs. These latter test components were prepared by Instrumental, Inc., and are documented elsewhere.

To help ensure the representativeness of the test package, a two-part survey of the DoD HPC community was conducted. In part one, scientists were asked to identify (a) the programmatic interface used to implement parallelism in their codes (e.g., Message Passing Interface (MPI) and Parallel Virtual Machine (**PVM**)); (b) the most important types of computations (e.g., flux calculations, linear equation solution); (c) which of computation, communication, or input/output was dominant; (d) any standard benchmarks or kernels that might already well represent work in their Computational Technology Area (CTA); and (e) the application codes representative of their CTA. Part two of the survey was tailored to each respondent's part one reply; it asked additional questions regarding the precise nature of the computations each scientist typically performed. Generally, the responses indicated that fast Fourier transforms, flux calculations, and various types of linear equation solution methods were common computations.

The surveys provided qualitative data on Major Shared Resource Center (MSRC) workloads; quantitative information came in the form of utilization data from all four HPCMP MSRCs. These data provided a breakdown of MSRC system usage by CTA and by number of central processing units (CPUs) used per job. This information was used to prepare an aggregate profile of the MSRC workloads, and jobs were selected to model that profile.

The selected codes were used in two different ways. First, the test package specified dedicated tests; these were jobs run on an otherwise empty system designed to measure peak application performance. Second, two throughput tests were included to measure system performance on simulated production workloads. Timing data for the dedicated tests were obtained on five Government-owned systems. Performance metrics relative to one of these systems, a Silicon Graphics Origin 3400, were calculated for the other four to better understand system behavior and to detect performance trends.

This test package should be viewed as a starting point for a more modular, and portable test package. It must be revised on a yearly basis so that it will remain up-to-date. Furthermore, a process must be instituted to periodically demonstrate the correlation between the test package workload and actual MSRC workloads. A graphical user interface should be added to relieve the tester of many of the installation and utilization tasks currently performed from the command line. Finally, the test package itself should be the object of further development. The development process should conform to generally accepted software engineering practices, and the package should be transformed into a programming systems product as opposed to merely a collection of programs.

1 Introduction

Benchmarking may be defined as a means of estimating the performance of systems by imposing one or more test workloads on them and then measuring their performance (Jones 1975). Within this definition there is considerable latitude for the application of benchmarking; CPUs, memory subsystems, input/output (I/O) subsystems, graphics subsystems, disk subsystems, compilers, operating systems, entire computer systems, multiprocessor computer systems, and local area networks all may be, and have been, benchmarked. Although there are other techniques that may be used to evaluate the performance of computer systems (e.g., analytical modeling and simulation), benchmarking is generally recognized as the most accurate, and its importance, particularly in the context of computer systems procurement, has long been recognized (Comptroller General of the United States 1982, Letmanyi 1984, National Bureau of Standards 1977, National Bureau of Standards 1980).

Overview of Benchmarking Goals

A benchmark test workload may be constructed using a variety of candidate test package components, and that workload may be imposed on the system under test (SUT) in a number of different ways. Therefore, benchmark test developers need a set of design goals to serve as an evaluation framework for their product; such a candidate set is presented in the following paragraphs. Like any realistic set of goals, these are inherently conflicting; thus, developers must strike an appropriate compromise in the degree to which each goal is met.

Benchmarking is used in three broad contexts: system selection, system improvement (tuning), and system design (Ferrari 1972). Since the source of these goals is a General Services Administration handbook (Federal Computer Performance Evaluation and Simulation Center 1979, Chapter 3), the goals are naturally oriented toward systems selection. Finally, it is important to realize that these goals cannot be addressed independently. Achievement of one may prohibit or facilitate the achievement of another.

Maximize benchmark representativeness

There are several aspects to maximizing benchmark representativeness. First, the test workload must be an accurate model of the projected actual workload. Use of programs still in the developmental stage may also be useful as a way to reflect future processing requirements in the benchmark test (Dongarra, Martin, and Worlton 1987). A second aspect of representativeness depends on the system configured to run the test; it should match the proposed configuration to the greatest extent possible. However, proposed systems with hundreds of attached terminals or networked nodes may be impossible to duplicate. A third aspect of this objective is the benchmarking methodology. If one is purchasing a large-scale system for scientific computing that will support a large number of users, then running one or two test jobs, or even a few actual programs, would scarcely approximate the actual operating environment. On the other hand, it is obviously not possible to incorporate all programs currently in use at the DoD MSRCs[1] into a benchmark test package. Finally, it is important to realize that lack of representativeness does not necessarily imply lack of effectiveness in measuring system performance.

Maximize benchmark believability/validity

If a benchmark is truly representative of DoD applications, then it becomes a valid test. As with any experiment (and a benchmark test of a computer is very much an experiment), the method of taking data must be accepted before the results are accepted. If the experimental method is, or is perceived to be, flawed, the results will be dismissed as meaningless. For example, if a test spends 90 percent of its time doing interprocess communication and the existing production system spends 90 percent of its time doing computation, then the test is not representative. Hence, by ensuring that the benchmark test is representative and that the benchmark guidelines are carefully followed, the results obtained become believable. Furthermore, it is equally important for the test applications to span the full spectrum of types of DoD HPC applications. Since no single MSRC, or machine for that matter, is devoted to a single computational technology area (CTA),[2] input must be gathered from all MSRCs and as many CTAs as possible.

[1] The HPCMP supports four MSRCs to provide DoD scientists access to the latest HPC hardware and software. These centers are the Army Research Laboratory (ARL) MSRC at Aberdeen Proving Ground, MD, the Aeronautical System Center (ASC) MSRC at Wright-Patterson Air Force Base, OH, the U.S. Army Engineer Research and Development Center (ERDC) MSRC at Vicksburg, MS, and the Naval Oceanographic Office (NAVO) MSRC at Stennis Space Center, MS. See High Performance Computing Modernization Program (2004) for more information on the MSRCs.

[2] The HPCMP CTAs are Computational Chemistry and Materials Science (CCM), Computational Electromagnetics and Acoustics (CEA), Computational Electronics and Nanoelectronics (CEN), Computational Fluid Dynamics (CFD), Computational Structural Mechanics (CSM), Climate/Weather/Ocean Modeling and Simulation (CWO), Environmental Quality Modeling and Simulation (EQM), Forces Modeling and Simulation/C4I (FMS), Integrated Modeling and Test Environments (IMT), and Signal/Image Processing (SIP). This CTA taxonomy provides the structure for the HPCMP's Common High Performance Computing Software Support Initiative (CHSSI), the intent of which is to provide "efficient, scalable, portable software codes, algorithms, tools, models, and simulations that run on a variety of HPC platforms and are needed by a large number of science and technology (S&T) and test and evaluation (T&E) scientists and engineers" (High Performance Computing Modernization Program 2004).

Only when representative samples of DoD applications are included within the benchmark, does it become valid.

Maximize benchmark scalability

Development of large-scale parallel systems has created the need for a new benchmark characteristic: scalability. If the problem size is fixed, then even so-called "embarrassingly parallel" codes that do no communication until an answer is obtained will eventually cease to benefit from the use of additional processors. Often, system performance will plateau simply due to having a fixed problem size that is too small to exploit additional processors. Sometimes, increasing the numbers of CPUs beyond some threshold stops improving a code's performance because of the way the code itself is written, but more often this "performance plateau" is due to having a fixed problem size that is too small to exploit additional processors. Consequently, as the number of processors in commercially available systems grows, the test package must be revised to include larger test problems that will exercise systems over a broad range of processor counts. Specifically, the codes in the test package should be scalable in two ways: (a) by increasing the number of processors used by such a code and (b) by increasing the size of the test problem solved by that code. Finally, the test package should be constructed so as to minimize the effort required of the test package user to perform a series of tests covering a range of processor counts and problem sizes.

Maximize benchmark durability

The durability of a benchmark is dependent on how it is perceived by the HPC community. The elegance and simplicity of the LINPACK benchmark (Dongarra 2004), for example, have made it one of the most durable HPC benchmark tests since its creation in the late 1970s. It is currently used in many acceptance tests run on current DoD machines, as well as being used to construct the "Top 500" list (National Energy Research Scientific Computing Center / Lawrence Berkeley National Laboratory 2003). Even though no DoD benchmarking test suite will be as elegant and simple as LINPACK, it will still need to exhibit LINPACK's ease of use, generality, portability, understandability, maintainability, and adaptability. In short, it should be a "programming systems product" and not just a collection of stand-alone programs (Brooks 1975, pp 4-6).

Minimize benchmark discrepancies

Every benchmark has its own "rules of the game" that specify how the tests are to be conducted. In this context, discrepancies are technical or procedural differences between these benchmark rules and the manner in which the vendor actually conducts the test. These differences may result from unintentional errors, deliberate misrepresentations on the part of the vendor, inadequate benchmark documentation provided by the test package developer, or system faults. If discrepancies are found and it is determined that their impact on quality of system sizing, on representativeness, or on some other goal is too severe, the evaluator may elect to invalidate the test.

Maximize benchmark uniformity

Benchmark uniformity implies that the benchmark test imposes the same workload on each system. This provides for a fair evaluation of vendors, or a fair evaluation of processor architectures, depending on the context. The intent here is to ensure a level playing field for each participant. Complete uniformity is impossible to attain, and an extremely high uniformity requirement may actually decrease representativeness and increase cost. For example, if vendors are required to use the same linear equation solver in a test, instead of an equivalent one tailored to their processor architecture, then the result may be lower observed performance that does not reflect the way the system will ultimately be used. In this instance, a generic specification of the problem to be solved, instead of a particular solver, may preserve both uniformity and representativeness. In some cases, uniformity may be deliberately violated. To detect benchmark discrepancies and prevent vendors from making special system modifications tailored to a particular benchmark job mix, benchmark test monitors sometimes make changes in file contents and order of job submission just prior to a test. Such changes should not, however, significantly alter the workload demands imposed on the system.

Maximize benchmark repeatability

"Repeatability" implies that the same benchmark test performed on the same system two different times will produce identical results. Because the results of unrepeatable experiments are inherently suspect, it is imperative that benchmark implementors pay particular attention to this issue. Complete repeatability is not possible in complex systems because there are many factors, some beyond the control of the test designer, affecting this goal. For example, differences in the order in which pages are loaded into memory may result in different paging or caching behavior. Furthermore, repeatability is not the same as uniformity, although they are interrelated and factors that affect the former also affect the latter. For example, random number generators are used to drive certain aspects of a benchmark test, and this may prevent both repeatability and uniformity. However, the differences produced by such factors should not be significant. Finally, as discussed in the following section, a lack of repeatability may have an adverse effect on quality of system sizing.

Maximize quality of system sizing

System sizing may be defined as "the process of determining a configuration of hardware and software components that can accomplish a specific set of workload demands at a required level of performance" (Federal Computer Performance Evaluation and Simulation Center 1979, paragraph 3.4.a). Other performance evaluation alternatives, such as analytic modeling or discrete event simulation, may be used to reach this goal. However, benchmarking typically exceeds all of them, not only in its ability to accurately size a target system, but in representativeness as well. Unfortunately, it also typically exceeds all of them in cost. Nevertheless, the possibility of obtaining an undersized system that cannot handle the projected workload, or an oversized system that is too expensive, often

compels agencies to require benchmarking. It is then incumbent upon the agency to specify a test workload that accurately models the projected actual workload so that vendors may propose correctly sized systems.

Minimize time and cost of acquisition

Attempts to make a benchmark test more thorough and representative generally make it larger and more complex. As benchmark size and complexity increase, additional time and money must be spent by the user to describe, implement, and validate a benchmark, to document benchmark procedures for vendors, to answer vendor questions, and to analyze the benchmark results. Therefore, the benchmark must generally strike a compromise between this goal and that of representativeness. Furthermore, complexity and lack of portability increase vendor costs. Vendors will respond by passing the costs on to the user or by declining to participate in the bid process, thus reducing competition. Maintaining a high level of competition results in lower system costs and in innovative solutions to problems.

Test Components and Methodology

To reach the above goals, appropriate test package components must be selected. Candidate components include synthetic tests, generic tests, application kernel tests, and complete codes, each of which supports the benchmarking goals to some degree, and has its own advantages and disadvantages.

Synthetic benchmarks

Among the least representative benchmarks is the synthetic job, that is, "a program which uses precisely specified amounts of computing resources, but which does no 'useful work' " (Kernighan and Hamilton 1973). Such jobs may be constructed to replicate the CPU and I/O requirements of a real job without regard to actual program features, or they may be constructed to represent a typical program written in a high-level language. If this latter approach is used, one first obtains statistics that quantify how often particular language features (e.g., statement types, data types, types of operators) occur. Ideally, these should be dynamic frequencies taken from program execution traces; when these are not available, static information obtained by examining source programs may be substituted. The data so obtained are then used to construct a program that exhibits the same frequencies of occurrence for all features of interest. Such a program may accept an input parameter to vary its resource utilization, or the program may be invoked within a loop to obtain a measurable amount of work. The results may then be expressed in instructions executed per second or, if the program is short enough, as program executions per second. Examples include the Buchholz benchmark (Buchholz 1969), the Whetstone benchmark (Curnow and Wichman 1976), and the Dhrystone benchmark (Weicker 1984, 1988). What these synthetic tests lack in representativeness, they often make up for in scalability and durability. They are generally quite portable and easy to use and provide a close measure of a system's peak performance.

Generic tests

The classic example of this category, LINPACK, has already been mentioned. For these types of benchmarks, usually a single problem is solved, such as a matrix-vector or matrix-matrix multiplication. These tests perform types of calculations similar to those found in many actual applications and may even use publicly available codes or vendor-specific libraries. In many cases, the number of floating-point operations or I/O operations is exactly known, and the rate of computation or memory transfer can be measured. However, one must be careful when quoting results from these types of tests, since, like synthetic tests, the correlation to the performance of actual application codes may be small. (It is rare to find an application with the same Mflop rate as LINPACK.)

Program kernels

A third methodology is the use of program kernels. To apply this technique effectively, heavily used programs from the actual workload are analyzed to determine what portions of the code use most of the computing time. These resource-intensive portions, or kernels, are extracted and packaged into a benchmark test. Early examples included the Livermore Fortran Kernels, commonly referred to as the "Livermore Loops" (McMahon 1986), and the National Aerospace Simulation (NAS) Kernel Benchmark Program (Bailey and Barton 1985). More recently, the NAS Parallel Benchmarks (Bailey et al. 1991, 1994, 1995, Saphir et al. 1996) have been developed to evaluate the performance of parallel systems. These two benchmarks are examples of what will here be termed *application kernels* because they contain code extracted from, or highly representative of, actual application programs. It is also possible to use *package kernels*, "off-the-shelf" subroutines from software packages such as LINPACK, LAPACK, ScaLAPACK, and FFTPACK, that form the computational core of many user applications. The definition of a kernel is inherently vague, although it intuitively seems to imply a small amount of code; as the amount of representative code extracted becomes larger, the kernel may be more appropriately termed a "mini-application." Although kernel benchmarks may be small and portable, their major disadvantages include the possibly significant work required to identify and extract them from a large application code and the problem of verifying that the kernel actually models the performance of the original code.

Actual programs

A final possible test component is an actual application program. Several currently popular benchmarks use this methodology, including the PERFECT (Cybenko 1990, Grassl and Schwarzmeier 1990, Saavedra-Barrera 1990) and SLALOM benchmarks (Gustafson et al. 1990). A third example is the SPEC benchmark suite (Henning 2000, Saavedra-Barrera 1990). The Systems Performance Evaluation Cooperative (SPEC) consists of a group of vendors who have agreed to use a standard test set to evaluate their systems. This set currently includes 26 different actual programs, ranging from a C compiler to a seismic wave propagation simulator, which exercise both the integer and floating-point capabilities of a system. The SPEC suite is designed to test the performance of

the CPU, memory hierarchy, and compiler. A performance ratio of the time on the SUT relative to the known execution time on a reference machine (originally a DEC VAX-11/780, currently a 300-MHz Sun Ultra-5/10) is calculated for each of the 26 programs. The geometric mean of these ratios is the performance rating (the "SPECmark") assigned to the SUT. The use of actual programs in this fashion suffers from many of the same problems as the kernel approach, including lack of representativeness due to a fixed workload, and possible lack of uniformity due to vendor modifications. However, these disadvantages are often outweighed by the perception that these are "real codes." Recently, SPEC has introduced a new test suite, SPEC HPC2002, specifically designed to measure the performance of HPC systems.

Methods for Running a Benchmark Test

The factor that generally serves to discriminate between benchmark techniques is the method each one uses to achieve representativeness, not only of the composition of the workload (as noted above), but also of the manner in which it is imposed on the SUT. Are the test programs to be run serially or simultaneously? If serially, then will the SUT be dedicated to the test or will it be loaded with other jobs? The answer to this last question depends on what is being measured. The dedicated serial test is intended to measure the best possible performance of a given program on a system, while a serial test under typical load should measure typical program performance.

If a stream of jobs is to be imposed on the SUT to measure system throughput, then a decision regarding use of internal or external test drivers must be made. This last decision is particularly crucial when a test package includes interactive jobs; in such cases, remote terminal emulation is often used to externally drive the SUT. Fortunately, that was not the case for these tests; instead, this benchmark test package contains dedicated synthetic tests, dedicated application tests, and internally driven throughput tests.

Terminology and Conventions Used in This Report

A report on subject matter such as this inevitably uses a variety of technical jargon and acronyms; some of these are defined in Appendix A. The following conventions[1] are used in this report:

Bold is used for statements and functions, identifiers, and program names.

Italic is used for file and directory names when they appear in the body of a paragraph as well as for names of test cases that consist of

[1] These conventions are reprinted with permission from *sed & awk*, 2nd ed. © 1997 O'Reilly Associates, Inc (Dougherty and Robbins 1997). For orders and information call 800-998-9938.

	multiple input files. It is also used for data types, for titles of books and journals, and for emphasizing new terms and concepts when they are introduced.
`Constant Width`	is used in examples to show the contents of files or the output from commands.
`Constant Bold`	is used in examples to show command lines and options that should be typed literally by the user.
"..."	are used to identify a code fragment in explanatory text. System messages, signs, symbols, and quotations from other sources are quoted as well.
[...]	surrounds optional elements in a description of program syntax. (The brackets themselves should never be typed, unless otherwise noted.)
. . .	stands for text (usually computer output) that has been omitted for clarity or to save space.

2 Design of the Test Package

Consideration of the factors discussed in the first chapter, as well as guidance from the HPCMP Office (HPCMPO), led the benchmark team to construct a three-part test package. Synthetic tests were included to measure peak system capabilities in a variety of areas: floating-point computation, memory transfer rate, and I/O performance, among others. These tests were constructed by Instrumental and are described elsewhere (Newman and Graham 2001). The second test component consisted of application codes executed on an otherwise idle system; these dedicated tests were intended to measure peak application performance. The final test component was made up of two throughput tests intended to represent production mode on a system. Other types of test components were considered, and some, particularly kernels, had technical merit. However, given the time constraints imposed on the team and the amount of money involved in the TI-01 procurement, use of the above three components seemed to allow timely construction of the test package and accurate sizing the prospective systems.

Perhaps the most crucial part of implementing the application portion of the test package was the selection of the application codes. The initial step in this process was a survey of users at all four DoD MSRCs. This was followed by a second, more detailed survey designed to provide deeper insight into the codes actually running on DoD HPC systems. Finally, a quantitative study of utilization data from all the MSRCs provided valuable information to the creation of the benchmark. A description of these data and their influence on the test package is provided in the following sections.

User Surveys

Credibility with the HPC community is an important feature of any HPC benchmark test package. Two surveys of DoD HPC users were conducted by e-mail to obtain user input for the design of the benchmark test package. This first survey instrument is shown in Figure 1. Survey recipients included leaders of all 10 DoD CTAs, onsite CTA leaders at the MSRCs, all Challenge Project principal investigators, as well as numerous other MSRC users. One hundred eighty-seven

Dear colleague:

At the request of the HPCMO, the Computational Migration Group (CMG) is constructing a scalable benchmark package. In order to make this package representative of your CTA, we urgently need your input. This is important because this package may be used for purposes of future machine sizing and procurement. Please help us out by answering a few questions. Each BT of a system will have two parts:

1. Generic performance tests designed to confirm vendor claims regarding peak computational, communication, and I/O performance, and
2. A series of tests using kernels representing each CTA. Ideally, each kernel selected should be parallelized to run efficiently on all DoD parallel systems.

We need advice from you to ensure that the results of these BTs are useful both to you and the HPCMO. We would be appreciative if you would address the following items:

1. What is the computational technology area (CTA) of your project?
2. Do most of the codes in your CTA use MPI, OpenMP, SHMEM, or some other programming interface (specify) to implement parallelism?
3. With respect to CPU time consumed, what types of computations characterize the codes in your CTA (e.g., dense/sparse linear system solution, gridding, flux calculations)?
4. With respect to overall time consumed, how do programs in your CTA spend their time: computation, communication, or I/O?
5. Are there any benchmark codes representative of your CTA that would be applicable for purposes of system evaluation and/or selection (specify)?
6. Suggest one or more parallelized computational kernels that represent your CTA; ideally, these should use MPI as the parallel programming interface and should run on as many of the above systems as possible.
7. Suggest one or more parallelized codes that represent your CTA; ideally, these should use MPI as the parallel programming interface and run on as many of the above systems as possible.
8. Is there a preferred platform among users in your CTA (specify)?
9. Is system performance, ease of use, or some other criterion (specify) the overriding factor in selecting a machine on which to run a code?
10. If appropriate, suggest the names of other engineers/scientists who might provide additional insight into these issues. Please provide their e-mail addresses if possible.
11. Make any other comments or suggestions you feel are appropriate to this effort.

Thank you for your help in making this effort a success.

Figure 1. Phase 1 user survey example

surveys were distributed. The responses of the 32 users who replied are summarized below; multiple responses to some questions cause some response totals to exceed 32.

a. *CTA distribution.* CCM: 7; CEA: 2; CEN: 1; CFD: 14; CSM: 3; CWO: 4; EQM: 2; SIP: 1.

b. *Parallel interface.* MPI: 27; OpenMP: 6; SHMEM: 6; other: 5; PVM: 2; threads: 2. Fourteen users mentioned only MPI, and two noted dual-level MPI/OpenMP usage.

c. *System preference.* SGI Origin: 15; Cray T3E: 12; IBM SP: 9; Sun: 3; Compaq: 1. No vector systems were mentioned.

d. *Type of computation.* Numerical linear algebra: 22; flux calculations: 10; FFTs: 5; quadrature: 3; other: 3.

e. *Component with significant time.* The majority believed that computation time dominated communication time and I/O time; communication-bound programs either were not used or had been rewritten to solve the problem. Only six responses indicated that I/O was a significant part of their usage.

