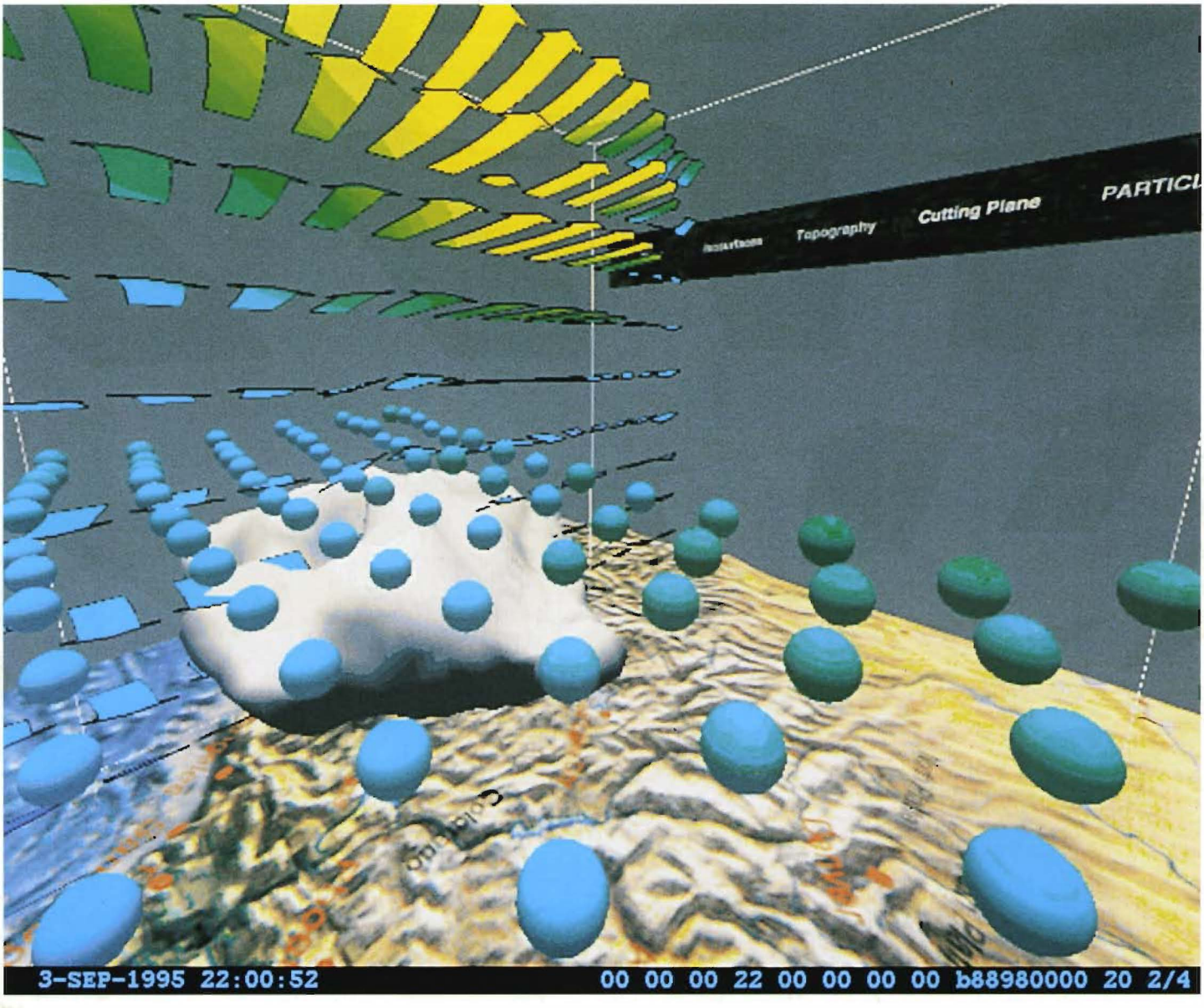
HIGH PERFORMANCE FORTRAN
IN PARALLEL ENVIRONMENTS

SEQUOIA 2000 RESEARCH

Digital Technical Journal

digital™



Volume 7 Number 3
1995

Cover Design

The images on the front and back covers of this issue are different visualizations of the same data output from a regional climate simulation program run by Dr. John Roads of the Scripps Institution of Oceanography. The data depicted contain measures of temperature, liquid and gaseous water content, and wind vectors; the topography represented by the data is the western U.S. in January 1990. Providing earth scientists with the ability to visualize such data is one of the objectives of the Sequoia 2000 research project—a joint effort of the University of California, government agencies, and industry to build a computing environment for global change research. This issue presents papers on several major areas explored by Sequoia 2000 researchers, including an electronic repository, networking, and visualization.

The cover was designed by Lucinda O'Neill of Digital's Design Group. Special thanks go to Peter Kochevar for supplying the cover images.

The *Digital Technical Journal* is a refereed journal published quarterly by Digital Equipment Corporation, 30 Porter Road LJO2/D10, Littleton, Massachusetts 01460. Subscriptions to the *Journal* are $40.00 (non-U.S. $60) for four issues and $75.00 (non-U.S. $115) for eight issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to dtj@digital.com. Single copies and back issues are available for $16.00 each by calling DECdirect at 1-800-DIGITAL (1-800-344-4825). Recent back issues of the *Journal* are also available on the Internet at http://www.digital.com/info/DTJ/home.html. Complete Digital Internet listings can be obtained by sending an electronic mail message to info@digital.com.

Digital employees may order subscriptions through Readers Choice by entering VTX PROFILE at the system prompt.

Comments on the content of any paper are welcomed and may be sent to the managing editor at the published-by or network address.

ISSN 0898-901X

Documentation Number EY-T838E-TJ

Book production was done by Quantic Communications, Inc.

Contents

Editor's Introduction

Scientists have long been motivators for the development of powerful computing environments. Two sections in this issue of the *Journal* address the requirements of scientific and technical computing. The first, from Digital's High Performance Technical Computing Group, looks at compiler and development tools that accelerate performance in parallel environments. The second section looks to the future of computing; University of California and Digital researchers present their work on a large, distributed computing environment suited to the needs of earth scientists studying global changes such as ocean dynamics, global warming, and ozone depletion. Digital was an early industry sponsor and participant in this joint research project, called Sequoia 2000.

