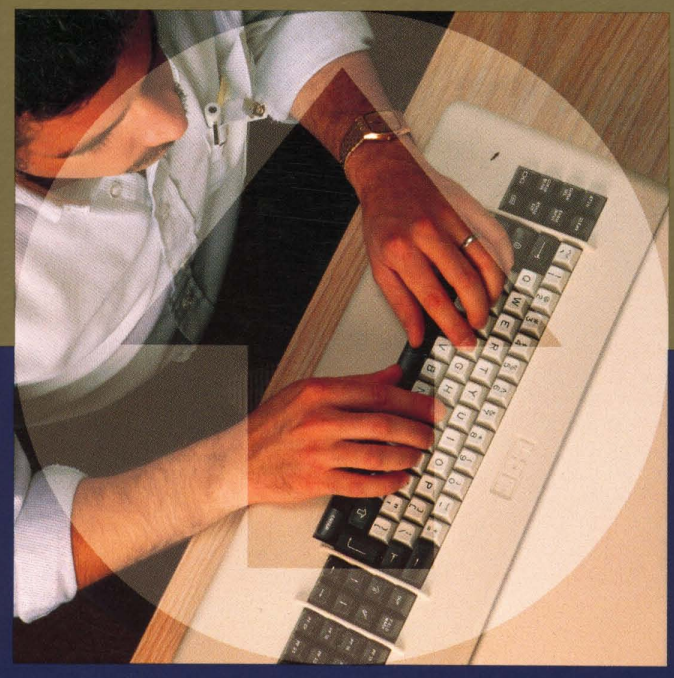
CRAY CHANNELS

FALL 1992 • A CRAY RESEARCH, INC. PUBLICATION

SOFTWARE DIRECTIONS



Announcing the CRAY S-MP Superserver

CRAY CHANNELS

In this issue

Cray Research devotes half of its R&D budget to software development each year. The company is committed to delivering reliable, high-performance software to users to maximize their productivity. Cray Research software helps users apply the full performance capabilities of Cray Research hardware systems to technical problem solving.

This issue of CRAY CHANNELS focuses on software, the engine that drives the computing machine. Cray Research's Executive Vice-President of Development, Bob Ewald, introduces the issue with an overview of the company's technical strategy. Feature articles cover Cray Research's MPP development project, visual tools for program development, the PVM and HeNCE networking environments, Fortran 90, and real-time applications. Our regular departments cover networking and applications software releases and recount the computerized quest to design a better boat for the America's Cup races.

Cray Research supercomputers—fueled by high-performance software—power the pursuit of solutions to the world's technical grand challenges. Cray Research will continue to invest in applications software, parallel processing capabilities, and greater operating system and compiler convenience and efficiency. These efforts support the mission-critical work of users in industries and noncommercial agencies worldwide.

Features

5

Departments

CRAY CHANNELS is a quarterly publication of the Cray Research, Inc. Marketing Communications Department, Tina M. Bonetti, Director. It is intended for users of Cray Research computer systems and others interested in the company and its products. Please mail feature story ideas, news items and Gallery submissions to CRAY CHANNELS at Cray Research, Inc., 1440 Northland Drive, Mendota Heights, Minnesota 55120. Subscription inquiries and address changes should be sent Attention Department D.

Volume 14, Number 4

Editorial staff
Ken Jopp, managing editor
Mary Miller, associate editor
Heidi Hagen, contributing editor
Rich Brueckner, contributing editor
Chris Kruell, contributing editor

Design and production
Barbara Cahlander, typesetter, production artist
Eric Hanson, graphic designer
James Morgan, art director
Cynthia Rykken, graphic designer

INTERVIEW ROBERT EWALD

Staying on the leading edge

Cray Research's Strategic Directions

Robert H. Ewald, executive vice president of development, is responsible for hardware and software product development at Cray Research. He joined the company in 1984 and has served as executive vice president of the software division, vice president of software, vice president of commercial marketing, and director of education industry marketing. Prior to joining Cray Research, Ewald was division leader for computing and communications at Los Alamos National Laboratory. In this interview, he outlines Cray Research's technical strategy and shares his thoughts on general trends that will characterize supercomputing in the 1990s.

CRAY CHANNELS: Cray Research has been very successful with its parallel-vector hardware architecture. What is the company's technical strategy for the future, in terms of high-end architectures?

Robert Ewald: Our high-end architectures will follow two paths: parallel-vector and parallel-scalar (MPP). We will develop the parallel-vector architecture to run much faster and include many more processors. In the near term, we will continue to develop the parallel-vector CRAY Y-MP C90 system and its successor, which we call the Triton project internally. The CRAY Y-MP C90 system has a clock cycle of about 4 nanoseconds. We see ways to cut that in half and then cut it in half again in the next five to eight years.

At the same time, we will provide a massively parallel solution to address highly parallel applications. We will provide customers with tools to help them get their work done in the most effective manner possible. Our solution is to offer the fastest parallel-vector architecture and complement that with an MPP system designed for real-world production environments.

Massively parallel processing at Cray Research

CC: Cray Research plans to offer the first of its MPP systems in 1993. What will be the key advantages of this system?

RE: Our massively parallel system will be the first MPP based on high-speed supercomputer technology combined with a very strong high-performance software base. We believe that we have learned a great deal by watching other projects, which has allowed us to add many new ideas and technologies to our design. We expect the result to be a machine that provides significantly better sustained performance than has been seen from early MPP systems.

Peak speeds are interesting to read about, but our customers want real, sustained performance on their applications. That is what we have always delivered on our parallel-vector machines, and will continue to deliver with our MPP. We will use the state-of-the-art Alpha RISC microprocessor from Digital Equipment Corporation in our system and provide a very high speed processor interconnect network that is capable of sustaining very high global memory bandwidth. Our architecture is MIMD with a globally-addressable, physically distributed memory subsystem that supports a flexible Fortran programming model. In addition, we have designed special hardware features to support the synchronization mechanisms that are needed for efficient parallel programming.

We expect our MPP system to be a very capable computational engine in its own right, and we will complement it with the proven UNICOS functionality and high-performance I/O capability of the CRAY Y-MP and CRAY Y-MP C90 systems. Most of our competitors are offering just the massively parallel component with weak I/O and immature supporting software. Our ability to leverage proven CRAY Y-MP and CRAY Y-MP C90 software enables us to offer users a complete environment with parallel processing, vector processing, and scalar processing integrated into a coherent whole. The vector/scalar component offers production quality I/O, proven software tools, high-speed networking, and the ability to distribute a single application between a vector/scalar environment and a highly parallel environment. This total computational capability simply does not exist on any

MPP or clustered workstation system in the market today.

CC: What bridge will connect Cray Research's parallel-vector system and its MPP system?

RE: There are two I/O paths: high-speed and low-speed channels over to the CRAY Y-MP system, and parallel I/O directly off the MPP system through our Model E I/O subsystem to a variety of peripherals. This allows us to provide a platform for heterogeneous computing and to leverage a great deal of technology and software that's been developed in the UNICOS operating system environment.

CC: As you incorporate new technologies to take advantage of performance improvements, how do you plan to protect customer software investments?

RE: As the hardware evolves on both the parallel-vector and the MPP systems, we will protect customer software investments by providing a common operating environment across the various architectures that supports industry standards. We will first build upon the proven UNICOS software platform on our parallel-vector systems and bridge that environment onto the MPP. The microkernel we will use for the MPP will lag behind the parallel-vector system in terms of capabilities and features, so they'll be out of sync initially, but our plan is to have them converge rather than go in different directions.

CC: What will the user interface look like for Cray Research's MPP system?

RE: The user environment will be almost transparent because the initial MPP Fortran compiler and the programming tools reside on the parallel-vector system and can generate code for either side. Our programming model allows users to distribute code across both machines. Programmers can use the tools, compilers, and network connections with which they are familiar.

CC: What tools will be available to help users take advantage of the MPP capabilities?

RE: We have an MPP emulator that runs on the parallel-vector system. Users can run their code through it to obtain a runtime statistical analysis of the code, which should help them structure their code for most efficiency. We also have a whole set of tools on the parallel-vector system that will be extended to operate on MPP codes. So from the CRAY Y-MP side you can look into your code running on the MPP and debug it. We have a new debugger that's going to run on both sides. It was developed by MPP pioneers Bolt, Beranek, and Newman, and we're confident that it's the best available product of its type.

We'll also provide users with tools similar to ATExpert so they can look into codes running on the MPP side and see the same types of profiling and performance monitoring information that they can see today on the CRAY Y-MP side. We'll also develop very fast links to run production codes on the MPP system. Over time, more of this software will move to the MPP system.

CC: Is there more to building a massively parallel system than just hooking together fast RISC processors?

RE: Yes. Any RISC processor, even Alpha, lacks some of the features necessary to build a massively parallel supercomputer, such as adequate physical memory address space, low latencies of memory access, and fast synchronization support. We are adding significant hardware around Alpha, using CRAY Y-MP C90 integrated circuits to cover these areas. That's going to be a key differentiator. We're not doing it in a vacuum—we're doing it with help from the DARPA community and a customer advisory group.

However, we believe that success with MPP is not really a hardware issue but a software issue—how do we use all these processors simultaneously? I think we'll go through a learning period with our initial MPP customers, just as we did with our customers in the early days of CRAY-1 systems and vectorization. Learning is not even the right term—it's more of a learning and building process together.

We can vectorize things today that people thought they'd never be able to vectorize ten years ago. The same thing will be true with MPP—there are problems today that we don't think will ever run effectively on MPP systems. However, that will not be the case in all instances—we will learn more over time about programming MPP systems and how to run a wider range of problems.

CC: How will your integrated MPP solution be superior to stand-alone MPP systems?

RE: Some codes will continue to run best on the parallel-vector side, and some codes will run best on the MPP side. We think that there are also some codes that will run best spread across both. By providing a high-performance bridge from the very strong hardware/software platform that we have established with our parallel-vector systems, we think people will be able to exploit the huge performance potential of MPP more easily.

SPARC technology

CC: Why is Cray Research endorsing SPARC technology with the new CRAY S-MP system and its technology agreements with Sun Microsystems?

RE: The key to our future is being able to solve our customers' problems with the best computational tools. More and more applications are being developed for workstations. When users run out of horsepower on their workstation, for whatever reason, we want them to be able to run their problems on our systems. To get those applications to run on a Cray Research system, we have to build bridges. More importantly, though, to make those bridges easy to operate, we want to take some of our existing software and move it down onto the workstation platform.

Robert Ewald

At the same time we started to implement this software strategy across the network, we were presented with a unique opportunity. Floating Point Systems had built a SPARC-based product with much higher performance than Sun's high-end products. They had gone out of business, and we thought that buying FPS would allow us to offer a product that picks up where Sun leaves off. We felt that with some of our core strengths in memory systems, I/O, and parallel processing, we could add value to the SPARC architecture. Then we could move some of our software onto the desktop through what we now call Cray Research Superservers.

In a general sense, that's what we're trying to do with our technical strategy: focus on the high end, push the technology down as far as it will go, implement lower-cost technology to come up the architectural chain, look at a different architecture where it has promise—for example, MPP because of the promise of real peak performance advantages—and drive some of our software out onto the desktop so applications being developed for the desktop can run across the entire network.

CC: What Cray Research technology will be available to Sun workstation users and what benefits will it provide?

RE: First, they probably will get a Fortran 90 compiler that they can't get from other vendors and our powerful math libraries. This Fortran 90 compiler will be optimized for workstations, so it will run well and be compatible with the Fortran code that exists on the big system. When users have big problems—huge problems that they can't run on their workstations—they can run them on the big systems easily.

CC: Does this sharing of technology provide a bridge to real supercomputing?

RE: Exactly. But these bridges work both ways—the bridge allows code to go off the Cray Research system onto servers and workstations. But I believe there is a much bigger potential for large workstation-initiated applications to move up to our supercomputers. The way to accomplish that is by providing compatibility between the workstation and the Cray Research system at a source-code level.

Looking ahead: Cray Research software

CC: What do you see in the future for Cray Research software?

RE: I think the biggest near-term challenge is to make MPP usable, which means everything from a fast, efficient operating system, through compilers, tools, debuggers, and all the rest. We also have to make sure the programming model is truly usable, efficient, and portable. So when users think about problems in their own terms, we can easily map those problems onto the hardware. I think that's the number one challenge.

The second big challenge is enabling the Cray Research supercomputer, the MPP, the workstations, and the file servers to all work cooperatively. This network supercomputing model really puts the Cray Research system on the user's desktop through the network. So when it becomes part of their workstation, they'll know they don't have to think about the Cray Research system—it's just there when they have a problem they want to run quickly. We envision automatic optimization software in the network that recognizes that a problem would take two hours to run on a server and that it makes more sense to run it on the Cray Research system in 10 seconds.

Quality

CC: How have customer expectations changed over the years, and how will you engineer quality into leading-edge systems in the future?

RE: Customer expectations for high-performance systems have changed dramatically. The CRAY-1 system ran at about one hundred hours mean time between interrupts in 1976. In 1976 that was terrific, but today that would be completely unacceptable.

I think the bottom line is that we're providing computational tools. Tools are like utilities. It's like electrical power—it has to be there day in and day out, and you have to be able to count on it. We have to build systems that run nearly nonstop and can monitor themselves. New systems have to have resilience features built into the hardware and software because, with the speeds and the technology we're pushing, we're going to have some failures in the chips themselves.

We have to build systems that are smart enough to be able, both in hardware and software, to identify a failing component and switch it out. At the same time, the system has to allow maintenance to be done concurrently so that we can continue to provide that computational tool and utility.

We've taken this approach on the MPP hardware side, in that we have a number of spare CPUs and we can reconfigure the network to compensate if one is failing. We have a UNICOS resiliency project and a hardware resiliency project, as well, so we have to push all those ahead.

As we look to the future of high-performance computing, I think it is essential for us to ensure that we meet our customers' expectations in everything we do. That is real quality and the key to our success.

A N N O U N C I N G

The CRAY S-MP Superserver

The world's most powerful SPARC computer

To bridge the gap between today's supercomputers and SPARC workstations, Cray Research, Inc., recently announced the CRAY S-MP Superserver, the world's most powerful SPARC computer. The CRAY S-MP Superserver opens the door to thousands of SPARC applications while providing a familiar interface to the supercomputer.

The CRAY S-MP Superserver is an ideal server for today's distributed environments. It meets the demands of large multiuser applications, such as E-CAD, mass storage management, and distributed graphics. Its flexible, modular design provides a wide range of capability and performance, satisfying requirements ranging from departmental compute servers to central data center systems. As a high performance compute server, the CRAY S-MP system accelerates scalar applications without recompiling code. It also can function as a high performance departmental visualization facility, freeing supercomputer resources for large computational tasks. Computationally intensive visualization packages such as Applications Visualization System (AVS), that do not run efficiently on a workstation, perform well on a CRAY S-MP system—especially one equipped with an attached parallel processor (APP).

CRAY S-MP systems are built on high performance SPARC scalar processors, with 67 MIPS performance per processor. Systems can have up to eight of these 15-nanosecond processors, yielding as much as 533 MIPS peak performance. The CRAY S-MP operating system provides symmetric multiprocessing capability, allowing multiple large jobs to be processed simultaneously. All of the system's processors can work together to solve large, complex problems, or they can be allocated to individual tasks.

The CRAY S-MP system can be configured with high-speed vector processors and APPs that complement the SPARC scalar processors to meet many types of application needs. The scalar processors, vector processors, and APPs are linked by a gigabyte-per-second system interconnect. More important, the CRAY S-MP system lets users access these different processor types from a common development environment.

Up to two SPARC vector processors can be configured with four scalar processors. Each vector processor has eight 1024-element vector registers and a peak rate of 67 MFLOPS.

One or two APPs can be configured with as many as 84 Intel i860 processing elements each. These parallel processors deliver up to 6 GFLOPS and excellent price/performance for compute-intensive applications such as signal processing, image processing, and electromagnetics.

All processors share up to 4 Gbytes of real memory, supplemented by a robust virtual memory management system designed for large jobs in multiuser environments. The memory system provides the ease-of-use of virtual memory while delivering the performance associated with a real memory architecture.

The CRAY S-MP Superserver is an ideal server for today's distributed environments. As a high-speed communications server, the CRAY S-MP system supports VMEbus and HIPPI channels, and FDDI, TCP/IP, UltraNet, Ethernet, and DECnet communications products. The native HIPPI implementation employs four full simplex channels per interface, yielding an aggregate bandwidth of 400 Mbytes per second. Using HIPPI connections, the CRAY S-MP system can provide up to 512 Gbytes of online disk storage at sustained bidirectional transfer rates of 128 Mbytes per second. Using these high-speed communications protocols and supported high performance mass storage devices, a CRAY S-MP Superserver running file storage and management software can be the nerve center of a file system that manages terabytes of data quickly and efficiently.

The operating system for the CRAY S-MP Superserver is an enhanced implementation of the UNIX-based Solaris operating system from Sun Microsystems, Inc. Extensions enable parallel processing, symmetric multiprocessing, vector processing, and parallel/vector processing to take advantage of the hardware's performance features. Fortran and C compilers for the CRAY S-MP system are compatible with both Sun Microsystems and Cray Research compilers.

The CRAY S-MP system can run more than 4500 SPARC application binary interface (ABI) applications without modification. The performance of many applications can be enhanced further by using optimized mathematical libraries that are loaded automatically at run time, or by recompiling to take advantage of the processing capability of SPARC vector processors or APPs. The CRAY S-MP Superserver can accelerate vector and parallel applications while providing the breadth of software and productivity tools available for SPARC systems.

Whether used as a high performance, high-bandwidth compute server, a powerful visualization server, a high-speed communications server, or a terabyte-capacity file server that fits seamlessly into a SPARC environment, the CRAY S-MP Superserver delivers.

Massively Parallel Performance at Cray Research

R. Kent Köninger, Cray Research, Inc.

Massively parallel processors (MPPs) offer exciting new opportunities for supercomputing. These computer systems, which combine potentially thousands of microprocessors able to operate concurrently on user problems, are not yet suited for general-purpose supercomputing. They do, however, offer unique opportunities for very high performance on certain problems. Production workloads in supercomputing environments typically comprise a mix of scalar, vector, and parallel codes. To maintain the highest levels of production throughput in these heterogeneous environments, Cray Research is developing a powerful MPP system that will be coupled closely to the parallel/vector/scalar architecture of the CRAY Y-MP supercomputer. The programming model we have developed for this system is a Fortran-based multiple-instruction, multiple-data (MIMD) model that allows programmers to specify data sharing and work sharing.

Our MPP development program is founded on three assumptions:

- ☐ MPPs are technology driven. Their success depends on advances in the performance and functionality of commodity reduced-instruction-set-computing (RISC) microprocessors. Although these technology advancements will be evolutionary, not revolutionary, they will proceed rapidly in the foreseeable future.
- ☐ MPPs will continue to be communications bound, not compute bound. Memory will drive affordability; interconnect technology will drive efficiency; and I/O will drive accessibility.
- ☐ MPPs must be optimized for next-generation problems. These problems are substantially larger and more complex than today's supercomputer problems. Problems that combine scientific disciplines, such as structural analysis and fluid dynamics, into integrated applications will require orders of magnitude more computing power than do today's typical applications. This multidisciplinary approach will require distributed, heterogeneous, and closely coupled computing. MPPs are necessary, but not sufficient, tools to solve these problems; integrated computing systems that combine diverse architectures are required.

Microarchitecture and macroarchitecture

Writing code for an MPP system is a substantial labor investment that should be preserved when upgrading to newer and faster MPPs. For this reason, we have adopted a two-tier architecture that comprises a macroarchitecture, which will remain consistent, and a microarchitecture, which will vary as technologies advance.

Macroarchitecture

Programs written for our initial MPPs will port easily to future systems because the con-

sistency of the macroarchitecture will maintain source code compatibility for programs written in our MPP programming model. This compatibility will protect customers' software investments. We have a three-phase plan to implement an MPP product using this macroarchitecture:

Phase	Performance rating	Time frame
1	300 GFLOPS peak	1993
2	1 TFLOPS peak	1995
3	1 TFLOPS sustained	1997

Macroarchitecture characteristics

The macroarchitecture will be characterized by

- Physically distributed memory. Each processing element (PE) will be coupled closely with its own dynamic random-access memory (DRAM).
- Logically shared memory. Each PE will be able to address every other PE's memory directly, as if it were its own; no message-passing instructions will be required. The system will be able to address terabytes of memory.
- Nonuniform Memory Access (NUMA). Each PE will be able to access its own DRAM faster than it can access other PEs' DRAMs.
- Latency hiding. Special communications hardware will allow data in remote PEs to be copied into a local PE before it is needed.
- Fast synchronization. The hardware will provide a rich set of synchronization primitives for both single-instruction, multiple-data (SIMD) and MIMD programming styles. These will include lightweight-barrier and data-driven synchronization primitives.
- High bandwidth parallel I/O. Our MPP products will connect to our I/O subsystems with multiple high-speed channels (200 Mbytes per second in each direction, per channel).
- Microkernel-based operating system. Each PE will have a microkernel that sends interprocessor communications (IPCs) to other PEs and sends IPCs to the closely coupled CRAY Y-MP vector processors.
- Heterogeneous computing. The MPP will be coupled closely to vector supercomputers. The combined systems should provide the highest levels of available performance across the full range of supercomputing needs.
- Scalability. The interconnect design will allow customers to scale up easily from hundreds to thousands of PEs.
- Reliability. Software-configurable redundant hardware will be included so that processing can continue, without hardware maintenance, should a PE fail.