The users responding to this first survey were sent a second, more detailed survey. Some of the questions in these second-round surveys were tailored to the round one survey responses, resulting in differences between individual survey instruments. An example round two survey instrument is shown in Figure 2. Twenty users responded to this second survey; several important trends were evident from their responses.

a. *Data type.* Double: 13; real: 2; double complex: 2; complex: 1.

b. *Source of software.* Used own solvers, FFTs, etc.: 10; used off-the-shelf software (e.g.; LAPACK): 7.

c. *Dimensionality of problem.* three-dimensional (3-D): 14; two-dimensional (2-D): 7; one-dimensional (1-D): 1.

d. *Discretization technique.* Finite differences: 10; finite volumes: 5; finite elements: 4.

e. *Linear equation solver.* Iterative: 15; direct: 3. A wide variety of iterative solvers was mentioned, including conjugate gradient: 6; Gauss-Seidel: 2; multigrid: 2; SOR: 2; and Jacobi: 1.

f. *Coefficient matrix.* Matrix features varied widely. Block cyclic, block diagonally dominant, block tridiagonal, dense nonsymmetric, diagonally dominant, positive definite, positive definite Hermitian, sparse-almost symmetric, sparse unstructured, sparse positive definite, symmetric, and symmetric block pentadiagonal matrices were all mentioned.

Dear colleague:

Thank you for your response to my earlier inquiry regarding construction of a benchmark package for the HPCMO. We need clarification of some of your answers to make sure our benchmark is representative.

1. Is the primary data type in your codes real, double precision, complex, and/or double complex?
2. Do you use any "off-the-shelf" subroutine/subroutine libraries (e.g., LINPACK, LAPACK, FFTPACK) or did you write your own?
3. Does your code(s) use a 1-D, 2-D, or 3-D grid to represent the problem?
4. Does your code(s) use finite differences? If yes, specify type, e.g., 2-D 5-point stencils, 3-D 27-point stencils.
5. Does your code(s) use finite elements/volumes? If yes, specify type, e.g., piecewise linear/planar, bicubic. Also specify what method, e.g., Galerkin.
6. How large are the linear systems?
7. Does the underlying matrix have any special properties, e.g., symmetric, band, block diagonal, diagonally dominant, positive definite?
8. Do you use a direct method, e.g., Gaussian elimination, LU decomposition, Cholesky factorization? If yes, specify method(s).
9. Do you use an iterative method, e.g., Gauss-Seidel, SOR, conjugate-gradient? If yes, specify method(s).
10. Do you use some other hard-to-classify method, e.g., multigrid, frontal? If yes, specify method(s).
11. Is the linear system stored as a 2-D array, or in some other special format?
12. Could you provide us with a subroutine or code(s) to help us ascertain the nature of the flux calculations?
13. Can you provide any additional information about the benchmarks/codes you mentioned (**NA825**, **NAVDAS**, **COAMPS**, **NOGAPS**)?
14. CGWAVE has been suggested by one of your colleagues as a candidate for kernel extraction. Do you think this code could be used to represent CWO codes in general?
15. You have said that solving a linear system is a dominant computational task in your code. Is this task in a loop (e.g., a time-step loop) as illustrated below?

    ```
    DO N times
      "before stuff"
      solve linear system
      "after stuff"
    END DO
    ```

    If yes, how large is N? What types of computations take place in "before stuff" and "after stuff" (e.g., calculate matrix coefficients, flux calculations)? Can you characterize the nature of these other tasks or provide code to illustrate them? How much time do they take relative to the linear system solution?

Thank you for your help in making this effort a success.

Figure 2. Phase 2 user survey example

Use of Utilization Data

Perhaps the most important source of data for design of the test package was utilization data for fiscal year 2000 from each of the MSRCs. A summary of MSRC system usage by CTA is shown in Tables 1 and 2; these data were used to identify the CTAs having the highest system utilization, and, as discussed later, to construct job mixes for the throughput tests.

Code usage across all of the MSRCs constituted another important source of workload data; Table 1 shows this usage data for codes using the most CPU hours across all four MSRCs. This table was compiled by using job accounting data from all of the MSRCs to identify the top users at each site, and then personally contacting them to determine which code they were running. If a user specified

Table 1
MSRC System Utilization in CPU Hours by CTA

CTA	ARL	ASC	ERDC	NAVO	Average
CCM	509,545	1,629,514	1,029,695	666,939	958,923
CEA	500,116	655,189	805,602	25,297	496,551
CEN	1,000	155,075	14,414	455,766	156,563
CFD	756,076	1,855,451	1,397,502	1,594,749	1,400,944
CSM	1,185,228	209,768	1,534,787	21,174	737,739
CWO	20,004	34	559,330	1,842,061	605,357
EQM	0	0	391,474	4,269	98,935
FMS	20,580	46,207	1,799	0	17,146
IMT	49,021	0	0	0	12,255
SIP	54,250	96,020	6,749	1,161	39,545
Other	63,055	0	100,854	425	41,083
Total	3,158,875	4,647,258	5,842,210	4,611,844	4,565,045

Table 2
Percent MSRC System Utilization by CTA

CTA	ARL	ASC	ERDC	NAVO	Average
CCM	16.1	35.1	17.6	14.5	20.8
CEA	15.8	14.1	13.8	0.5	11.1
CEN	0.0	3.3	0.2	9.9	3.4
CFD	23.9	39.9	23.9	34.6	30.6
CSM	37.5	4.5	26.3	0.5	17.2
CWO	0.6	0.0	9.6	39.9	12.5
EQM	0.0	0.0	6.7	0.1	1.7
FMS	0.7	1.0	0.0	0.0	0.4
IMT	1.6	0.0	0.0	0.0	0.4
SIP	1.7	2.1	0.1	0.0	1.0
Other	2.0	0.0	1.7	0.0	0.9
Total	100.0	100.0	100.0	100.0	100.0

two or more codes and did not give an estimate of the percent usage of each, then equal distribution of CPU hours among the codes was assumed. CPU hours on different systems are not necessarily equivalent due to differences in clock rate, processor architecture, compiler optimization, and other factors; this is not taken into account in these rankings. The Cray T3Es at ERDC and NAVO have large numbers of CPUs, so codes that run on those machines tend to be at the top of the list. Lastly, Cray vector machines at NAVO have powerful CPUs, but fewer of them, so codes that run only on those machines have much fewer CPU hours and do not appear in the table.

Selection of codes for inclusion in the test package proceeded in a systematic fashion. Beginning at the top of the ranking shown in Table 3, the following criteria were used to determine whether a candidate code was included in the test package:

Table 3
Code Ranking by CPU Hours Used

Code	CTA	Type	Usage
CTH	CSM	MPI	2,069,998
NLOM	CWO	MPI	1,969,755
$Cobalt_{60}$	CFD	MPI	989,124
GAMESS	CCM	MPI	514,070
Code A	CEN	MPI	434,559
LESLIE3D	CFD	MPI	404,308
Code B	CCM	MPI	371,848
NFA	CFD	HPF	363,367
Code C	CCM	MPI	327,194
Code D	CEA	SHMEM	319,867
Code E	CFD	SHMEM	315,745
Code F	CSM	SHMEM	303,374
Code G	CCM	MPI	287,858
MICOM	CWO	MPI	262,602
ICEPIC	CEA	MPI	259,937
VASP	CCM	MPI	215,086
GASP	CFD	OpenMP	199,864
Code H	CFD	Unknown	195,916
Code I	CFD	MPI	176,035
FEMD	CCM	MPI	170,751
USM3D	CFD	MPI	168,515
CHARGE	CEA	MPI	168,340
CRAFT	CFD	MPI	144,922
Gaussian	CCM	OpenMP	140,464
Code J	CCM	Unknown	118,212
Code K	CFD	Unknown	108,474
EIGER	CEA	MPI	101,328
Code L	CFD	SHMEM	94,648
MD-Multiscale	CEN	MPI	88,527
Code M	CFD	MPI	85,536
Code N	CFD	MPI	82,909
Code O	CEA	MPI	81,071
COAMPS	CWO	MPI	59,713

a. *Does the code use MPI or OpenMP?* One or the other of these parallel programming interfaces was required so that all prospective vendors could run the code; this ruled out codes that used only the Shared Memory (SHMEM) one-sided communication library and codes designed for best performance on vector processors.

b. *Does the code represent a CTA not already well represented in the test package?* The goal was to have two codes for each of the top seven CTAs.

c. *Does the code represent an MSRC not already well represented in the test package?* The goal was to have at least two codes representing each MSRC.

d. *Is the code readily obtainable?* Commercial codes requiring a fee and those that the developers were unable to release were discarded.

e. *Is the code portable?* If after a few days of effort it was not possible to get the code to execute on systems from two different vendors, then the code was discarded.

The final list of codes selected from this list for inclusion in the test package included the top code at each MSRC supplemented by others that satisfied the above constraints (see Table 4). Remarkably, a total of 6,634,810 CPU hours, or 36.3 percent, may be attributed to only nine codes: **CHARGE**, **Cobalt$_{60}$**, **CTH**, **FEMD**, **GAMESS**, **ICEPIC**, **LESLIE3D**, **MD-Multiscale,** and **NLOM**. For special reasons, four other codes were included for which no data was available: **FEMWATER123** to represent EQM, **PRONTO** to provide a second representative of CSM, **SARA-3D** because of its high I/O requirements, and **UNCLE** to represent CFD usage at ARL. A brief description of each code is provided in the next chapter.

Table 4
Characteristics of Benchmark Application Codes

Application	Language	Parallel Method	CTA	CHSSI
CHARGE	C	MPI	CEA	No
Cobalt$_{60}$	f90	MPI	CFD	Yes
CTH	f77/C	MPI/PVM	CSM	Yes
FEMD	f77	MPI	CCM	No
FEMWATER123	f77	MPI	EQM	Yes
GAMESS	f77	MPI/SHMEM	CCM	Yes
ICEPIC	f77	MPI	CEA	Yes
LESLIE3D	f77/f90	MPI/OpenMP	CFD	No
MD-Multiscale	C/f90	MPI	CEN	No
NLOM	f77	MPI/OpenMP/SHMEM	CWO	Yes
PRONTO	f77/f90/C	MPI	CSM	No
SARA-3D	f77/C	MPI/OpenMP	CEA	No
UNCLE	f90	MPI	CFD	No

3 Dedicated Run Results

As mentioned in the first chapter, a "dedicated" test is one in which the entire resources of a system are dedicated to running a single job. While it may seem wasteful, for instance, to empty a 256-processor system to run a job requiring only 64 processors, it is valuable to do so because it provides a measure of the best possible performance one could expect from the system for that job. This approach is useful in order to quantify the overhead penalty paid when using the machine in production mode, i.e., any competition with other jobs for memory or communication bandwidth is eliminated. To participate in the TI-01 procurement process, vendors were required to run the tests described in this chapter in dedicated mode. To prepare and validate the test package, the benchmark design team ran many, but not all, of these tests in that same dedicated mode. This chapter describes the results of the team's efforts.

The 13 application codes noted in the previous chapter were run on one sytem at the ASC MSRC, three at the ERDC MSRC, and one at the NAVO MSRC; the first section in this chapter describes these systems. For many of the codes, more than one set of input data was used; a code with a single set of input data is herein termed a "test case." All of the test cases were executed using different numbers of processors, and walltimes were recorded. To better understand the significance of these data, various performance metrics were computed; these are described in the second section of this chapter. Following that, there is a section for each application that discusses the code, the input data used to test the code, the times, and the associated scores. Appendix B discusses technical issues encountered while preparing the various codes for inclusion in the test package.

Systems Tested

The five SUTs are described in Table 5. The short names listed in this table will be used to refer to the various SUTs. When there is no possibility of confusion, these short names will also be used to refer to the particular system model of interest (e.g., the name "O3K" may refer to the particular SUT installed at ERDC, or to the entire class of Origin 3000 systems, depending on the context).

Table 5
Characteristics of SUTs

System	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
Name	seymour	osprey	cobalt	hpc03-5	sard
Site	NAVO	ERDC	ERDC	ASC	ERDC
Model	T3E-900	SP	Nighthawk I	Origin 2800	Origin 3800
CPUs	1048	255	512	128	256
CPUs/Node	1	1	8	2	4
Processor	Alpha 21164	RS/6000 POWER2SC	RS/6000 POWER3	MIPS R10000	MIPS R12000
Clock Rate	450 MHz	135 MHz	222 MHz	195 MHz	400 MHz
Level 1 I-Cache	8 KB, on-chip, direct mapped	32 KB, on-chip, 2-way set assoc.	32 KB, on-chip, 128-way set assoc.	32 KB, on-chip, 2-way set assoc.	32 KB, on-chip, 2-way set assoc.
Level 1 D-Cache	8 KB, on-chip, direct mapped	128 KB, on-chip, 4-way set assc.	64 KB, on-chip, 128-way set assoc.	32 KB, on-chip, 2-way set assoc.	32 KB, on-chip, 2-way set assoc.
Level 2 Cache	96 KB, on-chip, 3-way set assoc.	none	4 MB, external, direct mapped	4 MB, external, 2-way set assoc.	8 MB, external, 2-way set assoc.
Total Memory	384 GB	256 GB	256 GB	64 GB	512 GB
Total Disk Space	1.51 TB Mirrored	0.5 TB RAID	1.7 TB RAID	1 TB RAID	6.02 TB RAID
Operating System	UNICOSMK 2.0.5.47	AIX 3.4	AIX 3.4	IRIX 6.5	IRIX 6.5
Batch Scheduler	NQE 3.3	Open PBS 2.3	PBS Pro 5.0.1	Open PBS 2.3	PBS Pro 5.0.1
Fortran Compiler	Cray CF90 3.3.0.0	xlf 7.1.0.0	xlf 7.1.0.0	MIPSpro 7.3.1.1m	MIPSpro 7.3.1.1m
C Compiler	Cray C 6.3.0.0	xlc 3.1.4.10	xlc 3.1.4.10	MIPSpro 7.3.1.1m	MIPSpro 7.3.1.1m
Math Library	Craylibs 3.3.0.0	ESSL 3.1.2.0	ESSL 3.1.2.0	SCSL 1.3.0.0	SCSL 1.3.0.0
MPI Library	MPT 1.2.1.2	PSSP 3.11	PSSP 3.11	MPT 1.4	MPT 1.4.0.2

Interesting features of and special circumstances related to each of these systems will be discussed in turn.

The Cray T3E-900 used in these tests, referred to as the "T3E," is installed at the NAVO MSRC.[1] The 1,048 CPUs in the T3E are connected in a 3-D toroidal mesh. Each node in the T3E is based on a single 450-MHz DEC Alpha microprocessor and is capable of 2 flop/cycle (multiplies or adds), giving a peak rate of 900 Mflop/s per node. Memory on the T3E is distributed; furthermore, this T3E is configured so that 488 CPUs have 512 MB of memory each and 560 CPUs have 256 MB of memory each. System software uses 10 MB of memory on each CPU, reducing the amount available to a user, and 24 of the CPUs are "login" nodes unavailable for batch usage. As with the jobs on the other four systems, a batch queuing system was used to submit all of the jobs on the T3E; no jobs were run interactively.

The first IBM SP used in these tests is installed at the ERDC MSRC. The 255 nodes of this SP communicate via a packet switched interconnect; 249 of these are compute nodes available for running batch jobs. Each node in this system contains one CPU, a 135-MHz IBM RS/6000 POWER2 Super Chip (POWER2SC). Because it is an IBM SP equipped with POWER2SC CPUs, this system will be consistently referred to as the "SP2." The POWER2SC is a single-chip implementation of the POWER2 architecture that allows higher clock rates than its multichip predecessors. Like all members of the POWER2 and POWER3 families, the POWER2SC has a fused multiply-and-add (FMA) instruction. Each of the floating-point units on the POWER2SC is capable of completing one FMA instruction per cycle, giving it a peak rate of 540 Mflop/s. Memory is distributed on the SP2, and nodes have varying amounts of memory: 99 of the compute nodes have 512 MB of local memory, 97 have 1 GB, and 53 have 2 GB. During these tests, six nodes were dedicated to other purposes, leaving 249 nodes available for running batch jobs.

The second IBM system also has the SP architecture and is also installed at the ERDC MSRC. Each of the 64 nodes in this system is an 8-way symmetric multiprocessor (SMP) with 4 GB of shared memory. Each of the eight CPUs in a node is a POWER3 microprocessor running at 222 MHz. Because of this feature, this system is referred to as the "SP3." Just like the SP above, each CPU can execute two FMA instructions per cycle, giving a peak rate of 888 Mflop/s. The tests reported here were run before the installation of IBM's newest generation interconnect, the Switch2. Because of this, only four processors per node were available to MPI processes. Processes using threads could use all eight, but only two test codes, **NLOM** and **SARA-3D**, had this capability. Since the conclusion of these tests, the Switch2 has been installed on the SP3; an analysis of the pre- and post-Switch2 SP3 performance on several of the benchmark applications is presented in a separate report (Duffy et al. 2001).

The last two systems used in these tests were SGI Origin systems. The first is a 128-processor Origin 2800, an "O2K," installed at the ASC MSRC. The

[1] This system was moved to the ERDC MSRC in 2002 and combined with an already-installed T3E. The resulting system, the largest known T3E in the world, is shown on the cover.

second system is an Origin 3800, or "O3K," installed at the ERDC MSRC. At the time of these tests, the O3K was configured as two separate identical 256-CPU system images; these have since been merged into a single 512-CPU system. No distinction is made here between jobs run on the two O3K systems. The O2K and the O3K use 195-MHz MIPS R10000 and 400-MHz R12000 microprocessors, respectively. Both are capable of two floating-point operations per cycle, yielding respective per CPU rates of 390 and 800 Mflop/s. The Origin interconnect is a so-called "bristly hypercube," with two CPUs attached to a vertex on the O2K and with four on the O3K. Unlike the other systems reported here, the Origins use cache-coherent nonuniform memory access (ccNUMA) technology to provide a globally shared memory space to all of the processors.

Not all of the tests required of the vendors were run as part of these tests. In some cases, the SUT did not have a sufficient number of processors to perform a particular run, and, in other cases, limited local memory on some processors prevented the execution of some of the test cases. Such situations are denoted by blank entries in the tables.

To perform the tests reported here, exclusive access to the first four systems noted above was obtained so that jobs could run in dedicated mode. For various reasons, some jobs running during these test periods terminated abnormally or failed to complete within the dedicated time and ended up competing with other jobs. Because there was limited time for exclusive access, it was not possible to run or rerun some of the jobs in dedicated mode; to fill the gaps in the data, runs in nondedicated mode were used. No distinction is made here between times collected in dedicated mode and those collected in nondedicated mode. Furthermore, at the time of these tests, the O3K was in the shakedown phase of its installation; the system was available to "pioneer" users and it was not possible to obtain exclusive access to the system for dedicated test jobs. In spite of this, the second system image was often idle and de facto dedicated time was obtained for many of the tests.

Generally, MPI was the parallel programming interface used by all codes; however, there were several exceptions. The implementations of **GAMESS** and **NLOM** on the T3E used SHMEM. As noted earlier, **SARA-3D** uses MPI and threads to implement parallelism; this capability was used on the SP3 and the Origins.

Performance Metrics

The primary performance metric gathered for each run of a test case is "walltime;" that is, the elapsed time from when job execution begins until it ends. This does not include time spent waiting in a queue to execute. Furthermore, this is not the same as "CPU time," which measures the time consumed by the processor running the job. Rather, walltime is the time a job runs as recorded by a user watching an external clock; consequently, this is sometimes termed "wall-clock time." For jobs running on a single CPU, CPU time is always less than or equal to walltime, the difference between the two times being the result of time

spent waiting for temporarily unavailable system resources or performing system overhead tasks. When a job is run in dedicated mode, it is presumed that this difference will be at a minimum since there is no competition for system resources from another user's job.

For each run performed the elapsed walltime in seconds was recorded; tables, one for each code, present these data. As an aid in comparing systems, the O3K was used as a reference system and the performance relative to the O3K was computed. Specifically, if T_X is the time for some test case on system X and T_{O3K} is the time for that same run on the O3K, then the relative performance of system X is $P_X = T_{O3K}/T_X$. For example, if T_{O3K} is 50 sec and T_X is 100 sec, then P_X is 0.5, indicating that X has half the performance of the O3K. Obviously, this measure will vary depending on the code, input data, number of CPUs used, and other factors.

After this performance metric was computed for each observed walltime, a summary average performance was computed for each system by code. Those code averages were then averaged to produce an overall average performance for each system. Unweighted averages were used at each step. Further information on combining performance measures has been described by Hennessy and Patterson (1996).

CHARGE

Code description

CHARGE calculates the radar cross section (RCS) of complex scatterers using a finite-volume electromagnetics approach. The spatial discretization is a third-order, cell-centered Van-Leer splitting, while a second-order Runge-Kutta scheme is used for integration over time. Solving for an RCS using **CHARGE** requires six steps, generally performed in a single directory and each requiring a separate executable file. These steps are discussed below.

The first step uses the **GRIDGEN** program to define the input grid. Unlike the programs that follow, **GRIDGEN** is not a part of the **CHARGE** package, and it was not executed as part of the tests reported here. Next, **patchtrans** converts the **GRIDGEN** output file into a binary "patch" file, *patch.dat*, defining the boundary conditions for the problem.

In the third step, **mzconvert** restructures the grid data for effective parallelization. Its input consists of *patch.dat*, at least one ".grd" grid file in **PLOT3D** format, and a user-supplied file, *inputmzc.dat*. This last file allows the user to specify, among other things, the names of the input and output files. Output of this step is *grid.dat*, which is input for the next step, and *rcscells* and *surfcells*, which are input to the **CHARGE** step.

Running **mydecomp** is the fourth step. Using *grid.dat* from the previous step and the user-specified file *inputmyd.dat*, it partitions the domain among the processors. The output files are *cell.map*, *parallelgrid.dat*, and *parallelinterp.dat*.

The next step, **CHARGE**, is the heart of the calculation. It solves Maxwell's equations and calculates the RCS using an FFT approach. It requires *rcscells*, *surfcells*, *cell.map*, *parallelgrid.dat*, *solverinput.dat*, and *driverinput.dat* as input. Output files contain bistatic RCS data at 0.5° intervals.

The last step, **gridinterp**, is a post-processing step to facilitate visualization of the output. It was not a part of these tests.