To support the writing of parallel programs for computationally intense environments, Digital has extended DEC Fortran 90 by implementing most of High Performance Fortran (HPF) version 1.1. After reviewing the syntactic features of Fortran 90 and HPF, Jonathan Harris et al. focus on the HPF compiler design and explain the optimizations it performs to improve interprocessor communication in a distributed-memory environment, specifically, in workstation clusters (farms) based on Digital's 64-bit Alpha microprocessors.

The run-time support for this distributed environment is the Parallel Software Environment (PSE). Ed Benson, David LaFrance-Linden, Rich Warren, and Santa Wiryaman describe the PSE product, which is layered on the UNIX operating system and includes tools for developing parallel applications on clusters of up to 256 machines. They also examine design decisions relative to message-passing support in distributed systems and shared-memory systems; PSE supports network message passing, using TCP/IP or UDP/IP protocols, and shared memory.

Michael Stonebraker's paper opens the section featuring Sequoia 2000 research and is an overview of the project's objectives and status. The objectives encompassed support for high-performance I/O on terabyte data sets, placing all data in a DBMS, and providing new visualization tools and high-speed networking. After a discussion of the architectural layers, he reviews some lessons learned by participants—chief of which was to view the system as an end-to-end solution—and concludes with a look at future work.

An efficient means for locating and retrieving data from the vast stores in the Sequoia DBMS was the task addressed by the Sequoia 2000 Electronic Repository project team. Ray Larson, Chris Plaunt, Allison Woodruff, and Marti Hearst describe the Lassen text indexing and retrieval methods developed for the POSTGRES database system, the GIPSY system for automatic indexing of texts using geographic coordinates discussed in the text, and the TextTiling method for automatic partitioning of text documents to enhance retrieval.

The need for tools to browse through and to visualize Sequoia 2000 data was the impetus behind Tecate, a software platform on which browsing and visualization applications can be built. Peter Kochevar and Len Wanger present the features and functions of this research prototype and offer details of the object model and the role of the interpretive Abstract Visualization Language (AVL) for programming. They conclude with example applications that browse data spaces.

The challenge of high-speed networking for Sequoia 2000 is the subject of the paper by Joseph Pasquale, Eric Anderson, Kevin Fall, and Jon Kay. In designing a distributed system that efficiently retrieves, stores, and transfers very large objects (in excess of tens or hundreds of megabytes), they focused on operating system I/O and network software. They describe two I/O system software solutions—container shipping and peer-to-peer I/O—that avoid data copying. Their TCP/IP network software solutions center on avoiding or reducing checksum computation.

The editors thank Jean Bonney, Digital's Director of External Research, for her help in obtaining the papers on Sequoia 2000 research and for writing the Foreword to this issue.

Our next issue will feature papers on multimedia and UNIX clusters.

Jane C. Blake
Managing Editor

Foreword

Jean C. Bonney
Director, External Research

The Information Utility, the Information Highway, the Internet, the Infobahn, the Information Economy—the sound bytes of the 1990s. To make these concepts reality, a robust technology infrastructure is necessary. In 1990, Digital's research organization saw this need and set out to develop an experimental test bed that would examine assumptions and provide a basis for a technology edge in the '90s. The resulting project was Sequoia 2000, a three-year research collaboration between Digital, campuses of the University of California, and several other industry and government organizations. The Sequoia 2000 vision is

Petabytes (i.e., trillions of bytes) of data in a distributed archive, transparently managed, and logically viewed over a high-speed network with isochronous capabilities via a host of tools

—in other words, a big, fast, easy-to-use system.

Although the vision is still not reality today, our more than three years of participation in Sequoia 2000 research gave us the knowledge base we sought.

After a rigorous process of proposal development and review by experts at Digital and the University of California, Sequoia 2000 began in June 1991. The focus of the research was a high-speed, broadband network spanning University of California campuses from Berkeley to Santa Barbara, Los Angeles, and San Diego; a massive database; storage; a visualization system; and electronic collaboration. Driving the research requirements were earth scientists. The computing needs of these scientists push the state of the art. Current computing technologies lack the capabilities earth scientists need to assimilate and interpret the vast quantities of information collected from satellites. Once the data are collected and organized, there is the challenge of massive simulations, simulations that forecast world climate ten or even one hundred years from now. These were exactly the kinds of challenges the computer scientists needed.

Among the major results of three years of work on Sequoia 2000 was a set of product requirements for large data applications. These requirements have been validated through discussions with customers in financial, healthcare, and communications industries and in government. The requirements include

- A computing environment built on an object relational database, i.e., a data-centric computing system
- A database that handles a wide variety of nontraditional objects such as text, audio, video, graphics, and images
- Support for a variety of traditional databases and file systems
- The ability to perform necessary operations from computing environments that are intuitive and have the same look and feel; the interface to the environment should be generic, very high level, and easily tailored to the user application
- High-speed data migration between secondary and tertiary storage with the ability to handle very large data transfers
- Network bandwidth capable of handling image transmission across networks in an acceptable time frame with quality guarantees for the data
- High-quality remote visualization of any relevant data regardless of format; the user must be able to manipulate the visual data interactively
- Reliable, guaranteed, delivery of data from tertiary storage to the desktop

Sequoia 2000 was also a catalyst for maturing the POSTGRES research database software to the point where it was ready for commercialization. The commercial version, Illustra, is available on Alpha platforms and is enjoying success in the banking industry and in geographic information system (GIS) applications, as well as in other government applications with massive data requirements. Illustra is also making inroads into the Internet where it is used by on-line services.

Yet another major result of Sequoia 2000 was a grant from the National

Aeronautics and Space Administration (NASA) to develop an alternate architecture for the Earth Observing System Data and Information System (EOSDIS). EOSDIS will process the petabytes of real-time data from the Earth Observing System (EOS) satellites to be launched at the end of the decade. The alternate information architecture proposed by the University of California faculty was the Sequoia 2000 architecture. It will have a major influence on the EOSDIS project.

For the earth scientists, gains were made in simulation speeds and in access to large stores of organized data. These scientists used some of Digital's first Alpha workstation farms and software prototypes for their climate simulations. An eight-processor Alpha workstation farm provided a two-to-one price/performance advantage over the powerful, multimillion-dollar CRAY C90 machine. In another earth science application, scientists using Alpha and hierarchical storage systems could simulate two years' worth of climate data over the weekend without operator intervention; formerly, two months' worth of data took one day to simulate and required considerable operator intervention. Thus many more simulations could be processed in a fixed time and "time to discovery" was decreased considerably.