Microarchitecture

We will embed within our macroarchitecture a variable microarchitecture that will not compromise the macroarchitecture compatibility. The ability to vary the microarchitecture will allow us to incorporate the microprocessor advances required to achieve TFLOPS of sustained performance. Our flexible microarchitecture will be based on the fastest microprocessors available for each generation of MPPs. Consequently, instruction set compatibility will not necessarily be maintained between generations. Our first MPP microarchitecture is based on the Alpha microprocessor from Digital Equipment Corporation, a commodity RISC chip capable of executing 150 MFLOPS. It is cache based with pipelined functional units, can issue multiple instructions per cycle, and supports 64-bit floating-point and logical arithmetic.

Technology

Our MPP systems will take advantage of our proven supercomputer packaging techniques, including short wire lengths, fast clock rates, fast switching circuits, fast chips, and supplemental communication logic for latency hiding and synchronization. This approach should deliver a number of system benefits:

- Low latencies for remote memory accesses
- High bandwidth for remote memory references
- Fast global synchronization
- Highly effective latency hiding

These characteristics, in turn, should minimize communication overhead and maximize sustained computation rates.

Today's commodity RISC microprocessors lack some capabilities necessary for supercomputer-class performance. These shortcomings include:

- A lack of communication primitives necessary to assemble the chips into a coherent MPP supercomputer
- Too few address bits to reach the hundreds of gigabytes of memory included in our MPP systems
- An inability to hide the long memory latencies inherent in nonuniform-memory-access MPPs
- A lack of mechanisms for lightweight, fast, global-barrier and data-driven synchronization

We plan to circumvent these shortcomings by surrounding the RISC chips with appropriate communications hardware. In this way we plan to transform thousands of independent commodity RISC processors into a supercomputer-class MPP system—one that can address terabytes of memory, minimize communication overhead, and provide excellent lightweight synchronization.

By applying cooling technologies developed for our supercomputer systems, we can exploit the speed advantages of chips whose operating temperatures exceed conventional cooling capabilities. We will remain free to choose either commodity microprocessors or specialized microprocessors, whichever is appropriate, as the MPP technology advances toward a teraflops of sustained performance.

Cray Research's MPP operating system

Our MPP operating system will be a microkernel-based implementation of our UNICOS operating system. Each PE will run a minimal

microkernel to handle frequently used functions, such as IPC and memory management. Higher level UNIX system calls will be sent by way of an IPC mechanism to a closely coupled UNIX agent running on the UNICOS system.

This arrangement will provide maximum UNIX functionality for our initial MPP offering. Programs running on our MPP systems will be well integrated into a normal UNIX environment. For example, interactive MPP programs will remain connected to standard-in and standard-out. They will have access to the same file systems, network protocols, and batch queuing as all other UNICOS applications.

Our goal is to migrate functionality from the UNIX agent directly onto the PEs. Initially, all I/O control and data paths will pass through the UNIX agent. After the initial release of the MPP operating system, we will move the I/O data path to the PEs so that I/O will flow from the PEs to the I/O devices without passing through the UNIX agent. Eventually we also will move I/O control to the PEs. This incremental approach will allow the operating system to grow gracefully and will allow native PE functionalities to be added in order of importance.

Application focus

To ensure that customers have a rich selection of MPP application software, we have implemented a three-part applications strategy:

- ☐ Existing MPP codes will port easily to our MPP system, typically with improved performance. We will support message-passing, data-parallel, and HPF-like programming models.
- ☐ MPP application specialists at customer sites and at our Eagan, Minnesota, facility are converting key applications.
- ☐ Our Applications Department has developed longstanding relationships with many third-party application software vendors and is assisting companies in converting their supercomputing applications to the new MPP architecture.

Our application software conversion efforts focus on key MPP application areas:

- ☐ Seismic data processing for petroleum exploration
- ☐ Atmospheric modeling for weather prediction and climate research
- ☐ Computational fluid dynamics and structural analysis for the aerospace and automotive industries
- ☐ Computational chemistry for drug design and materials science applications
- ☐ Computational electromagnetics

MPP memory architectures

Memory architecture is one of the main differences between traditional vector computers and MPPs. For example, the CRAY Y-MP C90 system has 16 processors, each with very rapid access to all of the central memory. This uniform rapid memory access is very desirable, but it does not scale to support thousands of processors.

To support thousands of processors, MPPs use distributed memories; each processing element has its own local memory. This adds a new hierarchy to the access times of memory. A PE can access its local memory faster than it can access memories on other PEs. This nonuniform memory access (NUMA) characteristic is a critical factor in programming MPPs.

Distributed NUMA memories have two basic addressing models: a message-passing model and a logically shared memory model. In the message-passing model a PE can address only its own memory; it must send messages to request information from other PEs. In the logically shared memory model, any PE can address any word of memory on any PE, although local references are faster than remote references. Users of our MPP system will be free to choose message passing, logically shared memory, or both.

Cray Research's MPP programming models

Our MPP programming models provide tools that programmers can use to minimize interprocessor communication overhead and maximize the execution rates of their programs. To cover the spectrum of MPP programming needs, we support several programming models: one for message passing and two for logically shared memory. The shared memory models are the High Performance Fortran (HPF) and MPP Fortran programming models.

Message passing

Message passing will work on any parallel computer. Each processor addresses its own local memory and sends messages when it needs to share information with other processors. Message passing can be implemented in software without any special MPP hardware. For example, message passing will work on a group of workstations connected by an Ethernet network. MPPs usually use hardware to increase the speed of the messages. This universal applicability is one of several reasons message passing is a popular MPP programming paradigm.

Our message-passing model, based on the Parallel Virtual Machine (PVM) model developed at the Oak Ridge National Laboratory, will enable easy ports of existing MPP message-passing codes. Explicit messages are sent from user space (without operating system calls) with the assistance of very fast hardware. Our MPPs will be optimum message-passing machines, even though this is only one of the programming models we support. Cray Research supports PVM for the Fortran, C, and C++ languages.

High Performance Fortran

High Performance Fortran (HPF) is an emerging shared memory programming standard. We plan to implement this standard and expect to release it with our Fortran 90 compiler.

We are working closely with the High Performance Fortran Forum (HPFF), chaired by

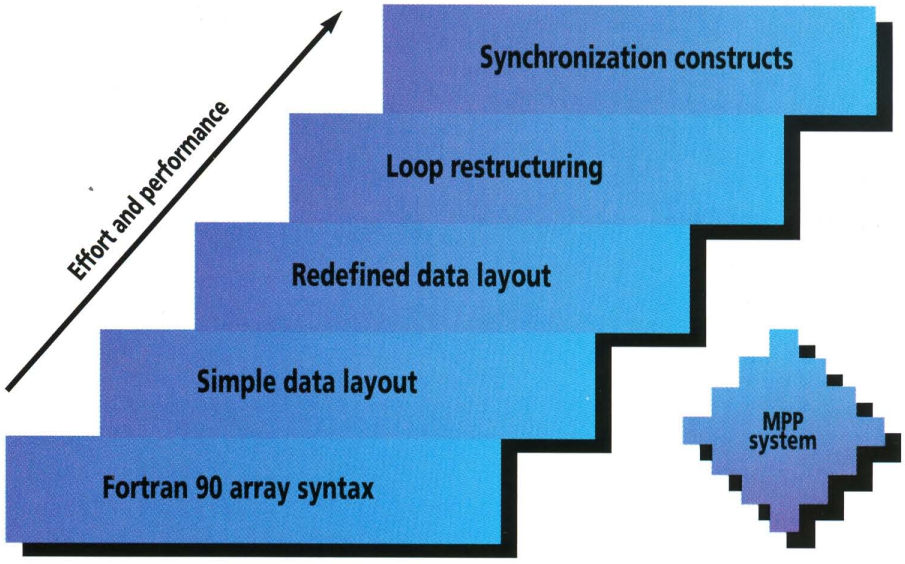
Ken Kennedy at Rice University. The HPFF is working toward a consensus on a shared memory Fortran programming model. Our involvement in the forum is part of our effort to standardize MPP programming methods to make it easier for software developers to write MPP codes that will run on multiple MPP implementations.

MPP Fortran

The HPF standard was not available when we started developing our shared memory programming model, so we began with a subset of the emerging HPF standard, enhanced it to provide direct access to our MPP hardware, and developed our MPP Fortran programming model. The model is standard Fortran with optimization directives. It provides an incremental approach to writing Fortran on MPPs. Programmers and developers using this model first specify a data sharing model by adding data sharing directives to distribute the data among the PEs. They then specify a work sharing model by using data parallel (Fortran 90 array syntax) statements. If the data parallel directives are too restrictive, DO SHARED directives can be used to control the indices and the work distribution of DO LOOPs more precisely. Finally, low-level synchronization directives, such as barrier synchronizations, locks, and events, can be added as necessary to fine-tune performance.

This model provides a number of programming benefits:

- ☐ It gives programmers the control needed to minimize communication and synchronization.
- ☐ It allows programmers to specify how and where data are located and how and where work will be performed.
- ☐ It combines work sharing and data sharing models with careful control of work and data alignment.



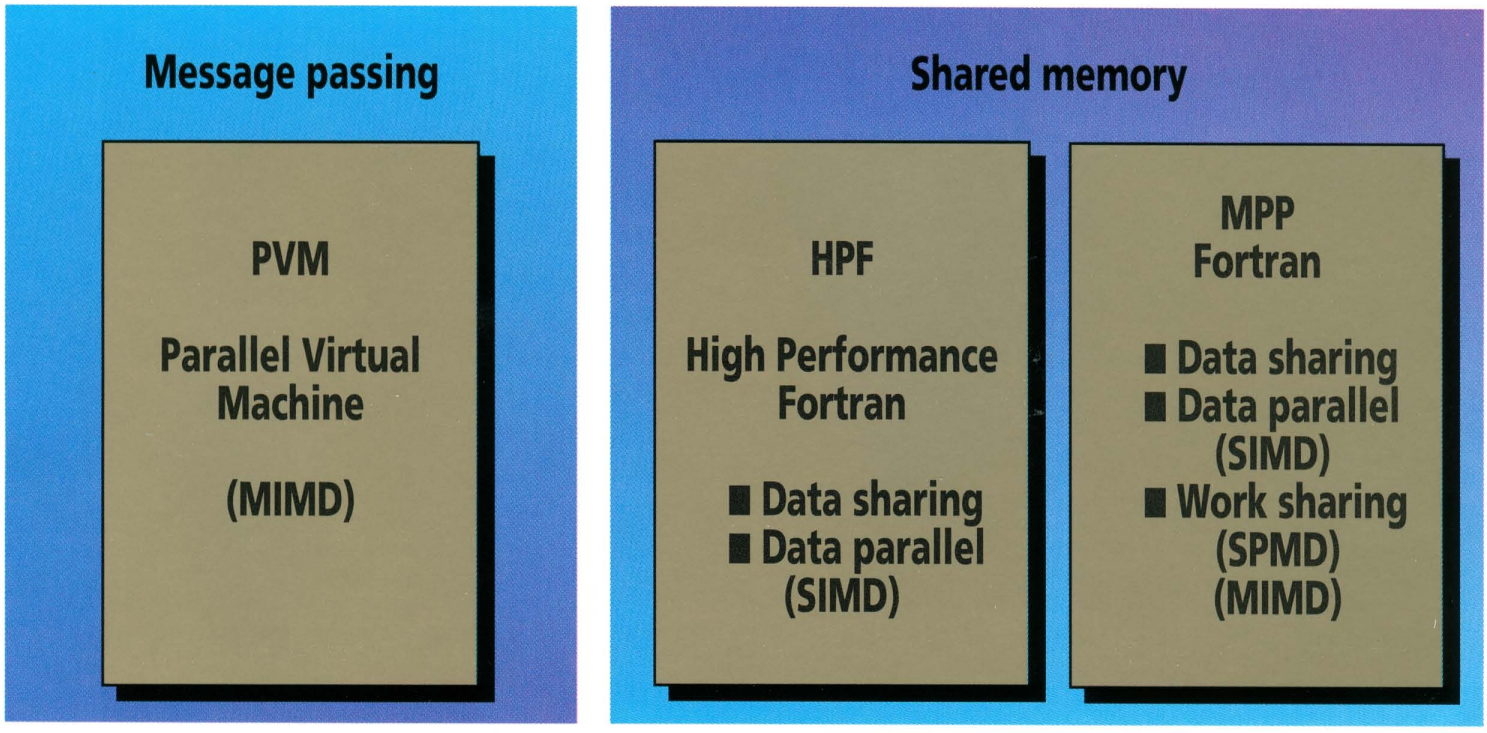
Above: Cray Research's MPP programming model provides an incremental approach to optimization. At a high level, users can indicate the areas of parallelism and let the system determine how best to exploit them. If the performance obtained is not sufficient, users can rewrite portions of their program, making decisions about the partitioning of data and work. The figure shows the steps users can take to improve code performance.

The following sections describe how to write Fortran directives for the MPP Fortran data-sharing and work sharing models. A full description of the programming model is available; send e-mail requests to mppgrp@cray.com.

Data sharing

Data objects can be either "shared" or "private." Shared objects are accessible to all tasks and generally are distributed with a piece of the object in each PE's memory. Private objects generally are replicated with an independent copy in each PE. Large matrices generally are shared; temporaries, such as loop counters, are private. For example,

```
COMMON /XXX/ A (131072), B (131072)
CDIR$ SHARED A (:BLOCK)
CDIR$ SHARED B (:BLOCK)
```



Below: Cray Research's MPP programming models. A choice of models lets programmers and software developers select the appropriate environment for their performance, ease-of-use, and portability needs.

This directive would start A(1) and B(1) in logical PE0 and keep A and B aligned so A(x) and B(x) always would be located in the same PE. The data would be distributed evenly across all the PEs. In a 128-PE partition, A(1) through A(1024) would be in PE0. A(1025) through A(2048) would be in PE1, etc. (131,072/128 = 1024, so the block size is 1024.)

One may optionally specify a block size. For example,

```
CDIR$ SHARED A (:BLOCK(32) )
```

In this case, the first 32 elements would be on PE0, the second 32 on PE1, and so on.

The use of a colon alone indicates that all occurrences of that dimension should be allocated on the same PE. For example,

```
   DIMENSION A(8192, 512)
CDIR$ SHARED A (:BLOCK(1),: )
```

In the case above, A(1, 1), A(1, 2), A(1, 3), ... A(1, 512) would be on PE0; A(2, 1), A(2, 2), A(2, 3), ... A(2, 512) would be on PE1, and so on.

Work sharing

Work can be shared across the PEs with either Fortran 90 data parallel statements or with standard Fortran 77 DO LOOPs. Some algorithms are easily expressed using array syntax. In these cases, data parallel directives simplify the programming effort. In other cases, DO LOOPs may be simpler. DO SHARED directives applied to standard DO LOOPs allow the programmer to directly control the indices and to align the data distribution with the work distribution.

Data parallel

Our fast synchronization hardware will provide exceptional performance on Fortran 90 SIMD-style MPP programs. Our CF77 Fortran compiling system already supports a significant subset of the Fortran 90 array syntax and intrinsics, and this syntax will generate SIMD-like parallel code on our MPP system. For example,

```
   DIMENSION A(1024), B(1024), C(1024)
CDIR$ SHARED A(:BLOCK), B(:BLOCK)
CDIR$ SHARED C(:BLOCK)

   A = B * C
```

multiplies all elements in B by all elements in C storing them in A.

DO SHARED

DO LOOPS can be "shared" or "private." Shared loops divide iterations among PEs. Private loops execute all iterations on one PE. Our programming model lets programmers direct loops to share work across many PEs and select from

The MPP emulator: a user perspective

Chris Hector and Jim Kohn, Cray Research, Inc.

Cray Research's MPP emulator allows users to run MPP applications on their CRAY Y-MP or CRAY Y-MP C90 systems. The emulator software executes code written in Cray Research's MPP Fortran programming model or the PVM programming model and provides feedback to help developers write more efficient parallel code. The emulator allows users to study data layout, data locality, and data reference patterns. Cray Research is making this emulator available to help customers develop programs for Cray Research's MPP system prior to the system's release.

Although the emulator provides feedback on certain aspects of code performance, it is not a simulation tool. It does not provide hardware timing information. It provides a runtime statistical analysis of the code that includes the following:

- ☐ Number and type of memory references (local and global) for each MPP processor element (PE)
- ☐ Ratios and weighting of the memory reference types
- ☐ An approximation of program parallelism determined by the ratio of memory references within parallel regions to total memory references
- ☐ Potential speedup using the program parallelism ratio and Amdahl's Law for parallel processing
- ☐ Information on application adherence to the Cray Research MPP Fortran Programming Model (including syntax checking)
- ☐ PVM message-passing statistics outlining the number and sizes of messages being passed

The phase-1 emulator can emulate a maximum of 32 PEs. This limitation is expected to be removed in the future, allowing for emulation of larger numbers of PEs.

The emulator software is made up of a library (libemu), a UNICOS command (emu), a compiler, and a version of cdbx that supports the compiler's additions to the symbol table. The compiler parses the MPP Fortran programming model declaratives, directives, and intrinsic functions, but generates code for a CRAY Y-MP or CRAY Y-MP C90 system. The emulator option causes the compiler to generate code and symbol table entries that pass information about the MPP declaratives and directives as well as information

several methods of scheduling the loops. Scheduling choices include

- Prescheduled aligned distribution of work. Iterations are placed on PEs that contain the data the iteration references. This option generally is the best one when load-balancing is not an issue. For example,

```
CDIR$ DOSHARED (K)   ON C (K)
    DO K= l,N
        A(K)=B(K)*C(K)+D(K)
    ENDDO
```

- Self-scheduled distribution of work. This option provides an adaptive runtime distribution of iterations to idle tasks. It is best used in MIMD-style loops where load-balancing is important. For example,

```
CDIR$ DOSHARED (K, L)        CHUNKSIZE(64)
    DO K= l,N
    DO L = 1, Z(K)
        A (K,L) = B (K,L) ,*C (K,L) + D (K,L)
    ENDDO
    ENDDO
```

Regardless of the value of Z(K), the work above will be serialized and portioned out to the PEs in chunks of 64 iterations. When a PE finishes its chunk, it will ask for another chunk of 64 until all the work is completed. This keeps the work balanced across the PEs.

Latency hiding

Our MPP Fortran programming model attempts to hide the latency of nonlocal memory references. Our MPP Fortran compiler will attempt to double buffer remote fetches so that blocks of data are transferred to local memory while previously transferred blocks are operated upon.

Massively parallel production

Cray Research's approach to massively parallel processing builds on 20 years of supercomputer design and engineering experience. We plan to provide the highest performing general-purpose systems by coupling massively parallel processing capabilities with the CRAY Y-MP and CRAY Y-MP C90 supercomputer architectures. Our stable MPP macroarchitecture, adaptable microarchitecture, and programming model will ensure that our MPP systems are supercomputer-class production computing systems from the outset. ▬

About the author

R. Kent Köninger is MPP software program manager at Cray Research. He has a B.S. degree in mathematics from the California State University.

about each user-code memory reference to the emulator library.

The emulator library uses the declarative, directive, and memory reference information to maintain an emulated MPP state. It can interpret memory references accurately because it includes knowledge of how the MPP hardware and compiler function. With few exceptions, programmers can obtain the same results executing an emulated program and executing a program on an MPP system. Floating point operations, which are not done with the IEEE floating point, may produce different results in emulated programs than in ones executed on a Cray Research MPP system.

Several options exist for the programmer to control the emulation. With these options, the programmer may increase or decrease the amount and the detail of the memory reference statistics. The emulator library makes the assumption that the reference pattern of variables on a CRAY Y-MP or CRAY Y-MP C90 system is essentially the same as on a phase-1 MPP emulator. The validity of these assumptions determines the quality of the emulator output.