CHARGE is written entirely in C and consists of 16,605 lines of code (LOC); the charge step itself is 6,950 LOC. The formats of the various input files are described in the user's manual (CHARGE Development Team 1999). All of above steps except for the **CHARGE** step execute serially.

Performance

The input data used for this test case specify a calculation of the RCS of a sphere. The three *500.dd* input files differ in the number of periods used; for these, the maximum number of steps is given by the digits *dd*. Thus, for *500.25*, a maximum of 25 time-steps is specified, even though the computation converges after 16. Six $59 \times 19 \times 19$ subgrids are used.

Walltimes for the **CHARGE** test cases are provided in Table 6. **CHARGE**'s performance scales well with the number of processors, particularly on the T3E and the SGI systems. The reduction in scalability at higher processor counts on the Origin systems indicates that the serial part of the code is becoming an increasingly significant fraction of the work. Performance relative to the O3K is given in Table 7; the O3K outperforms the next fastest system, the O2K, by almost a factor of 2.

Table 6
CHARGE Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
500.03	2	5725	7638	5965	4808	2629
	4	2931	4038	2999	2429	1437
500.05	64	437	1220	524	328	202
500.25	8	7659	10252	7962	6352	2918
	16	3966	5597	4401	3246	1476
	32	2072	3176	2285	1659	817
	64	1138	2265	1502	907	490
	128	665	1385	873	703	330

Table 7
CHARGE Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
500.03	2	0.46	0.34	0.44	0.55	1.00
	4	0.49	0.36	0.48	0.59	1.00
500.05	64	0.46	0.17	0.39	0.62	1.00
500.25	8	0.38	0.28	0.37	0.46	1.00
	16	0.37	0.26	0.34	0.45	1.00
	32	0.39	0.26	0.36	0.49	1.00
	64	0.43	0.22	0.33	0.54	1.00
	128	0.50	0.24	0.38	0.47	1.00
Average		**0.44**	**0.27**	**0.38**	**0.52**	**1.00**

Cobalt$_{60}$

Code description

Cobalt$_{60}$ is a parallel, implicit computational fluid dynamics (CFD) code that solves the compressible Euler and Navier-Stokes equations subject to the ideal gas equation of state (Computational Sciences Branch, Aeronautical Sciences Division, Air Vehicles Directorate 1999). Two-dimensional, 3-D, and axisymmetric spaces can be modeled. Unstructured grids with arbitrary cell types are permitted. The developers' goal was to make **Cobalt$_{60}$** as general, flexible, robust, accurate, and easy to use as possible.

The fundamental algorithm of **Cobalt$_{60}$** 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 (Godunov 1959, Holt 1996). However, in practice, Godunov's exact Riemann solver is very expensive, so the method of Gottlieb and Groth is employed (Gottlieb and Groth 1988, Strang 2000). 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 (van Leer 1979). The linear variations (gradients) are constructed by a central-difference, least-squares method that, in turn, is solved by QR factorization.[1] In cells requiring limiting, the gradients are corrected to give a one-sided least-squares scheme. First- and second-order temporal accuracy is achieved via 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 (MacCormack 1969), are

[1] *QR* is not an acronym; instead, it refers to the factorization of a matrix *A* into the matrix product of *Q* times *R* (*QR*) where *Q* is orthogonal and *R* is upper triangular. The *QR* algorithm is used to determine the eigenvalues of the matrix *A*.

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. The Spalart-Allmaras (Spalart and Allmaras 1992) and the Baldwin-Barth (Baldwin and Barth 1991) turbulence models are available to model the fine-scale effects of turbulence. The turbulence models are first-order accurate in space with temporal accuracy determined by the above point-implicit method. 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 groups, blocks, or zones, permitting parallel processing where each zone resides on a separate processor. This is accomplished using **ParMETIS**, the MPI-based, parallel grid-partitioning library that performs both static and dynamic graph partitioning and fill-reducing reordering (Karypis 2000, Karypis and Kumar 1995, Karypis and Kumar 1998). It partitions a multidimensional grid among a set of processors, ordering the grid points so that the amount of fill-in during direct solution of the resulting linear system is reduced. **ParMETIS** is highly portable and easy to install on a variety of platforms. Additionally, **ParMETIS** produces roughly equally sized zones, which produces good load balancing, and each zone has a minimized "surface area," thus reducing communications overhead. Consequently, **Cobalt$_{60}$**'s excellent scalability may be attributed to two characteristics: good load balancing with minimal communications overhead attributable to **ParMETIS**, and high computational intensity requiring little communication.

Development of **Cobalt$_{60}$** began in 1990 at the Air Force Research Laboratory (AFRL), Aeronautical Sciences Division, Computational Sciences Branch, with the three major developers being William Strang, Robert Tomaro, and Matthew Grismer. **Cobalt$_{60}$** is a project under the CFD CTA of the HPCMPO's Common High Performance Computing Software Support Initiative (CHSSI) and can only be distributed to U.S. citizens.

Cobalt$_{60}$ is written in Fortran 90 and uses only MPI to achieve parallelism. Hence, **Cobalt$_{60}$** is quite portable and has been successfully installed on IBM SPs, T3Es, Origins, Cray parallel vector processors, HP C240/C360 clusters, HP K460s, DEC AS8400s, and Linux-based systems (using the Portland Group's compilers).

Cobalt$_{60}$ is often used to model fluid flow and turbulence around objects moving through a fluid (e.g., missiles and jets traveling through the air). The number of cells in a model typically ranges from several hundred thousand to ten million, while the number of processors used ranges from tens to hundreds, depending on the size of the mesh.

Performance

Three cases of increasing size were run with **Cobalt$_{60}$**. *Missile* modeled a finned missile flying at Mach 2.5 and 14° angle of attack with 728,109 cells; *wingflap* modeled a wind tunnel model of a wing with a flap and endplates with 2,976,066 cells; and *trap-wing* modeled a body/wing/flap/slat configuration at 10° angle of attack with 7,009,212 cells. All cases were 3-D and modeled turbulent viscous flow over the geometries. The walltimes and relative performance for these cases are shown in Tables 8 and 9, respectively.

Table 8
Cobalt$_{60}$ Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
missile	16	5060			5294	2565
	32	2577	2650	1632	2204	1901
	64	1370	1274	776	1009	886
	128	808			686	536
wingflap	16	9471			8334	3664
	32	4831	4671	2760	4420	1733
	64	2643	2248	1343	2882	883
	128	1523			1004	533
trap_wing	32	8234	6856	4016	7481	2859
	64	4483	3854	2150	3166	1840
	128	2682				1225
	256	1784				817

Table 9
Cobalt$_{60}$ Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
missile	16	0.51			0.48	1.00
	32	0.74	0.72	1.16	0.86	1.00
	64	0.65	0.70	1.14	0.88	1.00
	128	0.66			0.78	1.00
wingflap	16	0.39			0.44	1.00
	32	0.36	0.37	0.63	0.39	1.00
	64	0.33	0.39	0.66	0.31	1.00
	128	0.35			0.53	1.00
trap_wing	32	0.35	0.42	0.71	0.38	1.00
	64	0.41	0.48	0.86	0.58	1.00
	128	0.46				1.00
	256	0.46				1.00
Average		**0.47**	**0.51**	**0.86**	**0.56**	**1.00**

For the *missile* test case, all systems except the O3K show a superlinear speedup. Presumably, this is due to the size of the problem and how it fits into the various levels of cache on the different chips. On the O3K, the problem fits into the second-level cache and shows a more representative speedup for this problem size. However, the *wingflap* and *trap-wing* models are larger and do not fit into cache except perhaps for runs with larger numbers of processors. Even so, the T3E shows the best speedup efficiency, though it takes the longest amount of walltime. All of the platforms perform well for the small *missile* test case. However, the performance drops dramatically for the larger test cases, with the O3K maintaining a high level of performance.

CTH

Code description

CTH[1] was developed by Sandia National Laboratories (SNL) for modeling complex multidimensional and multimaterial simulations involving large deformations and/or strong shock physics (Sandia National Laboratories 1998, Hertel et al. 1993, McGlaun et al. 1990). Hence, problems including penetration and perforation, compression, detonations, etc., can be explored with the collection of six codes that make up the **CTH** software package. These six components include the following:

a. **CTHGEN** sets up the initial configuration of the problem, including load balancing for parallel architectures.

b. **CTHREZ** rezones a problem or combines multiple problems.

c. **CTH** computes the time integration and conservation equations.

d. **CTHED** controls queries to the problem database for detailed information at the cell level.

e. **CTHPLT** produces graphics at a given time during the computation.

f. **HISPLT** produces graphics for user-defined variables as a function of time.

Over the past decade, **CTH** has become one of the most heavily used applications in the DoD research community and has been included in several prior DoD HPC benchmarks. Although it is export-controlled, its high utilization and portability plus being a CHSSI code made its inclusion in the test package imperative.

[1] "**CTH**" is an acronym of an acronym of an acronym; it stands for "**CSQ** to the Three-Halves." "**CSQ**" stands for "**CHARTD** SQuared." "**CHARTD**" stands for "Computational Hydrodynamics And Radiative Thermal Diffusion." **CHARTD**, **CSQ**, and **CTH** are 1-D, 2-D, and 3-D codes, respectively. Thanks to David Crawford of Sandia National Laboratories for supplying this crucial piece of information.

The CTH99 distribution contained instructions for installation on a variety of platforms. **CTH** is highly portable and can be built to run on a single processor or with multiple processors utilizing PVM, MPI, or NX message passing. **CTH** has been ported to and tested on numerous platforms under a variety of operating systems and programmatic interfaces, including the Cray T3E, IBM SP, SGI R10000 and R12000, SUN SPARC, and Intel Paragon. Only the MPI version of **CTH** was used for these tests. Furthermore, of the six components in the **CTH** distribution, only **CTHGEN** and **CTH** were used. **CTH** is also very scalable, making it a good candidate for testing the scalability limits of recently introduced parallel systems. Current usage of **CTH** varies greatly, with typical models using up to 512 CPUs.

Performance

Table 10 summarizes the test cases used for **CTH** by showing the number of cells in each dimension (N_x, N_y, N_z), the length of time simulated in the model, and the range of processors on which vendors were required to run. Vendors were allowed to supplement this list with runs using larger numbers of CPUs in order to demonstrate system scalability. The simulation time for test case *mpi-032* was reduced so that it could be easily included in the throughput tests.

The goal of having this many test cases was to exercise various paths through the code. Obviously not all paths could be exercised; otherwise, the number of test cases would have been quite large. Instead, representative inputs were obtained from DoD **CTH** users. Problems tackled with **CTH** by DoD users are growing in size. Fortuitously, some of these test cases are quite large and so are representative in that respect as well. For instance, the *arm.t1.in* test case may require as much as 20 GB of memory to execute.

Timings on five different machines are shown in Table 11, and system performance relative to the O3K is given in Table 12. Only the T3E was able to run the

Table 10
Characteristics of CTH Test Cases

Test Case	N_x	N_y	N_z	Simulated Time	Recommended Nos. of CPUs	Remarks
arm.t1.in	1580	500	40	10.0e-6	64, 96, 128, 256	From work at ERDC
efp3d.s1.in	80	80	80	50.0e-6	32, 48, 64	From the SNL distribution of **CTH**
efp3d.s2.in	80	80	40	50.0e-6	32, 48, 64	
mpi_001.in	215	30	60	40.0e-6	1, 2, 4, 8	Created at ARL to scale the size of the problem along with the number of CPUs
mpi_002.in	271	38	75	40.0e-6	2, 4, 8	
mpi_004.in	341	48	95	40.0e-6	4, 8, 16, 32, 64	
mpi_008.in	430	60	120	40.0e-6	8	
mpi_016.in	540	76	151	40.0e-6	16	
mpi_032.in	683	95	191	0.5e-6	16, 32, 64	

Table 11
CTH Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
arm.t1	64		5903	2516	4195	2034
	80	3518	4667	2123	3488	1718
	96	2975	4020	1864	3063	1498
	128	2237	3051	1450	2745	1248
	256	1218				1582
	512	688				
mpi_032	16	4853	5542	2371	4169	1914
	32	2447	2687	1331		1113
	64	1437	1905	877		679
	128	811	998	544		467
	256	492				639
	512	305				

Table 12
CTH Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
arm.t1	64		0.34	0.81	0.48	1.00
	80	0.49	0.37	0.81	0.49	1.00
	96	0.50	0.37	0.80	0.49	1.00
	128	0.56	0.41	0.86	0.45	1.00
	256	1.30				1.00
	512					
mpi_032	16	0.39	0.35	0.81	0.46	1.00
	32	0.45	0.41	0.84		1.00
	64	0.47	0.36	0.77		1.00
	128	0.58	0.47	0.86		1.00
	256	1.30				1.00
	512					
Average		**0.67**	**0.38**	**0.82**	**0.48**	**1.00**

test cases with more than 256 processors. Large **CTH** test cases scale well, and the *arm.t1.in* input is no exception. Typical efficiencies for 64- and 128-processor runs range between 80 and 96 percent, with the T3E scaling at nearly 100 percent for large processor counts. The increase in the time for the 256-processor run relative to the 128-processor run on the O3K is presumably caused by resource contention between **CTH** and operating system processes.

The performance of the O2K relative to the O3K (0.48) is almost exactly what would be expected from the improvement in clock speed (195 MHz / 400 MHz = 0.4875) (Hensley et al. 2001). This is not the case for the SP3 (0.82 versus 222 MHz / 400 MHz = 0.555), indicating that differences in architecture of

the CPU and interconnect are responsible for the additional performance. The scaling of the O3K is not ideal. It seems that the speed and efficiency of the O3K CPUs have increased to the point that the overhead time of initializing MPI, distributing the grid, reading in the input, etc., has become a significant portion of the total time of the run. This phenomenon seems most apparent for jobs with large numbers of CPUs and short run times.

FEMD

Code description

As technology advances, the need for materials with specific structural properties has increased dramatically. As an example, ceramics are highly desirable materials for applications requiring extreme operating conditions (i.e., high/low temperatures and pressures). Recent discoveries have resulted in ceramics that are much more ductile, opening up a larger number of applications to which ceramic materials may apply. Even with the extensive research being conducted, fundamental questions at the atomic level are still unanswered. Specifically, the understanding of the ceramic/metal interface is elusive (Kalia et al. 1999).

Calculations used to study such problems are extremely large scale and are based on first-principle methods. **FEMD** is a parallel pseudo-potential plane wave program used to make just such calculations. It is based on a preconditioned Krylov-space iterative Lanczos-method diagonalization procedure, which is an extension of an earlier Car-Parrinello molecular dynamics code developed at Queen's University, Belfast (Alavi et al. 1994).

Simulations on the order of hundreds of atoms are commonly performed. In order to treat such large systems, it was necessary to parallelize Fortran code using MPI. By exploiting the highly parallel architectures, a real space mesh of $108 \times 108 \times 108$ and 1,300 electron orbitals are expanded in about 90,000 plane waves. For problems of this size, 128 or more processors are generally used. Examples of the research performed using **FEMD** are available in Benedek, Minkoff, and Wang (1996), Benedek et al. (1999), and Benedek et al (2000).

FEMD is written in Fortran but contains C preprocessor directives that tailor the source code to a variety of platforms as well as make the resulting executable serial or parallel. **FEMD** uses the LAPACK library.

Performance

Two test cases were used for the **FEMD** tests. Both performed a calculation involving 32 atoms with a total of 116 electrons occupying 64 states. Three different types of atoms were used in the calculation: 16 titanium atoms, 12 aluminum atoms, and 4 carbon atoms. At each step of the code, the resulting k-space eigenvalues and eigenfunctions are computed and printed. After the final step, the total energy is calculated.

To analyze the time spent in initialization and problem setup, the first test case, *inp*, was run for five steps, while the second, *inp.2*, treated the same problem but was run for only one step. The times for these test cases are shown in Table 13, and relative performance is given in Table 14. If the setup time was negligible and if the work performed by the code in the first time-step was the same as that performed in subsequent time-steps, then *inp* should take five times longer than *inp.2*. Instead, roughly a factor of 2 is seen.

A speedup efficiency close to unity is seen when moving from 16 to 32 processors across all platforms. However, the test cases do not scale well beyond 32 CPUs. Furthermore, for every system except the T3E, the 64-processor runs were actually slower than the 32- and 48-processor runs. It seems likely that for this particular test case a communication bottleneck has been encountered. This is most likely due to the too small size of the test case. Further exploration of the performance of **FEMD** on a larger number of processors should use a more complex problem with more states and/or more atoms.

Table 13
FEMD Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
inp	16	10800	11574	5579	10571	4526
	32	5109	5711	2868	5321	2239
	48	4698	5895	2779		2173
	64	4537	8418	4416		2246
inp.2	16	6283	6574	3278		2646
	32	2958	3330	1701		1276
	48	2755	3398	1681		1249
	64	2601	4555	2554		1291

Table 14
FEMD Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
inp	16	0.42	0.39	0.81	0.43	1.00
	32	0.44	0.39	0.78	0.42	1.00
	48	0.46	0.37	0.78		1.00
	64	0.50	0.27	0.51		1.00
inp.2	16	0.42	0.40	0.81		1.00
	32	0.43	0.38	0.75		1.00
	48	0.45	0.37	0.74		1.00
	64	0.50	0.28	0.51		1.00
Average		**0.45**	**0.36**	**0.71**	**0.42**	**1.00**

For these test cases, the O3K shows a significant advantage over the other platforms. However, this result should be qualified by the fact that the code had to be compiled without optimizations on the IBM machines. With compiler optimizations enabled, the SP3 times would presumably be closer to the O3K.

FEMWATER123

Code description

FEMWATER123, written in Fortran 77, models groundwater flow through a 3-D volume of varying density and properties. The code is a result of a collaborative effort in the early 1990s between the Athens Environmental Research Laboratory[1] of the U.S. Environmental Protection Agency and the U.S. Army Engineer Waterways Experiment Station. It is an updated implementation of two older models: the flow model 3DFEMWATER (Yeh 1987) and the transport model 3DLEWASTE (Yeh 1990). A single coupled flow and transport model was created in a collaborative effort between Dr. Yeh and Dr. Hsin-Chi Lin at Waterways Experiment Station (Lin et al. 1997). A parallel version of **FEMWATER123**, using MPI, has recently been developed by Dr. Fred Tracy and Mr. Dave Richards of ERDC. Distribution of the computation across the processors is accomplished using **ParMETIS**, the parallel grid-partitioning library noted earlier. **FEMWATER123** has been tested on all the HPC systems at ERDC.

A typical **FEMWATER123** run has about 100,000 elements and typically uses up to about 40 processors. As an example, a 1-year simulation, with about 65,000 elements running on 40 processors on the T3E at ERDC, successfully completed in about 16.5 hr.

Performance

A classic test problem utilizing the 1-2-3-D coupling implemented in **FEMWATER123** is a Dade County, Florida, model (*fred.3bc.1*) where canals interact with the surface water and groundwater flow (Tracy and Richards 2000). The 1-D canal structures and 2-D surface model that are coupled to the 3-D flow model greatly complicate the computation. This complex system is modeled by a grid of 4,720 surface mesh nodes, 37,760 total nodes, and 65,429 tetrahedral elements. This case ran for 14,400 time-steps. Walltimes and relative performance are shown in Tables 15 and 16, respectively. These times are the sum of the times of all four programs required to run a **FEMWATER123** simulation to completion. Obviously, this problem is not scalable. Within the parallel version of **FEMWATER123**, there is a crossover from a computational bottleneck at lower numbers of processors to a communication bottleneck at higher numbers of processors. Hence, the optimum performance for this test case is around 24-32 processors. Increasing the number of time-steps will not make this problem more scalable; rather, the number of nodes or elements used for the calculation must be increased. Longer runs indicate that walltime scales linearly with the number of

[1] Now the National Environmental Research Laboratory (NERL).

Table 15
FEMWATER123 Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
fred.3bc.1	8	4277	4580	2832	3272	1516
	12	4125	4797	2837	3473	1528
	16	4011	4613	4127	3013	1418
	24	3415	4415	3988		1264
	32	3174	4333	2396	2933	1261
	48	3203	5202	2670	2605	1273

Table 16
FEMWATER123 Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
fred.3bc.1	8	0.35	0.33	0.54	0.46	1.00
	12	0.37	0.32	0.54	0.44	1.00
	16	0.35	0.31	0.34	0.47	1.00
	24	0.37	0.29	0.32		1.00
	32	0.40	0.29	0.53	0.43	1.00
	48	0.40	0.24	0.48	0.49	1.00
Average		**0.37**	**0.30**	**0.46**	**0.46**	**1.00**

time-steps, indicating that the combined setup time of all four programs is relatively small for this **FEMWATER123** simulation.

For this particular test case, the O3K outperforms the other platforms. Based on experience with the T3E and the observation from these data that the O3K is about 2.7 times faster than the T3E, an extension of this test case to 1 year of simulated time would require an estimated 6.3 hr using the optimum number of CPUs as discussed above.

GAMESS

Code description

The General Atomic and Molecular Electronic Structure System (**GAMESS**) is a code for computational quantum chemistry (Schmidt et al. 1993). It is an *ab initio* code in that material properties are calculated from first principles (i.e., the number, type, and position of the atoms in a molecule must be specified as initial conditions for the computation). **GAMESS** evolved from several earlier quantum chemistry codes, in particular, **HONDO**, which was developed using funding from the National Science Foundation and the Department of Energy.

Development continues at Iowa State University with sponsorship from the Air Force Office of Scientific Research (Department of Chemistry, Iowa State University 2000).

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, and generalized valence bond. A complete description of the program's capabilities and the input language interface may be found in the user's guide (Department of Chemistry, Iowa State University 2000).

The version of **GAMESS** used in these tests was dated 25 March 2000; it consists of 238,315 nonblank, noncomment LOC, with 782 LOC written in C, and the remainder in Fortran 77. The Fortran source files must be tailored to a specific system using a custom preprocessor supplied as part of the **GAMESS** distribution.

Performance

Two test cases were used to test the performance of **GAMESS**. The first case, *cycl*, models cyclic AMP ($C_{10}H_{11}N_5O_6P$); this 33-atom molecular model has 356 Cartesian Gaussian basis functions and 85 occupied orbitals. This simulation specifies a restricted Hartree-Fock calculation providing both the molecular energy and the gradient, and requires minimal disk storage. The energy computation converges in 14 iterations.

The second test case, *hedm*, models an HEDM molecule ($C_2N_{10}O_4$) without N_2. This 16-atom molecular model has 560 Cartesian Gaussian basis functions and 57 occupied orbitals. A restricted Hartree-Fock calculation that computes only the molecular energy is specified. Even though this model has fewer atoms than the first test case, the larger number of basis functions makes it a more complex problem to solve. The energy computation converges in 18 iterations.