Now that we can look at Sequoia 2000 in retrospect, would we do such a project again? The answer is a resounding "yes" from all of us involved. It was a complex project that included 12 University of California faculty members, 25 graduate students, and 20 staff. Another 8 faculty members and students provided additional expertise. Four of Digital's engineers worked on site, and a variety of support personnel from other industry sponsors participated, including SAIC, the California Department of Water Resources, Hewlett-Packard, Metrum, United States Geological Survey (USGS), Hughes Application Information Services, and the Army Corps of Engineers.

But as is the case with such ambitious projects, there were unanticipated and difficult lessons for all to learn. To experiment with real-life test beds means considerably more than writing a rigorous set of hypotheses in a proposal. Michael Stonebraker, in his paper, notes a number of challenges we faced and the lessons learned. One of the issues that kept surfacing was the "grease and glue" for the infrastructure, that is, the interoperability of pieces of software and hardware that composed the end-to-end system. This remains a challenge that needs research if we are going to achieve the promised goals of internetworking. Another sticky point was scalability. On the one hand, it is difficult to build a very large networked system from scratch. On the other hand, as we slowly built the mass storage system to the point of minimal critical mass, we found that the current off-the-shelf technologies for mass storage were not ready to be put use for our purposes. So, yes, we believe the project was worthwhile with some caveats. We gained critical knowledge about the technology, and we also came a long way in learning the art of directing and leading the type of project that is necessary to assist the Information Technology industry in its quest for the ubiquitous distributed information system.

How else are we going to get insight into the critical issues of building and reliably operating a robust information infrastructure without building a large test bed with real end users whose needs push the state of the art at each point along the way? We believe that large projects similar to Sequoia are crucial. The papers that follow attest to the important knowledge gained. We have focused specifically on the end-to-end system —from the scientists' desktops to the mass storage system, the challenge of building and using a large data repository, the timely and fast movement of very large objects over the network, and browsing and visualizing data from networked sources.

Compiling High Performance Fortran for Distributed-memory Systems

Jonathan Harris
John A. Bircsak
M. Regina Bolduc
Jill Ann Diewald
Israel Gale
Neil W. Johnson
Shin Lee
C. Alexander Nelson
Carl D. Offner

Digital's DEC Fortran 90 compiler implements most of High Performance Fortran version 1.1, a language for writing parallel programs. The compiler generates code for distributed-memory machines consisting of interconnected workstations or servers powered by Digital's Alpha microprocessors. The DEC Fortran 90 compiler efficiently implements the features of Fortran 90 and HPF that support parallelism. HPF programs compiled with Digital's compiler yield performance that scales linearly or even superlinearly on significant applications on both distributed-memory and shared-memory architectures.

High Performance Fortran (HPF) is a new programming language for writing parallel programs. It is based on the Fortran 90 language, with extensions that enable the programmer to specify how array operations can be divided among multiple processors for increased performance. In HPF, the program specifies only the pattern in which the data is divided among the processors; the compiler automates the low-level details of synchronization and communication of data between processors.

Digital's DEC Fortran 90 compiler is the first implementation of the full HPF version 1.1 language (except for transcriptive argument passing, dynamic remapping, and nested FORALL and WHERE constructs). The compiler was designed for a distributed-memory machine made up of a cluster (or farm) of workstations and/or servers powered by Digital's Alpha microprocessors.

In a distributed-memory machine, communication between processors must be kept to an absolute minimum, because communication across the network is enormously more time-consuming than any operation done locally. Digital's DEC Fortran 90 compiler includes a number of optimizations to minimize the cost of communication between processors.

This paper briefly reviews the features of Fortran 90 and HPF that support parallelism, describes how the compiler implements these features efficiently, and concludes with some recent performance results showing that HPF programs compiled with Digital's compiler yield performance that scales linearly or even superlinearly on significant applications on both distributed-memory and shared-memory architectures.

Historical Background

The desire to write parallel programs dates back to the 1950s, at least, and probably earlier. The mathematician John von Neumann, credited with the invention of the basic architecture of today's serial computers, also invented cellular automata, the precursor of today's massively parallel machines. The continuing motivation for parallelism is provided by the need to solve computationally intense problems in a reasonable time and at an affordable price. Today's parallel machines,

which range from collections of workstations connected by standard fiber-optic networks to tightly coupled CPUs with custom high-speed interconnection networks, are cheaper than single-processor systems with equivalent performance. In many cases, equivalent single-processor systems do not exist and could not be constructed with existing technology.

Historically, one of the difficulties with parallel machines has been writing parallel programs. The work of parallelizing a program was far from the original science being explored; it required programmers to keep track of a great deal of information unrelated to the actual computations; and it was done using ad hoc methods that were not portable to other machines.

The experience gained from this work, however, led to a consensus on a better way to write portable Fortran programs that would perform well on a variety of parallel machines. The High Performance Fortran Forum, an international consortium of more than 100 commercial parallel machine users, academics, and computer vendors, captured and refined these ideas, producing the language now known as High Performance Fortran.[1-3] HPF programming systems are now being developed by most vendors of parallel machines and software. HPF is included as part of the DEC Fortran 90 language.[4]

One obvious and reasonable question is: Why invent a new language rather than have compilers automatically generate parallel code? The answer is straightforward: it is generally conceded that automatic parallelization technology is not yet sufficiently advanced. Although parallelization for particular architectures (e.g., vector machines and shared-memory multiprocessors) has been successful, it is not fully automatic but requires substantial assistance from the programmer to obtain good performance. That assistance usually comes in the form of hints to the compiler and rewritten sections of code that are more parallelizable. These hints, and in some cases the rewritten code, are not usually portable to other architectures or compilers. Agreement was widespread at the HPF Forum that a set of hints could be standardized and done in a portable way. Automatic parallelization technology is an active field of research; consequently, it is expected that compilers will become increasingly adept.[5-12] Thus, these hints are cast as comments—called *compiler directives*—in the source code. HPF actually contains very little new language beyond this; it consists primarily of these compiler directives.