The emulator library currently does not support all features available on an actual Cray Research MPP system. As resources become available, some unsupported features may be added. The most important of these unsupported features is the set of MPP array intrinsic functions supported by the MPP compiler. In addition, the emulation statistics do not reflect use of the prefetch queue or the block transfer engine.

PVM support

Programs that run under the emu command can use the Cray Research MPP programming model or the PVM message-passing library (from Fortran or C), as parallel programming techniques. The emulator library can generate and output performance statistics for either the Cray Research model or the PVM library, although the format and type of statistics differ from one case to the other.

PVM statistics are designed to be processed by the procview command, where message transfers between each pair of PEs are considered a separate file.

The emulator package is planned for release in the second quarter of 1993, following release 6.0 of the CF77 Fortran compiling system. Customers who want to obtain a copy of the emulator software must receive a license from Cray Research.

About the authors

Chris Hector and Jim Kohn are senior programmer analysts in Cray Research's software division.

Visual tools unlock peak performance

David Metcalfe and Kathy Nottingham
Cray Research, Inc.

A primary goal of Cray Research software is to deliver peak supercomputing performance to users. To help users develop the highest performing applications, the Application Development Toolset has been developed. This toolset is part of the UNICOS operating system and is significantly enhanced with the 7.0 release of UNICOS. The set contains a wide variety of visual tools that can be used for file management, process monitoring, source code browsing and editing, debugging, and performance analysis. These tools go beyond simple displays of information; they provide graphical and verbal interpretations of data to help users reduce analysis time. More specifically, the toolset includes

UNICOS visual interface tools:
- ☐ Cray File Manager (xfm)
- ☐ Process Monitor (xproc)

Static analysis and dynamic debugging tools:
- ☐ Program Browser (xbrowse)
- ☐ Interactive debugger (cdbx)

Application performance analysis tools:
- ☐ Program, subroutine, and loop performance analyzers (flowview, perfview, jumpview, and profview)
- ☐ I/O performance and memory activity (procview)
- ☐ Autotasking performance (ATExpert)

All these tools have visual interfaces based on the X Window System, providing users with an easy-to-use, point-and-click programming environment that increases productivity at all levels of experience. The novice user can use the UNICOS visual interface without having to learn the UNICOS

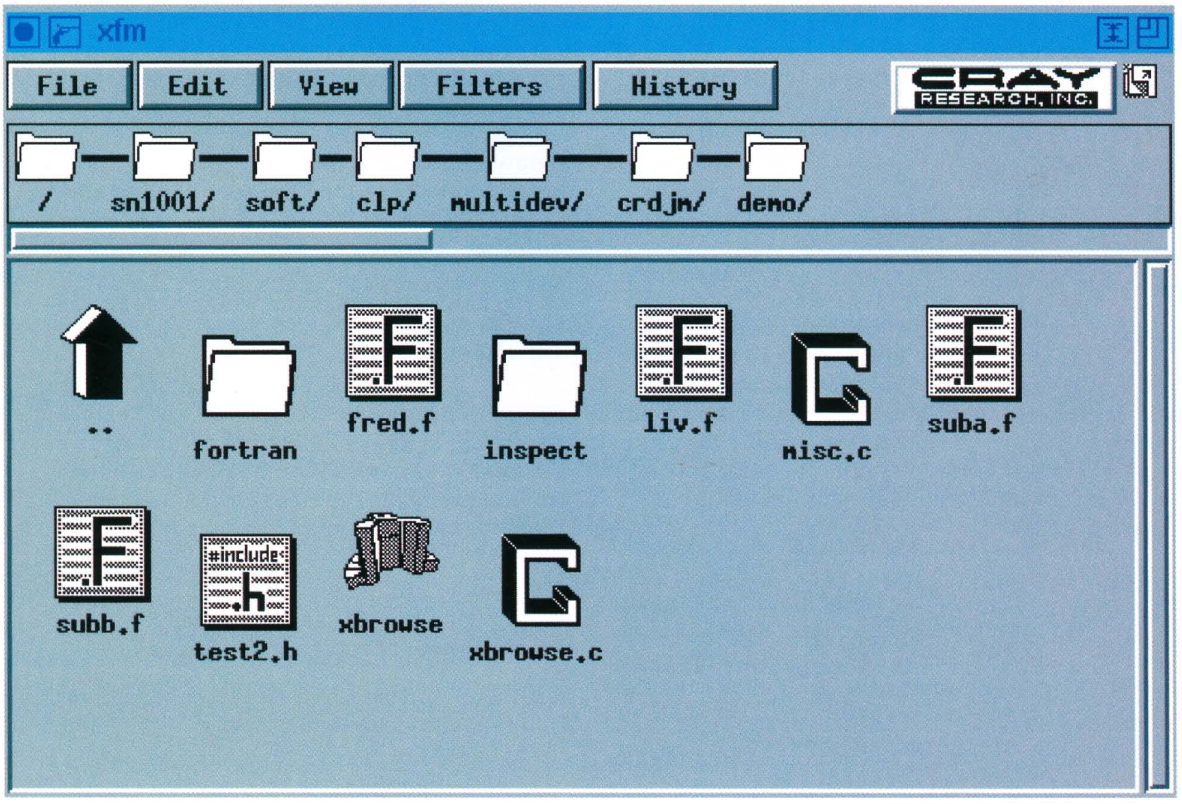


Figure 1. Xfm main window display showing a variety of file types.

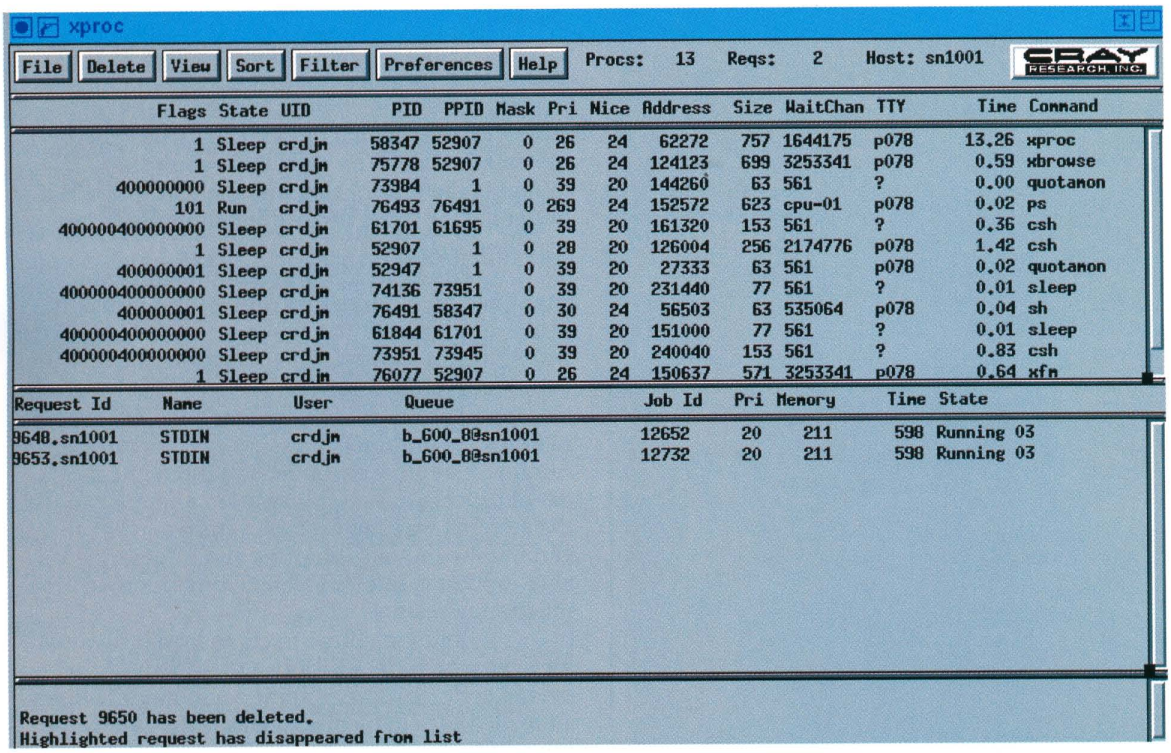


Figure 2. Xproc main window showing several interactive and NQS processes.

command language. More experienced users can benefit from the advanced debugging and performance analysis features provided in the toolset.

The visual interface for the Application Development Toolset was developed to run on a Cray Research system connected to a heterogeneous network. The use of the X Window System provides support for a variety of visual servers, including UNIX workstations running TCP/IP, terminals running the X Window System, Digital workstations running Cray Research Superlink/VMS, Apple Macintosh running MacX, and personal computers running an X server.

With the UNICOS 7.0 release, a number of new visual tools have been added: Program Browser (xbrowse), the Cray File Manager (xfm), the Process Monitor (xproc), and the performance analysis tools jumpview and procview. In addition, the Autotasking analysis tool (ATExpert) has been significantly enhanced to provide detailed observations about a user's code. These tools provide programmers and system users with convenient, fast access to requested information and present the results in an easy-to-read, customizable format. Simple point-and-click interfaces allow users who become familiar with one tool to be able to use others easily.

UNICOS visual interface

The UNICOS visual interface component of the Application Development Toolset is comprised of two tools: the file manager and the process monitor. These tools allow a user to use a Cray Research supercomputer without having to type UNICOS commands. Through the use of the visual interface, scientists and engineers can execute an application on a Cray Research system, monitor the status of the process, and view the results.

xfm

The Cray Research File Manager (xfm) is a graphical user interface to UNICOS that provides users with the ability to execute, copy, and remove files, and navigate through directory structures without typing UNICOS commands. Files and directories are represented by icons (Figure 1), and commands are executed by clicking a mouse button on the icons or pull-down menu selections. Default actions (which may be altered) are assigned to different file types. For example, selecting a text or source code file will automatically bring up an editor showing the file; selecting an executable file will automatically invoke it.

Menu options allow many UNICOS commands to be invoked directly. These options include copying, removing, renaming, and creating new files. More complex command options also are available, including the ability to locate files within a file structure.

Copying files with xfm may be achieved through a process known as "drag and drop." With this process, the user views two file structures in their icon forms, selects the file to copy, and "drags" it into the other directory. This "drag" technique also may be used to load files into the program browser, xbrowse.

xproc

The second component of the UNICOS visual interface is the Cray Research process monitor, xproc, which provides a visual interface for users to monitor the status of all their processes running on a Cray Research system.

Xproc visually displays information from the Network Queueing System (NQS) qstat and UNICOS ps commands, and allows the user to customize the display in various ways (Figure 2). Filters may be applied to sort the data selectively

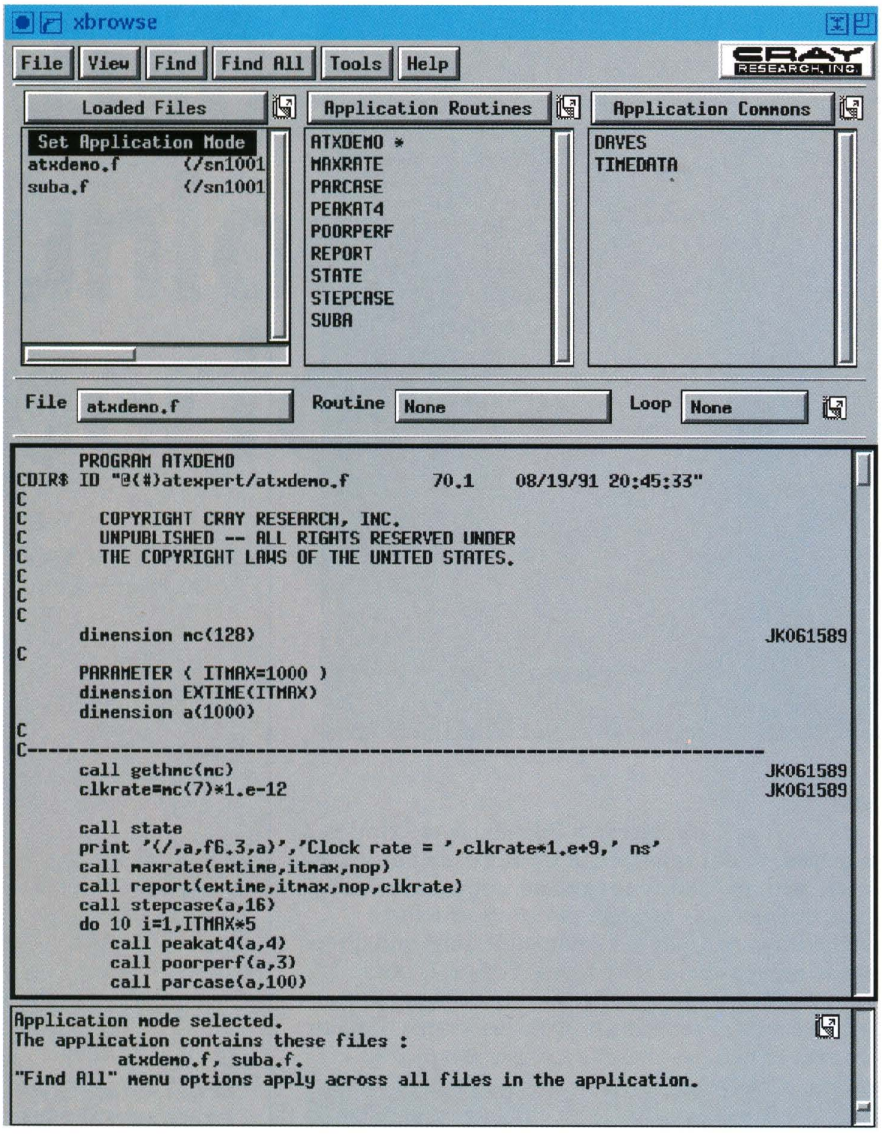


Figure 3. Browsing multiple files using the xbrowse tool.

on any of the fields and to remove processes that are of no interest to the user. For example, a user may choose to view only those processes that have been running for longer than 10 seconds, sorted by execution time. If the user were interested only in the process id, command, and execution time, all other fields could be removed from the display.

Xproc also provides information that is not directly available from the ps or qstat commands alone. By selecting a menu entry, users can view a tree of processes that shows parent/child relationships. Another menu entry provides an English translation of the process flags, translating the code values into a meaningful description of the status of the process. Through xproc, users have control over jobs or processes, enabling them to send a selected signal to, for example, kill the job or process.

Static analysis and dynamic debugging

The Cray Research Application Development Toolset provides a visual interface for debugging, and source browsing and editing, that simplifies program development.

xbrowse

When full screen editors such as vi or emacs were introduced, the programmer's ability to navigate around source code was vastly improved. However, it remained difficult to extract such information as "Who calls this subroutine?" or "What is the type of this variable and where is it declared?" without performing multiple string searches.

The UNICOS program browser (xbrowse) simplifies such queries by providing a visual, interactive environment for viewing and editing Fortran codes. Xbrowse provides language-sensitive source code analysis such as variable cross reference, variable tracing, and call tree information. For example, to find a list of subroutines that call a particular subroutine, a single menu selection offers the user a list of callers. To find information about a variable, the selected variable is chosen from a list of all variables within a subroutine and a "describe variable" option is selected. A few keystrokes achieve the desired results.

Xbrowse allows users to load individual files, with each file possibly a small part of the overall code. Selecting "application mode" allows all loaded files to be browsed as if they were one source file (Figure 3). Queries then can be made against the entire application, single files, or individual subroutines. For example, "application mode" can be used to obtain a list of all of the routines used in the application.

When a file is first loaded, or later selected, a list of all subroutines and functions is shown, together with all common blocks used. This is an example of an underlying principle of xbrowse and the other visual tools: the most effective way to get more information about an item is to point the cursor at the item and select it. (Or in the case of an executable file, to invoke it.) Action appropriate to the selected item will be performed. For example, selecting a routine name will display that particular routine; selecting a common block will display a list of all routines in which this common block is referenced. Various other high-level information can be accessed, including a list of all of the calls made within the source file, the external routines referenced (a call made to a routine not in this source file), or subroutines not called by any other user.

When browsing individual subroutines, more detailed information is available. Aside from local details as to the common blocks used or calls made, detailed information such as the variables used or the loops defined also may be accessed.

At the variable level, xbrowse allows users to query more than just the type and storage method, enabling them to "trace" the variable down a call tree. If a variable is passed as an argument into another subroutine (or via a common block reference), this is reported, even if the variable name changes. When browsing in application mode, the trace will work across subroutines in separate files, reporting when a variable is passed on to an external routine. Routines also may be edited from within xbrowse via the user-selected editor. All information lists are updated automatically when the altered source code is saved.

To simplify the comparison of multiple source files, xbrowse incorporates a peel-off mechanism. This feature allows users to select any information or text area of the screen and effectively peel it off into its own window, allowing the old and new screens to be viewed side by side (Figure 4).

The source code browser also acts as a base for other Cray Research analysis tools. For example, data scoping assistance previously supplied through a separate utility (atscope) is now incorporated as an option of xbrowse. For many complex applications, manual insertion of Autotasking directives is necessary when compiling systems cannot automatically resolve variable scopes sufficiently and must conservatively inhibit parallelization. This is particularly true for cross-module parallelism where variables are passed down into called subroutines. Using the tracing facilities of xbrowse previously described, atscope helps the user insert Autotasking directives.

cdbx

The Cray Research debugger (cdbx) provides symbolic, source-level interactive debugging capabilities with a visual interface based on the X Window System. Symbolic debugging allows users to reference variable and module names, as well as the labels that comprise a program. Source code references are based on original source code provided to the compiler. The visual interface to cdbx provides automatic source code look-up and display throughout the debugging process, eliminating the need for separate source code listings or viewing windows.

Cdbx has standard symbolic debugging functionality along with additional capabilities required for debugging optimized code (both vectorized and multitasked). With cdbx, users can connect to running processes and set breakpoints to stop execution within multiprocessing tasks.

Application performance analysis

Cray Research also provides users with a wide variety of tools for gathering performance information about applications executing on Cray Research computer systems. The primary performance analysis tools flowview, perfview, profview, jumpview, and procview provide a visual interface for users to study the data generated by the performance analysis tools: flowtrace, perftrace, prof, jumptrace, procstat, and the Hardware Performance Monitor, HPM. Each of the "view" tools first displays an overview of the program's performance. Additional options and reports then allow the user to focus on increasingly finer details of the performance information.

flowview

Flowview processes the data generated by the flowtrace library. Flowtrace gathers execution timing information about user routines while the program is running.

Users can view the performance data in a variety of ways. For example, users can traverse a graphical calling tree of their program. This tree shows the interrelationships of different routines and allows study of portions of programs (subtrees) with a simple point-and-click mouse interface. Flowview also provides information for all subroutines regarding the number of invocations, total time, and average time per call.

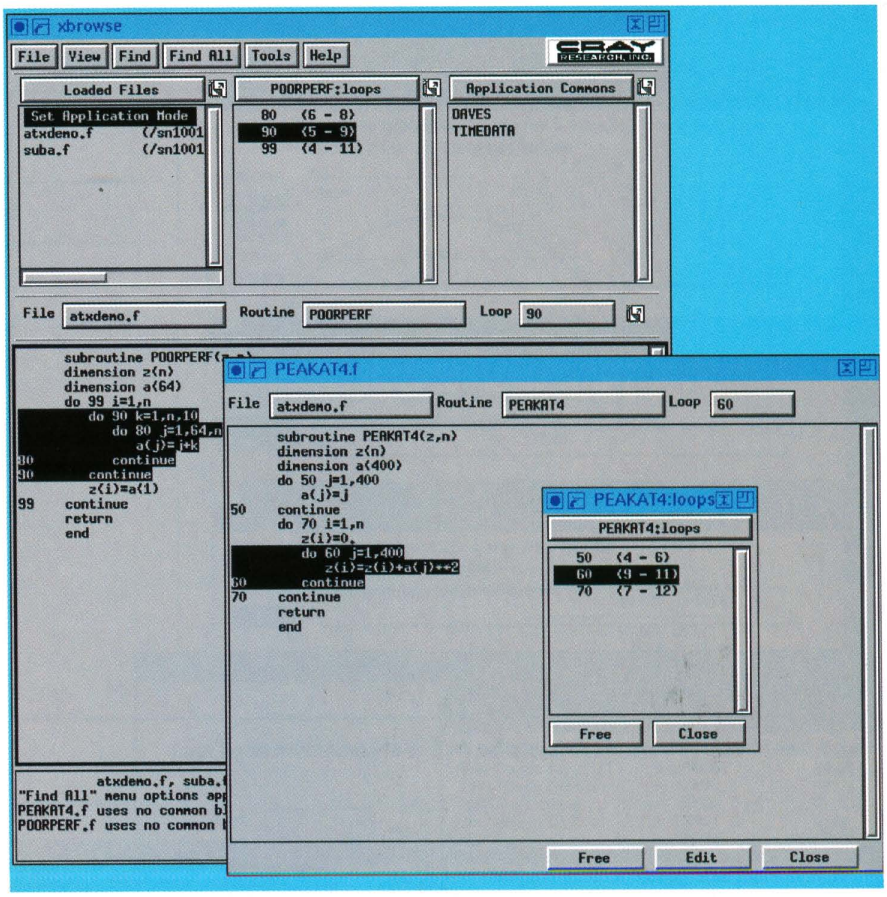


Figure 4. An example of peel-off windows allowing different areas of source code to be viewed simultaneously.

perfview

Perfview processes the data generated by the perftrace library. Perftrace times user routines, using HPM on the CRAY Y-MP and CRAY X-MP systems. A number of hardware statistics, such as the MFLOPS rating of a subroutine, are gathered by this device. Perfview can process up to four sets of statistics for the same user program: execution summary, hold issue conditions, memory activity, and vector activity. This tool also can process the output of the UNICOS hpm command, which gathers HPM statistics for an entire program.