The walltimes for the the *cycl* and *hedm* test cases are included in Table 17 and the relative performance is given in Table 18. It is clear from the times presented in the table that both cases exhibit quite acceptable scaling, and that the O3K significantly outperforms the other systems.

ICEPIC

Code description

The Improved Concurrent Electromagnetic Particle-In-Cell (**ICEPIC**) code is a parallel, 3-D PIC computer simulation tool for fully relativistic problems involving collisionless or low-collisonality plasmas in complex geometries.

Table 17
GAMESS Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
cycl	4	3880	7815	4378	6080	
	8	2064	4297	2306	3103	
	16	1131	2318	1257	1632	769
	32	660	1372	847	895	427
	64	417	877	696	516	260
hedm	32	6273	15079	8471	9517	4371
	64	3441	8168	4914	4997	2356
	96	2495	6342	3346	3492	1633
	128	2022	5031	3971	2804	1270
	192		4199			

Table 18
GAMESS Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
cycl	4					
	8					
	16	0.68	0.33	0.61	0.47	1.00
	32	0.65	0.31	0.50	0.48	1.00
	64	0.62	0.30	0.37	0.50	1.00
hedm	32	0.70	0.29	0.52	0.46	1.00
	64	0.68	0.29	0.48	0.47	1.00
	96	0.65	0.26	0.49	0.47	1.00
	128	0.63	0.25	0.32	0.45	1.00
	192					
Average		**0.66**	**0.29**	**0.47**	**0.47**	**1.00**

ICEPIC was designed to take full advantage of the speed and power of parallel architectures and is one of the few PIC codes that is scalable to a large number of processors. To enable the code to operate efficiently on parallel platforms, **ICEPIC** was equipped with several special features, including automated partitioning, an advanced parallel PIC algorithm, and dynamic load balancing. **ICEPIC** is written entirely in ANSI C and uses the standard MPI communications interface to maintain portability.

Originally, **ICEPIC** was developed as a parallel, fully 3-D PIC code with a variable Cartesian mesh. Its capabilities have since been expanded to include 2-D Cartesian and 2-D and 3-D cylindrical simulations. **ICEPIC** can execute in serial or parallel mode for all 2-D and 3-D simulations. It has been tested on a variety of systems, including SP2s, SP3s, and other RS/6000 systems running AIX, Origins running IRIX, T3Es running UNICOS, Sun workstations and

clusters running Solaris and MPICH, and Intel/AMD workstations and clusters running LINUX and MPICH. **ICEPIC** is export-controlled. It was distributed by ERDC after the appropriate Air Force release forms were signed.

Performance

Typical production runs of **ICEPIC** last 48 hr or more on 64-96 CPUs. In this benchmark, however, the average walltime is significantly less, typically on the order of 1 hr. Vendors were required to run **ICEPIC** on several different input sets and processor ranges as summarized in Table 19. The first three test cases involve similar problems of increasing size.

Timings are presented for the test cases *ice.dat.128* and *mitl.dat* (a total of five tests) in Table 20, and performance relative to the O3K is given in Table 21. Clearly, the O3K outperforms the other platforms for these test cases. The SP3's performance reflects its lower processor speed of 222 MHz. Running this code on a 375-MHz SP3 presumably would result in performance much closer to the O3K. For the test case *ice.dat.128*, the SP3, the T3E, and the O3K all show a superlinear speedup as the number of processors is increased. Experience indicates that this probably results from program data becoming cache-resident.[1]

Table 19
ICEPIC Test Cases

Test Case	Number of CPUs
ice.dat.16	16, 32, 64, 128
ice.dat.64	16, 32, 64, 128
ice.dat.128	16, 32, 64, 128
mitl.dat	16, 32

Table 20
ICEPIC Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
ice.dat.128	16			17577		11009
	32		10448	8196	7261	5080
	64	5917	5496	3862	4653	2154
	128	2832	3000	1808	2380	916
mitl.dat	32	1345	2093	954	1129	614

[1] For a well-written code running a fixed size problem, memory requirements should grow slowly as the number of CPUs is increased. Available cache memory will grow linearly with the number of CPUs, so more and more of the data formerly in memory will reside in cache. Significantly, the system that has the poorest speedup for this test case, the SP2, has no second-level cache.

Table 21
ICEPIC Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
ice.dat.128	16			0.63		1.00
	32		0.49	0.62	0.70	1.00
	64	0.36	0.39	0.56	0.46	1.00
	128	0.32	0.31	0.51	0.38	1.00
mitl.dat	32	0.46	0.29	0.64	0.54	1.00
Average		**0.38**	**0.37**	**0.59**	**0.52**	**1.00**

LESLIE3D

Code description

LESLIE3D is a 3-D, Navier-Stokes solver for turbulent reacting flows over structured rectangular or cylindrical grids; it was developed at Computational Combustion Laboratory at the Georgia Institute of Technology. **LESLIE3D** solves the fully compressible Filtered Navier-Stokes equations, the conservation of mass, momentum, energy, and chemical species equations using an explicit finite-volume scheme that is fourth-order accurate in space and second-order (explicit) in time. MacCormack's predictor-corrector method (MacCormack 1969) for the time integration was used. The code is designed to operate either as a direct numerical simulation approach or as a large-eddy simulation (LES) approach. In the LES approach (which is used for the benchmark), all scales larger than the grid are resolved using the finite-volume scheme, and only scales smaller than the grid are modeled using a subgrid model. **LESLIE3D** is not a CHSSI code and has no export control restrictions.

In addition to the basic conservation equations, **LESLIE3D** solves a one-equation model of the subgrid kinetic energy of the turbulence (K_{sgs}). K_{sgs} is used to close the unresolved terms in the LES model that arise from the spatial filtering of the governing equations. **LESLIE3D** is capable of employing a localized dynamic evaluation of several of the scaling factors in the turbulence models, such as the coefficients in the subgrid closure. This process, known as Dynamic K_{sgs}, was developed by W.-W. Kim and Suresh Menon of the Georgia Institute of Technology.

The code has been used for various reacting and nonreacting flows such as swirl-stabilized combustion in a gas turbine (Kim, Menon, and Mongia 1999), spatial compressible mixing layers (Fernand and Menon 1993), and combustion instability in ramjet engines (Menon 1995). An extensive bibliography on LES research may be found at the Georgia Institute of Technology web site (Computational Combustion Laboratory, School of Aerospace Engineering 2000).

However, the version used for the benchmark solves a simpler flow of a temporal mixing layer formed between two parallel plates that are moving in opposite directions. A uniform grid distribution in all three directions is used to resolve the flow in a cubic volume. Two scalar species are also simulated to mimic fuel and air mixing in the mixing layer. Thus, a total of eight conservation equations are solved in the code used for the benchmark. For the fourth-order scheme, two layers of ghost cells are required in each processor at every time-step. For the localized dynamic evaluation of the subgrid model (not used for the benchmark), three layers of ghost cells are required.

There exist solvers for both cylindrical and square configurations. The algorithm is suitable for high-resolution simulations of free shear flows (mixing layers, isotropic turbulence, flame propagation, etc.). Curvilinear coordinate transformations are not performed, so a uniform grid is required. The code allows a choice for the order of spatial accuracy (2 or 4), the number of chemical species (none, flame model, or multiple species), the LES turbulence model (standard or k-equation model), and inviscid or viscous flow. The benchmark version of the code was set up for fourth-order spatial accuracy, two chemical species, viscous flow, and the standard LES turbulence model.

LESLIE3D is written entirely in Fortran 77 and is parallelized using MPI. Some parallelization has also been implemented using OpenMP, but this was not used in these tests. **LESLIE3D** is quite portable, requiring only a Fortran 77 compiler and the MPI library. The current code was written and tested on the following operating system/compiler combinations: SGI IRIX64/**f77**/**f90**, Cray UNICOS/**f90**, IBM AIX/**xlf**, GNU/**g77**, and PGI/**pgf77**. Only slight modifications were required, e.g., changing the `access=` keyword in the `open` statement to `position=` when using **f90**.

Performance

Only one test case was used for the **LESLIE3D** runs. The problem simulated is a 3-D temporally evolving mixing layer in which the shear layer rollup is due to the nonlinear growth of the fundamental mode of instability. For the benchmark tests, the top half is made up of fuel and the bottom half is made up of oxidizer, and, thus, the problem is of fuel-air mixing between two species. This test case takes only 500 time-steps and the simulation model employs an LES approach that involved solution of another equation for K_{sgs}. Thus, a total of eight equations are solved on the computational domain. The grid size was $128 \times 128 \times 128$ for all runs, but the processor grids were $1 \times 2 \times 4$, $1 \times 4 \times 4$, and $2 \times 4 \times 4$, for a total of 8, 16, and 32 MPI processes, respectively.

Table 22 shows the resulting walltimes, while Table 23 gives the relative performance. Good speedup is observed on all the platforms tested, with the O3K running the code the fastest. It is interesting to note, though, that the difference in timings between the O3K and the SP3 is much less than that observed in other application runs.

Table 22
LESLIE3D Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
128x128x128	8	4805	5766	3081	5711	2747
	16	2492	3115	1653	2926	1406
	32	1322	1670	927	1612	729

Table 23
LESLIE3D Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
128x128x128	8	0.57	0.48	0.89	0.48	1.00
	16	0.56	0.45	0.85	0.48	1.00
	32	0.55	0.44	0.79	0.45	1.00
Average		**0.56**	**0.45**	**0.84**	**0.47**	**1.00**

MD-Multiscale

Code description

MD-Multiscale implements the coupling of length scales method for simulating multiscale materials phenomena. It uses a molecular dynamics approach to give an atomistic description of the material, using a quantum-mechanical description of bonding in a limited region where it is required and a faster, but less accurate, empirical description in the rest of the system. In this benchmark, the code is used to simulate the fracture of silicon, where the quantum-mechanical description of bonding near the tip of the crack is needed to reproduce experimental results. Improved understanding of fracture will enhance the ability to design stronger materials from silicon and related materials used for micromachines and to design ceramics used for structural applications.

The version of **MD-Multiscale** used in these tests consists of 18,560 nonblank, noncomment LOC, with 1,698 LOC written in C and the remainder in Fortran 90. Both the C and Fortran source files must be fed to the C preprocessor to tailor the code to a particular system and to consistently substitute formulas in the program. Attempts to migrate the code to the T3E were unsuccessful, so no timings or scores for the T3E will be presented.

Performance

To fit into the throughput tests, the 100 time-step test data **test.100** was reduced to 10 time-steps in *test.10*; all other input values remained the same. Table 24 shows the walltimes for the **MD-Multiscale** test cases, while Table 25 gives the relative performance.

Running the code on large numbers of processors does not appear profitable on the O3K, and its scalability is not particularly good on the IBM systems. A possible explanation for this lackluster performance may be the calls to the **ps** command from within the code and the considerable amount of formatted debugging output produced.

Table 24
MD-Multiscale Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
test.10	64		1292	912		
test.100	32 52 101		12765 10061 7634	8107 6605 4746	9861	5657 4601 5472

Table 25
MD-Multiscale Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
test.10	64					
test.100	32 52 101		0.44 0.46 0.72	0.70 0.70 1.15	0.47	1.00 1.00 1.00
Average			**0.54**	**0.85**	**0.47**	**1.00**

NLOM

Code description

The NRL Layered Ocean Model (**NLOM**) has a development history extending back to at least 1980 (Hurlburt and Thompson 1980). It has been used to model semi-enclosed seas, major ocean basins, and the global ocean, and uses a tiled, data-parallel programming approach (Wallcraft and Moore 1997). Details on its usage may be found in the user's guide (Wallcraft 2000).

NLOM is a highly portable code that runs on a variety of parallel systems using MPI, MPI-2 PUT/GET, SHMEM, High Performance Fortran, Co-Array Fortran, OpenMP, and even Fortran autotasking. **NLOM** has been included in prior DoD HPC benchmark exercises; thus, many vendors had some familiarity with the code. However, the test case used here is four times larger than that used in 1998. The **NLOM** distribution contained instructions for installing **NLOM** on a variety of platforms. **NLOM** has been ported and tested on SP2s, SP3s, O2Ks, O3Ks, T3Es, Sun E10000s, and HP Exemplars. The MPI version was required in the benchmark, but vendors were allowed to report results from dual-level MPI/OpenMP and SHMEM versions.

Performance

The *NA825* test case simulates 3.05 model days on a 1/64 degree 5-layer Atlantic Subtropical Gyre region (grid size $4096 \times 2688 \times 5$). About 12 GB of memory and 20 GB of scratch disk space were required. The run included I/O and data sampling typical of a production run but did not include an initialization step, which was performed as a separate job. Vendors were required to run this case at 28, 56, and 112 CPUs; optionally, runs at 14, 84, 168, 224, 336, 448, 672, and 896 CPUs were requested to aid the analysis of system scalability.

The walltimes for this test case are shown in Table 26 and the relative performance in Table 27. Only the T3E was able to run the test cases with more than 256 processors. The results indicate that **NLOM** does not scale well at larger processor counts for this model size. The T3E times were obtained using the SHMEM version of **NLOM**, and the SP3 times were obtained before installation of the IBM "Switch2" interconnect. Counter to expectations, the performance of the O2K relative to the O3K (0.41) is somewhat slower than what would be indicated by the improvement in clock speed (195 MHz / 400 MHz = 0.4875) (Hensley et al. 2001). Relative performance of the Cray and IBM systems decreases as the number of CPUs increases; this falloff is most severe for the SP3.

Table 26
NLOM Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
NA825	28	9213	14457	6728	14495	6300
	56	4459	8231	3664	7963	2933
	112	2497	7057	2112	3807	1602
	168		4956	1631		1101
	224	1499	4109	1497		917
	336	1240				
	448	1123				
	672	1068				
	896	1091				

Table 27
NLOM Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
NA825	28	0.68	0.44	0.94	0.43	1.00
	56	0.66	0.36	0.80	0.37	1.00
	112	0.64	0.23	0.76	0.42	1.00
	168		0.22	0.68		1.00
	224	0.61	0.22	0.61		1.00
	336					
	448					
	672					
	896					
Average		**0.65**	**0.29**	**0.76**	**0.41**	**1.00**

PRONTO

Code description

PRONTO is a 3-D transient dynamics code developed at SNL that models the deformations of nonlinear materials subjected to extremely high strain rates (Flanagan and Flanagan 1989, Sandia National Laboratories 2000). Transient dynamics analysis is important in a variety of industrial applications; examples include simulation of vehicle crashes, forging metal, and container denting and deformation. **PRONTO**'s smoothed particle hydrodynamics capability enables it to simulate cases with extremely high strains (e.g., explosive events) as well as coupled structure interactions (Attaway et al. 1997). **PRONTO** is part of the Sandia Engineering Analysis Code Access System (SEACAS). This system includes tools for preprocessing, postprocessing, database translation, and graphics. **PRONTO** does not perform any mesh generation or postprocessing analysis; rather it relies on external applications to perform those processes.

Two test cases were chosen as part of the benchmark suite; both of these tests are included with the **PRONTO** code distribution. The first test, *brick-wall*, simulates a brick wall being hit by an elastic rod using approximately 6,100 elements. The second test, *beam-large-mesh*, uses more than 200,000 elements to model the deformation of a beam under a load. Mesh generation was not part of the benchmark; only the execution time for **PRONTO** itself was recorded.

Performance

PRONTO was included in the test suite late in the benchmarking effort. Consequently, it was not possible to get the code running on non-SGI systems, and a discussion of relative performance is not possible. Tables 28 and 29 provide the performance data that were obtained.

Table 28
PRONTO Walltimes in Seconds on Various Systems

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
beam_large_mesh	16					1350
	32					684
	64					353
	128					
brick_wall	4				2458	1098
	8					675
	16					479
	32					399

Table 29
PRONTO Performance Relative to the O3K

Test Case	CPUs	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
beam_large_mesh	16					1.00
	32					1.00
	64					1.00
	128					
brick_wall	4				0.45	1.00
	8					1.00
	16					1.00
	32					1.00
Average					**0.45**	**1.00**

SARA-3D

Code description

The Structural Acoustic Radiation Analyzer (SARA) family of finite-element codes (**SARA-2D** and **SARA-3D**) has been designed to efficiently solve complex structural acoustic problems by using finite elements to model both the structure and the fluid (BBN Technologies 2000). **SARA-3D** is a dual-level MPI/OpenMP parallel code developed at BBN Technologies, with help from the ERDC MSRC's Computational Science and Engineering Group (CS&E). A variety of quadratic structural and continuum elements are available for modeling the structure, while pressure-type acoustic fluid elements are used to model internal fluids and the external field near the structure (the "near field"). The unbounded external fluid (the "far field") is modeled with infinite elements that include in their formulation the outward traveling and decaying wave shape. The excitation may be by mechanical forces or acoustic sources using either plane or spherical

waves. The combination of finite and infinite elements has been shown to be significantly more economical than using finite and boundary elements for comparable problems. If near or far field pressure fields are desired, they are obtained via the Helmholtz integral equation from velocities and pressures computed at the fluid-structure interface. The infinite/finite element technique used in **SARA-3D** provides an effective method for solving the fluid-structure acoustics problem. Important features of **SARA-3D**'s solution approach are as follows:

a. The resulting fluid-structure interaction equations are symmetric and banded, and, hence, can be solved efficiently.

b. Infinite elements reduce the exterior fluid field to a few element layers.

c. Exterior fluid field normal to the structure is constant with frequency, but element size is varied.

d. As **SARA-3D** problems get larger, the infinite element method is more efficient than the boundary element method.

The **SARA** software consists of **SARA-2D**, which solves axisymmetric and 2-D geometries, and **SARA-3D**, which treats general 3-D geometries with arbitrary loadings. The following types of problems ve been solved using **SARA**:

a. *Structural vibrations*. Compressible isotropic and anisotropic materials as well as incompressible isotropic materials can be analyzed. Frequency-dependent moduli and loss factors may be included. Continuum and shell elements are available for modeling.

b. *Structural acoustics*. Pressure-type fluid elements are provided for modeling internal or external fluids. The representation of infinite fluids is enhanced by infinite elements that include in their complex representation the outward traveling and decaying wave shapes. Separate fluid and structure models are created and connected via coupling elements.

c. *Radiation problems*. A variety of loadings may be applied to the structure to calculate radiated sound in the field.

d. *Scattering problems*. Appropriately directed and phased loadings are applied at the fluid-structure interface to represent incident plane scattered or spherical waves, and the field pressures are computed.

e. *Electroelasticity*. Piezoelectric elements, materials, and boundary conditions are provided to facilitate the modeling of transducer structures.

Additional features include built-in mesh generation for the structural and interior fluid models and automated generation of a frequency-dependent exterior fluid model. Structures may consist of plates and shells with discrete or smeared stiffeners, compressible or incompressible solids, and/or coatings and absorptive layers. In addition, a continuum beam, which provides an exact solution of Timoshenko's beam theory, is available. Materials include various types of anisotropy, frequency-dependent moduli, and structures. Results such as velocities,

pressures, and power can be postprocessed in a variety of ways and output to graphics programs for line plots, color contour plots, etc.

Capabilities also exist for efficient solving of axisymmetric structures with limited nonsymmetric components by using a substructure approach that combines the fluid-loaded shell from **SARA-2D** with any **SARA-3D** model. The axisymmetric fluid-loaded shell can be solved as a series of uncoupled 2-D problems far more economically than as a full 3-D problem. Coupling this solution to 3-D parts of the problem can result in significant savings in computation costs.

Typically, **SARA-3D** is used to model the frequency response to incident waves of a structure such as a submarine or airplane surrounded by a fluid. The calculations for each freqency are completely independent. **SARA-3D** exploits dual-level or nested parallelism by using MPI to parallelize the code that cycles through the input frequencies (the "frequency loop") and using OpenMP to parallelize certain calculations for each frequency. In particular, each frequency leads to a large sparse system of linear equations, and the frontal solver used to solve the system has been parallelized using OpenMP.

SARA-3D is written primarily in Fortran 77, but includes a few C routines. It runs on many systems, including Cray T3Es, IBM SPs, and SGI Origins, and Convex, DEC, HP, and Sun systems. It may only be distributed to U.S. citizens.

Performance

The basic input file describes a parameterized model of a submerged, stiffened cylindrical shell with hemispherical endcaps subjected to forces on the shell. Parameter *P*2, the circumferential direction, was set to either 10 for a small problem or to 20 for a large problem, while the number of frequencies was either 64 or 128. This resulted in the four test cases shown in Table 30.

Table 30
Characteristics of SARA-3D Test Cases

Test Case	Size	P2	Frequencies
tape5.10.064	small	10	64
tape5.10.128	small	10	128
tape5.20.064	large	20	64
tape5.20.128	large	20	128

Table 31 gives the walltime **SARA-3D** on various computers for different MPI/OpenMP processor configurations. The column labeled "Threads/Process" gives the number of OpenMP threads per MPI process, and the product of "MPI Processes" and "Threads/Process" is equal to the number of CPUs. The timings show that doubling the number of frequencies from 64 to 128 for a given problem and MPI/OpenMP processor configuration increases the execution time by less than a factor of 2. This is because the frequency loop is parallelized using both MPI and OpenMP, while a significant amount of time is needed to perform

particular one-time operations (e.g., the "prefront" and postprocessing phases), which are outside the frequency loop and are only parallelized in OpenMP mode. The cost of these one-time operations is amortized as the number of frequencies handled by each MPI process grows, thus increading MPI parallel efficiency.

For a fixed number of processors on systems that support OpenMP (e.g., the SP3 and the Origins), use of mixed-mode parallelism (i.e., Threads/Process > 1) may reduce the walltime relative to a pure MPI run (Threads/Process = 1); this effect is more pronounced for 64 frequencies than for 128 frequencies. As the number of frequencies grows, MPI parallel efficiency dominates OpenMP efficiency, and thus it may be more efficient to use fewer OpenMP threads, as can be seen in Table 31. Performance relative to the O3K is given in Table 32.