The HPF language was shaped by certain key considerations in parallel programming:

- The need to identify computations that can be done in parallel
- The need to minimize communication between processors on machines with nonuniform memory access costs
- The need to keep processors as busy as possible by balancing the computation load across processors

It is not always obvious which computations in a Fortran program are parallelizable. Although some DO loops express parallelizable computations, other DO loops express computations in which later iterations of the loop require the results of earlier iterations. This forces the computation to be done in order (serially), rather than simultaneously (in parallel). Also, whether or not a computation is parallelizable sometimes depends on user data that may vary from run to run of the program. Accordingly, HPF contains a new statement (FORALL) for describing parallel computations, and a new directive (INDEPENDENT) to identify additional parallel computations to the compiler. These features are equally useful for distributed- or shared-memory machines.

HPF's data distribution directives are particularly important for distributed-memory machines. The HPF directives were designed primarily to increase performance on "computers with nonuniform memory access costs."[1] Of all parallel architectures, distributed memory is the architecture in which the location of data has the greatest effect on access cost. On distributed-memory machines, interprocessor communication is very expensive compared to the cost of fetching local data, typically by several orders of magnitude. Thus the effect of suboptimal distribution of data across processors can be catastrophic. HPF directives tell the compiler how to distribute data across processors; based on knowledge of the algorithm, programmers choose directives that will minimize communication time. These directives can also help achieve good load balance: by spreading data appropriately across processors, the computations on those data will also be spread across processors.

Finally, a number of idioms that are important in parallel programming either are awkward to express in Fortran or are greatly dependent on machine architecture for their efficient implementation. To be useful in a portable language, these idioms must be easy to express and implement efficiently. HPF has captured some of these idioms as library routines for efficient implementation on very different architectures.

For example, consider the Fortran 77 program in Figure 1, which repeatedly replaces each element of a two-dimensional array with the average of its north, south, east, and west neighbors. This kind of computation arises in a number of programs, including iterative solvers for partial differential equations and image-filtering applications. Figure 2 shows how this code can be expressed in HPF.

On a machine with four processors, a single HPF directive causes the array *A* to be distributed across the processors as shown in Figure 3. The program

```
      integer n, number_of_iterations, i,j,k
      parameter(n=16)
      real A(n,n), Temp(n,n)
     ... (Initialize A, number_of_iterations) ...
      do k=1, number_of_iterations
C
        Update non-edge elements only
        do i=2, n-1
          do j=2, n-1
             Temp(i, j)=(A(i, j-1)+A(i, j+1)+A(i+1, j)+A(i-1, j))*0.25
          enddo
        enddo
        do i=2, n-1
          do j=2, n-1
            A(i, j)=Temp(i,j)
          enddo
        enddo
      enddo
```

Figure 1
A Computation Expressed in Fortran 77

```
          integer n, number_of_iterations, i, j, k
          parameter (n=16)
          real A(n, n)
!hpf$     distribute A(block, block)
         ...(Initialize A, number_of_iterations) ...
          do k=1, number_of_iterations
            forall (i=2:n-1, j=2:n-1) !Update non-edge elements only
               A(i, j)=(A(i, j-1)+A(i, j+1)+A(i+1, j)+A(i-1, j))*0.25
            endforall
          enddo
```

Figure 2
The Same Computation Expressed in HPF

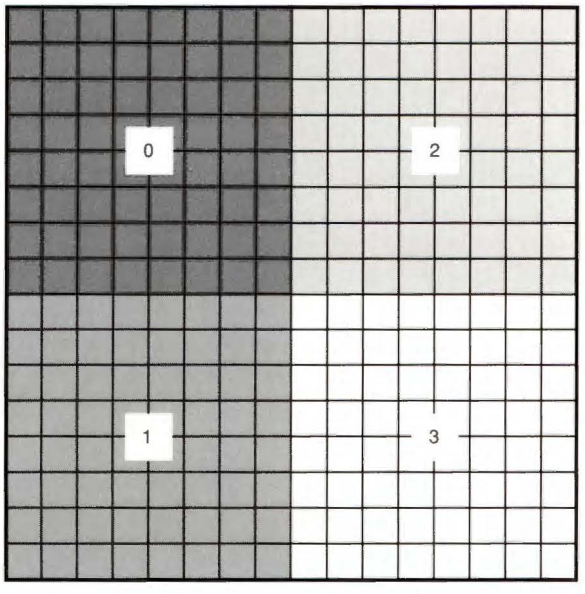


Figure 3
An Array Distributed over Four Processors

executes in parallel on the four processors, with each processor performing the updates to the array elements it owns. This update, however, requires interprocessor communication (or "data motion"). To compute a new value for $A(8, 2)$, which lives on processor 0, the value of $A(9, 2)$, which lives on processor 1, is needed. In fact, processor 0 requires the seven values $A(9, 2)$, $A(9, 3)$, ... $A(9, 8)$ from processor 1, and the seven values $A(2, 9)$, $A(3, 9)$, ... $A(8, 9)$ from processor 2.[13] Each processor, then, needs seven values apiece from two neighbors. By knowing the layout of the data and the computation being performed, the compiler can automatically generate the interprocessor communication instructions needed to execute the code.

Even for seemingly simple cases, the communication instructions can be complex. Figure 4 shows the communication instructions that are generated for the code that implements the FORALL statement for a distributed-memory parallel processor.

Processor 0	Processor 1	Processor 2	Processor 3
SEND $A(8, 2)...A(8, 8)$ to Processor 1	SEND $A(9, 2)...A(9, 8)$ to Processor 0	SEND $A(2, 9)...A(8, 9)$ to Processor 0	SEND $A(9, 9)...A(15, 9)$ to Processor 1
SEND $A(2, 8)...A(8, 8)$ to Processor 2	SEND $A(9, 8)...A(15, 8)$ to Processor 3	SEND $A(8, 9)...A(8, 15)$ to Processor 3	SEND $A(9, 9)...A(9, 9)$ to Processor 2
RECEIVE $A(9, 2)...A(9, 8)$ from Processor 1	RECEIVE $A(8, 2)...A(8, 8)$ from Processor 0	RECEIVE $A(2, 8)...A(8, 8)$ from Processor 0	RECEIVE $A(9, 8)...A(15, 8)$ from Processor 1
RECEIVE $A(2, 9)...A(8, 9)$ from Processor 2	RECEIVE $A(9, 9)...A(15, 9)$ from Processor 3	RECEIVE $A(9, 9)...A(9, 15)$ from Processor 3	RECEIVE $A(8, 9)...A(8, 15)$ from Processor 2

Figure 4
Compiler-generated Communication for a FORALL Statement

Although the communication needed in this simple example is not difficult to figure out by hand, keeping track of the communication needed for higher-dimensional arrays, distributed onto more processors, with more complicated computations, can be a very difficult, bug-prone task. In addition, a number of the optimizations that can be performed would be extremely tedious to figure out by hand. Nevertheless, distributed-memory parallel processors are programmed almost exclusively today by writing programs that contain explicit hand-generated calls to the SEND and RECEIVE communication routines. The difference between this kind of programming and programming in HPF is comparable to the difference between assembly language programming and high-level language programming.