In addition to reporting the perftrace data in several forms and arrangements, this tool provides observations to help the user interpret performance data. These observations are based on general rules governing the expected behavior of the HPM counters for certain types of codes. For instance, this tool can determine and report if a routine appears to be well vectorized or if memory conflicts exist. The help facility incorporated in this tool gives the user further assistance by providing explanations and offering examples of solutions to common performance problems.

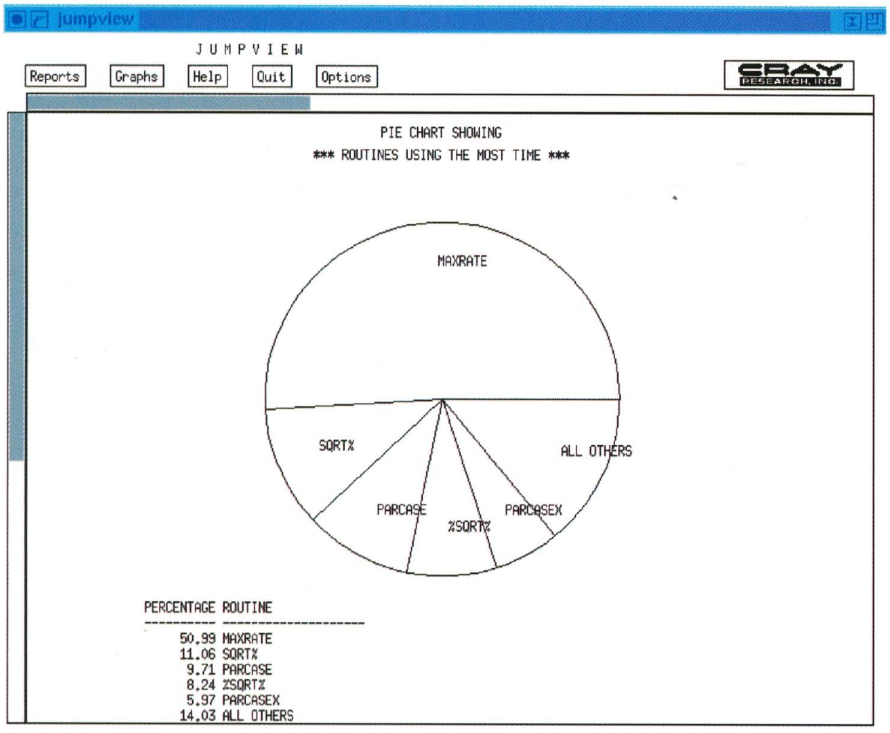


Figure 5. Main jumpview window illustrating top five routines where execution time is spent.

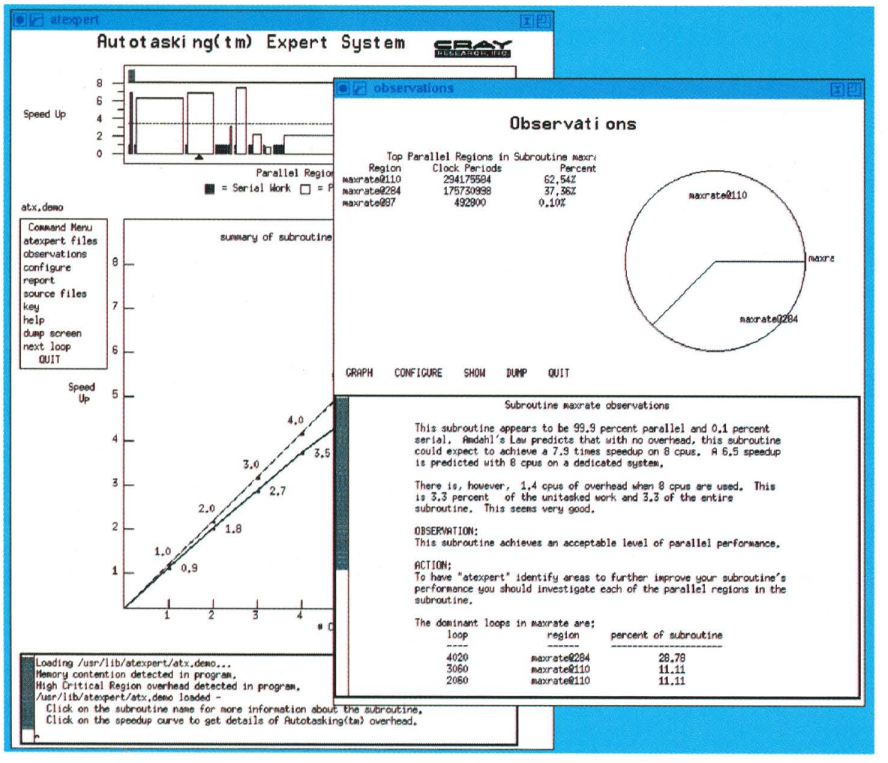


Figure 6. ATExpert with observations, providing interpretation of performance data.

jumpview

Jumpview processes the data generated by the jumptrace library, which combines simulation and runtime statistics. This tool helps users optimize their programs by providing highly detailed, fine-grained code timings (Figure 5). The tracking library isolates each code block in a user's program and gathers basic execution statistics. The jumpview program combines these runtime statistics with hardware simulation to provide accurate timings. For example, the jumptrace library can report the floating-point operation rate (MFLOPS) for a detailed area as small as a single block of user code as well as summarize timings for a subroutine, function, or the user's entire program. The jumptrace feature differs from other timing tools provided by Cray Research in that it uses simulation to provide repeatable results.

profview

Profview provides a statistical profile of program execution times by processing the data generated from the profiling system comprised of prof and the profiling library. Performance statistics are based on a regular sampling of the program's current executing instruction (P) address. Areas of the program that use larger amounts of CPU time show higher statistical samples, or "hits." A map of these hits, combined with the known memory locations of routines, can provide users with a highly detailed view showing precisely where CPU time is spent in their program and routines.

Profview users can view the statistical information in a variety of ways and at several levels of detail. For instance, if the debugging symbols were enabled during the original program's compilation, profview can show performance percentages down to the level of one loop or line of Fortran code.

This tool also provides a "memory-oriented" view of performance. By directing the mouse, users can point to spikes of high activity and view detailed information about the routine at that memory location.

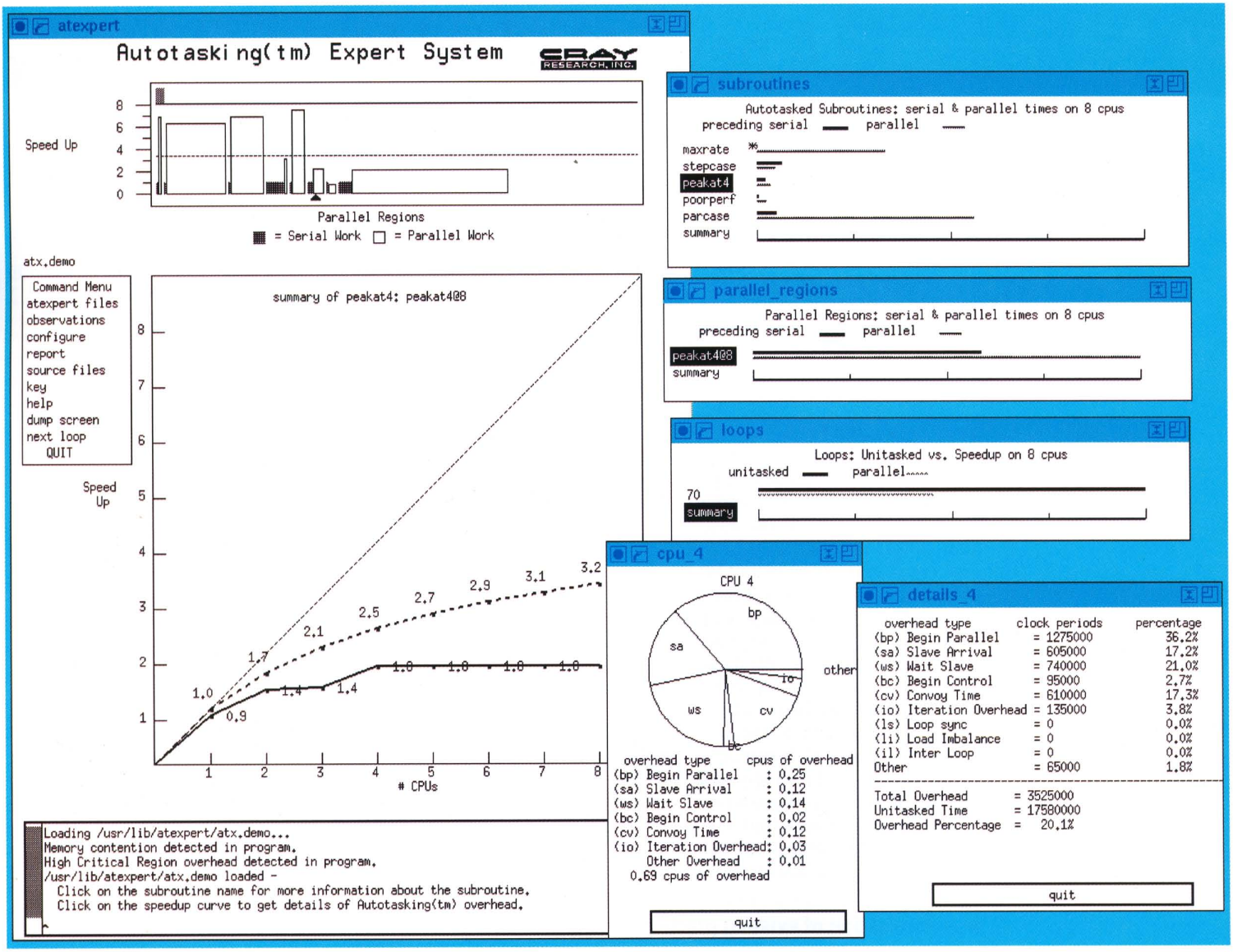
procview

Procview allows a user to view and interact with performance statistics to determine if an application is performing efficient I/O requests. Procview processes the data generated from the procstat utility, which monitors process execution and generates statistics about I/O performance and memory activity. Procstat maintains statistics for all processes executed in conjunction with a user program, including any processes initiated by a multitasked program.

Procview statistics are available in report format or graphical displays. Procview produces graphs of a wide variety of data at the user's request. Information available in graphical output includes Fortran I/O buffer efficiency, number of bytes read/written, I/O wait time, number of system calls, and file size.

Autotasking expert system (ATExpert)

Because the majority of Cray Research system users only have access to busy production systems, the already complex task of determining and improving the extent of parallel operations for their programs is compounded. The Autotasking performance monitor (ATExpert) was developed to help users predict parallel performance without access to a dedicated system and to make changes easily to increase performance.



ATExpert was first released with version 6.0 of the UNICOS operating system. Using statistics gathered during execution of a program on a nondedicated production system, ATExpert can predict parallel performance speedups that could be achieved if the program ran on a dedicated system. ATExpert also indicates where a program is spending its time, and if those areas are executing serially or with multiple processors. With release 7.0 of the UNICOS operating system, observations are provided as an interpretation of the performance data with suggestions for changes that could improve performance (Figures 6 and 7). This simplifies analysis of performance data and provides direction for programmers wishing to increase parallel performance.

The overhead for ATExpert timings is kept low (10-20 percent) by careful location of the timing points to be sampled and by avoiding large numbers of additional subroutine calls. This strategy allows ATExpert to make accurate projections of speedups associated with using any number of processors, up to the maximum number of processors available on the system.

The performance key

Scientists and engineers require peak supercomputing performance to solve difficult problems in a competitive marketplace. The Application Development Toolset is designed to help users achieve peak performance from their systems using a visual, interactive environment. These tools allow scientists and engineers to develop applications with the performance to get the job done.

Figure 7. Typical ATExpert display with multiple windows to view various aspects of a program's performance. The tool shows that code used in this example does not access more than four processors. The user in this case might try to improve performance by modifying the code to make it more parallel.

About the authors

David Metcalfe is a senior programmer analyst with Cray Research, responsible for producing visual development tools. He received a B.Sc. in computing and information systems from Manchester University, England.

Kathy Nottingham is the product marketing manager for end user software at Cray Research. Formerly she worked as an application development consultant and benchmarking analyst. Nottingham received a B.S. degree in computer science from Saint Cloud State University, Minnesota.

Fortran 90 A new standard for productivity

Richard Maine, NASA Dryden Flight Research Facility, Edwards Air Force Base, California

The Fortran programming language has been the preferred language for scientific and technical programming since the 1960s. Although the Fortran of the 1990s differs from the Fortran of the 1960s, the various versions of the language share recognizable qualities that distinguish them from other programming languages. Fortran continues to evolve, and the newest standard version, Fortran 90, provides many improvements over its immediate predecessor, the 14-year-old FORTRAN 77 standard.

Compatibility with existing FORTRAN 77 programs and coding styles was considered mandatory for the Fortran 90 standard. This compatibility has been maintained, and as a result, any program that conforms to the FORTRAN 77 standard also will conform to the Fortran 90 standard. However, in a few cases specifically referenced in the standard, a program may have a different interpretation under the Fortran 90 standard. Fortran 90 also includes some new features that differ substantially in style from the older FORTRAN 77 features. Programmers undoubtedly will develop a variety of Fortran 90 programming styles based partly on their selection of old and new language features.

Standardization of existing practice

Many features of the extended FORTRAN 77 standard, which are supported by a number of existing FORTRAN 77 compilers, are standardized in Fortran 90. These include

- ☐ Long names. Fortran 90 standardizes names up to 31 characters long and allows underscores in names.
- ☐ Lower case. Fortran 90 standardizes the use of mixed case on processors that support both upper and lower case.
- ☐ Free form source. Fortran 90 supports both the old source form and a new form, which removes the dependence on specific columns and allows up to 132 significant columns of input.
- ☐ Inline comments. Fortran 90 standardizes the exclamation mark for trailing comments. (This is the only way to signify a comment in the new source form, where column 1 is no longer reserved.)
- ☐ INCLUDE. Fortran 90 standardizes the INCLUDE line.
- ☐ IMPLICIT NONE. Fortran 90 standardizes the IMPLICIT NONE statement.
- ☐ END DO. Fortran 90 standardizes the END DO statement.
- ☐ All features of the MIL-STD-1753 standard have been adopted in Fortran 90, including bit manipulation intrinsic functions, the DO WHILE construct, and binary, octal, and hexadecimal constants.
- ☐ NAMELIST I/O. FORTRAN 90 standardizes NAMELIST input/output operations.

Although most of these features do not add functionality, they add enough convenience to have become widely used. Programmers who adhered strictly to the standard, either by choice or by dictum, now can adopt these features. Programmers who already had adopted these features as extensions to FORTRAN 77 are assured that the features will be a part of every Fortran 90 implementation and will use a consistent syntax.

Portable control of precision

Fortran 90 provides portable means of specifying and inquiring about precision. The only choices available in FORTRAN 77 for floating point were real and double precision. Applications often needed to be coded as real on 64-bit systems and as double precision on 32-bit systems and could be converted only by editing the source code. Coding practices to facilitate automated editing could be adopted, but the editing could not be avoided.

Fortran 90 addresses this issue by introducing "kind type parameters." The type name "real" can encompass several different "kinds" distinguished by an integer constant kind type parameter. One value of the kind type parameter corresponds to the FORTRAN 77 single precision real and one value corresponds to double precision. The kind type parameter has several advantages over the old real and double precision declarations:

- ☐ It can be specified in a portable way with the SELECTED_REAL_KIND intrinsic. The Fortran 90 statements

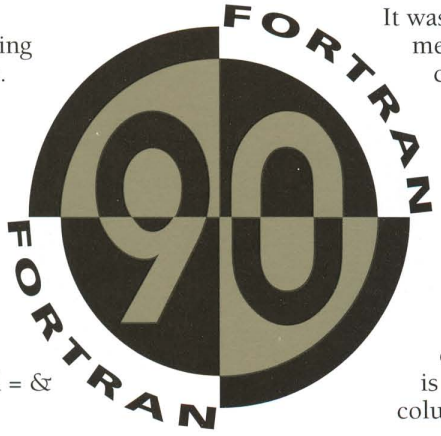
```
INTEGER, PARAMETER :: r_kind = &
      SELECTED_ REAL_KIND(12,30)
REAL(r_kind) :: a
```

declare the variable *a* to be of a type that has at least 12 decimal digits of precision and a range of at least 10^{-30} to 10^{30}. (In practice a programmer probably would put the "r_kind" declaration in a module or include file.) This code requires no

editing to port between 32- and 64-bit systems. Standard Fortran 90 intrinsics inquire about several of the machine-dependent numeric models.

- It provides a standard syntax for supporting multiple precisions. Many current systems have three floating-point precisions (and a few have more), but no standard FORTRAN 77 syntax exists to support more than two. The Fortran 90 standard does not require that more than two precisions exist, but it does standardize the syntax used to support an arbitrary number of precisions.
- It standardizes support for different precisions of COMPLEX. The FORTRAN 77 standard defines COMPLEX only in single precision. Although many compilers extend the standard to include a DOUBLE PRECISION COMPLEX, the lack of standardization in this area forces many programs to avoid use of COMPLEX entirely. The Fortran 90 standard defines a complex kind corresponding to every real kind.

Fortran 90 also introduces kind type parameters to support multiple integer ranges. The standard does not require that more than one integer kind exist, but it does standardize the syntax for supporting an arbitrary number of precisions.



Array features

Fortran 90 adds major new features in array syntax. A useful feature for many applications is dynamic allocation. Arrays can be allocated with an ALLOCATE statement as in

```
INTEGER,PARAMETER :: r_kind = &

SELECTED_REAL_KIND(12,30)
REAL(r_kind), ALLOCATABLE :: a(:)
INTEGER :: n
...
READ(*,*) n
ALLOCATE(a(n))
```

or automatically as in

```
SUBROUTINE sub(x)
INTEGER,PARAMETER :: r_kind = &
        SELECTED_REAL_KIND(12,30)
REAL(r_kind) :: x(:,:)
REAL(r_kind) :: scratch(SIZE(x,1),SIZE(x,2))
```

The second example also illustrates another important new array feature, assumed-shape array dummy arguments. The "x" array here automatically assumes the shape (dimensions) of the passed array. Assumed-shape arrays are more functional than the FORTRAN 77 assumed-size arrays (declared with "*" in the last dimension) in two ways, both illustrated here. First, assumed-shape arrays allow all the dimensions to be assumed-size, whereas assumed-size arrays allowed the "*" only for the last dimension. Second, assumed-shape arrays automatically pass the dimension information at run time. The SIZE intrinsic can be used as shown here to inquire about the resulting dimensions; the "scratch" array in this example is an automatically allocated array with the same dimensions as "x."

Fortran 90 supports array expressions, also called array syntax. A simple sample is

```
A = B + C
```

where A, B, and C are arrays of the same shape. Fortran 90 also supports array sections. For example, the Fortran 90 expression

```
x(:,j)
```

explicitly represents column j of the x array. This expression can be used as a vector in an expression or passed as an argument. To pass a column of an array as a FORTRAN 77 argument, the top element of the column was passed, as in

```
x(1,j)
```

It was ambiguous whether this really meant to pass the element or the column; all that really got passed was a starting address.