Table 31
SARA-3D Walltimes in Seconds on Various Systems

Test Case	CPUs	MPI Processes	Threads/ Process	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
tape5.10.064	16	16	1	7344	3599	1922	2679	1496
tape5.10.128	16	16	1	10866	5803	3444	3806	2266
tape5.20.064	32	32	1	21464	10145	5956	8784	6561
	8	1	8			11443	28540	
	32	16	2			5681	8673	3550
	32	8	4			5278	16987	3660
tape5.20.128	32	32	1	32109	16044	8304	13756	8919
	8	1	8			19046	50076	
	32	16	2			8326	10719	5694
	32	8	4			8251	12336	6268

Table 32
SARA-3D Performance Relative to the O3K

Test Case	CPUs	MPI Processes	Threads/ Process	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
tape5.10.064	16	16	1	0.20	0.42	0.78	0.56	1.00
tape5.10.128	16	16	1	0.21	0.39	0.66	0.60	1.00
tape5.20.064	32	32	1	0.31	0.65	1.10	0.75	1.00
	8	1	8					
	32	16	2			0.62	0.41	1.00
	32	8	4			0.69	0.22	1.00
tape5.20.128	32	32	1	0.28	0.56	1.07	0.65	1.00
	8	1	8					
	32	16	2			0.68	0.53	1.00
	32	8	4			0.76	0.51	1.00
Average				**0.25**	**0.50**	**0.80**	**0.53**	**1.00**

UNCLE

Code description

UNsteady Computation fieLd Equations (**UNCLE**) is a parallel incompressible flow simulator developed and maintained by the Computational Simulation and Design Center Engineering Research Center for Computational Field Simulation at Mississippi State University on behalf of the Office of Naval Research. **UNCLE** was designed to solve Unsteady Reynolds-Averaged Navier-Stokes equations. Complex geometries, such as submarines and surface ships, may be represented by multiblock structured grids with arbitrary block connectivity (Computational Simulation and Design Center, College of Engineering 2000). Written in Fortran 90 and using MPI, **UNCLE** runs on a variety of systems.

Along with **UNCLE**, Utilities for Solver Setup (**USS-UNCLE**) contains various tools to simplify the use of **UNCLE** for the analysis and design of a variety of naval applications. The graphical capabilities of **USS-UNCLE** include preprocessing of grid and boundary conditions, definition of input files, documentation, and the actual flow simulation code. For the purposes of the DoD benchmark, prerelease version 1.3.2 of **UNCLE** was used.

Performance

Depending on the complexity of the input geometry, the grid sizes can grow quite large. Furthermore, the grid sizes will dictate the number of processors on which a specific problem will be run. The test cases were created for use in the benchmark by the developers of the **UNCLE** code. The grids created for these inputs were generated for only 8 and 16 processors. Thus, vendors were not able to run the problem for anything other than 8 or 16 processors. Hence, the scaling of the code for this particular test case was not explored.

The two **UNCLE** test cases, *sub-grid8* and *sub-grid16*, differ only in grid size, which was scaled with the number of processors in order to keep the amount of work required for 8 and 16 processors roughly equal. Furthermore, both of these test cases were run for 30, 60, 90, and 120 cycles each. Hence, not only can the startup time be analyzed, but the scaling of the code as the number of cycles grows can also be studied. The grid files for both the 8- and 16-processor runs are slightly greater than 6 MB. At the beginning of a run, each processor had to read in this file. Periodically during the run, updated grid files are generated that can be used to restart the code.

Though vendors were required to run both with and without the restart files, the timings contained in Table 33 were all started from scratch. For a given number of cycles, the 8- and 16-CPU runs required approximately the same amount of walltime. Surprisingly, the 60-cycle runs for both grid sizes took nearly three times longer than the corresponding 30-cycle runs. Finally, when compiler optimizations were enabled on the IBM systems, **UNCLE** produced incorrect results, so the times obtained here are for unoptimized code.

Performance relative to the O3K is given in Table 34. The relative performance when running on the O3K versus the O2K is again slightly greater than 2 and is probably due to faster clock rate. For small numbers of processors (e.g., 8-16), it is typical for a code to be bound by computation rather than communication. To observe the improved performance of the O3K versus the O2K due to factors other than a faster CPU, these test cases would have to be run using more CPUs.

Having a grid specific to only two CPU counts, coupled with the absence of software to generate new grids, severely limited the usefulness of **UNCLE** in the test package. If **UNCLE** is to be included in future benchmarks, then such software must be made available.

Table 33
UNCLE Walltimes in Seconds on Various Systems

Test Case	CPUs	Cycles	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
sub_grid8.1.in	8	30	578	3183	1940	664	404
sub_grid8.2.in	8	60	1720	8171	5153	1922	929
sub_grid8.3.in	8	90	2862	13415	8346	3204	1771
sub_grid8.4.in	8	120	4003	18383	11212	4484	2632
sub_grid16.1.in	16	30	630	3976	2460	713	359
sub_grid16.2.in	16	60	1772	9057	5553	2003	967
sub_grid16.3.in	16	90	2914	14378	8660	3296	1586
sub_grid16.4.in	16	120	4058	19875	12067	4583	2202

Table 34
UNCLE Performance Relative to the O3K

Test Case	CPUs	Cycles	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
sub_grid8.1.in	8	30	0.70	0.13	0.21	0.61	1.00
sub_grid8.2.in	8	60	0.54	0.11	0.18	0.48	1.00
sub_grid8.3.in	8	90	0.62	0.13	0.21	0.55	1.00
sub_grid8.4.in	8	120	0.66	0.14	0.23	0.59	1.00
sub_grid16.1.in	16	30	0.57	0.09	0.15	0.50	1.00
sub_grid16.2.in	16	60	0.55	0.11	0.17	0.48	1.00
sub_grid16.3.in	16	90	0.54	0.11	0.18	0.48	1.00
sub_grid16.4.in	16	120	0.54	0.11	0.18	0.48	1.00
Average			**0.59**	**0.12**	**0.19**	**0.52**	**1.00**

Summary of Results

Table 35 summarizes the relative performance results obtained in this study; it contains the "Average" line from the preceding relative performance tables for each code. It also gives an unweighted overall average relative performance for each SUT. The same results are illustrated in Figure 3.

In summary, the O3K is the newest system evaluated in this benchmark exercise. Based on the applications used here, it is more than 1.5 times faster than the SP3, twice as fast as the T3E and O2K, and nearly three times faster than the SP2. It would have been instructive to run this suite of tests on a system of comparable vintage to the O3K (e.g., a 375-MHz SP3), but such a system was unavailable.

Table 35
Summary of Relative Performance

Code	Cray T3E	IBM SP2	IBM SP3	SGI O2K	SGI O3K
CHARGE	0.44	0.27	0.38	0.52	1.00
$Cobalt_{60}$	0.47	0.51	0.86	0.56	1.00
CTH	0.67	0.38	0.82	0.48	1.00
FEMD	0.45	0.36	0.71	0.42	1.00
FEMWATER123	0.37	0.30	0.46	0.46	1.00
GAMESS	0.66	0.29	0.47	0.47	1.00
ICEPIC	0.38	0.37	0.59	0.52	1.00
LESLIE3D	0.56	0.45	0.84	0.47	1.00
MD-Multiscale		0.54	0.85	0.47	1.00
NLOM	0.65	0.29	0.76	0.41	1.00
PRONTO				0.45	1.00
SARA-3D	0.25	0.50	0.80	0.53	1.00
UNCLE	0.59	0.12	0.19	0.52	1.00
Overall	**0.50**	**0.37**	**0.64**	**0.48**	**1.00**

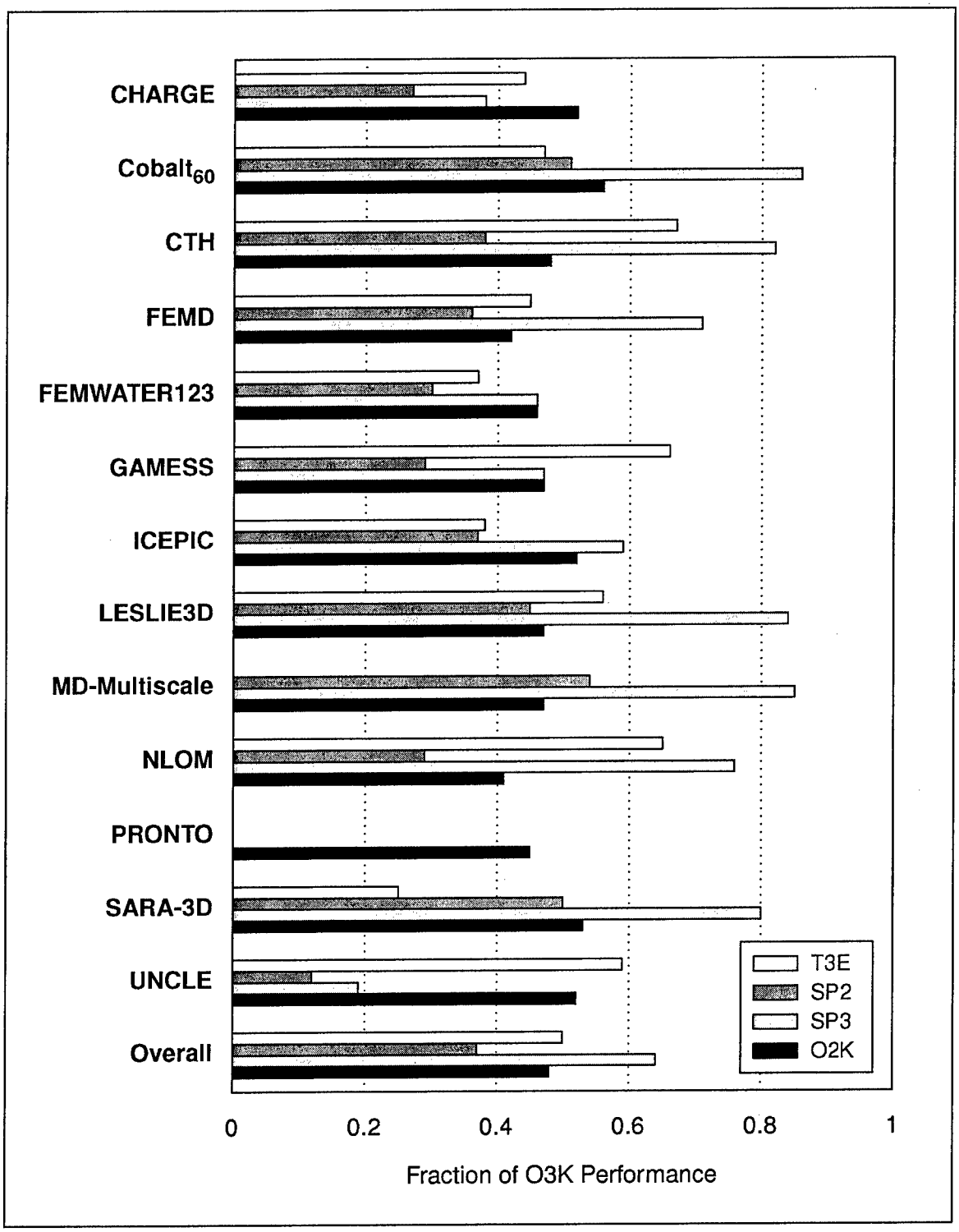


Figure 3. Summary of relative performance

4 Throughput Tests

Introduction

Many computer platforms may perform quite well when codes are executed in dedicated mode, but when the same codes are executed in an operational setting, a much lower level of performance is observed. Thus, one of the main weaknesses of dedicated tests is that they fail to model important features of an actual production environment; such features are described below.

a. In a production environment with hundreds of users, the batch queuing system and job scheduler play a vital role in maintaining a high utilization of any HPC. Along with the large number of users, the queues are complicated by differences in priorities and various resource limits, such as time and memory. An efficient scheduler could result in a much higher utilization and, hence, throughput for a system.

b. With multiple applications running on a system, there will be some contention for resources between the jobs. For example, an application requiring significant I/O could slow down I/O requests by other jobs. Similarly, a job with large memory requirements could reduce the amount of memory available to other jobs and impair system performance by committing numerous page faults.

c. Constructing a throughput test that models the type and frequency of actual job submissions is a daunting task. A significant problem is that, just as there is no typical job, there is no typical mix. There are variations in workload size and character over a workday, a work week, a fiscal year, and, of course, due to project deadlines.

d. System outages are catastrophic events for jobs. The method of recovery from such an occurrence is an important system characteristic. Though no system reboots or artificial outages were forced upon a system in these tests, throughput tests can be used to measure a system's ability to recover. For example, if a system employs checkpoint/restart, jobs will generally be delayed only by the amount of time spent rebooting since the

job state will be saved prior to rebooting, and restored at system startup. Without this feature, jobs must be restarted from the beginning, thus wasting the time used by the job before the reboot.

Resolving these issues using a synthetic benchmark test would be difficult to construct. It would also be difficult to convince fellow scientists that such a test was representative and meaningful. Although arguably just as difficult to construct, a throughput test constructed using actual codes is probably the best way to approach the problem.

Vendors were required to complete two throughput tests. This chapter describes the steps taken to create, test, and validate the job mixes used in those tests. How the system usage data from various sites was used to create these mixes is explained. An algorithm was developed to select applications and test cases subject to weighted constraints on the number of CPUs used and CTA type. This algorithm will be described, and results from throughput tests conducted on Government systems will be presented.

The throughput tests discussed here were run on the ERDC SGI O2K; its specifications have already been given in Table 5. The batch scheduler, PBS, provides several scheduling algorithms, various prioritization schemes, and multiple queues. Here, however, the default method was used, and test jobs were submitted to a single queue. PBS applied a "first-fit" criterion to select the next job to run (i.e., jobs waiting to run are ordered by arrival time, and when CPUs become available, the job list is searched and the first job that requires that many CPUs or less is placed in execution).

Analysis of ERDC's Usage

The first step in the creation of these throughput tests was an analysis of system usage throughout the HPCMP. Chapter 2 described the data received from all four MSRCs and how these data were used to select the 13 applications and the associated test cases used in the dedicated tests. To simplify the task of the benchmark team and the vendors, these same test cases were used to construct the throughput test. However, a representative mix (i.e., the number of instances of each case and the time of submission of each instance), still had to be determined. In order to create this list of jobs, job logs from all of the HPC systems at ERDC covering 1 January 1999 to 8 September 2000 were analyzed.

In order to create a site-wide profile, a histogram was created showing the number of CPU hours for each job versus the percentage of the total number of CPUs in that particular system used by that job. This allowed data from multiple machines with differing numbers of CPUs to be combined.[1] Hence, the CPU hours for a job which uses 128 CPUs on a machine with a total of 256 CPUs would be placed in the 50 percent bin; a job requiring 64 out of 512 CPUs would be placed in the 12.5 percent bin. After all jobs in the sample were categorized in

[1] Only aggregate usage of ERDC HPC systems is discussed here; usage of individual HPCs at ERDC is presented elsewhere (Duffy et al. 2000).

this fashion, the bins were normalized so that the area under the histogram was 100 percent.

To simplify this analysis, no distinction was made between a job requesting the same percentage of CPUs but running for a significantly different amount of walltime; such a distinction could be quite important. For example, suppose a system runs a significant number of jobs with only a few processors and those jobs run for a long period of time. Then a throughput test should ideally include jobs of that sort to allow evaluation of system behavior in those circumstances. Although some adjustments in these mixes were made by hand to avoid artifacts of scheduling that might bias the test (e.g., scheduling of long jobs late in the test that would continue long after all other jobs had completed), no attempt was made to match the walltimes of jobs in the actual ERDC workload.

Furthermore, in the current working environment, not all jobs go into a single queue. Different queues are defined with various priorities in order to expedite certain jobs and to apply differing time and memory limits. In these tests, only a single queue was used, and all jobs were run at the same priority.

Figure 4 shows the resulting histograms for all the machines at ERDC. The top graph in the figure uses 128 bins to display the aggregate workload data. Moving from top to bottom in Figure 4, the bin size is doubled and the number of bins halved for each successive graph. Use of larger bins smooths out small variations in the data and makes it easier to identify usage patterns. The bottom two graphs clearly illustrate user preferences for jobs that use a power of two number of CPUs. More specifically, peaks occur at 12.5, 25, 50, 75, and 100 percent of the CPUs used. Furthermore, a large number of jobs request less than 25 percent of a system. The large plateau of usage that occurs for a relatively low number of processors is to be expected; many users will ask for fewer CPUs so as to get their job to begin execution sooner, even if that means the job will run longer.

Selection of Jobs for the Throughput Tests

Determining the specific jobs to be used in the mixes was the next step in the preparation of the throughput tests. Crucial to this process was assigning each job a score, the computation of which is described in Appendix C, reflecting how well it fit into the job mix. The simple greedy algorithm shown in Figure 5 uses those scores to select jobs for inclusion in each mix.

Though a large number of application codes and test cases were used in the creation of the throughput tests, it was difficult to fit the actual usage histograms shown in Figure 4. To solve this problem, a simplified histogram, shown in Figure 6, was created by combining adjacent bins; note that the usage peaks now fall in the middle of a bin. Also, due to the large number of jobs that use one or two CPUs, a small bin for those jobs was created. The resulting profile was easier to fit, but still preserved the important characteristics of the workload.

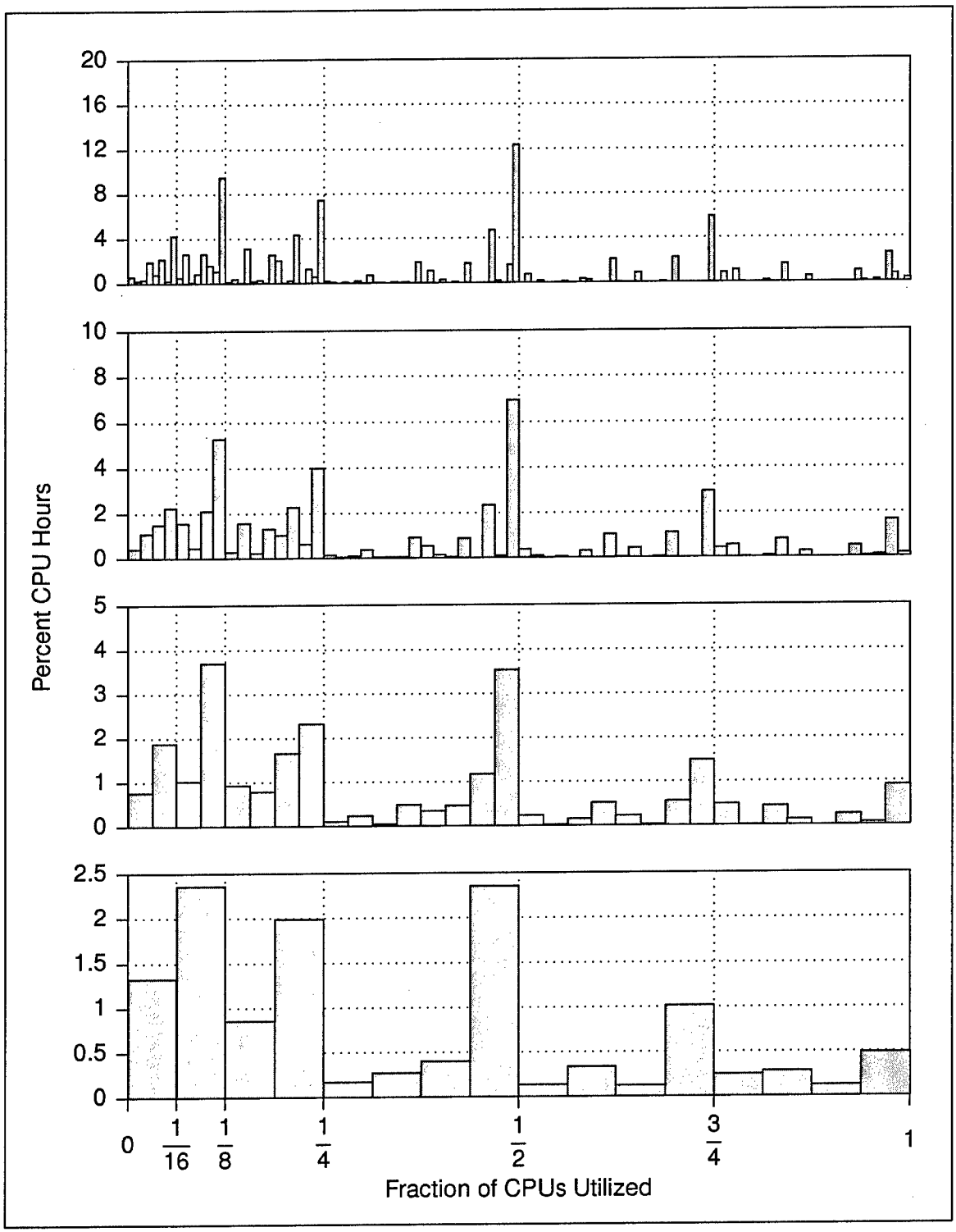


Figure 4. Aggregate ERDC CPU usage using equal intervals

```
SET max_cpu_hours = num_nodes * num_hours
WHILE tot_cpu_hours < max_cpu_hours DO
   FOR each candidate job
      COMPUTE a score measuring how well it meets mix requirements
   END FOR
   FOR the job with the best score DO
      ADD job to the throughput mix
      SET tot_cpu_hours = tot_cpu_hours + job_cpu_hours
      SET tot_cta_cpu_hours = tot_cta_cpu_hours + job_cpu_hours
      SET tot_np_cpu_hours = tot_np_cpu_hours + job_cpu_hours
      SET job_priority = job_priority / 2
   END FOR
END WHILE
```

Figure 5. Greedy algorithm used to select jobs

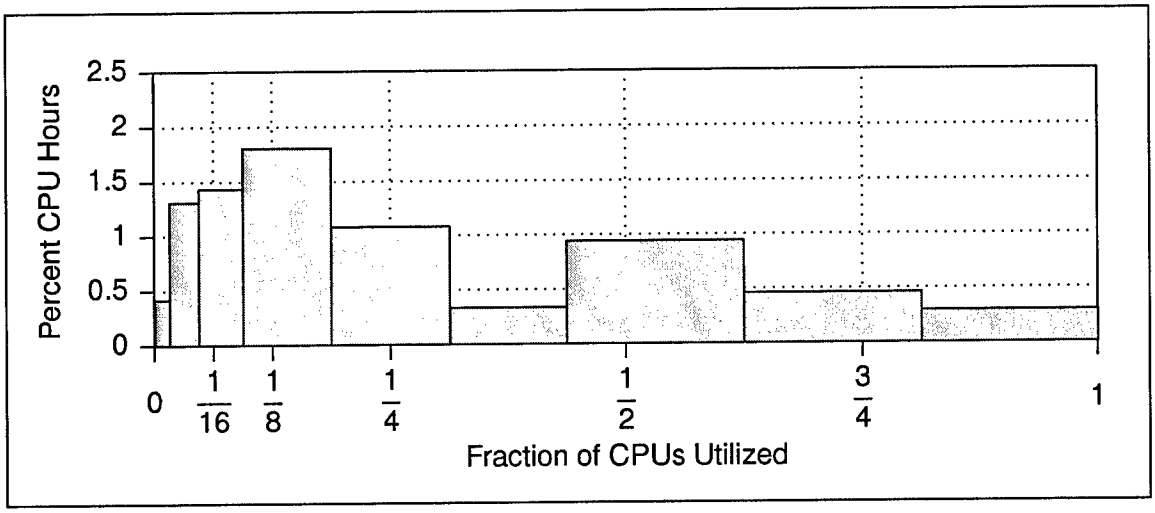


Figure 6. Aggregate ERDC CPU usage using unequal intervals

Rules of the Game

The guidelines for running the throughput tests were as follows:

a. The executable binary programs used for the throughput tests had to be the same as those used for the dedicated runs. Practically speaking, there was no way to verify vendor compliance with this requirement without a vendor site visit, so the vendors had to be trusted to do this.

b. The tests were to be run on a 128-processor machine with the same configuration as that used in the dedicated runs.

c. The codes, test cases, and number of processors were defined and could not be modified by the vendors.

d. All jobs had to be submitted to a single batch queue and could not be hand scheduled in any way.

e. Only commercially available schedulers without any special modifications could be used for the throughput tests.

f. The order and time of job submission were defined in advance by the benchmark team and could not be modified.

g. Names of job submission scripts had to end with *.bat*. The vendors were provided a skeleton of each batch file to be used in the throughput tests; it included the required commands to record walltime and list directory contents (see Figure 7). This item greatly simplified validation of the results. Although many of the codes contained internal timers, the date stamps in the output files provided an easy, standard way to verify walltimes. Furthermore, the **ls -al** command provided a diagnostic to ensure that nothing unforeseen had occurred during the running of an application code.

h. A queue snapshot, consisting of the output equivalent to a **qstat** command, was taken once per minute and saved in a log file.

i. Finally, all the required files as specified for each code, along with the log file of the snapshots, had to be returned with the vendor's response.

```
# This file should contain the batch commands used
# to submit this job to the queue and any commands
# needed to execute the application.

# Required to be the first executable statement.
echo "Job started: `date`"

# Commands required to execute the code: see the
# application's readme file for more information.

# Required to be executed before the final echo
echo "Directory Listing:"
ls -al

# Required to be the last executable statement.
echo "Job ended: `date`"
```

Figure 7. Sample batch submission script

Execution of the Throughput Tests

Two resulting throughput tests were defined to be run by the vendors. Table 36 lists the test cases used in throughput test "A" (Mix A). This mix had the following properties:

a. The 50 jobs in the mix represented a hypothetical site-wide igeneric workload. As a point of reference, each job had to be run outside the throughput test in dedicated mode.

b. The test was designed to run 6 hr on 128 CPUs and so consume 768 CPU hours. Further, its resulting histogram approximately matched that shown in Figure 7.

c. One full configuration job requested all 128 processors.

d. The sequence of submission was given by the order in the table. This could not be changed by the vendors.

e. The number of CPUs to be used for a test case was specified and could not be changed.

f. Conduct of the test was controlled by a simple script that submitted the jobs intermittently over the first hour of the mix. Ten jobs were submitted at the beginning of the test. Every 10 min thereafter, a set of five more jobs was submitted until all of the jobs were queued.