This paper continues with an overview of the HPF language, a discussion of the machine architecture targeted by the compiler, the architecture of the compiler itself, and a discussion of some optimizations performed by its components. It concludes with recent performance results, showing that HPF programs compiled with Digital's compiler scale linearly in significant cases.

Overview of the High Performance Fortran Language

High Performance Fortran consists of a small set of extensions to Fortran 90. It is a data-parallel programming language, meaning that parallelism is made possible by the explicit distribution of large arrays of data across processors, as opposed to a control-parallel language, in which threads of computation are distributed. Like the standard Fortran 77, Fortran 90, and C models, the HPF programming model contains a single thread of control; the language itself has no notion of process or thread.

Conceptually, the program executes on all the processors simultaneously. Since each processor contains only a subset of the distributed data, occasionally a processor may need to access data stored in the memory of another processor. The compiler determines the actual details of the interprocessor communication needed to support this access; that is, rather than being specified explicitly, the details are implicit in the program.

The compiler translates HPF programs into low-level code that contains explicit calls to SEND and RECEIVE message-passing routines. All addresses in this translated code are modified so that they refer to data local to a processor. As part of this translation, addressing expressions and loop bounds become expressions involving the processor number on which the code is executing. Thus, the compiler needs to generate only one program: the generated code is parametrized by the processor number and so can be executed on all processors with appropriate results on each processor. This generated code is called explicit single-program multiple-data code, or explicit-SPMD code.

In some cases, the programmer may find it useful to write explicit-SPMD code at the source code level. To accommodate this, the HPF language includes an escape hatch called EXTRINSIC procedures that is used to leave data-parallel mode and enter explicit-SPMD mode.

We now describe some of the HPF language extensions used to manage parallel data.

Distributing Data over Processors

Data is distributed over processors by the DISTRIBUTE directive, the ALIGN directive, or the default distribution.

The DISTRIBUTE Directive For parallel execution of array operations, each array must be divided in memory, with each processor storing some portion of the array in its own local memory. Dividing the array into parts is known as distributing the array. The HPF DISTRIBUTE directive controls the distribution of arrays across each processor's local memory. It does this by specifying a mapping pattern of data objects onto processors. Many mappings are possible; we illustrate only a few.

Consider first the case of a 16 × 16 array *A* in an environment with four processors. One possible specification for *A* is

```
        real A(16, 16)
!hpf$   distribute A(*, block)
```

The asterisk (*) for the first dimension of *A* means that the array elements are not distributed along the first (vertical) axis. In other words, the elements in any given column are not divided among different processors, but are assigned as a single block to one processor. This type of mapping is referred to as serial distribution. Figure 5 illustrates this distribution.

The BLOCK keyword for the second dimension means that for any given row, the array elements are distributed over each processor in large blocks. The blocks are of approximately equal size—in this case, they are exactly equal—with each processor holding one block. As a result, A is broken into four contiguous groups of columns, with each group assigned to a separate processor.

Another possibility is a (*, CYCLIC) distribution. As in (*, BLOCK), all the elements in each column are assigned to one processor. The elements in any given row, however, are dealt out to the processors in round-robin order, like playing cards dealt out to players around a table. When elements are distributed over *n* processors, each processor contains every *n*th column, starting from a different offset. Figure 6 shows the same array and processor arrangement, distributed CYCLIC instead of BLOCK.

As these examples indicate, the distributions of the separate dimensions are independent.

A (BLOCK, BLOCK) distribution, as in Figure 3, divides the array into large rectangles. In that figure, the array elements in any given column or any given row are divided into two large blocks: Processor 0 gets *A*(1:8, 1:8), processor 1 gets *A*(9:16, 1:8), processor 2 gets *A*(1:8, 9:16), and processor 3 gets *A*(9:16,9:16).

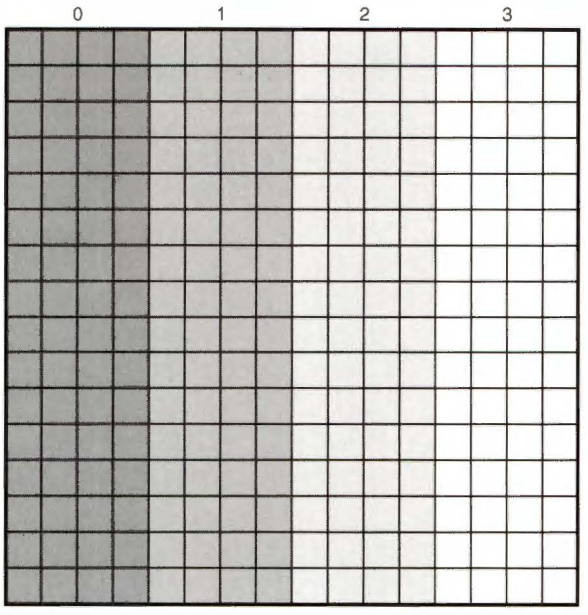


Figure 5
A (*, BLOCK) Distribution

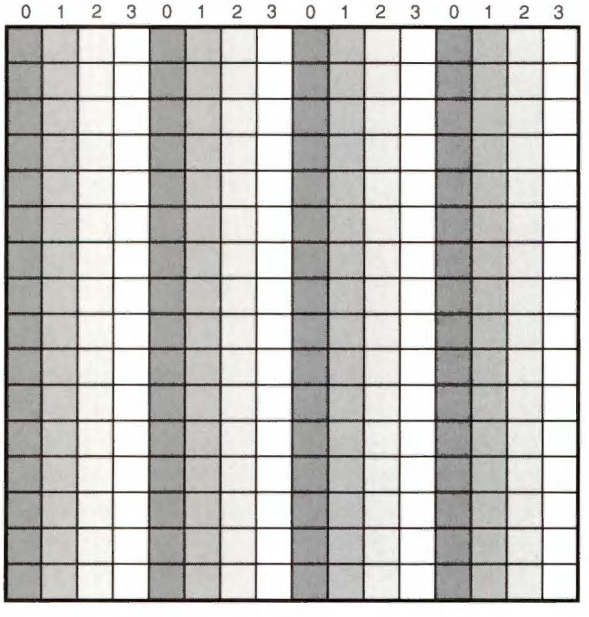


Figure 6
A (*, CYCLIC) Distribution

The ALIGN Directive The ALIGN directive is used to specify the mapping of arrays relative to one another. Corresponding elements in aligned arrays are always mapped to the same processor; array operations between aligned arrays are in most cases more efficient than array operations between arrays that are not known to be aligned.