FORTRAN 77 provides no natural way to pass a row of an array as an argument; the workarounds involve index manipulations based on a knowledge of how the array was laid out in memory. In Fortran 90, passing a row is no different from passing a column. The expression

```
x(i,:)
```

explicitly represents row i of the x array. Note that the elements of the row are not contiguous in memory. The Fortran 90 compiler handles this automatically and transparently. When this row is passed as an argument to a subroutine, the compiler passes all the information needed to compute the memory locations from index values. This extends to more complicated array slices such as

```
x(1:3,1:4,7)
```

which is a 3 by 4 array slice from a three-dimensional array. The index manipulation is similar to that required in FORTRAN 77, but in Fortran 90 the compiler performs the manipulation.

This is one of the significant changes in Fortran 90. The new standard allows the programmer to code in a more abstract manner, closer to the mathematical statement of the problem, which results in clearer code. This is consistent with Fortran's heritage of "FORmula TRANslation." The compiler is responsible for mapping the abstractions into machine implementations and storage layout. Because memory layout issues are left to the

compiler, programmers can take better advantage of the architectural features of the particular system. Programmers adopting the philosophy of the "new language" part of Fortran 90 will seldom use indexing "tricks" that depend on knowing how objects are laid out in memory.

Fortran 90 also supports array-valued functions; that is, functions whose results are arbitrarily shaped arrays.

Data structures

Fortran 90 supports data structures, also known as derived types. The only kind of composite data construct supported by FORTRAN 77 is the array. In an array, the data items are identified by indices, and all of the data items must be the same type. Most other recent languages also support a construct that goes by different names in different languages but that we will refer to here as a data structure. The defining characteristics of a data structure are that its constituent data items are identified by name and that each data item may have a different type. Many programmers consider the lack of data structures to be a major shortcoming of FORTRAN 77.

Fortran 90 also supports pointers, which greatly simplify the implementation of some kinds of algorithms. A Fortran 90 pointer may point to a scalar variable, an array variable, or an array section. The appearance of a pointer name in an expression is an automatic reference to the object to which the pointer points.

Modules, interface blocks, and internal procedures

One of the more significant new features of Fortran 90 is the module. A Fortran 90 module is a collection of declarations and/or procedures. Although modules are conceptually simple, their implications and uses are broad enough to cause significant changes in coding styles.

Modules are a substitute for and an improvement over common blocks for communication among program units. Any program unit that has a USE statement for the module can access the public entities of the module. A fundamental difference between common blocks and modules is that common blocks communicate variables by address, whereas modules communicate by name. This means that modules are not subject to the bugs that often arise when different declarations are used accidentally for the same common block in different program units. It also means that modules can be used for dynamically sized objects, parameters, and derived type definitions that cannot go in common blocks.

By replacing common blocks, modules also replace block data subprograms. The variables in a module can be initialized directly in the module, with none of the complications of initializing data in common blocks.

Modules can contain both declarations and procedures. When a procedure is in a module, the compiler verifies that all calls to the procedure have compatible argument lists. This avoids a large class of program bugs common in FORTRAN 77. There are other ways to enforce this kind of checking, but having the procedure in a module is the easiest. Module procedures also can have optional arguments and can use a keyword syntax for arguments.

Keyworded argument passing is similar to FORTRAN 77's I/O control list syntax. This allows arguments to be passed in any order other than that specified in the procedure declaration. Keyworded argument passing and checking of argument types at call sites also can be accomplished with Fortran 90's interface blocks and internal procedures. Interface blocks also allow users to define their own operators, overload generic or intrinsic operators, and define or overload their own generic procedures. The module/use feature allows for name hiding of procedures, as do internal procedures. This is useful for developers of libraries and for third-party vendor libraries. The USE statement, which specifies a module to be imported, allows local renaming of module names to avoid name collisions within the local subprogram.

Input/output enhancements

FORTRAN 77 I/O is record oriented. Fortran 90 introduces the notion of nonadvancing I/O, which allows the reading or writing of a partial record. This may be a single character, a character string, a scalar variable, an array, a subobject of a structure, or a whole structure without moving beyond a record boundary in a file.

Other I/O enhancements include a binary, octal, and hexadecimal format edit descriptor, the ability to specify how records are padded (blank or zero pad), and the position at which to open an I/O file, that is, to overwrite or append an output file.

Fortran 90 provides many improvements over FORTRAN 77. It adds important capabilities and features that promote the portability, clarity, modularity, and reliability of code. As a result of these improvements, programmers should be able to increase their productivity and the performance of their codes. ▬

About the author

Richard Maine is an aerospace engineer at the NASA Dryden Flight Research Facility at Edwards Air Force Base, California. He has a B.S. degree in aeronautical engineering from Purdue University and M.S. and ENGR degrees in system science from the University of California, Los Angeles. He has over 20 years of experience in scientific computation in Fortran and several other languages on numerous computer platforms. His current projects include a system for managing all flight data archival and access at NASA Dryden, implemented mostly in Fortran 90.

Fortran 90 at Cray Research

Kathy Nottingham, Cray Research, Inc.

Cray Research is developing a new Fortran 90 compiling system that will conform to the full Fortran 90 standard. The new Fortran compiler frontend will connect to the common modular backend used in the company's industry-leading compiling systems. Our development goal is to provide not only a fast and efficient Fortran 90 compiler, but also the features and performance that are available with the CF77 Fortran compiling system. We expect to release the new compiler in mid-1993.

Two options currently exist for developing applications using Fortran 90 features on Cray Research systems. Developers can use either the Fortran 90 features available in the current CF77 compiling system or a commercially available Fortran 90 translator.

Cray Research has included a number of Fortran 90 features in the CF77 Fortran compiling system. These features, particularly array syntax, allow users to develop applications with Fortran 90 features. The Fortran 90 features supported in the CF77 compiling system include

- ☐ Long names
- ☐ Inline comments
- ☐ INCLUDE line
- ☐ DO WHILE, END DO, and nonlabeled DO statements
- ☐ 31-character names, underscore, and mixed upper/lower cases
- ☐ Exclamation mark for inline comments
- ☐ IMPLICIT NONE statement
- ☐ NAMELIST input/output operation
- ☐ RECURSIVE functions and subroutines
- ☐ Vectorizable math functions, such as sin, sqrt, max, and abs
- ☐ Mil-spec bit intrinsics
- ☐ Quotation marks as string delimiters
- ☐ Hexadecimal (Z) and octal (O) constants
- ☐ Mixed character and noncharacter data in the same COMMON block
- ☐ Automatic arrays
- ☐ All of the OPEN and INQUIRE specifiers
- ☐ Character string edit descriptor delimited with quotation marks
- ☐ Repeat count for slash editor, for example, 3///.

The CF77 compiling system also supports a subset of the array syntax including whole and partial array assignment and subscript triplet notation. It currently does not support nonsequence arrays as actual or dummy arguments, passing vector valued subscripted arrays, or ALLOCATABLE arrays. The CF77 compiler also includes pointers, though they differ from those defined in the Fortran 90 standard.

Until Cray Research's Fortran 90 compiler becomes available, users can use software translators to run Fortran 90 code on Cray Research systems. A number of third-party translators are available that convert Fortran 90 code to FORTRAN 77 or C code:

- ☐ VAST90 from Pacific Sierra Research translates Fortran 90 to FORTRAN 77, and vice versa, and runs on Sun SPARC and IBM RISC System/6000 platforms.
- ☐ The F90 product from ParaSoft translates Fortran 90 to FORTRAN 77. F90 runs on a variety of platforms, including Cray Research, SUN, IBM 3090 and RISC System/6000, DecStation, Silicon Graphics, Convex, and personal computers.
- ☐ The Fortran 90 translator from Numerical Algorithms Group (NAG), Inc., converts Fortran 90 to C. It is available on workstation platforms, including Sun Microsystems, IBM RISC System/6000, and Digital Equipment Corporation Ultrix systems.
- ☐ The Forge 90 translator from Applied Parallel Research translates FORTRAN 77 to Fortran 90. The product runs on a variety of UNIX platforms, including Cray Research, IBM RISC System/6000, Digital Equipment, Sun SPARC workstations, Silicon Graphics, Convex, and Intel.

About the author

Kathy Nottingham is the product marketing manager for end user software at Cray Research.

PVM and HeNCE

Traversing the parallel environment

Adam Beguelin, Carnegie-Mellon University, Pittsburgh, Pennsylvania
Jack Dongarra and Al Geist, Oak Ridge National Laboratory, Oak Ridge, Tennessee
Robert Manchek, University of Tennessee
Vaidy Sunderam, Emory University, Atlanta, Georgia

To take advantage of today's heterogeneous networks, users need to distribute problems across diverse architectures. To this end, researchers at Oak Ridge National Laboratory, the University of Tennessee, and Emory University have teamed up to create two software packages that allow diverse computer systems to work together automatically to run applications. The two packages, Parallel Virtual Machine (PVM)[1] and the Heterogeneous Network Computing Environment (HeNCE),[2] were designed with heterogeneity and portability as primary goals, enabling the most cost-effective use of networked resources. Currently, 100 institutions worldwide are running PVM. Cray Research will be releasing PVM and HeNCE and providing an implementation of PVM as a tool for users developing software for Cray Research MPP systems.

PVM

PVM enables a user to define a heterogeneous networked collection of serial, parallel, and vector computers to function as one large computer. It can be used as stand-alone software or as a foundation for other heterogeneous network software. PVM may be configured to contain various machine architectures including sequential processors, vector processors, and multicomputers. The present version of the software has been tested with various combinations of CRAY Y-MP, CRAY-2, Sun 3, SPARCstation, MicroVAX, DECstation, IBM RISC System/6000, HP-9000, Silicon Graphics IRIS, NeXT, Sequent Symmetry, Alliant FX, IBM 3090, Intel iPSC/860, Thinking Machines CM-2, KSR-1, and

Convex systems. In addition, users can port PVM to new architectures simply by modifying a generic "makefile" supplied with the source and recompiling. PVM therefore allows users to exploit the aggregate power of workstations and supercomputers distributed around the world to solve computational grand challenges. To the user, PVM appears as a loosely coupled distributed-memory computer programmed in C or Fortran with message-passing extensions. The hardware that composes the user's personal parallel virtual machine may be any UNIX-based machine on which the user has a valid login and network access.

With PVM, users can configure their own parallel virtual computer, which can overlap other users' virtual computers. A personal parallel virtual computer can be configured simply by listing the names of the machines in a file that is read when PVM is started. Several physical networks can coexist inside a virtual machine. For example, a local Ethernet, HIPPI, and fiber optic network all can be parts of a user's virtual machine. While each user can have only one virtual machine active at a time, PVM multitasks, allowing several applications to run simultaneously on a parallel virtual machine.

The PVM package is small (less than 400 Kbytes of C source code) and easy to install. It needs to be installed only once on each machine to be accessible to all users. Moreover, the installation does not require special privileges on any of the machines and therefore can be accomplished by any user.

Application programs that use PVM are composed of subtasks, making PVM extremely flexible. The subtasks can be generic serial codes or codes specific to a particular machine. With PVM, resources may be accessed in three modes: in the transparent mode subtasks are automatically executed at appropriate sites; in the architecture-dependent mode the user may indicate specific architectures on which particular subtasks are to execute; and in the machine-specific mode a particular machine may be specified. Such flexibility allows different subtasks of a heterogeneous application to exploit particular strengths of individual machines on the network.

The PVM interface requires that all message data be explicitly typed, freeing users from data conversion worries. PVM performs machine-independent data conversions when required, thus allowing machines with different integer and floating-point representations to pass data. Applications access PVM resources via a library of standard interface routines. These allow the initiation and termination of processes across the network as well as communication and synchronization between processes. Communication constructs include those for the exchange of data structures as well as high-level primitives such as broadcast, barrier synchronization, and rendezvous.

Application programs under PVM may possess arbitrary control and dependency structures. This means that at any point in the execution of concurrent applications, the processes in existence may have arbitrary relationships between each other and may communicate and/or synchronize with any other. Users therefore benefit from increased flexibility without the restrictions of master and slave processes.

HeNCE

The second software package, HeNCE, is built on a PVM foundation. While PVM provides low-level tools for implementing parallel programs, HeNCE provides the programmer with a higher-level environment for using heterogeneous networks. The goal is to make network computing accessible to scientists and engineers without the need for extensive training in parallel computing and to enable them to use resources best suited for a particular phase of the computation. The HeNCE philosophy of parallel programming is to have the programmer explicitly specify the parallelism of a computation and to automate, as much as possible, the tasks of writing, compiling, executing, debugging, and analyzing the parallel computation. Central to HeNCE is an interface based on the X Window System that the programmer uses to perform these functions. Through this interface, users can use a *compose* tool to explicitly specify parallelism by drawing a graph of the parallel application. If an X Window System interface is not available, then textual graph descriptions can be entered.

To increase user productivity, HeNCE is designed to enhance procedure reuse. To this end, each box in a HeNCE graph represents a procedure written in either Fortran or C. The procedure can be a subroutine from an established library or a special-purpose subroutine supplied by the user. Arcs between boxes visually represent data dependency and control flow. A dependency arc from one box to another, for example, reminds the user that the tail box of the arc must run before the head of the arc. Data is sent to a node from its ancestors in the graph (usually its parents).

PVM/HeNCE on Cray Research systems

Peter Rigsbee, Cray Research, Inc.

Cray Research will be providing PVM and HeNCE as a common interface for programming distributed applications involving Cray Research computer systems. This interface will accommodate the following network environments:

- ☐ Multiple Cray Research systems connected in a network
- ☐ Cray Research systems connected with other systems in a network
- ☐ A Cray Research massively parallel processing (MPP) system (where PVM would be used for message passing between processing elements)
- ☐ A Cray Research MPP system and a CRAY Y-MP system combined to work on a single application

Any application will be able to use the PVM library as the single interface for message passing throughout the network. Message passing is an explicit programming paradigm for interprocess communication that has been used for many years in a variety of applications. Although most existing MPP applications use message passing, most interfaces are proprietary and unique to a particular vendor. Cray Research supports PVM for its wide availability and portability.

As the first step in supporting PVM and HeNCE, Cray Research has released versions of these software products. These are source releases of the same codes that can be obtained through netlib from the Heterogeneous Network Project, which is provided with limited support from Cray Research, Inc.

Cray Research plans to enhance PVM and HeNCE much further. For example, development is under way to provide a version of PVM as the primary message-passing library on future Cray Research MPP systems. This version will provide very efficient communication within the MPP system and, with the same interface, will allow communications with processes running on other systems in the network. Cray Research also plans to continue releasing new versions of PVM and HeNCE to keep customers up-to-date with the work of the Heterogeneous Network Project.

Although PVM and HeNCE are not the only systems available for programming distributed applications, Cray Research decided to support and enhance them for a number of reasons. PVM was designed for scientific network computing, with support for heterogeneity and high performance as major goals for the design. Although standards in message passing have yet to be defined, PVM is viewed by many as the de facto standard. PVM runs on many platforms, many of which our customers use in their everyday work. HeNCE, the higher-level programming environment, offers unmatched ease of use with a powerful approach to distributed applications that eliminates much of the programming burden. Most important, Cray Research users at a number of sites already are using PVM and HeNCE to distribute applications.

Perhaps the most exciting prospect in network supercomputing is the ability to harness the power of multiple supercomputers to work on a single application. Cray Research believes PVM and HeNCE offer the means to make this prospect a reality. For more information on ordering PVM/HeNCE 1.0, contact your local Cray Research representative.

About the author

Peter Rigsbee is a software developer with Cray Research, working on distributed computing and MPP development. He received B.S. degrees in civil engineering and management from the Massachusetts Institute of Technology and an M.B.A. degree from the College of St. Thomas, St. Paul, Minnesota.

In addition to simple boxes, four types of visual control constructs are available in the HeNCE graph language representing looping, conditional dependency, a fan-out to a variable number of identical subgraphs, and pipelining. These visual constructs make programming easier and notify the user of certain programming errors as they occur. A loop construct directs HeNCE to execute a subgraph a number of times based on the expression in the loop construct. Using a conditional construct, a section of the graph can be executed or bypassed based on an expression that will be evaluated at runtime. A variable fan-out (and subsequent fan-in) construct is available while composing the graph. The width of the fan-out is specified as an expression that is evaluated at runtime. The construct is similar to a parallel-do construct found in several parallel Fortrans. With the pipelined construct, when a node finishes with one set of input data, it reruns with the next piece of pipelined data.

Once the dynamic graph is specified, the configuration tool in HeNCE can be used to specify the configuration of machines that will compose the user's parallel virtual machine. The *configuration* tool also assists users in setting up a cost matrix. The cost matrix allows users to describe which machines will perform which tasks and to give priority to certain machines. HeNCE will use this cost matrix at runtime to determine the most effective machine on which to execute a particular procedure in the graph.

HeNCE also contains a *build* tool to perform three tasks. After analyzing the graph, HeNCE automatically generates the parallel program using PVM calls for all the communication and synchronization required by the application. Second, by knowing the desired PVM configuration, HeNCE automatically compiles the node procedures for the various heterogeneous architectures. Finally, the *build* tool installs the executable modules on the particular machines in the PVM configuration.

The *execute* tool in HeNCE starts up the requested virtual machine and begins execution of the application. During execution, HeNCE automatically maps procedures to machines in the heterogeneous network based on the cost matrix and the HeNCE graph. Trace and scheduling information saved during the execution can be displayed in real time or replayed later.

The HeNCE environment has a *trace* tool that allows visualization of the parallel run. The trace tool is based on the X Window System and consists of two windows. One window shows a representation of a network and machines underlying PVM. In this window, icons of the active machines are illuminated with different colors depending on whether they are computing or communicating. Under each icon is a list of the node procedures mapped to this machine at any given instant. The second window displays the user's graph of the application, which changes dynamically to show the actual paths and parameters taken during a run. The nodes in the graph change colors to indicate the various activities going on in each procedure. Figure 1 shows a snapshot of the *trace* tool in action.

PVM results

One of the computational grand challenges being addressed at Oak Ridge National Laboratory is the calculation of the electronic structure of superconductors.[3] After modifying this application to run using PVM, a heterogeneous network of workstations was used to achieve execution rates exceeding 250 MFLOPS.

A second set of experiments designed to show the capability of PVM to connect several supercomputers also used this grand challenge code. In one test, an Intel iPSC/860, a CRAY X-MP system, and an IBM RISC System/6000 were configured as a metacomputer. In another test, several CRAY Y-MP8 systems and a CRAY Y-MP2 system distributed over a WAN were configured into a metacomputer. Although the input datasets were too small to drive either of the metacomputers to near their peak rates, the experiments demonstrate the viability of using a collection of supercomputers. Production runs are now being completed using the CRAY Y-MP4 system at the Florida Supercomputer Center configured with workstations at Oak Ridge. PVM applications also are being developed at Oak Ridge National Laboratory in the areas of molecular dynamics, statistical modeling, and climate modeling.

Current status and availability

PVM became publicly available in March 1991. The current version, version 2.4, is available through netlib. To obtain a description of related information, such as a copy of the *PVM User's Guide* or source code, send e-mail to netlib@ornl.gov with the message "send index from pvm." To obtain a description of what is available related to HeNCE, send e-mail to netlib@ornl.gov with the message: "send index from hence." The same software also can be obtained through the Cray Research, Inc. PVM/HeNCE 1.0 release.