Included in Table 36 are two timings taken for each application benchmark during the throughput test. First, the amount of time (in hours) spent in the queue waiting to run is given in the column labeled "Queue Time." This time is just a measure of how long the jobs spent waiting to be executed. Next, the amount of walltime (in hours) of execution of each code is given in the column labeled "Walltime." Hence, the total number of CPU hours used for a single job is the number of processors multiplied by the walltime.

Figure 8 shows the percentage of the total number of processors (in this case, 128) in use throughout the length of the throughput test A. The utilization remained close to 100 percent for the first 3 hr of the test. At that point, the scheduler began to starve the queue in order to allow the 128-processor **CTH** job to run. This job began execution around the 4-hr mark and ended shortly thereafter. Thus, for this case, the scheduler sacrificed some efficiency in order to allow a job that required the entire system to run.

Another aspect of the throughput test that is very important to measure is the total amount of memory used throughout the test. Most entries created in the accounting files on HPC systems do not keep a record of the maximum amount of memory utilized by a single job. This data would be extremely useful in creating a more representative set of benchmark tests.

In the absence of this data, several assumptions were made about the total amount of memory required during the running of a mix. First, the mixes were

Table 36
Jobs in Throughput Test A

Job	Code	Test Case	CPUs	Queue Time, hr	Wall-time, hr	CPU Hours
1	CTH	*mpi-001*	4	4.93	1.10	4.41
2	CTH	*mpi-032*	32	1.70	1.61	51.43
3	PRONTO	*brick-wall*	4	3.19	0.67	2.69
4	UNCLE	*sub-grid16.1*	16	0.03	0.18	2.88
5	GAMESS	*hedm*	96	0.04	0.23	22.16
6	LESLIE3D	*128x128x128*	32	1.87	0.91	28.97
7	GAMESS	*hedm*	96	1.41	0.24	22.93
8	UNCLE	*sub-grid16.1*	16	0.08	0.19	3.00
9	$Cobalt_{60}$	*wingflap*	16	0.35	2.83	45.30
10	NLOM	*na825*	112	0.00	1.08	121.36
11	MD-Multiscale	*test.10*	64	1.63	0.38	24.11
12	$Cobalt_{60}$	*wingflap*	16	4.88	2.23	35.76
13	PRONTO	*brick-wall*	4	3.03	0.67	2.67
14	CHARGE	*500.03*	2	4.77	1.37	2.73
15	GAMESS	*cycl*	12	2.45	0.19	2.29
16	LESLIE3D	*128x128x128*	16	0.00	1.24	19.76
17	CHARGE	*500.03*	4	2.88	0.64	2.57
18	GAMESS	*cycl*	12	2.13	0.19	2.29
19	GAMESS	*cycl*	64	1.70	0.05	3.38
20	LESLIE3D	*128x128x128*	8	1.05	2.03	16.24
21	GAMESS	*cycl*	4	2.27	0.54	2.15
22	GAMESS	*cycl*	64	1.70	0.05	3.11
23	ICEPIC	*ice.dat.16*	64	1.70	0.10	6.56
24	GAMESS	*cycl*	64	1.85	0.04	2.49
25	CTH	*arm.t1*	128	3.27	0.64	82.28
26	FEMD	*inp*	48	5.74	1.90	91.32
27	GAMESS	*cycl*	64	1.75	0.05	3.09
28	FEMWATER123	*frec.3bc.0*	16	0.66	0.41	6.55
29	ICEPIC	*ice.dat.16*	64	1.76	0.14	8.69
30	LESLIE3D	*128x128x128*	8	1.05	2.31	18.51
31	ICEPIC	*ice.dat.16*	64	1.84	0.13	8.25
32	UNCLE	*sub-grid16.1*	16	0.93	0.26	4.23
33	PRONTO	*brick-wall*	8	2.22	0.39	3.09
34	GAMESS	*cycl*	12	1.96	0.19	2.26
35	LESLIE3D	*128x128x128*	16	1.06	1.19	19.00
36	GAMESS	*cycl*	4	2.14	0.54	2.15
37	CHARGE	*500.03*	2	3.76	1.30	2.61
38	$Cobalt_{60}$	*missile*	64	3.45	0.30	19.06
39	$Cobalt_{60}$	*missile*	64	3.37	0.28	17.87
40	CTH	*mpi-001*	2	3.67	1.98	3.97
41	UNCLE	*sub-grid16.1*	16	0.83	0.25	3.97
42	UNCLE	*sub-grid16.1*	16	1.83	0.21	3.30
43	GAMESS	*cycl*	64	1.60	0.06	3.63
44	LESLIE3D	*128x128x128*	32	1.87	0.90	28.84
45	UNCLE	*sub-grid16.1*	16	1.83	0.19	3.08
46	PRONTO	*brick-wall*	4	1.90	0.70	2.78
47	GAMESS	*cycl*	4	1.90	0.53	2.11
48	CHARGE	*500.03*	4	1.90	0.65	2.62
49	CHARGE	*500.05*	64	1.68	0.09	5.46
50	SARA-3D	*sample8-064*	32	3.63	1.09	34.73

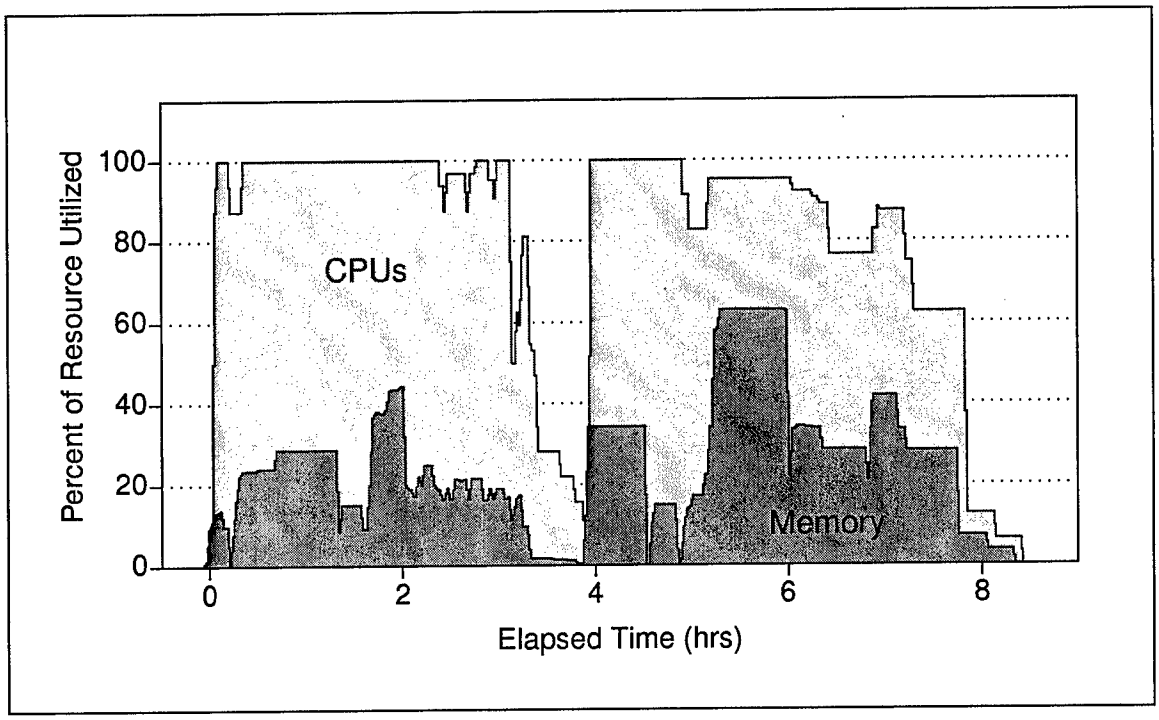


Figure 8. Resource utilization during throughput test A

not designed to test swap space. Thus, all memory requirements had to remain lower than that amount, which would force a machine to use swap space. Furthermore, at least once during the test, a large percentage of the total memory of the platform should be in use.

Figure 8 also shows the memory usage profile for Mix A; the average memory used for the duration of the test is about 16 GB, or about one-fourth of the total memory of the O2K. Between the fifth and sixth hours of the test, the total memory used jumped to around 40 GB, or more than 60 percent of the total memory.

The codes used in the second throughput test, Mix B, are given in Table 37. This second mix has the following properties:

a. A subset of the 13 application codes was used.

b. The test was designed to run 3 hr on a 128-processor machine for a total of 384 CPU hours.

c. Again, at least one full configuration job was included, and no specification was made on when it could be executed.

d. The sequence of submissions was given by the order shown in Table 38 and could not be changed.

e. The code, test case, and number of processors were specified.

f. The time of submission of the codes followed the same pattern as that used for Mix A.

Included in Table 37 are the same measurements for the applications run in Mix B as those presented for Mix A. Note that in this case the large configuration job actually failed to run; thus, its CPU hours are set to zero in the table.

Figure 9 shows the resulting utilization versus time for Mix B. The utilization profile is markedly different than that seen for the previous throughput test. This same figure also presents the memory utilization profile during Mix B. The average amount of memory use is about 21 GB, while the peak memory usage is close to the system limit but below that which would require swapping.

Table 37
Jobs in Throughput Test B

Job	Code	Test Case	CPUs	Queue Time, hr	Wall-time, hr	CPU Hours
1	CHARGE	*500.03*	4	0.05	0.68	2.72
2	CTH	*mpi-032*	32	2.26	4.02	128.59
3	CTH	*efp3d.s2*	48	0.02	0.14	3.96
4	GAMESS	*cycl*	4	0.06	0.55	2.18
5	UNCLE	*sub-grid16.2*	16	2.68	0.54	8.71
6	SARA-3D	*sample8-064*	32	0.47	1.68	53.89
7	UNCLE	*sub-grid8.2*	8	0.04	0.54	4.32
8	CTH	*efp3d.s2*	48	0.02	0.14	6.64
9	UNCLE	*sub-grid8.2*	8	0.04	0.56	4.46
10	UNCLE	*sub-grid8.2*	8	0.04	0.60	4.82
11	CTH	*mpi-032*	16	2.60	2.23	35.72
12	CTH	*mpi-001*	16	1.08	0.44	7.06
13	GAMESS	*hedm*	96	0.03	0.23	22.29
14	CTH	*arm.t1*	64	2.32	1.38	88.55
15	GAMESS	*cycl*	4	0.37	0.41	1.63
16	UNCLE	*sub-grid8.2*	8	2.43	0.87	7.00
17	CTH	*mpi-001*	4	2.35	1.34	5.35
18	GAMESS	*cycl*	4	0.27	0.71	2.83
19	SARA-3D	*sample8-064*	32	0.05	1.68	53.80
20	CTH	*mpi-001*	16	1.27	0.51	8.20
21	UNCLE	*sub-grid8.2*	8	2.19	0.87	6.95
22	GAMESS	*cycl*	4	0.10	0.67	2.67
23	CTH	*arm.t1*	128	0.00	0.00	0.00
24	UNCLE	*sub-grid8.2*	8	2.19	0.83	6.65
25	CTH	*mpi-032*	16	2.20	2.25	35.93
26	UNCLE	*sub-grid16.2*	16	2.20	0.86	13.76
27	CHARGE	*500.03*	2	2.21	1.55	3.10
28	CTH	*mpi-001*	16	1.18	0.51	8.19
29	GAMESS	*hedm*	96	1.76	0.22	20.80

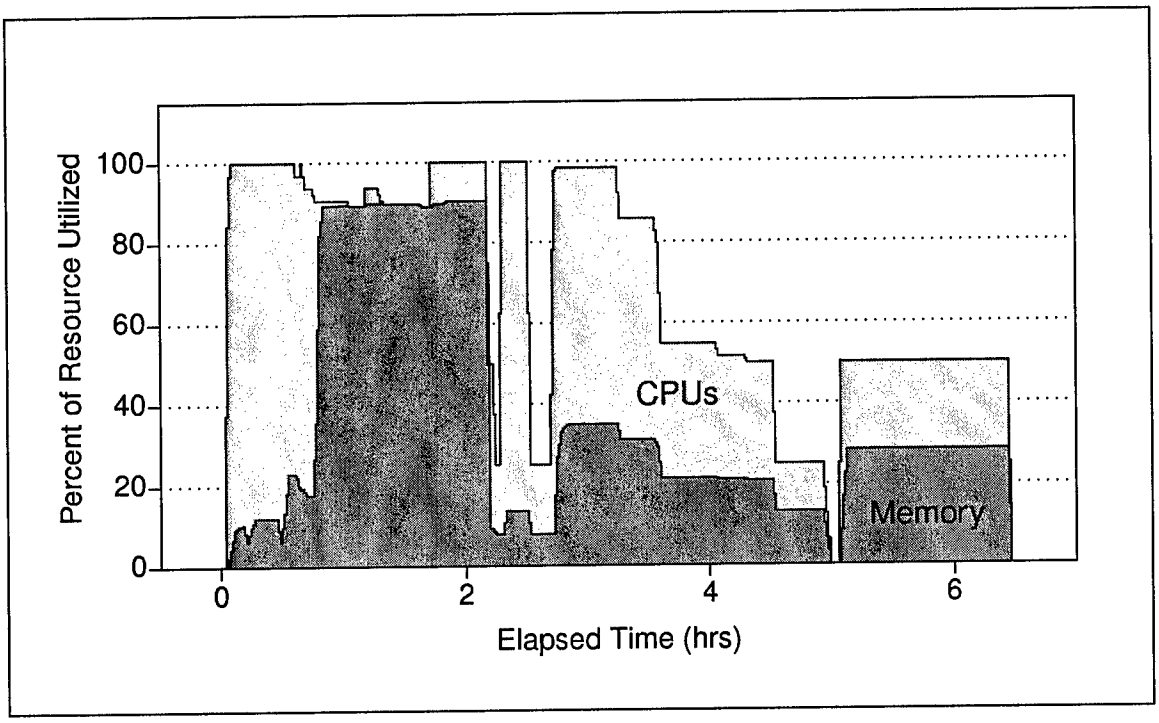


Figure 9. Resource utilization during throughput test B

Scoring the Throughput Tests

A simple method of quantifying the quality of a throughput test is to calculate the utilization efficiency, E, of each mix (Wong et al. 2000). The utilization efficiency is a rough measure of how effectively the jobs were scheduled to run on a given platform. This can be measured as the ratio of the total amount of CPU hours spent in the execution of all the individual jobs with respect to the total amount of CPU hours required to run a mix. If N_i is the number of processors requested by job i, and T_i is the amount of walltime required for that job to be run in the throughput test, then the efficiency can be caluclated as

$$E = \frac{\sum_i N_i T_i}{NT} \quad (1)$$

where N is the total number of processors available on the machine, and T is the total amount of walltime from the beginning of the first job to the end of the last job.

Table 38 lists the elapsed time, T, for Mixes A and B. Although Mix A was designed to run 6 hr, and Mix B for 3 hr, the elapsed walltime from the beginning of the first job to the end of the last job was significantly longer. Since the

Table 38
System Efficiency During Throughput Tests

Metric	Mix A	Mix B
Elapsed Time, *T* (hr)	8.42	6.67
CPU Hours	814.67	550.78
Efficiency, *E*	0.76	0.63
Total Efficiency, *TE*	0.54	0.50

timings used to create the mixes were taken on a faster architecture, it makes sense that the mixes would take longer than expected on the O2K. Also in Table 39 is a measure of the total amount of CPU hours consumed by all the individual jobs throughout the mixes. Thus, the efficiencies are easily calculated to be 0.76 for Mix A and slightly worse for Mix B at 0.63.

There are several possible explanations for why Mix B was less efficient than Mix A. First, the more jobs that occur in a mix, the more opportunity the scheduler has to backfill jobs. As more jobs are submitted to the scheduler, PBS has the luxury of pulling from a larger pool of jobs with a variety of requested resources in order to insert smaller jobs ahead of larger ones to better utilize the system. The difference between 50 jobs of Mix A and only 29 jobs in Mix B was enough to cause a significant difference in the utilization efficiency.

Another difference is the type of jobs used in the mixes. Note that for Mix A the longest running job was an 8-processor **LESLIE3D** job that ran for 2.31 hr. In Mix B, a single **CTH** job requiring 32 processors ran for more than 4 hr. Thus, for 4 hr of Mix B, the scheduler had only 96 processors in which to schedule fewer jobs than the initial 29.

For the measurements taken in this work, the utilization efficiency is actually just a measure of the scheduler's ability to increase the throughput of a system. Other specific scheduling strategies or system features were not tested with these mixes. For example, most platforms have the ability to checkpoint and restart jobs if the machine suffers from some catastrophic failure. Thus, the entire time spent in computation before the job was killed is not completely lost. Only that time since the last checkpoint cannot be recovered. Also, other features, such as gang scheduling or swapping, that might improve efficiency have not been investigated (Wong et al. 1999).

Though the efficiency is an excellent measure of the quality of a throughput test, it does not take into account any machine-specific features. In other words, a single throughput test may be measured to have a high efficiency on two different platforms, and yet, the total walltime required to run the mix may be dramatically different. To be more specific, a job run in dedicated mode will probably take less walltime than a job run in production mode (i.e., a throughput test). Because jobs run in nondedicated mode must compete with each other for

resources, they will probably run slower. No measure of this contention between jobs is included in Equation 1.

In order to better describe the total efficiency of a throughput test, it is necessary to time each run of a throughput test in dedicated mode. Since in dedicated mode only a single job is running (ignoring any contention that may occur between the system and the job), the best possible, or shortest, time is obtained for a job on a specific architecture. These times can be used to further extend the concept of throughput efficiency to compute a total efficiency by

$$TE = \frac{\sum_i N_i t_i}{\sum_i N_i T_i} \tag{2}$$

where t_i is the dedicated time of the ith job run on N_i number of processors, and T_i is the amount of walltime required for the ith job executed in the throughput on the same number of processors.

In this way, if the architecture handles the contention of resources perfectly, then the time of run in a throughput test, T_i, will be equal to the dedicated time, t_i, and a score of unity will be obtained.[1] Therefore, the closer to this perfect score, the better a system handles multiple jobs executing at the same time.

Throughout the creation of this benchmark, dedicated and nondedicated timings were gathered for all the representative codes and test cases. Using these "best" times, the total efficiency of both throughput tests as calculated by Equation 2 was computed and is also included in Table 39. Note that in both cases a significant drop in the total efficiency was realized, with Mix A again showing a higher efficiency. However, here the disparity between the two mixes is much less than for the total efficiency.

An obvious effect of this type of comparison is that Equation 2 can be used to compare different architectures. By obtaining the dedicated timings of all the jobs in the throughput tests across all the SUTs, a measure of the best possible throughput efficiency can be obtained. By using the shortest dedicated time across all platforms, t_i in Equation 2, for a given job, a theoretical limit of the total number of CPU hours needed for the throughput test can be obtained. Comparing this to the actual number of CPU hours allows for the comparison of throughput tests run across multiple platforms.

Also, in Mix B, the large **CTH** job did not complete for this particular running of the test. One way to compensate for failure of a job in a throughput test is to assume that the job completed in some specified amount of time (e.g., the dedicated time for the job on that platform), and then add the corresponding number of CPU hours to the total for the throughput test. However, since the purpose of this particular job was to test the scheduling of a full configuration run during the mix, it is significant that this job did not complete. Therefore, the authors recommend that the failure of a full configuration job in a throughput test

[1] This is, of course, assuming that consistent timings can be obtained. There will be variations in the timings of a job from run to run.

invalidate the test. The importance of running a job that uses all of the processors is key to the understanding and scoring system performance.