The most common use of ALIGN is to specify that the corresponding elements of two or more arrays be mapped identically, as in the following example:

```
!hpf$   align A(i) with B(i)
```

This example specifies that the two arrays *A* and *B* are always mapped the same way. More complex alignments can also be specified. For example:

```
!hpf$   align E(i) with F(2*i-1)
```

In this example, the elements of *E* are aligned with the odd elements of *F*. In this case, *E* can have at most half as many elements as *F*.

An array can be aligned with the interior of a larger array:

```
        real A(12, 12)
        real B(16, 16)
!hpf$   align A(i, j) with B(i+2, j+2)
```

In this example, the 12 × 12 array *A* is aligned with the interior of the 16 × 16 array *B* (see Figure 7). Each interior element of *B* is always stored on the same processor as the corresponding element of *A*.

The Default Distribution Variables that are not explicitly distributed or aligned are given a default distribution by the compiler. The default distribution is not specified by the language: different compilers can choose different default distributions, usually based on constraints of the target architecture. In the DEC Fortran 90 language, an array or scalar with the default distribution is completely replicated. This decision was made because the large arrays in the program are the significant ones that the programmer has to distribute explicitly to get good performance. Any other arrays or scalars will be small and generally will benefit from being replicated since their values will then be available everywhere. Of course, the programmer retains complete control and can specify a different distribution for these arrays.

Replicated data is cheap to read but generally expensive to write. Programmers typically use replicated data for information that is computed infrequently but used often.

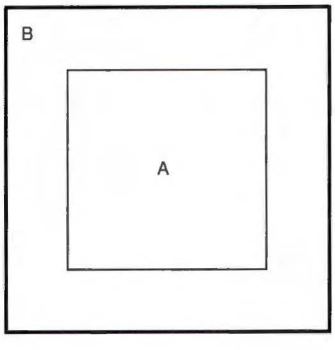


Figure 7
An Example of Array Alignment

Data Mapping and Procedure Calls

The distribution of arrays across processors introduces a new complication for procedure calls: the interface between the procedure and the calling program must take into account not only the type and size of the relevant objects but also their mapping across processors. The HPF language includes special forms of the ALIGN and DISTRIBUTE directives for procedure interfaces. These allow the program to specify whether array arguments can be handled by the procedure as they are currently distributed, or whether (and how) they need to be redistributed across the processors.

Expressing Parallel Computations

Parallel computations in HPF can be identified in four ways:

- Fortran 90 array assignments
- FORALL statements
- The INDEPENDENT directive, applied to DO loops and FORALL statements
- Fortran 90 and HPF intrinsics and library functions

In addition, a compiler may be able to discover parallelism in other constructs. In this section, we discuss the first two of these parallel constructions.

Fortran 90 Array Assignment In Fortran 77, operations on whole arrays can be accomplished only through explicit DO loops that access array elements one at a time. Fortran 90 array assignment statements allow operations on entire arrays to be expressed more simply.

In Fortran 90, the usual intrinsic operations for scalars (arithmetic, comparison, and logical) can be applied to arrays, provided the arrays are of the same shape. For example, if *A, B,* and *C* are two-dimensional arrays of the same shape, the statement $C = A + B$ assigns to each element of *C* a value equal to the sum of the corresponding elements of *A* and *B*.

In more complex cases, this assignment syntax can have the effect of drastically simplifying the code. For instance, consider the case of three-dimensional arrays, such as the arrays dimensioned in the following declaration:

```
real D(10, 5:24, -5:M), E(0:9, 20, M+6)
```

In Fortran 77 syntax, an assignment to every element of D requires triple-nested loops such as the example shown in Figure 8.

In Fortran 90, this code can be expressed in a single line:

```
D = 2.5*D+E+2.0
```

The FORALL Statement The FORALL statement is an HPF extension to the American National Standards Institute (ANSI) Fortran 90 standard but has been included in the draft Fortran 95 standard.

```
do i = 1, 10
  do j = 5, 24
    do k = -5, M
      D(i, j, k) = 2.5*D(i, j, k) + E(i-1, j-4, k+6) + 2.0
    end do
  end do
end do
```

Figure 8
An Example of a Triple-nested Loop

FORALL is a generalized form of Fortran 90 array assignment syntax that allows a wider variety of array assignments to be expressed. For example, the diagonal of an array cannot be represented as a single Fortran 90 array section. Therefore, the assignment of a value to every element of the diagonal cannot be expressed in a single array assignment statement. It can be expressed in a FORALL statement:

```
real, dimension(n, n) :: A
forall (i = 1:n) A(i, i) = 1
```

Although FORALL structures serve the same purpose as some DO loops do in Fortran 77, a FORALL structure is a parallel assignment statement, not a loop, and in many cases produces a different result from an analogous DO loop. For example, the FORALL statement

```
forall (i = 2:5) C(i, i) = C(i-1, i-1)
```

applied to the matrix

$$C = \begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 22 & 0 & 0 & 0 \\ 0 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 44 & 0 \\ 0 & 0 & 0 & 0 & 55 \end{bmatrix}$$

produces the following result:

$$C = \begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 22 & 0 & 0 \\ 0 & 0 & 0 & 33 & 0 \\ 0 & 0 & 0 & 0 & 44 \end{bmatrix}$$

On the other hand, the apparently similar DO loop

```
do i = 2, 5
  C(i, i) = C(i-1, i-1)
end do
```

produces

$$C = \begin{bmatrix} 11 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 11 \end{bmatrix}$$

This happens because the DO loop iterations are performed sequentially, so that each successive element of the diagonal is updated before it is used in the next iteration. In contrast, in the FORALL statement, all the diagonal elements are fetched and used before any stores happen.