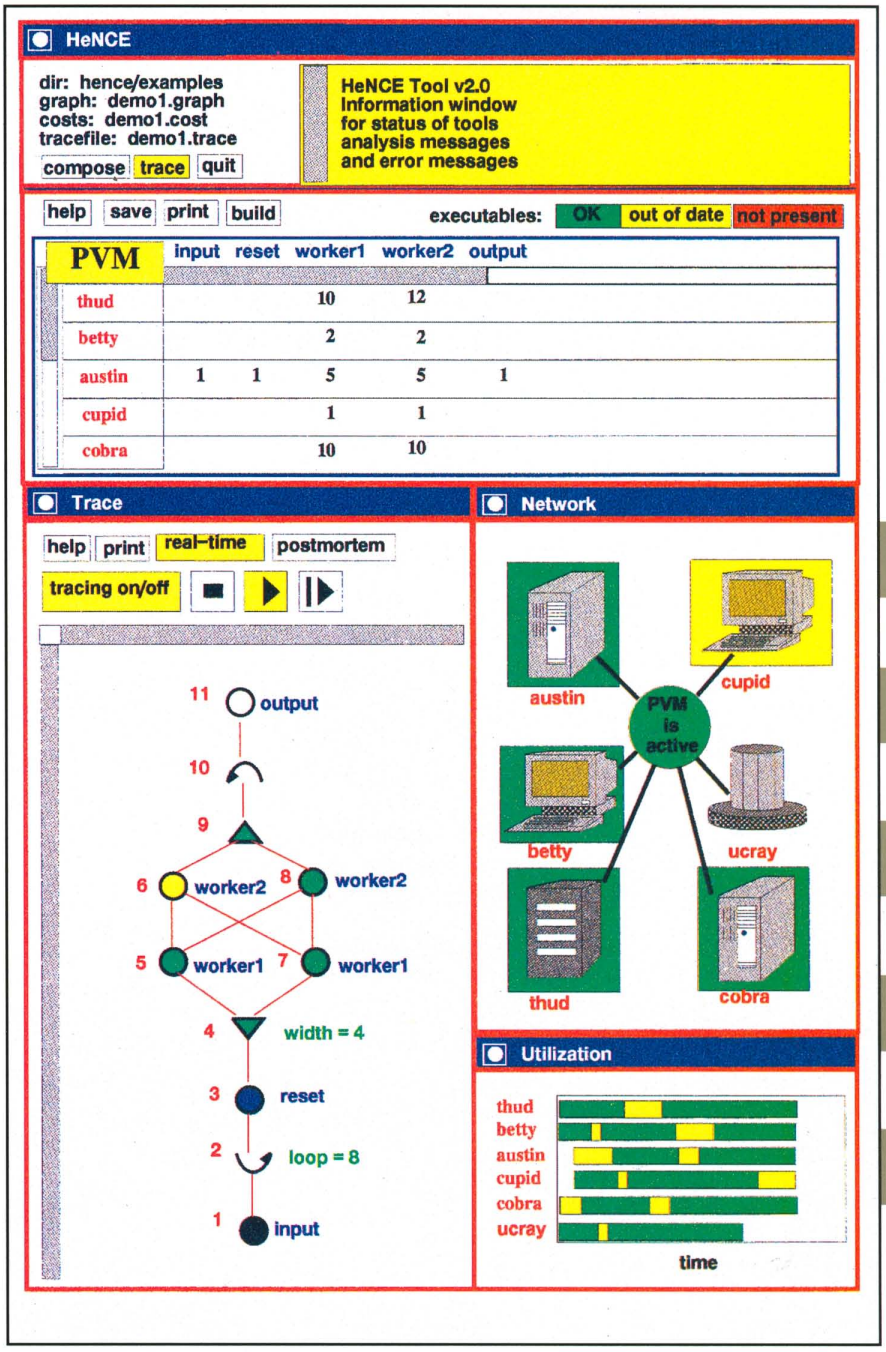


Figure 1. The HeNCE interface is based on the X Window System. Here the *trace* tool is shown in operation.

About the authors

Adam Beguelin is a research scientist in the School of Computer Science at Carnegie Mellon University specializing in the design and development of programming tools and environments for high performance parallel and distributed computing. He received M.S. and Ph.D. degrees in computer science from the University of Colorado.

Jack Dongarra holds a joint appointment as distinguished professor of computer science in the computer science department at the University of Tennessee and as Distinguished Scientist in the Mathematical Sciences Section at Oak Ridge National Laboratory under the Science Alliance Program. He received an M.S. degree in computer science from the Illinois Institute of Technology and a Ph.D. degree in applied mathematics from the University of New Mexico.

Al Geist is a researcher at Oak Ridge National Laboratory in computational sciences. He received a B.S. degree from North Carolina State University in mechanical engineering.

Robert Manchek is a research associate at the University of Tennessee, Knoxville, in the department of computer science. He received a B.S. degree in electrical and computer engineering from the University of Colorado in Boulder.

Vaidy Sunderam is a faculty member in the department of mathematics and computer science at Emory University. He received a Ph.D. degree in computer science from the University of Kent, England.

References

1. Beguelin, A., J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, *A User's Guide to PVM Parallel Virtual Machine*, Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.
2. Beguelin, A., J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, *Solving Computational Grand Challenges using a Network of Supercomputers*, Proceedings of the Fifth SIAM Conference on Parallel Processing, Danny Sorensen, ed. and SIAM, Philadelphia, PA, 1991.
3. Beguelin, A., J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, "Opening the Door to Heterogeneous Network Supercomputing," *Supercomputer Review*, September 1991, pp. 44-45.

Real-time systems

Cray Research meets the "real(time)" world

Lance Miller, Cray Research, Inc.

A real-time system is a collection of hardware and software that performs its functions and responds to external, asynchronous events in a predictable (or deterministic) period of time. Typically, these systems are monitoring, controlling, simulating, collecting, or communicating with events from the external, physical world.

Real-time systems are classified as "hard" real-time and "soft" real-time. Hard real-time systems are driven by real-world stimuli, such as power plant monitors, anti-lock brakes, or flight simulators, that generate events which require a response in a fixed and predictable (deterministic) period of time. Soft real-time systems are driven by less stringent time frames and often are priority driven. Examples of soft real-time applications include scheduling, data acquisition, and seismic applications.

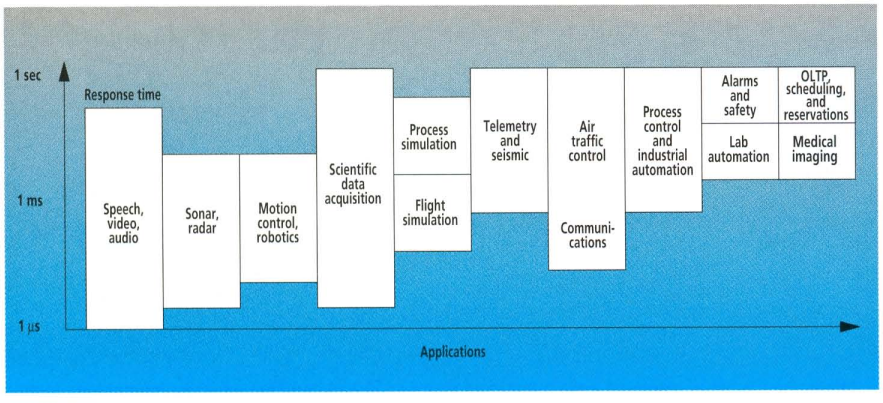
Real-time problems require a combination of computational power, high throughput I/O processing, rapid deterministic interrupt response, and handling of asynchronous events to meet the needs of real-time applications. While no single machine spans all applications and all performance needs, Cray Research's real-time systems cover a wide spectrum of high response and intensive, computational applications.

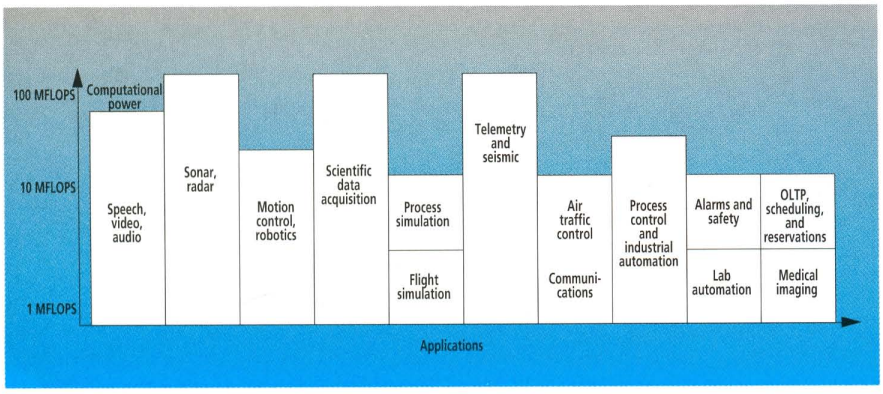
Response time requirements of a real-time system are application-specific. Figure 1 depicts the various applications (x-axis) and their respective response times (y-axis). For example, real-time telemetry applications generally require interrupt responses in the 100 µs to 1 second range.

Computational requirements of a real-time system also are application-specific. Figure 2 depicts the same types of applications on the x-axis and their corresponding computational needs on the y-axis. For example, real-time telemetry applications can require from 1 to 100 MFLOPS of performance.

Figure 1 (top). Real-time applications and response times.

Figure 2 (bottom). Computational requirements of real-time applications.





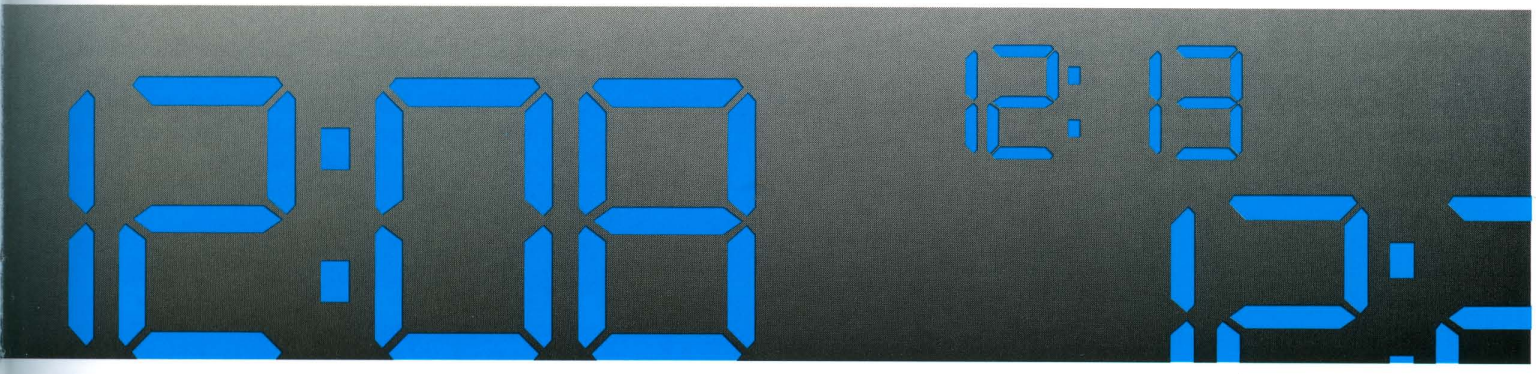
Real-time characteristics

Real-time problems place different stresses on a computing system than do most general-purpose problems. Compared to general-purpose requirements, real-time system requirements are distinguished by the following:

- ☐ Deterministic system response
- ☐ Preemptibility/interruptibility for events
- ☐ High performance and high data rates
- ☐ Fast interrupt response
- ☐ System algorithms dedicated to real-time
- ☐ I/O support for real-time peripherals
- ☐ Support for segregation of system resources

It may appear as if real-time computing equals high-performance computing. Although some portions of real-time processing depend on high performance, the common theme shared by all real-time problems is the predictability of the system's response. In real-time, the right answer at the wrong time is the wrong answer. For example, if a series of computationally intense calculations is needed to predict the behavior of a nuclear power plant and the answer arrives just a few milliseconds late, it may mean the difference between containment and disaster.

When a real-time system fails to meet its targets for interrupt response, predictability, or overall performance, there are a number of consequences that differentiate the real-time system from the general-purpose system. These consequences



can be catastrophic to humans, systems, or missions—usually stated as life threatening or mission critical.

In a real-time system, loss of data, failure to act at the prescribed intervals, failure to respond to an interrupt, or inability to complete necessary processing in the time interval allocated may show up as lost satellite transmissions, nuclear plant shutdown, or the collapse of a production process, for example.

Hardware-in-the-loop/human-in-the-loop

Real-time systems often involve machine-to-machine interactions (for example, computers interacting with experiments, tools, simulators, computers, or other systems). This segment of real-time computing is often called "hardware-in-the-loop" computing. At times, the system interacts with a human controller, such as a commanding officer in a force-on-force simulation, or a technician in a power plant. This is referred to as "human-in-the-loop" computing.

Real-time markets and applications

Real-time users seek differentiation, productivity improvements, ease of use, standardization, stable vendors, simpler systems, cost-effective solutions, and new ways to solve their problems. Cray Research answers those needs in three broad markets: data acquisition, simulation, and innovative markets. These real-time markets, which are served by the CRAY Y-MP EL real-time system, can be characterized by the application attributes listed in Table 1.

Real-time data acquisition systems

A data acquisition system is designed to handle the stringent needs of capturing steady streams of high-volume, high-frequency data. Data acquisition involves performing the same operation over and over on each item in a constant data stream. Examples of data acquisition systems applications include

- ☐ Engine test bed
- ☐ Factory control/supervisory control and data acquisition (SCADA)
- ☐ Imaging
- ☐ Mapping
- ☐ Radar and sonar
- ☐ Ranges/telemetry
- ☐ Seismic
- ☐ Signal intelligence (SIGINT)
- ☐ Telecommunications
- ☐ Transportation/traffic
- ☐ Weather/environment
- ☐ Wind tunnels/test stands

Engine test bed

Data acquisition from an aircraft engine test bed is an example of a moderate-to-high-end problem. Acquiring test data provides design engineers with important data to verify design points, determine operational characteristics, and refine models/simulations of the engine. It also permits safe operation of the engine in the test cell. Even with advances in simulation techniques, an engine has to be certified, stressed, and calibrated against real-world performance.

Table 1. Attributes of various real-time system markets

Target market attributes	Data acquisition	Simulation	Innovative markets
High data volumes	Y	N	Y
High bandwidth (CPU, I/O, and memory)	Y	Y	Y
Large memory systems	Y	Y	Y
Large I/O configurations and capacity	Y	Y	Y
Large amounts of data storage	Y	N	Y
Extensive vector processing	Y	N	Y
Extensive scalar processing	N	Y	Y
Mixed vector/scalar processing	Y	Y	Y
Multiprocessing	Y	Y	Y
Preprocessing/postprocessing of data	Y	N	Y
Extensible and easily configured systems	Y	N	Y
Price/performance sensitivity	N	Y	N
Need to solve leading edge application problems	N	Y	Y
Need to solve "large system" problems	Y	N	Y
Deterministic behavior	Y	Y	Y
Specialized I/O interfaces	Y	Y	N
Time critical	Y	Y	N
Response critical	N	N	Y
High speed response to interrupts	Y	Y	Y

The real-time characteristics of this application are tied to the monitoring and sensor data coming from the engine during operation. A fully instrumented engine in a test cell is expensive to operate and can be very expensive if the engine fails or is damaged.

When an engine is being monitored, a number of analog to digital (A/D) interfaces connect the engine to the system to obtain the engine's operational values. The operator of the engine test cell guides the engine through the test sequences and operations while the monitoring equipment gathers data on temperature, pressure, rotational velocity, fluid flow, and so on. This data is sampled at very high rates, processed, analyzed, displayed, and recorded for post-operation analysis.

Simulation

Closed-loop simulation systems with high-frequency models in the control loop must accept, process, and feed back information within stringent time frames. Examples of simulation applications include

- ☐ Braking systems
- ☐ Missile, aircraft, spacecraft
- ☐ Power plants
- ☐ Rotors
- ☐ Test beds
- ☐ Training
- ☐ Vehicles
- ☐ Visual systems

Braking systems simulation

In a sample application for an aircraft anti-skid/anti-lock braking device, systems are used to "flight-test" the brakes before actual airborne testing. This type of "hardware-in-the-loop" testing simulates extremes in weather, runway, and aircraft conditions in a closed-loop manner.

This application models wheel rotation, brake torque, aircraft performance, tire/runway friction, and gear motion. The computer models are used to take the place of the aircraft, and all other systems are physical mock-ups (hydraulic lines, wheels, struts, and braking electronics). These are supplemented by facilities for vibration, shock, temperature, EMI/RFI, and altitude variations.

Anti-skid brakes sense and maintain optimal friction between the tire(s) and the runway to minimize stopping distances under a variety of conditions. Continuous, controlled brake pressure helps maintain brake torque at optimal values. A/D sensors connected to the apparatus emulate the pilot's pressure on the brakes and hydraulics, and D/A interfaces direct the braking electronics to apply or reduce pressure to lower skid possibilities.

The entire simulation integrates models of aircraft dynamics (with six degrees of freedom), landing gear dynamics, tire/gear shimmy, strut dynamics, thrust/velocity, and braking drag. The numerous nonlinearities in the braking system, the multiple parameters represented by each model, and the variety of aircraft characteristics necessitate the use of real-time systems.

The CRAY Y-MP EL system brings supercomputing power to the real-time marketplace.

Real-time simulation and interaction with the physical hardware can be viewed by an operator many times before a braking system is actually deployed. Through testing and evaluation, the engineer can adjust braking performance to obtain optimal performance. Using real-time systems allows rigorous, safe testing before a system is flight-tested.

Innovative markets

Innovative applications are "response critical," whereas data acquisition and simulation applications are "time critical." In these "soft" real-time applications, rapid response is more critical than time-driven constraints. Examples of innovative market applications include

- ☐ Air traffic control
- ☐ Decision support systems (DSS)
- ☐ Genetics/biomedical
- ☐ Graphics/animation
- ☐ Online transaction processing (OLTP)
- ☐ Safety/monitoring
- ☐ Scheduling/reservations
- ☐ Virtual reality

Real-time DSS

In the fast-paced world of interlinked economic financial markets, timely decisions are necessary for success. Automated trading systems are finding their way into day-to-day operations. Real-time systems, albeit with "soft" time constraints, can be used to perform the automated tasks of securities/options trading across multiple markets.

Virtual reality

Simulated environments, sometimes called "virtual reality" systems, can be thought of as the next stage in the evolution of simulation and visualization. These systems involve human-in-the-loop interactions with simulated environments. Real-time system performance, response, capacity/bandwidth, and detailed rendering capabilities can be used to enhance the realistic appearance of the system. Obtaining realistic displays and feedback for larger and more complex systems will require mid-range to high-end real-time systems.

Limitations to today's real-time system approaches

Real-time applications are used to simulate, control, and monitor real-world systems. These applications are too complex for older, simpler systems, which cannot cope with this increase in complexity either because of hardware limitations, software restrictions, or both.

Real-time simulations demand greater detail, improved accuracy, computational power, and fidelity. In many real-time shops, users want to combine multiple models and multiple simulations into a single system. For example, a car manufacturer wants to model the engine, drive train, clutch, transmission, differential, wheels, and brakes

as a combined system. In past solutions, these models lacked tight integration. Multiple systems require greater flexibility for simultaneous simulations, greater adaptability for the configuration of multipurpose subsystems, and extensibility to accommodate everchanging models and lab environments. Prior systems cannot meet these needs.

In data acquisition applications there is an abundance of data to be collected. The amount, speed, storage, and retrieval tasks are larger and faster than past systems, and users are increasing their use of simulations instead of live testing. It is no longer economical to experiment without the use of simulation. In particular, repeated runs can be made without regard to life and equipment safety. Newer systems have a much larger capacity to run multiple simulations.

Users want to combine their standard computing, development, and real-time systems into a tightly coupled complex. Switching between systems is an unproductive vestige of past implementations. Today's real-time systems combine these elements into an integrated system combining modeling, programming, and deployment.

UNIX and real-time extensions

Real-time extensions to the UNIX operating system offer the optimal blend of deterministic, real-time performance with the application environment and compatibility of standard UNIX. As UNIX grew from a development environment for program developers into a more general-purpose interactive system, several features evolved which ease the transition from general-purpose to real-time systems. These include support for multiple processes, synchronization of processes through signals, and various communication mechanisms such as IPCs and pipes.

An important element of real-time processing is the relationship between asynchronous events of the real world and the multiprocessing/multitasking model used for many real-time applications. Tasking is used to parallelize computational tasks, to respond to I/O and interrupts, or to respond to asynchronous events. UNIX has a strong heritage of support for processes; however, a standard tasking model for real-time performance has not emerged.