Finally, suppose when checking for correctness it is found that certain jobs ran to completion but did not produce the right answer. For an isolated instance of this in a throughput test, a solution similar to that applied above can be used. One suggestion is to use the "longest" dedicated time measured across all platforms to apply somewhat of a penalty for not producing an acceptable answer. This too is arbitrary and would have to be set in the rules of the throughput tests before it was given to the vendors.

Future Throughput Tests

Throughput tests give an excellent measure of the ability of a new system to perform given tasks representative of a specific site's workload. Currently, no synthetic benchmark or tool is capable of taking measurements on a working environment in order to model the behavior of the environment on a new platform. Until such time when that ability becomes available, defining mixes of application codes with representative test cases is by far the best measure of a new system's capability. Combining this knowledge with that obtained from dedicated runs gives the purchaser of new architecture confidence that the best decision has been made.

In addition to the above tests, more can be studied during the running of the throughput mixes. One of the greatest concerns of HPCs today is their ability to recover from a catastrophic failure. In the event of a system panic and reboot, how much time is lost in the recovery process? Therefore, a simple addition to the throughput test is to schedule a system shutdown and reboot in the mix. This would redefine the efficiency of the throughput test slightly since the time of shutdown/reboot would need to be included. Hence, Equation 1 becomes

$$E = \frac{\sum_i N_i T_i}{N(T+S)} \tag{3}$$

where S is the total amount of time of the shutdown/reboot. With the inclusion of the shutdown/reboot, a new system may showcase its ability to checkpoint and restart jobs. Without checkpointing, not only is the walltime of the reboot lost, but also the time spent by the jobs currently running at the time of the shutdown. As systems and users' jobs get larger, this time could be substantial.

Furthermore, in the throughput tests described here, the full configuration jobs often ended up waiting a large amount of time in the queue. Hence, these jobs were usually run last. In a working environment where new jobs are constantly being submitted, a large configuration job may never run. Thus, the queue must force the job to be run by increasing its priority to a higher level than the newly submitted jobs. This will cause the queue to starve jobs for resources; that is, processors will be idle even though valid jobs may be run. The ability of a

queuing system to push large jobs through can easily be tested by requiring that the full configuration jobs be run at certain times during the throughput test. Although this idea was discussed for TI-01 benchmark, it was not implemented.

Another novel idea is to allow a vendor or the system itself to generate either a throughput mix or the most efficient order of submission. In this work, the order and time of the submission of jobs was dictated to the vendors. One can imagine, though, that slight changes in the order of submission could result in dramatic changes in the efficiencies of the throughput tests. Also, since the vendors know their architectures best, the vendors could define a throughput test with certain criteria to best showcase their platform. This would aid in the allocation of resources of CPU hours on specific architectures for certain types of applications.

5 Final Remarks

Directions for Future Research

Throughout the creation and testing of this benchmark test package, several practical considerations were noted that would have enhanced its quality. With additional time, more applications could have been selected, obtained, tested, and included in the final suite of codes, and the test package's portability, compactness, and scalability improved. The end result would have been a more representative and easy-to-use test package. Thus, the vendors would have been able to do a more thorough job of running the required tests, perhaps even performing extensive code optimizations. These considerations suggest several possible avenues for further work; these are discussed below.

Construction of a kernel-based HPC benchmark

As noted in the introduction, kernel-based benchmark test packages are an alternative to the use of entire application codes. Because they are significantly smaller than full-fledged applications, they are inherently more portable and maintainable. If properly constructed, they may be designed to accept an input parameter that controls the size of the problem to be solved; this helps address the issue of benchmark scalability and helps prevent test package obsolescence. Specifically, this effort would proceed as follows:

a. Identify representative codes from the DoD HPC user community, drawing from the experiences of the TI-01 test suite.

b. Use appropriate performance tools (e.g., **TotalView** and PAPI) to identify the computation/communication kernels in these codes.

c. Extract those kernels and prepare driver programs to read or construct on the fly appropriate scalable test data.

d. Validate the kernels using appropriate statistical techniques, showing the correlation between the performance of the kernel and the performance of the parent code.

e. "Productize" the test package so that it may be installed and executed conveniently on a range of HPC platforms.

HPC scheduler simulation

Crucial to the performance of any computer system is the effectiveness of the job scheduler. In the case of HPC systems, this is particularly true due to the large number of CPUs that may be idle if the batch scheduler is unable to pack jobs tightly. Another issue related to the job scheduler involves maintaining high utilization while also providing short job turnaround times. Unfortunately, these two goals are often conflicting, and each can be only partially achieved.

Studying these issues is facilitated by a scheduler simulator, which accepts a stream of simulated jobs as input and records utilization and wait time metrics as outputs. Each simulated job consists of a number of CPUs, estimated walltime, actual walltime, and start time. A prototype simulator would serve as the starting point for this work. Specifically, this effort would proceed as follows:

a. Gather actual historical job data from ERDC HPC systems as input for the simulator.

b. Modify the simulator to handle multiple job classes (challenge, primary, background).

c. Compare the effect of using different scheduling algorithms on utilization and wait time; specifically study no-backfill, strict-backfill, and relaxed-backfill strategies with various tuning parameters.

d. Study the effect of more accurate user estimates of job walltime on utilization and wait time; currently there seems to be little penalty with greatly overestimating a job's run time.

HPC system simulation

The TI-01 benchmark activity is evidence of the HPCMPO's interest in forecasting the performance of application codes on future HPC systems. One approach, of course, is to extrapolate from benchmark data; however, using trend lines beyond the range of the gathered data is always risky. Another approach is to gather data on application programs (e.g., number of CPUs, number and size of messages passed, number of floating-point operations, and number and size of I/O operations) and construct a statistical, least-squares type model of system performance tailored to a system with known or projected hardware performance characteristics. This approach is dependent on gathering sufficient data points on which to base the model and may not accurately forecast the effects of interactions between the independent variables. A third approach, the one proposed here, involves the construction of a simulation model of a system's performance. Specifically, each of the CPUs on a parallel system would be represented by several queues: one to provide compute services, one for I/O services, one for sending messages, and one for receiving messages. The number, configuration, and service times associated with these queues would be chosen to reflect the

hardware and software architecture of the actual or hypothetical system being studied. To run a job, profiling data would be gathered, using tools such as **Vampir**, on candidate applications. These data, represented as a series of requests for service on each CPU, would be fed as input to the simulator. Specifically, this effort would proceed as follows:

a. Gather job profile data from one or more actual DoD applications to serve as input for the simulator.

b. Write a prototype simulator to model one of the existing DoD HPC systems (e.g., IBM SP3); ideally, a simulation language such as Simscript would be used for this task, but it could be written in a scripting language.

c. Compare the simulator's predicted performance with the modeled system's actual performance in order to calibrate the model.

d. Perform a simulation of the same code with more CPUs and/or a different code to validate the simulator's performance.

e. Integrate this simulator with the scheduler simulator above to simulate a job stream.

Conclusions and Recommendations

The benchmark test package documented in this report was developed to respond to a critical need of the DoD HPCMP. In the absence of a standard DoD HPC benchmark and with the necessity to procure the next generation of HPCs, a benchmark test package was constructed using actual application codes in use at all of the MSRCs. Based on the analysis of system usage at each site, a suite of 13 codes was selected to represent the different CTAs. These 13 codes, with multiple inputs and processor configurations, were used to construct both dedicated and throughput tests to be run by the vendors.

Practically, the construction and execution of a complicated benchmark test package is a nontrivial task. Even excluding the considerable effort required to execute individual test cases, construct the throughput mixes, and document the package, merely identifying and acquiring the codes and their inputs is a time-consuming activity.

Though the resulting benchmark test package provides a useful tool for the evaluation of HPC systems, the authors sincerely hope that the research directions noted above will be pursued. Specifically, the quality of the HPCMP benchmark test package would be greatly improved if the package was revised and enhanced on a yearly basis. Perhaps CHSSI and Challenge Projects should be required to produce a representative kernel (code and input data) to help keep the test package up-to-date.

As part of such an ongoing activity, some effort should be devoted to automating the installation and execution of the test package. This would include

construction of a software framework written in **sh**, **Perl**, **make**, and/or some other combination of languages and tools. Compilation, test execution, and gathering of metrics would be handled by software in a reliable and consistent fashion. The end result would be a more representative, easy to use, and, most importantly, believable test package that could better evaluate new architectures for DoD use.

References

Alavi, A., Kohaniff, J., Parrinello, M., and Frenkel, D. (November 1994). "Ab initio molecular dynamics with excited electrons," *Phys. Rev. Lett.* 73 (19), 2599-2602.

Attaway, S., Barragy, T., Brown, K., Gardner, D., Hendrickson, B., Plimpton, S.J., and Vaughan, C. (November 1997). "Transient solid dynamics simulations on the Sandia/Intel teraflop computer." *Supercomputing '97.* IEEE Computer Society, San Jose, California.

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. (March 1994). "The NAS parallel benchmarks" RNR-94-0007, NASA Ames Research Center, Moffett Field, California.

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. (Fall 1991). "The NAS parallel benchmarks," *International Journal of Supercomputing Applications* 5 (3), 63-73.

Bailey, D., Harris, T., Saphir, W., Wijngaart, R. van der, Woo, A., and Yarrow, M. (December 1995). "The NAS parallel benchmarks 2.0" NAS-95-020, NASA Ames Research Center, Moffett Field, California.

Bailey, D.H. and Barton, J.T. (August 1985). "The NAS kernel benchmark program" NASA Technical Memorandum 86711, National Aeronautics and Space Administration Ames Research Center, Moffett Field, California.

Baldwin, B.S. and Barth, T.J. (1991). "A one-equation turbulence transport model for high Reynolds number wall-bounded flows" AIAA Paper No. 91-0610, American Institute of Aeronautics and Astronautics.

BBN Technologies (2000). "Applied physics and tactical sonar" http://www.gte.com/aboutgte/gto/bbnt/apts/offerings/sara.html, BBN Technologies, Cambridge, Massachusetts.

Benedek, R., Alavi, A., Seidman, D.N., Yang, L.H., Muller, D.A., and Woodward, C. (10 April 2000). "First principles simulation of a ceramic/metal interface with misfit," *Phys. Rev. Lett.* 84 (15), 3362-3365.

Benedek, R., Minkoff, M., and Yang, L.H. (15 September 1996). "Adhesive energy and charge transfer for MgO/Cu heterophase interfaces," *Physical Review B* 54 (11), 7697-7700.

Benedek, R., Seidman, D.N., Minkoff, M., Yang, L.H., and Alavi, A. (15 December 1999). "Atomic and electronic structure and interatomic potentials at a polar ceramic/metal interface: 222MgO/Cu," *Physical Review B* 60 (23), 16094-16102.

Brooks, F.P., Jr. (1975). *The mythical man-month.* Addison-Wesley, Reading, Massachusetts.

Buchholz, W.A. (1969). "A synthetic job for measuring system performance," *IBM Systems Journal* 8 (4), 309-318.

CHARGE Development Team (9 April 1999). "CHARGE user's manual" Version 0.1.2, unpublished document.

Comptroller General of the United States (22 October 1982). "Benchmarking: Costly and difficult but often necessary when buying computer equipment and services" GAO/AFMD-83-5, General Accounting Office, Washington, DC.

Computational Combustion Laboratory, School of Aerospace Engineering (2000). "Georgia Tech Computational Combustion Laboratory" http://heron.ae.gatech.edu/ccl/index.html, Georgia Institute of Technology, Atlanta, Georgia.

Computational Sciences Branch, Aeronautical Sciences Division, Air Vehicles Directorate (June 1999). "$Cobalt_{60}$ Home Page" http://www.va.afrl.af.mil/vaa/vaac/cobalt/, Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio.

Computational Simulation and Design Center, College of Engineering (2000). "SimCenter: Research" http://www.erc.msstate.edu/simcenter/research.html, Mississippi State University, Mississippi State, Mississippi.

Curnow, H.J. and Wichman, B.A. (February 1976). "A synthetic benchmark," *The Computer Journal* 19 (1), 43-49.

Cybenko, G. (1990). "Supercomputer performance evaluation and the perfect club." *Proceedings of the 1990 International Conference on Supercomputing.* Association for Computing Machinery, New York, 254-266.

Department of Chemistry, Iowa State University (25 March 2000). *GAMESS user's guide.* Department of Chemistry, Iowa State University, Ames, Iowa.

Dongarra, J.J. (January 2004). "Performance of various computers using standard linear equations software" CS-89-85, Computer Science Department, University of Tennessee, Knoxville, Tennessee.

Dongarra, J.J., Martin, J.L., and Worlton, J. (July 1987). "Computer benchmarking: Paths and pitfalls," *IEEE Spectrum* 24 (7), 38-43.

Dougherty, D. and Robbins, A. (1997). *sed & awk.* O'Reilly & Associates, Sebastopol, California.

Duffy, D., Fahey, M., Hensley, J., Oppe, T., Ward, W., and Alter, R. (2001). "Performance comparison of the IBM SMP POWER3 pre- and post-Switch2 on applications codes" ERDC/MSRC/PET 01-05, U.S. Army Engineer Research and Development Center Major Shared Resource Center, Vicksburg, Mississippi.

Duffy, D., Ward, W.A., Jr., Fahey, M., Fahey, R., and Oppe, T. (2000). "Analysis of HPC usage: ERDC MSRC" ERDC/MSRC/PET TR/00-36, U.S. Army Engineer Research and Development Center Major Shared Resource Center, Vicksburg, Mississippi.

Federal Computer Performance Evaluation and Simulation Center (August 1979). *Use and specifications of remote terminal emulation in ADP system acquisitions.*, FPR 1-4.11 General Services Administration Automated Data and Telecommunications Services, Washington, DC.

Fernand, E. and Menon, S. (1993). "Mixing enhancement in compressible mixing layers: An experimental study," *AIAA J.* 31, 278-285.

Ferrari, D. (July-August 1972). "Workload characterization and selection in computer performance measurement," *Computer* 5 (4), 18-24.

Flanagan, L.M. and Flanagan, D.P. (March 1989). "PRONTO3D: A three-dimensional transient solid dynamics program" SAND87-1912, Sandia National Laboratories, Albuquerque, New Mexico.

Godunov, S.K. (1959). "A finite difference method for the numerical computation and discontinuous solutions of the equations of fluid dynamics," *Math. Sb.* 47, 271-295.

Gottlieb, J.J. and Groth, C.P.T. (1988). "Assessment of Riemann solvers for unsteady one-dimensional inviscid flows of perfect gases," *J. Comp. Phy.* 78, 437-458.

Grassl, C.M. and Schwarzmeier, J.L. (Spring 1990). "A new measure of supercomputer performance: Results from the perfect benchmarks," *Cray Channels* 12 (1), 14-16.

Gustafson, J., Rover, D., Elbert, S., and Carter, M. (November 1990). "SLALOM: The first scalable supercomputer benchmark," *Supercomputing Review*, 56-61.

Hennessy, J.L. and Patterson, D.A. (1996). *Computer architecture: A quantitative approach, 2nd ed.*. Morgan Kaufman, San Francisco, California.

Henning, J.L. (July 2000). "SPEC CPU2000: Measuring CPU performance in the new millennium," *Computer*, 28-35.

Hensley, J., Duffy, D., Fahey, M., Oppe, T., Ward, W.A., Jr., and Alter, R. (2001). "Peformance comparison of SGI Origin 2800 and SGI Origin 3800 on application codes" in preparation, ERDC MSRC/PET, Vicksburg, Mississippi.

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. (July 1993). "CTH: A software family for multi-dimensional shock physics analysis." *Proceedings of the 19th International Symposium on Shock Waves.*, R. Brun and L.D. Dumitrescu, eds., I, Marseille, France, 377-382.

High Performance Computing Modernization Program (2004). "Welcome to the HPCMP" http://www.hpcmo.hpc.mil/, High Performance Computing Modernization Program Office, U.S. Department of Defense, Arlington, Virginia.

Holt, M. (October 1996). "Review of Godunov methods" NASA Contractor Report 198332, ICASE Report No. 96-25, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia.

Hurlburt, H.E. and Thompson, J.D. (1980). "A numerical study of loop current intrusions and eddy shedding," *Journal of Physical Oceanography* 10, 1611-1651.

Jones, R. (1975). "A survey of benchmarking: The state of the art." *Benchmarking: Computer Evalaution and Measurement.*, N. Benwell, ed., Hemisphere Publishing Corporation, Washington, DC, 15-23.

Kalia, R.K., Vashishta, P., Benedek, R., Woodward, C., Rao, S.I., and Dimiduk, D.M. (7-10 June 1999). "Computational assisted development of high temperature structural materials." *Department of Defense High Performance Computing Modernization Program Users Group Meeting.*, S. Scherr, ed.,, *http://www.hpcmo.hpc.mil/Htdocs/UGC/UGC99/papers/chal2-3/chal2-3.pdf.*

Karypis, G. (2000). "ParMETIS: Parallel graph partitioning" http://www-users.cs.umn.edu/~karypis/metis/parmetis/, Department of Computer Science and Engineering, University of Minnesota, St. Paul, Minnesota.

Karypis, G. and Kumar, V. (1995). "Multilevel graph partitioning and sparse matrix ordering." *Proceedings of the 1995 International Conference on Parallel Processing.*.

Karypis, G. and Kumar, V. (1998). "A parallel algorithm for multilevel graph partitioning sparse matrix ordering," *Journal of Parallel and Distributed Computing* 48, 71-85.

Kernighan, B.W. and Hamilton, P.A. (1973). "Synthetically generated performance test loads for operating systems." *1st Annual SIGME Symposium on Measurement and Evaluation.* Association for Computing Machinery, New York, 121-126.

Kim, W.-W., Menon, S., and Mongia, H. (1999). "Numerical simulations of reacting flows in a gas turbine combustor," *Combustion Science and Technology* 143, 25-62.

Letmanyi, H. (March 1984). "Assessment of techniques for evaluating computer systems for federal agency procurements" NBS Special Publication 500-113, National Bureau of Standards, Washington, DC.

Lin, H.C.J., Richards, D.R., Talbot, C.A., Yeh, G.T., Cheng, J.R., Cheng, H.P., and Jones, N.L. (1997). "FEMWATER: A three-dimensional finite element computer model for simulating density-dependent flow and transport in variably saturated media" CHL-97-12, Coastal and Hydraulics Laboratory, U.S. Engineer Waterways Experiment Station, Vicksburg, Mississippi.

MacCormack, R. (1969). "The effect of viscosity in hypervelocity impact cratering" AIAA Paper No. 69-354, American Institute of Aeronautics and Astronautics.

McGlaun, J.M., Thompson, S.L., Kmetyk, L.N., and Elrick, M.G. (1990). "CTH: A three-dimensional shock wave physics code," *Int. J. Impact Engng.* 10, 351.

McMahon, F. (December 1986). "The Livermore Fortran kernels: Computer test of the numerical performance range" UCRL-53745, Lawrence Livermore National Laboratory, Livermore, California.

Menon, S. (1995). "Secondary fuel injection control of combustion instability in a ramjet," *Combustion Science and Technology* 100, 385-393.

National Bureau of Standards (15 May 1977). "Guidelines for benchmarking ADP systems in the competitive procurement environment" FIPS PUB 42-1, National Bureau of Standards, Washington, DC.

National Bureau of Standards (18 September 1980). "Guidelines for constructing benchmarks for ADP system acquisition" FIPS PUB 75, National Bureau of Standards, Washington, DC.

National Energy Research Scientific Computing Center / Lawrence Berkeley National Laboratory (November 2003). "Top 500 supercomputer sites" http://www.top500.org/,, Celle, Germany.

Newman, H. and Graham, R. (February 2001). "HPCMPO synthetic benchmark results" Powerpoint presentation and associated documentation on CD-ROM prepared for the HPCMPO,.

Saavedra-Barrera, R.H. (August 1990). "The SPEC and perfect club benchmarks: Promises and limitations." *Hot Chips Symposium 2.*, Santa Clara, California.

Sandia National Laboratories (October 1998). "CTH 3D Eulerian shock code" http://sherpa.sandia.gov/9231home/cth/cth-frame.html, Sandia National Laboratories, Albuquerque, New Mexico.

Sandia National Laboratories (2000). "Sandia Engineering Analysis Code Access" http://endo.sandia.gov/SEACAS/Documentation/SEACAS.html, Sandia National Laboratories, Albuquerque, New Mexico.

Saphir, W., Wijngaart, R. van der, Woo, A., and Yarrow, M. (August 1996). "The NAS parallel benchmarks 2.1 results" NAS-96-010, NASA Ames Research Center, Moffett Field, California.

Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M., and Montgomery, J.A. (1993). "General Atomic and Molecular Electronic Structure System," *J. Comput. Chem.* 14, 1347-1363.

Spalart, P.R. and Allmaras, S.R. (1992). "A one-equation turbulence model for aerodynamic flows" AIAA Paper No. 92-0439, American Institute of Aeronautics and Astronautics.

Strang, W.Z. (21 July 2000). *$Cobalt_{60}$ user's manual.* CFD Research Branch Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio.

Tracy, F. and Richards, D. (Summer 2000). "Interview with users...Dr. Fred Tracy," *The Resource* U.S. Army Engineer Research and Development Center Major Shared Resource Center, Vicksburg, Mississippi, 13-15.

van Leer, B. (1979). "Towards the ultimate conservative difference scheme, V. A second-order sequel to Godunov's method," *J. Comp. Phy.* 32, 101-136.

Wallcraft, A.J. (2000). "The NRL Layered Ocean Model users guide" NOARL 35, Naval Research Laboratory, Stennis Space Center, Mississippi.

Wallcraft, A.J. and Moore, D.R. (1997). "The NRL Layered Ocean Model," *Parallel Computing* 23, 2227-2242.

Weicker, R.P. (October 1984). "Dhrystone: A synthetic systems programming benchmark," *Communications of the ACM* 27 (10), 1013-1030.

Weicker, R.P. (August 1988). "Dhrystone benchmark: Rationale for version 2 and measurement rules," *SIGPLAN Notices* 23 (8), 49-62.

Wong, A.T., Oliker, L., Kramer, W.T.C., Kaltz, T.L., and Bailey, D.H. (May 1999). "System utilization benchmark on the Cray T3E and IBM SP." *Fifth Workshop on Job Scheduling..*

Wong, A.T., Oliker, L., Kramer, W.T.C., Kaltz, T.L., and Bailey, D.H. (November 2000). "ESP: A system utilization benchmark." *Proceedings of SC2000.*.

Yeh, G.T. (1987). "3DFEMWATER: A three-dimensional finite element model of water flow through saturated-unsaturated media" ORNL-6368, Oak Ridge National Laboratory, Oak Ridge, Tennessee.