The Target Machine

Digital's DEC Fortran 90 compiler generates code for clusters of Alpha processors running the Digital UNIX operating system. These clusters can be separate Alpha workstations or servers connected by a fiber distributed data interface (FDDI) or other network devices. (Digital's high-speed GIGAswitch/FDDI system is particularly appropriate.[14]) A shared-memory, symmetric multiprocessing (SMP) system like the AlphaServer 8400 system can also be used. In the case of an SMP system, the message-passing library uses shared memory as the message-passing medium; the generated code is otherwise identical. The same executable can run on a distributed-memory cluster or an SMP shared-memory cluster without recompiling. DEC Fortran 90 programs use the execution environment provided by Digital's Parallel Software Environment (PSE), a companion product.[3,15] PSE is responsible for invoking the program on multiple processors and for performing the message passing requested by the generated code.

The Architecture of the Compiler

Figure 9 illustrates the high-level architecture of the compiler. The curved path is the path taken when compiler command-line switches are set for compiling programs that will not execute in parallel, or when the scoping unit being compiled is declared as EXTRINSIC(HPF_LOCAL).

Figure 9 shows the front end, transform, middle end, and GEM back end components of the compiler. These components function in the following ways:

- The front end parses the input code and produces an internal representation containing an abstract syntax tree and a symbol table. It performs extensive semantic checking.[16]

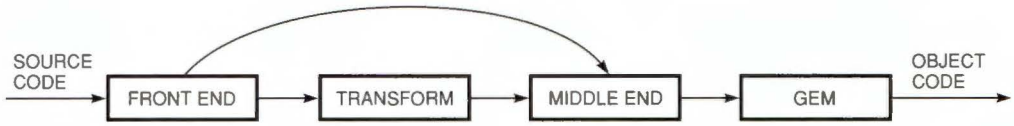


Figure 9
Compiler Components

- The transform component performs the transformation from global-HPF to explicit-SPMD form. To do this, it localizes the addressing of data, inserts communication where necessary, and distributes parallel computations over processors.
- The middle end translates the internal representation into another form of internal representation suitable for GEM.
- The GEM back end, also used by other Digital compilers, performs local and global optimization, storage allocation, code generation, register allocation, and emits binary object code.[17]

In this paper, we are mainly concerned with the transform component of the compiler.

An Overview of Transform

Figure 10 shows the transform phases discussed in this paper. These phases perform the following key tasks:

- LOWER. Transforms array assignments so that they look internally like FORALL statements.
- DATA. Fills in the data space information for each symbol using information from HPF directives where available. This determines where each data object lives, i.e., how it is distributed over the processors.
- ITER. Fills in the iteration space information for each computational expression node. This determines where each computation takes place and indicates where communication is necessary.
- ARG. Pulls functions in the interior of expressions up to the statement level. It also compares the mapping of actual arguments to that of their corresponding dummies and generates remapping into compiler-generated temporaries if necessary.

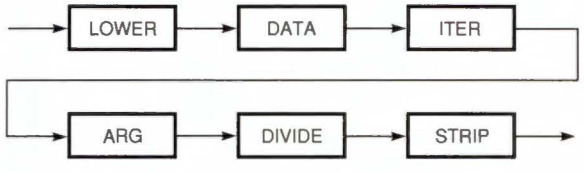


Figure 10
The Transform Phases

- DIVIDE. Pulls all communication inside expressions (identified by ITER) up to the statement level and identifies what kind of communication is needed. It also ensures that information needed for flow of control is available at each processor.
- STRIP. Turns global-HPF code into explicit-SPMD code by localizing the addressing of all data objects and inserting explicit SEND and RECEIVE calls to make communication explicit. In the process, it performs strip mining and loop optimizations, vectorizes communication, and optimizes nearest-neighbor computations.

Transform uses the following main data structures:

- Symbol table. This is the symbol table created by the front end. It is extended by the transform phase to include dope information for array and scalar symbols.
- Dotree. Transform uses the dotree form of the abstract syntax tree as an internal representation of the program.
- Dependence graph. This is a graph whose nodes are expression nodes in the dotree and whose edges represent dependence edges.
- Data spaces. A data space is associated with each data symbol (i.e., each array and each scalar). The data space information describes how each data object is distributed over the processors. This information is derived from HPF directives.
- Iteration spaces. An iteration space is associated with each computational node in the dotree. The iteration space information describes how computations are distributed over the processors. This information is not specified in the source code but is produced by the compiler.

The interrelationship of these data structures is discussed in Reference 18. The data and iteration spaces are central to the processing performed by transform.

The Transform Phases

LOWER

Since the FORALL statement is a generalization of a Fortran 90 array assignment and includes it as a special case, it is convenient for the compiler to have a uniform representation for these two constructions. The

LOWER phase implements this by turning each Fortran 90 array assignment into an equivalent FORALL statement (actually, into the dotree representation of one). This uniform representation means that the compiler has far fewer special cases to consider than otherwise might be necessary and leads to no degradation of the generated code.

DATA

The DATA phase specifies where data lives. Placing and addressing data correctly is one of the major tasks of transform. There are a large number of possibilities:

When a value is available on every processor, it is said to be *replicated.* When it is available on more than one but not all processors, it is said to be *partially replicated.* For instance, a scalar may live on only one processor, or on more than one processor. Typically, a scalar is replicated—it lives on all processors. The replication of scalar data makes fetches cheap because each processor has a copy of the requested value. Stores to replicated scalar data can be expensive, however, if the value to be stored has not been replicated. In that case, the value to be stored must be sent to each processor.

The same consideration applies to arrays. Arrays may be replicated, in which case each processor has a copy of an entire array; or arrays may be partially replicated, in which case each element of the array is available on a subset of the processors.

Furthermore, arrays that are not replicated may be distributed across the processors in several different fashions, as explained above. In fact, each dimension of each array may be distributed independently of the other dimensions. The HPF mapping directives, principally ALIGN and DISTRIBUTE, give the programmer the ability to specify completely how each dimension of each array is laid out. DATA uses the information in these directives to construct an internal description or *data space* of the layout of each array.

ITER

The ITER phase determines where the intermediate results of calculations should live. Its relationship to DATA can be expressed as:

- DATA decides where parallel data lives.
- ITER decides where parallel computations happen.