As UNIX evolved, it remained unsuitable for real-time applications. Several functions prevented real-time performance. These shortcomings are discussed in the following paragraphs:

- ☐ Determinism. Because UNIX is not a preemptive kernel and because of its heritage as an interactive development system, many activities of differing lengths can occur at any time. When an event must occur as in real-time processing, the kernel may be preoccupied or it may be ready to respond—this lack of predictable response is a problem for real-time processing.
- ☐ Preemption. Preemption within the system ensures that a higher priority process will preempt a lower priority process. The issues associated with preemption within the kernel are shown in Figure 3 as three different approaches, ranging from normal UNIX (nonpreemptive) to full preemption, with an interim compromise based on preemption points.
- ☐ Interruptibility. The kernel is not fully interruptible; therefore, long delays may inhibit rapid response for real-time processing.
- ☐ Interrupt latency. Figure 4 shows the components of interrupt response. In UNIX there is no optimization for low interrupt response times.
- ☐ Context switch. One key measure of a real-time system is the time taken to switch from one process to another. In general-purpose systems like UNIX, the system seeks to maximize the best "average" case for most measures of throughput and response. In real-time systems, the design of a system is biased toward the best "worst" case—in other words, the goal is to constrain all system behavior to a predictable worst case.
- ☐ Timers. UNIX does not support high-resolution timers and clocks for time-based events.
- ☐ Preallocation of resources. Real-time applications avoid high overhead within the system by preallocating and dedicating resources for real-time operation.
- ☐ Segregate resources. UNIX does not have a provision for segregating system resources (CPUs,

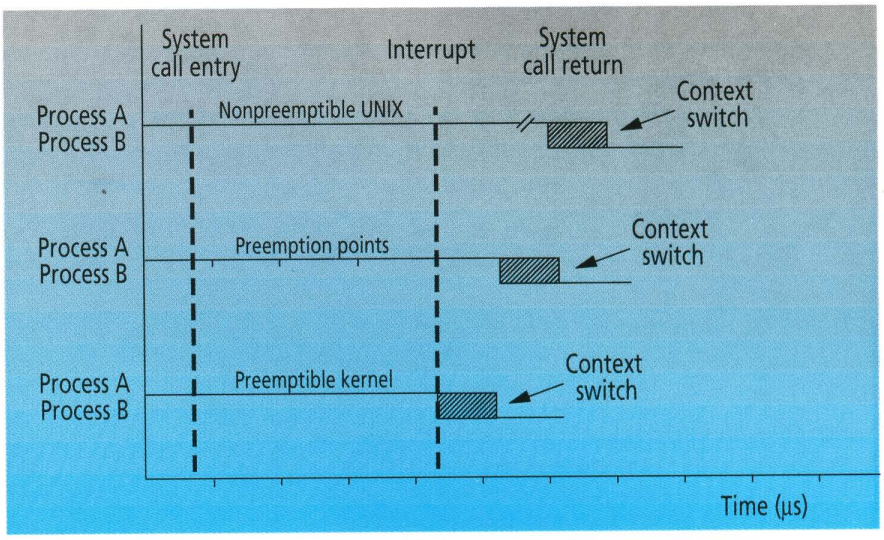


Figure 3. Three approaches to preemption within the operating system.

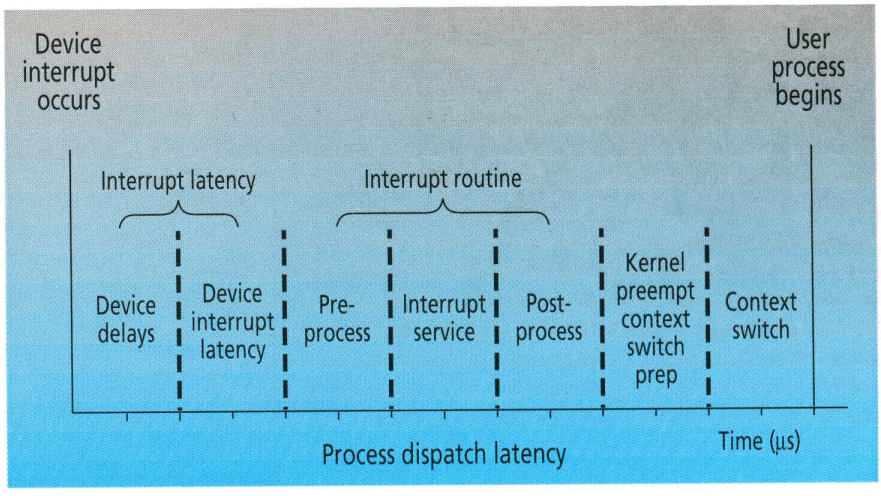
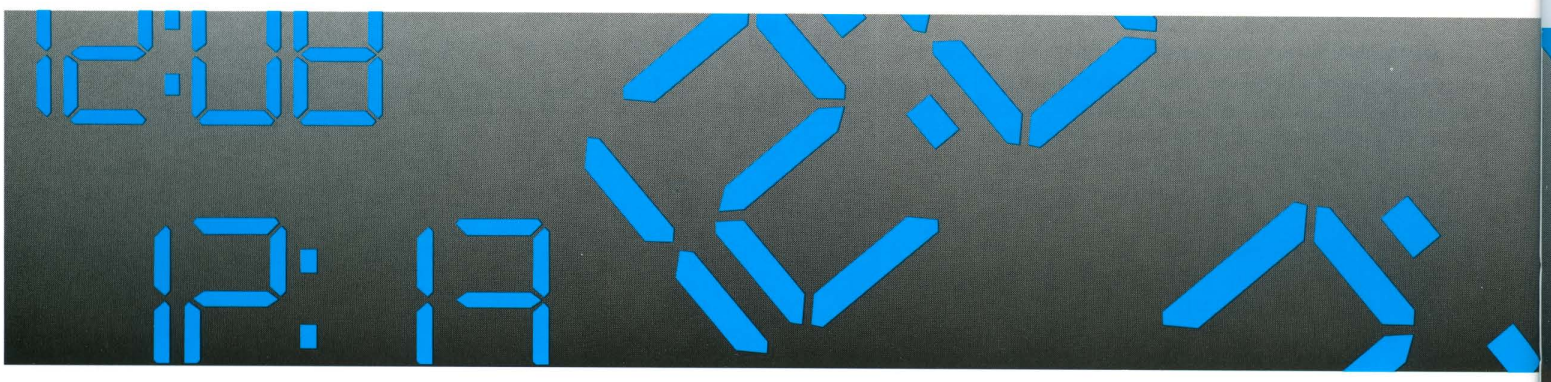


Figure 4. Interrupt response components.



memory, files, and I/O connections) for real-time versus non-real-time processing.

- ☐ Real-time algorithms. The system algorithms in UNIX are not tailored for real-time processing. Algorithms for memory management, CPU scheduling, and other resources are not designed to support real-time processing.
- ☐ Synchronous kernel. The UNIX kernel is synchronous by design. In real-time processing, most of the events from the real world occur asynchronously and cannot be blocked by kernel synchronization.
- ☐ Frame times. Frame times are crucial to real-time processing. A frame is the unit of time during which a real-time application performs its periodic processing. In data acquisition, for example, a 20 ms frame time defines the time between the external interrupt/event, the system's overhead to begin execution in the user's application, the application code itself, and any wrap-up processing. UNIX is not designed to constrain work within a frame.

A real-time kernel based on UNIX needs several additional features. It must be able to support real-time scheduling, be preemptive, guarantee interrupt response times, support interprocess communications, perform high-speed data acquisition, provide I/O support, and allow user control of system resources.

With multiple features lacking in UNIX, several vendors began adding real-time features to their kernels. Left unconstrained, this evolution would have deviated from the standard UNIX users want. Therefore, a push for standardization of real-time extensions appeared.

POSIX 1003.4

The IEEE has defined a standard Portable Operating System Interface for Computer Environments (POSIX). The first of the POSIX standards (1003.1) defines the interface between portable applications and the operating system. The real-time extensions to POSIX 1003.x are identified as P1003.4.

The industry's move to standardization offers users compelling advantages to move away from proprietary kernels towards the POSIX 1003.4 standard. These include vendor independence, availability of applications, preservation of software investment, availability of trained personnel, connectivity to other systems, and some degree of insulation from technology obsolescence. In contrast, many of today's aging proprietary operating systems make real-time application development cumbersome, complicated, and expensive.

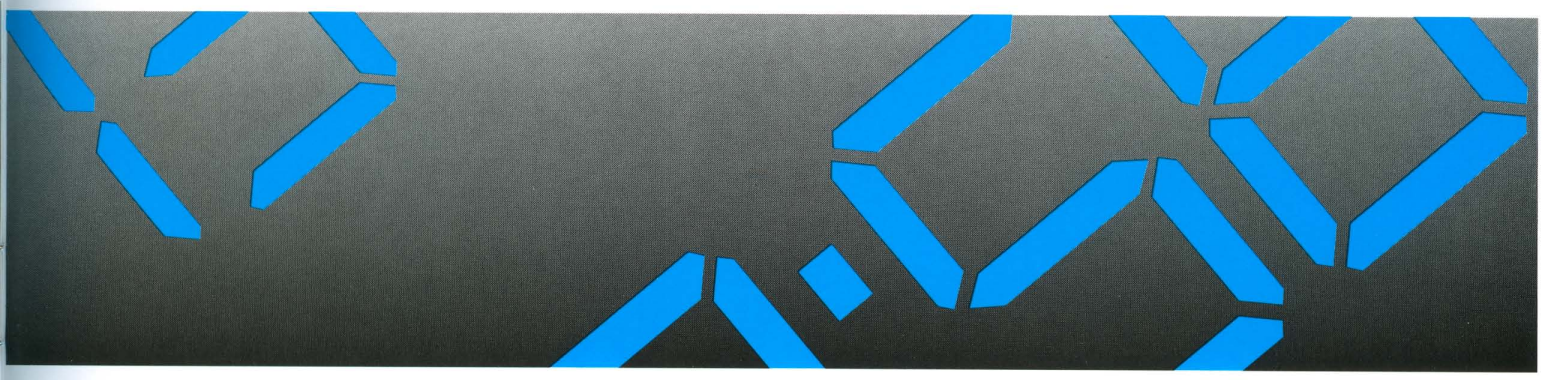
The real-time POSIX features being incorporated into Cray Research's real-time software are

- ☐ Binary semaphores
- ☐ Memory locking and shared memory
- ☐ Priority scheduling
- ☐ Asynchronous events
- ☐ Clocks and timers
- ☐ IPC messaging
- ☐ Asynchronous I/O
- ☐ Synchronous I/O
- ☐ Real-time files

Real-time hardware

The initial hardware platform for Cray Research's real-time systems will be the CRAY Y-MP EL system. Real-time features for timers, interrupts, and I/O interfaces will be available through the incorporation of third-party products into the CRAY Y-MP EL VME chassis. Use of standard VME connections allow vendor and customer interfaces to be developed expediently. Initial releases of real-time software will support both standard CRAY Y-MP EL peripherals and real-time interfaces as described below:

- ☐ Standard peripherals. The CRAY Y-MP EL system supports a wide selection of industry-standard peripherals, including IPI and ESDI disks, disk arrays, removable disks, Ethernet, DAT tapes, 9TR tapes, 8 mm tapes, 3480 tapes, FDDI, HIPPI, and SCSI.
- ☐ Real-time peripherals. Real-time markets often have unique connectivity requirements. An initial complement of real-time peripherals will be updated continually, based on customer needs. Initial real-time support includes VME-based interfaces to A/D, D/A, MILSTD-1553B, Encore's HSD, IEEE-488, DR11W, DRB-32, non-host real-time Ethernet, SCSI, RS232/422, digital I/O, video frame grabbers, discretes, and graphics accelerators.
- ☐ Real-time I/O systems. Under terms of a recently announced agreement, Cray Research and Applied Dynamics International, Inc. (ADI) are integrat-



ing their two systems and will jointly market the integrated product to customers needing robust, supercomputer real-time performance. The combined CRI/ADI system will address applications such as "hardware-in-the-loop," high bandwidth data acquisition, and closed-loop process control.

Real-time software

Cray Research's real-time group is building software products, integrating new real-time peripherals, and melding them with CRAY Y-MP EL hardware and software to produce Cray Research's real-time software products. The real-time software products will include a real-time operating system, runtime libraries, networking, peripheral drivers, utilities, tools, and end-user development interfaces.

Product overview

The suite of Cray Research real-time products consists of CRAY Y-MP EL hardware, IOS, real-time I/O interfaces, and the following components:

Real-time kernel
- ☐ POSIX 1003.1/1003.4 support
- ☐ Preemptive/interruptible
- ☐ Fast interrupt response and context switch
- ☐ System algorithms optimized for real-time
- ☐ Real-time memory/swap management
- ☐ Administrative and resource controls

Kernel options
- ☐ NFS
- ☐ Real-time file system
- ☐ Memory-based file system
- ☐ UNICOS file system interface
- ☐ Swap accelerator
- ☐ File system accelerator

I/O products based on the standard CRAY Y-MP EL IOS
- ☐ Synchronous and asynchronous I/O
- ☐ Standard drivers (tapes, for example)
- ☐ BMR support and drivers
- ☐ Real-time specific peripherals and drivers such as A/D and D/A
- ☐ Graphics accelerators and drivers
- ☐ Driver development kit
- ☐ Support for the customer-defined VME modules/interfaces
- ☐ Support for user-written drivers

UNICOS compilers
- ☐ Supported compilers include the Cray Standard C Compiler, Cray Research CF77 compiling system, Cray C++ compiling system, Cray Pascal, and the Cray Ada compiler.

Networking
- ☐ Real-time communications and networking products

Libraries
- ☐ Language-specific libraries for the Cray Standard C Compiler, CF77, Cray C++, Cray Pascal, and Cray Ada
- ☐ Real-time library extensions
- ☐ Modified multitasking library
- ☐ Runtime support for UNICOS system calls

Tools, commands, and utilities
- ☐ Standard and real-time commands
- ☐ New tools for administrators and developers
- ☐ Optimized debuggers

Graphics and visualization software such as the Cray Visualization Toolkit (CVT) and Cray Research IRIS Explorer

Third-party tools

Documentation

By combining real-time features with the UNIX operating system and making the software available on the CRAY Y-MP EL supercomputer, Cray Research brings supercomputing power and a rich software environment to the real-time marketplace. ▬

About the author

Lance Miller is a senior director managing the development and product marketing functions for Cray Research's Real-time Systems Group. Miller received a B.S. degree in computer science from the United States Air Force Academy. His involvement with real-time systems began in the Air Force and continued with real-time development during the past two years after working in general-purpose computing.

CORPORATE REGISTER

Minnesota Supercomputer Center orders most advanced Cray Research supercomputers

The **Minnesota Supercomputer Center, Inc. (MSCI)** and Cray Research announced an agreement in July that calls for MSCI to acquire Cray Research's newest and most advanced supercomputer technology. Under the terms of the agreement, Cray Research will install a CRAY Y-MP C90 system in the second quarter of 1993. The company also will install a CRAY Y-MP M92 system at MSCI during the fourth quarter of 1992.

John Sell, president of MSCI, described the CRAY Y-MP C90 system as "the most powerful general-purpose supercomputer in the world. It will be an essential component of the computing resources we offer to our commercial, government, and academic users." The CRAY Y-MP M92 system, part of a product series announced by Cray Research in May that features the world's largest central memories, will provide MSCI with important new capabilities until the arrival of the CRAY Y-MP C90 system.

"Supercomputing technology is increasingly vital for Minnesota's economic development, global competitiveness, and academic research leadership," said Minnesota Governor Arne H. Carlson. "We are fortunate to have the world's leading supercomputer manufacturer, Cray Research, here in our state, and we are very pleased that the Minnesota Supercomputer Center has chosen to strengthen its ties to Cray Research to ensure it maintains its cutting-edge in computing capability."

AUDI AG, the German automotive manufacturer and member of the Volkswagen group, ordered and installed a CRAY Y-MP 2E supercomputer in August. AUDI will use the system for research and development in crash analysis, finite element analysis, and computational fluid dynamics. As part of the order, AUDI has licensed Cray Research's Multipurpose Graphic System (MPGS), an easy-to-use postprocessing and visualization software tool for engineering applications.

The UK **Science and Engineering Research Council (SERC)** in June ordered a CRAY Y-MP 8I system to support a broad range of science and engineering projects. The system will be installed at the Atlas Centre of SERC's Rutherford Appleton Laboratory in the United Kingdom, where it will replace a CRAY X-MP system. Immediate projects for the supercomputer include oceanographic and atmospheric studies and engineering projects in structural analysis and fluid flow.

The U.S. **Environmental Protection Agency (EPA)** ordered a two-processor CRAY Y-MP system in the second quarter. The order contract includes an option for another supercomputer for the EPA's Research Triangle Park, North Carolina, facility within an eight-year time frame. EPA researchers at the National Environmental Supercomputing Center in Bay City, Michigan, will use the new supercomputer to study problems such as the effect of automobile exhaust on the ozone layer, the creation of acid in the atmosphere, and pollution in the Great Lakes and Chesapeake Bay region.

Cray Research announces new version of Ada software programming environment

Cray Research announced in June the availability of Cray Ada 3.0, the company's enhanced Ada language programming environment. When combined with the fast processing speeds of Cray Research systems, Cray Ada 3.0 provides the world's highest-performing Ada development platform for a variety of applications.

This software environment lets users write efficient Ada computer programs without extensive hardware knowledge. The Cray Ada environment includes

- ☐ A compiler that generates code for execution on Cray Research computer systems
- ☐ An execution environment, or runtime system, that provides support for executing Ada programs
- ☐ Library manager tools
- ☐ A linker that binds and links compiled Ada source code into executable programs
- ☐ A debugger
- ☐ Language tools
- ☐ A profiler that gathers subprogram calling statistics and program timing information
- ☐ An optimizer that increases the efficiency of compiled programs

A significant enhancement in Cray Ada 3.0 is multitasking, which allows Ada tasks to take advantage of the parallel processing capabilities of Cray Research systems.

Cray Research sponsors award honoring pioneers in computational science

Cray Research announced in June that next year the company will sponsor a Benefactor Award, the highest level of sponsorship, at the 1993 Computerworld Smithsonian Awards.

Established in 1989, the Computerworld Smithsonian Core Program Awards recognize and publicly honor information technology innovations that have contributed to social progress. Winning entries are selected in ten categories and showcased in a major permanent exhibit at the Smithsonian Institution's Museum of American History.

Benefactors such as Cray Research establish their own nominating committees, with a maximum of eight members who must be approved by the Foundation. Each member may submit up to four nominations. The winner is selected by the Computerworld Smithsonian Foundation.

Cray Research has been involved with the Technology Leadership Awards since 1991. Last year, the company submitted nominations for the general award program and contributed $20,000 to the program. This year, John Rollwagen, chairman and chief executive officer of Cray Research, was a member of the MCI Benefactors Award nominating committee.

APPLICATIONS UPDATE

UniChem unbundled

Since Cray Research introduced its quantum chemistry software environment, UniChem, in 1989, some computational chemists have expressed interest

in obtaining one or two of the package's programs. The company has responded to this demand by unbundling the UniChem package so customers can obtain the exact components they require.

The following UniChem components may be chosen individually or in combination:

- ☐ CADPAC 5.0, a Hartree-Fock-based ab initio program written and maintained by Roger Amos and associates at Cambridge University in England
- ☐ DGAUSS, a density-functional-theory-based ab initio program developed at Cray Research
- ☐ Gaussian 92, a Hartree-Fock-based ab initio program developed by John Pople and associates at Carnegie-Mellon University. Gaussian 92 is distributed and maintained by Gaussian, Inc.
- ☐ MNDO91, a semiempirical program developed by Walter Thiel and associates at the University of Wuppertal, Germany

Existing Cray Research customers who already have Gaussian 92 can obtain the UniChem interface for their Silicon Graphics workstations.

For additional information regarding the unbundling of UniChem, contact Mark Cole, 655E Lone Oak Drive, Eagan, MN 55121; telephone: 612/683-3688, fax: 612/683-3099.

Usage-based pricing for UniChem

Cray Research now offers a usage-based pricing option for its UniChem computational quantum chemistry product. This option allows installation of UniChem codes at supercomputer centers around the world. The UniChem interface is installed on workstations at customer sites. Users can build molecular models, launch calculations to the Cray Research supercomputer via a network to the supercomputer center, and visualize and analyze the results at their desktop. UniChem fees are based on an hourly charge for CPU time at the supercomputer center.

For additional information regarding the UniChem usage-based pricing option, contact Mark Cole, 655E Lone Oak Drive, Eagan, MN 55121; telephone: 612/683-3688, fax: 612/683-3099.

Application Integration Toolkit (AIT) 2.0

The Application Integration Toolkit (AIT) 2.0 has been released and is now available from Cray Research. AIT provides application developers with the tools to create distributed applications between "client" workstations and Cray Research supercomputer "servers." AIT is intended for applications that follow the client-server paradigm. One example of such an application is a graphical user interface running on a workstation and interacting with a server application running on a Cray Research supercomputer.

AIT 2.0 functionality for Cray Research supercomputer/client applications enables the user to

- ☐ Initiate a job on a Cray Research system for immediate execution or send a job to an NQS queue.
- ☐ Monitor and control a job. Retrieve a variety of statistics on the supercomputer/server part of a distributed application and control or steer the remote application.
- ☐ Establish connections and transfer binary data between processes. Transfer is bidirectional and data are translated into the appropriate formats for each machine.

AIT functionality for Cray Research supercomputer/server applications includes:

- ☐ Application status to the AIT system
- ☐ Checkpointing and restarting of the Cray Research resident part of a distributed application

AIT 2.0 is available for any Cray Research supercomputer running UNICOS 6.1 or 7.0. Client libraries only are supported on Sun Microsystems, Inc., Sun-4 and SPARC workstations running SunOS 4.1; Silicon Graphics, Inc., IRIS 4-D running IRIX 3.3 or higher; IBM RS/6000 workstations running AIX 3.1; and DECstation and VAX systems running ULTRIX 4.1. For more information on AIT, contact Russ Loucks, 655E Lone Oak Drive, Eagan, MN 55121; telephone: 800/284-2729.