Yeh, G.T. (1990). *3DLEWASTE: A hybrid Lagrangian-Eulerian finite element model of waste transport through saturated-unsaturated media.* Department of Civil Engineering, The Pennsylvania State University, University Park, Pennsylvania.

Appendix A
Glossary of Acronyms

1-D	One-dimensional
2-D	Two-dimensional
3-D	Three-dimensional
AFB	Air Force Base
AFRL	Air Force Research Laboratory (at Wright-Patterson AFB, Ohio)
ARL	Army Research Laboratory (MSRC site at Aberdeen, Maryland)
ASC	Aeronautical Systems Center (MSRC site at Wright-Patterson AFB, Ohio)
BLAS	Basic Linear Algebra Subroutines
BWG	Benchmark Working Group
CCM	Computational Chemistry and Materials Science (CTA)
ccNUMA	Cache-coherent nonuniform memory access
CEA	Computational Electromagnetics and Acoustics (CTA)
CEN	Computational Electronics and Nanoelectronics (CTA)
CFD	Computational Fluid Dynamics (CTA)
CFS	Computational Field Simulation
CHSSI	Common HPC Software Support Initiative (HPCMP initiative)
CMG	Computational Migration Group (former name of CS&E at the ERDC MSRC)

CPU	Central processing unit
CS&E	Computational Science and Engineering Group (new name of CMG at the ERDC MSRC)
CSC	Computer Sciences Corporation
CSM	Computational Structural Mechanics (CTA)
CTA	Computational Technology Area
CTH	**CSQ** to the Three-Halves (application program). **CSQ** stands for "**CHARTD** SQuared;" **CHARTD** stands for "Computational Hydrodynamics And Radiative Thermal Diffusion;" **CHARTD**, **CSQ**, and **CTH** are 1-D, 2-D, and 3-D codes, respectively.
CWO	Climate/Weather/Ocean Modeling and Simulation (CTA)
DoD	Department of Defense
EQM	Environmental Quality Modeling and Simulation (CTA)
ERDC	Engineer Research and Development Center (MSRC site at Vicksburg, Mississippi)
FEMD	Finite Element Molecular Dynamics (application program)
FFT	Fast Fourier transform
flop	Floating-point operation(s); plural determined by context
fma	Fused multiply-and-add (IBM RS/6000 machine instruction)
FMS	Forces Modeling and Simulation/C4I (CTA)
GAMESS	General Atomic Molecular Electronic Structures System (application program)
GB	Gigabyte, 2^{30} bytes (approximately one billion bytes)
HPC	High Performance Computing
HPCMP	HPC Modernization Program
HPCMPO	HPCMP Office (at Arlington, Virginia)
HPF	High Performance Fortran
IBM	International Business Machines
ICEPIC	Improved Concurrent Electromagnetic Particle-In-Cell (application program)

IMT	Integrated Modeling and Test Environments (CTA)
KB	Kilobyte, 2^{10} bytes (approximately one thousand bytes)
Ksgs	Kinetic energy at the subgrid scale, same as the kinetic energy of the turbulence
LES	Large-eddy simulation
LESLIE3D	Large-Eddy Simulation LInear Eddy model in 3 Dimensions (application program)
LOC	Lines of code
MB	Megabyte, 2^{20} bytes (approximately one million bytes)
Metis	A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices; "metis" is the Greek word for wisdom'
Mflop/s	Millions of floating-point operations per second
MHz	Millions of hertz
MIPS	Millions of instrictions per second, or the name of a manufacturer of microprocessors, depending on the context
MPI	Message Passing Interface
MSRC	Major Shared Resource Center (HPCMP initiative)
MSU	Mississippi State University (at Starkville, Mississippi)
MUSCL	Monotone upwind scheme for scalar conservation laws (due to van Leer)
NAS	National Aerospace Simulation
NAVO	Naval Oceanographic Office (MSRC site at Stennis Space Center, Mississippi)
NLOM	NRL Layered Ocean Model (application program)
NP	Number of processors
NRL	Naval Research Laboratory (at Washington, D.C.)
NSWC	Naval Surface Warfare Center (at Dahlgren, Virginia)
O2K	SGI Origin 2000
O3K	SGI Origin 3000

ParMetis	Parallel version of **Metis**
PBS	Portable Batch System
POWER	Performance Optimization With Enhanced RISC (first generation implementation of IBM's RS/6000 microprocessor architecture)
POWER2SC	POWER2 Super Chip (single-chip second generation implementation of IBM's RS/6000 microprocessor architecture)
PVM	Parallel Virtual Machine (application program to facilitate running parallel applications)
QR	Not an acronym; refers to the factorization of a matrix A into the matrix product of Q times R (QR) where Q is orthogonal and R is upper triangular. The QR algorithm is used to determine the eigenvalues of the matrix A.
RCS	Radar cross section
SARA	Structural Acoustic Radiation Analyzer (family of application programs)
SARA-2D	SARA for 2-D problems (application program)
SARA-3D	SARA for 3-D problems (application program)
SEACAS	Sandia National Laboratories Engineering Analysis Code Access System
SGI	Silicon Graphics, Inc.
SHMEM	Shared Memory Access Library (Cray software product)
SIP	Signal/Image Processing (CTA)
SMP	Symmetric multiprocessor
SNL	Sandia National Laboratories (at Albuquerque, New Mexico)
SP2	IBM SP with POWER2SC nodes
SPAWAR	Space and Naval Warfare
SUT	System under test
TB	Terabyte, 2^{40} bytes (approximately one trillion bytes)
TI-01	Technology Insertion 2001 (HPCMP acquisition program)
UNCLE	UNsteady Computation fieLd Equations (application program)
USS_UNCLE	Utilities for Solver Setup for **UNCLE**

Appendix B
Benchmark Code Issues

This appendix discusses issues for each code that had to be overcome before the test application was included in the final benchmark package. These issues include the portability of the codes as well as compiliation problems/solutions encountered by the authors.

CHARGE

CHARGE is quite portable; the benchmark team experienced minimal difficulty in running it on a T3E, SP2, SMP, and O2K. No source code modifications were performed. To improve performance, compiler options were specified to inline the functions **BuildCell**, **GetU1StencilData**, **LoadFluxMatrices**, **ComputeFaceFlux**, and **UpdateCellSolution**.

As noted above, four steps (**patchtrans** through **CHARGE**) are required to run **CHARGE**. Furthermore, input data resided in remotely mounted disks, while the better performance was attained when running **CHARGE** from local workspace directories. To facilitate this process and spare the user the task of interactively setting up the various input files, a script, **charge.run**, was created to perform all of the job steps. Even though the first three serial steps were run on the parallel system, their total run time compared to the charge step was negligible.

$Cobalt_{60}$

On SGI platforms, it was discovered that the **$Cobalt_{60}$** and **ParMETIS** source codes should be compiled with the **`-64`** option (for 64-bit pointers) to enable the larger problems to run. Thus, the compiler flags for the Origins in *make-cobalt* were

```
FFLAGS="-64 -freeform -O2 -mips4 -OPT:fast_sqrt=ON \
   -TARG:platform=ip35"
CFLAGS="-64 -I ParMetis.v1.0 -Dsgi"
```

Also, some MPI environment variables had to be increased from their default SGI values to the following to be sufficient for all three test problems:

```
setenv MPI_TYPE_MAX 200000
setenv MPI_GROUP_MAX 500
setenv MPI_COMM_MAX 1000
```

Finally, the vendor submissions from SGI indicated that the **dplace** utility was used for optimal memory and process placement in a dedicated run. Only logical placement was requested. The placement files (named *dplace.in*) were all of the form

```
memories (NPROCS+3)/4 in topology cube
threads NPROCS+1
distribute threads 1:NPROCS across memories
```

where **NPROCS** was edited to be the number of processors requested by the job. In general, **dplace** was used for the dedicated applications, but not used for the jobs in the throughput mixes. The execute line in the *CoMPIRUN* file was then changed by SGI to

```
timex -pmt $RUN -np $proc $DPLACE $SCRATCH/cobalt \
   < $SCRATCH/CoIN.$$ \
   > $SCRATCH/CoOUT.$$
```

where the **DPLACE** environment variable was set to **`dplace -place dplace.i`** to allow the use of **dplace** during dedicated runs.

CTH

It should be noted that for some platforms, namely the Cray T3E, patches need to be applied to the CTH99 version of the code. These patches are *Mar99.patch1* and *Mar99.patch2*. After applying both patches, the code still did not execute correctly on the T3E. After some investigation, it was discovered that Sandia does not recommend that the second of the two patches be applied. After applying only the first patch, the code was successfully compiled and produced the correct results on the T3E.

FEMD

During the process of compiling and testing the code for different platforms, makefiles were created for different architectures. With minor modifications, namely to some timing routines and to the C preprocessor statements, the code successfully compiled and completed on the SP2, SMP, O2K, and T3E at ERDC. However, it was found that on some platforms, higher levels of optimization could cause the code to hang. Further work will be needed to find and explore those routines which may be corrupted by the compiler's optimization.

FEMWATER123

Since **FEMWATER123** was highly portable and representative of the type of simulations being performed in the CTA of Environmental Quality Management (EQM), it was selected as the sole representative of EQM. However, for a fixed problem size, **FEMWATER123** does not scale very well as the number of processors increases. For the problem size that was included in the benchmark, the best timings were obtained between 32 and 40 processors. Due to a crossover between being computationally bound at low numbers of processors and communicationally bound at high numbers of processors, using more than 40 processors will actually slow the code down.

FEMWATER, like many other parallel codes, has poor scaling behavior (i.e., for the problem sizes of interest to researchers, using more than 32-64 CPUs does little to improve performance). In fact, it was precisely because of this type of behavior that **FEMWATER123** was included in the benchmark suite, since it represents a significant class of DoD usage of HPC resources.

GAMESS

GAMESS is quite portable; the benchmark team experienced minimal difficulty in running it on a T3E, SP2, SMP, O2K, and O3K. The T3E version used SHMEM as the communication mechanism, while all other versions used MPI. No source code modifications were performed. Some minor changes were also made to path names in the compile and execute scripts supplied with the code's distribution in order to conform to the local programming environment.

As with several other codes, input data resided in remotely mounted disks, while better performance was attained when running **CHARGE** from a workspace directory local to the parallel system. To facilitate the process of setting up a job, a script was created to perform all of the job steps. Also, **GAMESS** uses the Basic Linear Algebra Subroutines (BLAS); linking to a BLAS version tuned for a particular platform should improve performance.

ICEPIC

During compilation of **ICEPIC**, messages of the following form may result:

```
"objects.h", line 82.18: 1506-159 (E)
   Bit-field type specified for ZShim is not valid.
   Type unsigned assumed.
```

These warnings concern enumerations. According to some compiler **man**-pages, the enumerations are legal in spite of the warnings. They may be ignored.

The following compiler command:

```
mpcc -I../mp -DSP2 -DMPI -Q -DKR_C -O3 -qstrict \
   -qarch=pwr2 -qtune=pwr2 -c VT.c
```

may cause the following warning message:

```
"VT.c", line 26.7: 1506-356 (W)
   Compilation unit is empty.
```

It is empty unless **VAMPIR** is used. This message may also be ignored.

LESLIE3D

Before compiling the code, the user must specify the dimensions of the domain with a PARAMETER statement in an include file, as in

```
parameter (igsiz1=216, jgsiz1=216, kgsiz1=216)
```

as well as a processor grid configuration, as in

```
parameter (isiz1 = igsiz1/4)
parameter (isiz2 = jgsiz1/4)
parameter (isiz3 = kgsiz1/4)
```

for a $4 \times 4 \times 4$ processor grid. A recompilation of the code is necessary for any change in the domain or processor grid dimensions. After compilation, an executable *mixing.x* must be run to generate a binary file *mixing.data* prior to execution of the **LESLIE3D** executable. The *mixing.data* file is the same for any processor grid configuration for a given domain grid, but must be regenerated with any change in the domain grid dimensions.

The code was modified slightly to use the **access = 'append'** clause for opening a particular file when using a Fortran 77 and to use the **position = 'append'** clause when using Fortran 90. The programmer uses the pre-processor directive **-DF90** in the **FFLAGS** definition in the makefile to select the Fortran 90 syntax.

To turn off OpenMP on SGI platforms, the code was compiled without the **-mp** flag and the following environment variables were set:

```
export OMP_NUM_THREADS=1
export MPC_NUM_THREADS=1
export MP_SET_NUMTHREADS=1
```

For the SMP runs, the code was compiled without the `-qsmp=omp` flag and the following environment variables were set:

```
export XLSMPOPTS="parthds=1"
export OMP_NUM_THREADS=1
```

On SGI platforms, the **LESLIE3D** source codes were compiled with the **-64** option (for 64-bit pointers).

Finally, the vendor submissions from SGI indicated that the **dplace** utility was used for optimal memory and process placement in a dedicated run. Only logical placement was requested. The placement files (all named *dplace.in*) were all of the form

```
memories (NPROCS+1)/4 in topology cube
threads NPROCS+1
distribute threads 1:NPROCS across memories
```

where **NPROCS** was edited to be the number of processors requested by the job. In general, **dplace** was used for the dedicated applications, but not used for the jobs in the throughput mixes.

MD-Multiscale

MD-Multiscale was less portable than some of the other codes in this test package. The benchmark team was unable to get the code to produce correct results on the T3E, and two vendors that participated in these tests reported difficulties related to excessive memory requirements. There are 229 allocate and 119 deallocate statements in the code, so a memory leak of some sort may be the cause. Also the computational difficulties associated with solving multiscale problems and with the physics of fractures may have contributed to the problems encountered. In spite of this, the code ran correctly on an SP2, SMP, O2K, and O3K.

A subtle error that appears to affect only execution on IBM POWER3-based systems was discovered and corrected. The MPI library call **MPI-CART-CREATE** permits rank reassignments. Reassignment is not performed in IBM Parallel Environment 2.4, but is performed in Parallel Environment 3.1. This reordering can be defeated simply by changing the fifth argument to **MPI-CART-CREATE** from `.true.` to `.false.`[1]

[1] The authors acknowledge Farid Parpia of IBM for discovering this error.

A few source code modifications were performed on four C source code files: *force-interface.edip.c*, *etime.c*, *pmine.c*, and *timing.c*. These changes were made during the attempt to move the code to the T3E and involved simplifying the system-dependent way in which underscores were added to function names for C-Fortran interfaces. Another minor change required by the T3E involved replacing the call to the LAPACK double complex routine **zhegv** with the single complex routine **chegv**. This involved defining appropriate C preprocessor variables. Finally, the system of makefiles used to build **MD-Multiscale** was extensively reworked to simplify building programs on multiple platforms. A top level makefile was created to replace a shell script, and the makefiles in the subdirectories were modified. However, additional work should be done in this area to centralize the way in which compiler options are specified for compilations in different subdirectories.

Finally, input data resided on disks remote to the parallel system, while the better performance was attained when running **MD-Multiscale** from local workspace directories. To facilitate this process and spare the user the task of interactively setting up the various input files, a script, **mdmulti.run**, was created to perform all of the job steps.

NLOM

Since **NLOM** has been ported and tested on many platforms, only one significant issue had to be resolved. On the ERDC O3K, if the latest SGI scientific library (scs) was used for **NLOM**'s FFT routines, the code would crash. If, however, the code was linked to the previous scientific library *complib.sgimath*, the code would work properly. Note that the code also includes its own FFT routines if a vendor library is not available. The supplied FFT routines were used for benchmarking on the O3K.

PRONTO

It is necessary to install a significant portion of the SEACAS system in order to use **PRONTO**. **PRONTO** has been compiled and run on a wide variety of platforms and the distribution includes scripts to build and install on many machines. However, **PRONTO** was not successfully ported to the Cray T3E.

SARA-3D

The vendor submissions from SGI indicated that the **dplace** utility was used for optimal memory and process placement in a dedicated MPI-only run. Only logical placement was requested. The placement files (all named **dplace.in**) were all of the form

```
memories (NPROCS+1)/4 in topology cube
threads NPROCS+1
distribute threads 1:NPROCS across memories
```

where **NPROCS** was set to the number of MPI processes requested by the job. In general, **dplace** was used for the dedicated applications, but not used for the jobs in the throughput mixes. **dplace** was not used for OpenMP-only or hybrid MPI/OpenMP runs.

For pure OpenMP or hybrid MPI/OpenMP runs, the **OMP-NUM-THREADS** environment variable was set to the number of OpenMP threads. For hybrid MPI/OpenMP runs, the **MPI-DSM-PPM** environment variable was set to 4/**OMP-NUM-THREADS** to allow the threads to operate on the same memory as its parent MPI process. On the O3K, 4 CPUs share a common memory, so for 4 OpenMP threads,

```
setenv MPI_DSM_PPM 1
```

while for 2 OpenMP threads,

```
setenv MPI_DSM_PPM 2.
```

UNCLE

Only one input problem was obtained from the developers for the benchmark tests; the developers supplied the grid files for the problem to be executed for only 8 and 16 processors. Hence, the vendors were unable to perform any scaling analysis. Thus, a problem that can be scaled and have the grid distributed for arbitrary numbers of processors would have been more desirable and representative.

The code uses **gmake** to compile, and some of the routines have to be compiled with different optimization levels on different platforms. Furthermore, as of this writing, a bug had been found in the new MIPSpro version 7.3.1.1 compiler on the Origins. The code had to be compiled using version 7.3 in order to create an executable program.

Appendix C
Test Case Prioritization

This appendix describes the methodology for assigning scores to each job, defining how well each job would fit into a mix. These scores were used in a simple greedy algorithm to define the set of codes and input decks used in the throughput tests.

First the notation for subscripts and superscripts is explained.

CTA	Subscript denoting data associated with the CTA of a job's application program.
JOB	Subscript denoting data associated with a particular job.
NPB	Subscript denoting data associated with a job's number-of-processors bin. For purposes of categorization, the range of processors is subdivided into "bins" (e.g., a job requiring 12 CPUs falls into the 9-16 CPUs bin).
PRG	Subscript denoting data associated with a job's application program.
TOT	Subscript denoting data associated with overall total node hours.
cur	As the job selection algorithm proceeds, it keeps running totals of various node hour quantities. This superscript denotes the current value of such a total.
prj	This superscript denotes the projected value of a total after some job's node hours are added.
tgt	This superscript denotes the target value of a node hour total specified by the algorithm's input.
NP	Number of processors on the SUT; in this case, 128.

WT	Target walltime for the throughput test; here, 6 hr for Test A, and 4 hr for Test B.
CH^{tgt}_{TOT}	Total target node hours for the throughput test; = $NP * WT$.
CH^{tgt}_{CTA}	Target node hours for current job's CTA.
CH^{cur}_{CTA}	Current node hours for current job's CTA.
CH^{tgt}_{NPB}	Target node hours for current job's NPB.
CH^{cur}_{NPB}	Current node hours for current job's NPB.
CH^{tgt}_{PRG}	Target node hours for current job's PRG.
CH^{cur}_{PRG}	Current node hours for current job's PRG.
F^{cur}_{CTA}	Current fraction of node hours needed by CTA of current job. $= (CH^{tgt}_{CTA} - CH^{cur}_{CTA})/CH^{tgt}_{TOT}$
F^{prj}_{CTA}	Projected fraction of node hours needed by CTA of current job. $= (CH^{tgt}_{CTA} - CH^{cur}_{CTA} - CH_{JOB})/CH^{tgt}_{TOT}$
F^{cur}_{NPB}	Current fraction of node hours needed by NPB of current job. $= (CH^{tgt}_{NPB} - CH^{cur}_{NPB})/CH^{tgt}_{TOT}$
F^{prj}_{NPB}	Projected fraction of node hours needed by NPB of current job. $= (CH^{tgt}_{NPB} - CH^{cur}_{NPB} - CH_{JOB})/CH^{tgt}_{TOT}$
F^{cur}_{PRG}	Current fraction of node hours needed by PRG of current job. $= (CH^{tgt}_{PRG} - CH^{cur}_{PRG})/CH^{tgt}_{PRG}$
S_{CTA}	CTA score for current job; weights are subjective. $= 0.6F^{cur}_{CTA} + 0.4F^{prj}_{CTA}$
S_{NPB}	NPB score for current job; weights are subjective. $= 0.4F^{cur}_{NPB} + 0.6F^{prj}_{NPB}$
S_{PRG}	PRG score for current job. $= F^{cur}_{PRG}$
$S_{CTA \times NPB}$	$CTA \times NPB$ cross product score for current job; favors jobs that have both the neediest CTA and neediest NPB..br = $\sqrt{\max 0, S_{CTA} * S_{NPB}}$
$S_{CTA \times PRG}$	$CTA \times PRG$ cross product score for current job; favors jobs that have both the neediest CTA and neediest PRG..br = $\sqrt{\max 0, S_{CTA} * S_{PRG}}$
P	Subjective priority for current job.
S	Overall score for current job .br $= 0.5S_{CTA} + 0.3S_{NPB} + 0.1S_{CTA \times NPB} + 0.1S_{CTA \times PRG} + P$

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
August 2004	Final report	

4. TITLE AND SUBTITLE

Development of the TI-01 Application Benchmark

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Robert W. Alter, Daniel Q. Duffy, Mark R. Fahey, Rebecca A. Fahey,
Jeffrey L. Hensley, Thomas C. Oppe, William A. Ward, Jr.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Sciences Corporation, Major Shared Resource Center, U.S. Army Engineer Research and Development Center, 3909 Halls Ferry Road, Vicksburg, MS 39180-6199
Data Management Consultants, Major Shared Resource Center, U.S. Army Engineer Research and Development Center, 3909 Halls Ferry Road, Vicksburg, MS 39810-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC/ITL TR-04-4

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road, Vicksburg, MS 39180-6199

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report documents the design, construction, and validation of the Technology Insertion 2001 (TI-01) application benchmark test package for the Department of Defense (DoD) High Performance Computing Modernization Program Office (HPCMPO). This test suite contained 12 application programs chosen to be representative of the DoD's HPC workload. The report justifies this claim of representativeness. On a given HPC system, each application program was typically tested using at least two sets of input data, and each program/input data test case was always tested at multiple numbers of processors. These tests were done in dedicated mode with no other applications on the system, so as to obtain a presumably "best possible" performance. This collection of tests was repeated on five different HPC systems: a Cray T3E, an IBM POWER2-based SP, and IBM POWER3-based SP, an SGI Origin 2000, and an SGI Origin 3800. Additionally, two throughput tests constructed to represent HPCMPO workloads were conducted. Again, the methodology used to obtain representativeness is described.

15. SUBJECT TERMS

Computer benchmarking
Computer performance evaluation
High Performance Computing Modernization Program (HPCMP)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED		100	