MOLDFLOW and Cray Research introduce supercomputer versions of molding analysis software

Three new plastic molding analysis software packages, MF/FLOW, MF/COOL, and MF/WARP, specially designed to operate on Cray Research's CRAY Y-MP computer systems, were announced jointly by Cray Research and MOLDFLOW PTY LTD in July.

MF/FLOW is used for flow analysis of the filling and packing phases of plastic injection molding; MF/COOL is used to analyze heat exchange and temperature control in the mold as the plastic cools; and MF/WARP is used to determine net distortion of the molded component, diagnose the causes of the distortion, and determine component strength and stiffness under load.

More than 1000 commercial sites around the world use MOLDFLOW's programs to improve the quality of their products and achieve cost savings through improved product performance, reduced scrap, minimized design lead-times, and lowered design and manufacturing costs. With the full library of MOLDFLOW analysis packages available on Cray Research supercomputers, users can design products and the tooling and manufacturing processes needed for

trouble-free production more quickly and effectively.

Apple Computer, Inc., the first company to purchase a license for the new supercomputer versions of the MOLDFLOW packages, uses the software to design its personal computer cabinetry.

Apple recently used MOLDFLOW on its CRAY Y-MP 2E system to analyze various designs and injection processes for the plastic cabinet of Apple's successful Macintosh PowerBook series of laptop computers.

"The personal computer industry moves at such a rapid pace that it allows very little time to analyze the complex geometries of our computer cabinetry," said Sandra Morgan, molding analyst from Apple. "With MOLDFLOW software on our Cray Research system we were able to analyze nearly all molded components of the PowerBook prior to committing huge sums of money for tooling. This powerful simulation capability enabled us to produce an extremely high-quality product and bring the PowerBook series to market more quickly than we could have without the supercomputer capability."

For more information on MF/FLOW, MF/COOL, and MF/WARP contact Bill Hicock; MOLDFLOW PTY LTD; 2 Corporate Drive, Suite 232, Shelton, CN 06484; telephone: 203/925-0552 or Mike Obermier, Cray Research, 655E Lone Oak Drive, Eagan MN 55121; telephone: 612/683-3650.

Cray Research IRIS Explorer 1.0 available for use on Cray Research systems

Developing customized computer applications can be a time-consuming task, often too complex for anyone but full-time computer programmers. To make the task easier for programmers with varying degrees of expertise, Cray Research and Silicon Graphics, Inc. have cooperated to deliver the Cray Research IRIS Explorer distributed application building environment. Cray Research IRIS Explorer puts the computational power and speed of Cray Research systems and the three-dimensional graphics capabilities of Silicon Graphics workstations in the hands of users who want to solve problems instead of investing their resources in programming.

The Cray Research IRIS Explorer visually based object-oriented application building environment combines the development and use of distributed applications. Creating applications is easy. Users simply select modules with a mouse and drag them into position. With a few clicks of the mouse, the modules are connected into an intuitive, easy-to-read data flow diagram. Another click, and the application runs. With a few simple movements of the mouse, users have created a sophisticated distributed processing application.

To create an application, users place modules in the windows and connect them with lines, creating the data flow diagram or "map."

Each module is a single, self-contained, preprogrammed software tool dedicated to performing a single task. Users can choose from modules residing on the workstation and on the Cray Research system. Because some modules on the Cray Research system perform the same function as some of those on the workstation, users decide which modules to select, based on problem requirements.

Modules can be added to or taken from a map without disturbing the code in other modules. Users can easily insert a function, such as a visualization module, in the middle of the process to check intermediate results. Each module has clearly defined input and output ports that control the flow of data through the module. Preexisting code may be added using the mouse to match variables. While an application is running, all module parameter controls are visible and can be changed during the application run to review possible results.

Cray Research IRIS Explorer requires version 6.1 or later of Cray Research's UNICOS operating system and a workstation running Silicon Graphics IRIS Explorer. For more information on Cray Research IRIS Explorer, contact your Cray Research representative.

Khoros software development environment now available on Cray Research systems

The Khoros system, created by the Khoros Group at the University of New Mexico, provides engineers and scientists with a flexible tool for information processing, data visualization, and software development. Khoros utilizes current industry standards and is based on the X Window System. As a member of the Khoros Consortium, Cray Research has worked closely with the Khoros Group to port this environment to its systems. Currently, over 14 sites are running Khoros 1.0 on Cray Research systems.

Components of the Khoros software system include

- ☐ Visual programming language
- ☐ Code generators for extending the visual language and adding new applications to the system
- ☐ Interactive user interface editor
- ☐ Interactive image display package
- ☐ Two- and three-dimensional plotting packages
- ☐ Animation program
- ☐ Image warping application
- ☐ Imagery/elevation display program

An interactive user interface editor enables Khoros programs to combine a UNIX-like command with an X Window System-based graphical user interface (GUI) to extend and add applications to the system. These extensions include the animation display program, which enables users to layer a sequence of images on their X Window System workstation. Khoros also features a surface-rendering application that arranges data in a presentable form and allows data to be "draped" over other data on the screen. For example, a curved image of the earth and a weather map can be warped and then registered together to present a realistic picture of the combined data on the screen.

Khoros contains extensive libraries of nearly 300 UNIX programs used for image processing, digital signal processing, numerical analysis, classification, feature extraction, data conversion, and many other areas.

Khoros has been applied in such diverse areas as virtual reality, telecommunications, medical imaging, remote sensing, and optics. It uses Cray Research's vector and parallel processing capabilities effectively and runs on 16 of the major UNIX X Window System workstation platforms such as Sun Microsystems, Digital Equipment, and Silicon Graphics. Currently, porting efforts are under way for 386/486 machines as well as the Apple Macintosh.

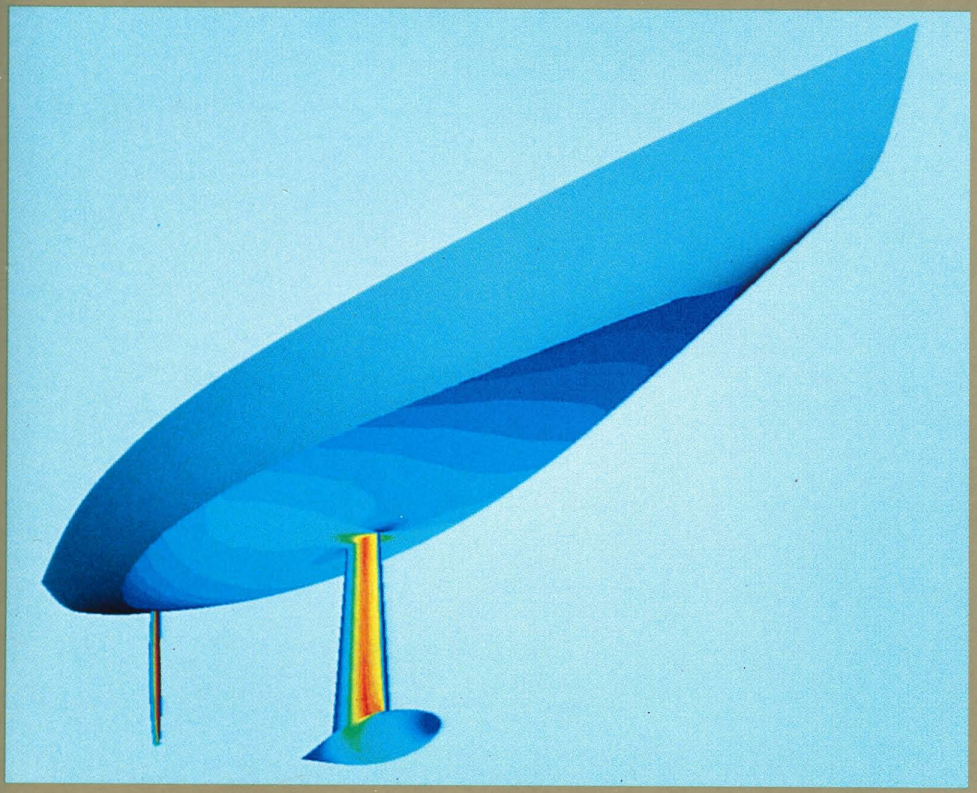
The Khoros system is available via anonymous ftp from pprg.eece.unm.edu at no charge, and directly from Cray Research. Support is provided via both an e-mail users group and USENET news. For more information about Khoros on Cray Research computer systems, contact Bill Samayoa, Applications Department, Cray Research, Inc., 655E Lone Oak Drive, Eagan, MN 55121; telephone: 612/683-3645; or Tom Sauer, The Khoros Group, University of New Mexico, Department of EECE, Rm. 110 EECE Building, Albuquerque, NM 87131; telephone: 505/277-6563.

USER NEWS

Building a better boat: Cray Research contributes to America's Cup technology

"It is fundamentally a technology contest," said John Marshall, general manager of the Partnership for America's Cup Technology (PACT), the joint effort to develop basic technology to aid American syndicates in the 1992 America's Cup defense. "The America's Cup competition tests a nation's technological ability to develop the best boat."

A CRAY Y-MP supercomputer, a key weapon in the United States' arsenal, helped Bill Koch's "America Cubed" (America^3) and Dennis Conner's "Stars and Stripes" prepare for competition in the America's Cup elimination series and defender trials. America^3 won the race. The America^3 Foundation ultimately completed its own final design, having used PACT resources to verify and validate a variety of design options throughout its campaign.



CFD results of a generic America's Cup yacht display the induced drag characteristics of a boat's underwater parts.

Cray Research joined The Boeing Company, IBM Corp., and Science Applications International Corporation (SAIC) in PACT in March 1991. PACT and a team of researchers at Boeing modified computational fluid dynamics (CFD) codes used in aircraft design to be applied to yacht design. The design of the keel (a fin resembling an airplane wing and usually made of lead, that extends from the bottom of the boat into the water) became the primary focus. Because the keel serves two purposes—to hold the boat upright and to act as a lift force to counterbalance the wind—an optimal keel design must have a low center of gravity, low drag, and high lift.

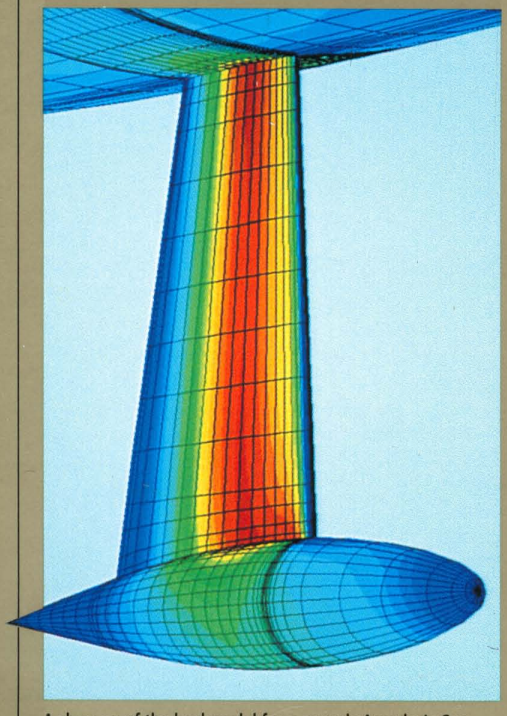
"Because the keel is trying to perform two tasks at once, optimizing this kind of configuration is quite complex," said Marshall. "With a sophisticated tool like the CRAY Y-MP system, we were able to begin with a fundamental mathematical and analytical approach and then use massive computing power to branch out and find two or three regimes which looked promising, instead of just one. We got some answers that not everyone expected, and that's when it got exciting.

American keel development was significantly more advanced than the competition's, and that was due largely to supercomputing."

For Conner, turnaround and throughput were the main advantages of supercomputing. "Dennis came to us on a Friday and said he needed to start cutting metal on Monday," said Ed Tinoco, associate technical fellow and technical leader of the Boeing research team. "With the CRAY Y-MP system we were able to run about a dozen design analyses a day, as opposed to the two or three we could and did run on some of our workstations. We couldn't have done the cutting-edge Navier-Stokes work in the short timeframe without the supercomputer."

Two additional teams, one at South Bay Simulations, Babylon, New York, and another at SAIC in San Diego, conducted complementary CFD code work using donated CRAY Y-MP supercomputing time from Cray Research to focus on calculating boat hull drag as well as keel performance.

Cray Research resources also were important in the area of rough water performance optimization, where the task is to predict which of two alternate designs will be best in rough water, where ocean waves cause the boat to pitch and heave and thus slow it down. Paul Sclavounos at the Massachusetts Institute of Technology (MIT) developed a three-dimensional CFD code that ran on the Cray Research system at MIT and provided the theoretical capability to make this prediction. Said Marshall, "Once we made this prediction, we built a model of the boat and towed it through a tank where waves were being made. Our objective was to validate the code's predictive ability. We measured the boat's motion—how much it pitched and heaved—and resistance—how much force was needed to keep the boat moving. We compared the resistance results to towing the boat through a tank with no waves. The difference in the two is called the added resistance in waves, and that's what we needed to minimize."



A close up of the keel model for a generic America's Cup yacht.

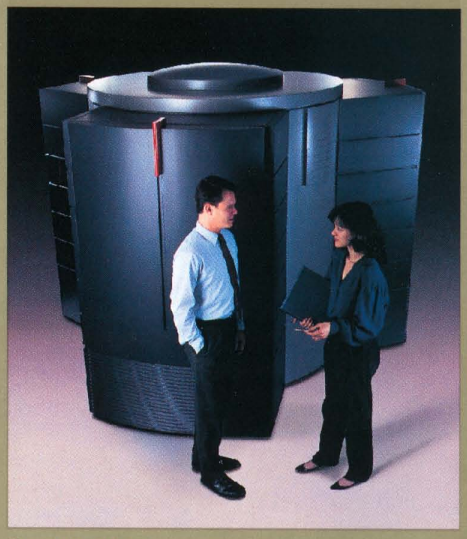
Minimizing the wave-making drag of the hull was a major objective of the South Bay Simulations Team. Wave-making drag refers to those waves the boat itself makes as it moves through the water, representing energy lost by the boat. "If you have the tools to predict what a particular boat's wave-making drag will be and can therefore avoid full prototype testing, you're far ahead of someone who has to actually build the boat and sail it in real life," said Marshall. "For these kinds of codes, which are extremely complex and massive computationally, supercomputers really are the only tools that can accommodate the many designs and the many different speeds at which you're running them.

"Throughout the America's Cup competition, we were attacking substantially different major technologies simultaneously in universities and research labs around the country," Marshall continued. "Technology is a long-term commitment, a building block process. If you let it lie for a long time, it becomes obsolete. We already are putting together the PACT program for 1995, and we hope to involve Cray Research in a big way."

CRAY Y-MP C90 system achieves an easy 10 GFLOPS on seismic migration codes

Through an international network linked to a CRAY Y-MP C90 supercomputer located in the United States, Cray Research showed approximately 1500 attendees of the European Association of Exploration Geophysicists (EAEG) Technical Exposition, held in Paris in June, the value of using Cray Research technology for three-dimensional seismic migration—consistent and easily achievable speed on production-size data sets.

The company demonstrated sustained 6, 8, and 10 GFLOPS performance on three popular three-dimensional post- and prestack seismic migration codes. These codes are used for processing and analyzing field data to image the earth's subsurface for the purpose of more easily and readily locating oil. This application is the most important computer problem for geophysicists and petroleum engineers. Users can run their code easily on the CRAY Y-MP C90 system with little or no reprogramming.

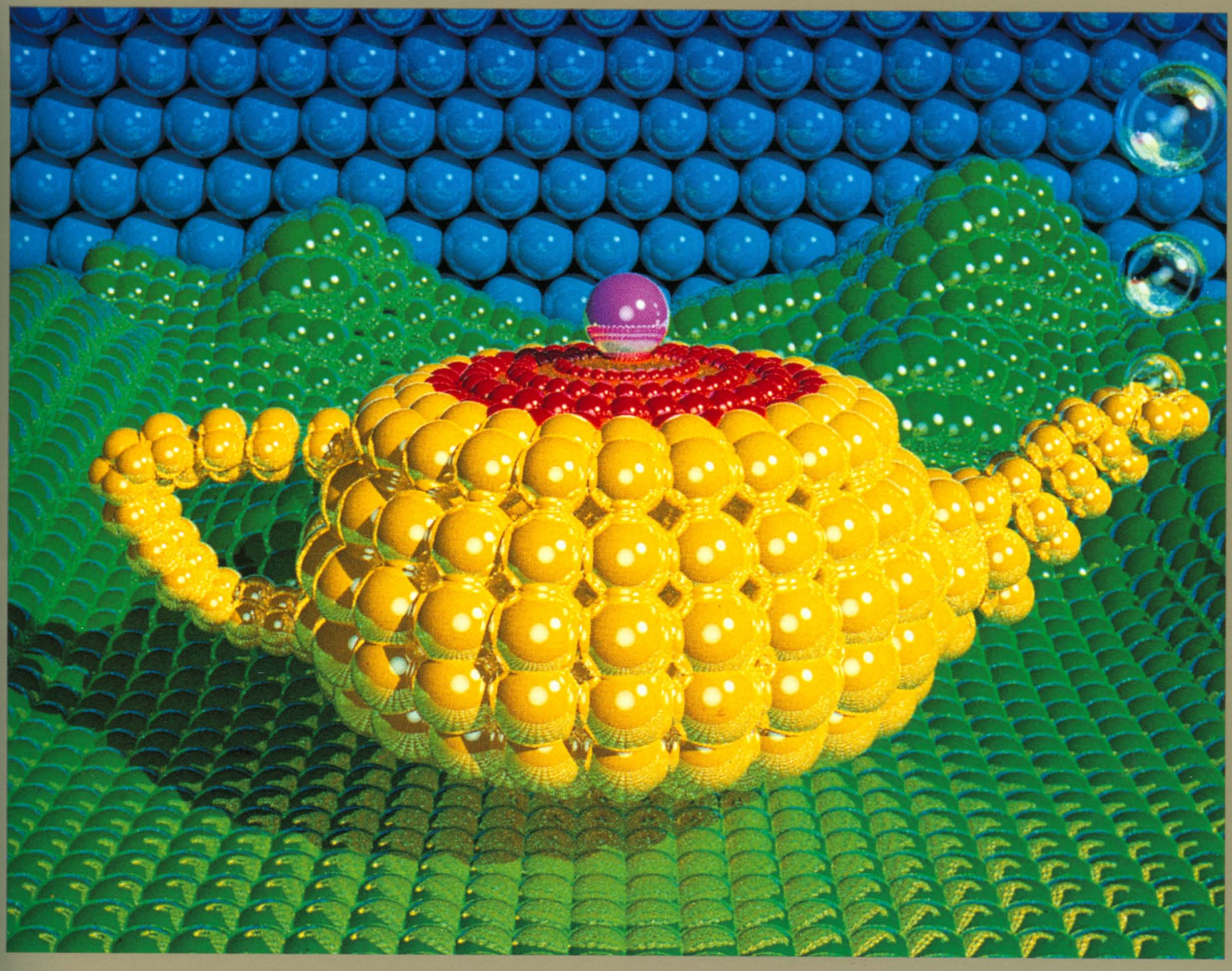


The CRAY Y-MP C90 computer system.

The production-scale problems demonstrated at the Paris show involved segments of land approximately 10 by 10 kilometers and 5 kilometers deep, as well as 15 by 7.5 kilometers and 3 kilometers deep. The codes demonstrated were GEOSYS by GECO-PRAKLA, a 3-D, poststacked seismic depth migration code, which ran in excess of 10 GFLOPS; a Cray Research-written code for 3-D, Kirchhoff, prestack migration analysis, which ran in excess of eight GFLOPS; and the Split Step Fourier 3-D depth migration code written by Paul Stoffa, professor of geological sciences at the University of Texas (UT), called SP3D, which ran in excess of six GFLOPS.

"Ease is the real benefit we're demonstrating," said Tom Eliseuson, Cray Research's petroleum industry marketing director. "The message is not so much achieving the superstar numbers, but that Cray Research has a uniquely well-balanced architecture that's ideally suited for this kind of capability. Years of software development combined with continuous hardware enhancements make Cray Research systems easy to use. Most well written and reasonably structured seismic migration codes can be expected to achieve over 6 GFLOPS on the CRAY Y-MP C90 system. You can't get that kind of performance today on any other system."

GALLERY



"Spherical Universe" was produced on a CRAY Y-MP supercomputer at the Los Alamos National Laboratory using the SCOPE program. In this illustration of ray tracing, all objects, including the sky, consist of spheres. Image courtesy of Melvin Prueitt, C-6, LANL.