Each array has a fixed number of dimensions and an extent in each of those dimensions; these properties together determine the shape of an array. After DATA has finished processing, the shape and mapping of each array is known. Similarly, the result of a computation has a particular shape and mapping. This shape may be different from that of the data used in the computation. As a simple example, the computation

```
A(:,:,3) + B(:,:,3)
```

has a two-dimensional shape, even though both arrays A and B have three-dimensional shapes. The data space data structure is used to describe the shape of each array and its layout in memory and across processors; similarly, *iteration space* is used to describe the shape of each computation and its layout across processors. One of the main tasks of transform is to construct the iteration space for each computation so that it leads to as little interprocessor communication as possible: this construction happens in ITER. The compiler's view of this construction and the interaction of these spaces are explained in Reference 18.

Shapes can change within an expression: while some operators return a result having the shape of their operands (e.g., adding two arrays of the same shape returns an array of the same shape), other operators can return a result having a different shape than the shape of their operands. For example, reductions like SUM return a result having a shape with lower rank than that of the input expression being reduced.

One well-known method of determining where computations happen is the "owner-computes" rule. With this method, all the values needed to construct the computation on the right-hand side of an assignment statement are fetched (using interprocessor communication if necessary) and computed on the processor that contains the left-hand-side location. Then they are stored to that left-hand-side location (on the same processor on which they were computed). Thus a description of where computations occur is derived from the output of DATA. There are, however, simple examples where this method leads to less than optimal performance. For instance, in the code

```
        real A(n, n), B(n, n), C(n, n)
!hpf$   distribute A(block, block)
!hpf$   distribute B(cyclic, cyclic)
!hpf$   distribute C(cyclic, cyclic)

        forall (i=1:n, j=1:n)
          A(i, j) = B(i, j) + C(i, j)
        end forall
```

the owner-computes rule would move B and C to align with A, and then add the moved values of B and C and assign to A. It is certainly more efficient, however, to add B and C together where they are aligned with each other and then communicate the result to where it needs to be stored to A. With this procedure, we need to communicate only one set of values rather than two. The compiler identifies cases such as these and generates the computation, as indicated here, to minimize the communication.

ARG

The ARG phase performs any necessary remapping of actual arguments at subroutine call sites. It does this by comparing the mapping of the actuals (as determined by ITER) to the mapping of the corresponding dummies (as determined by DATA).

In our implementation, the caller performs all remapping. If remapping is necessary, ARG exposes that remapping by inserting an assignment statement that remaps the actual to a temporary that is mapped the way the dummy is mapped. This guarantees that references to a dummy will access the correct data as specified by the programmer. Of course, if the parameter is an OUT argument, a similar copy-out remapping has to be inserted after the subroutine call.

DIVIDE

The DIVIDE phase partitions ("divides") each expression in the dotree into regions. Each region contains computations that can happen without interprocessor communication. When region R uses the values of a subexpression computed in region S, for example, interprocessor communication is required to remap the computed values from their locations in S to their desired locations in R. DIVIDE makes a temporary mapped the way region R needs it and makes an explicit assignment statement whose left-hand side is that temporary and whose right-hand side is the subexpression computed in region S. In this way, DIVIDE makes explicit the interprocessor communication that is implicit in the iteration space information attached to each expression node.

DIVIDE also performs other processing:

- DIVIDE replicates expressions needed to manage control flow, such as an expression representing a bound of a DO loop or the condition in an IF statement. Consequently, each processor can do the necessary branching.
- For each statement requiring communication, DIVIDE identifies the kind of communication needed.

  Depending on what knowledge the two sides of the communication (i.e., the sender and the receiver) have, we distinguish two kinds of communication:

  - Full knowledge. The sender knows what it is sending and to whom, and the receiver knows what it is receiving and from whom.
  - Partial knowledge. Either the sender knows what it is sending and to whom, or the receiver knows what it is receiving and from whom, but the other party knows nothing.

    This kind of message is typical of code dealing with irregular data accesses, for instance, code with array references containing vector-valued subscripts.

STRIP

The STRIP phase (shortened from "strip miner"; probably a better term would be the "localizer") takes the statements categorized by DIVIDE as needing communication and inserts calls to library routines to move the data from where it is to where it needs to be.

It then localizes parallel assignments coming from vector assignments and FORALL constructs. In other words, each processor has some (possibly zero) number of array locations that must be stored to. A set of loops is generated that calculates the value to be stored and stores it. The bounds for these loops are dependent on the distribution of the array being assigned to and the section of the array being assigned to. These bounds may be explicit numbers known at compile time, or they may be expressions (when the array size is not known at compile time). In any case, they are exposed so that they may be optimized by later phases. They are not calls to run-time routines.

The subscripts of each dimension of each array in the statement are then rewritten in terms of the loop variable. This modification effectively turns the original global subscript into a local subscript. Scalar subscripts are also converted to local subscripts, but in this case the subscript expression does not involve loop indices. Similarly, scalar assignments that reference array elements have their subscripts converted from global addressing to local addressing, based on the original subscript and the distribution of the corresponding dimension of the array. They do not require strip loops. For example, consider the code fragment shown in Figure 11a.

Here k is some variable whose value has been assigned before the FORALL. Let us assume that A and B have been distributed over a 4×5 processor array in such a way that the first dimensions of A and B are distributed CYCLIC over the first dimension of the processor array (which has extent 4), and the second dimensions of A and B are distributed BLOCK over the second dimension of the processor array (which has extent 5). (The programmer can express this through a facility in HPF.) The generated code is shown in Figure 11b.

If the array assigned to on the left-hand side of such a statement is also referenced on the right-hand side, then replacing the parallel FORALL by a DO loop may violate the "fetch before store" semantics of the original statement. That is, an array element may be assigned to on one iteration of the DO loop, and this new value may subsequently be read on a later iteration. In the original meaning of the statement, however, all values read would be the original values.

This problem can always be resolved by evaluating the right-hand side of the statement in its entirety into a temporary array, and then—in a second set of DO loops—assigning that temporary to the left-hand side. We use dependence analysis to determine if such a problem occurs at all. Even if it does, there are cases in which loop transformations can be used to eliminate the need for a temporary, as outlined in Reference 19.

```
        real A(100, 20), B(100, 20)
!hpf$   distribute A(cyclic, block), B(cyclic, block)
        ...
        forall (i = 2:99)
          A(i, k) = B(i, k)
        end forall
```

(a) Code Fragment

```
m = my_processor()
...
if k mod 5 = Lm/4J then
   do i = (if m mod 4 = 0 then 2 else 1), (if m mod 4 = 3 then 24 else 25)
     A(i, Lk/5J) = B(i, Lk/5J)
   end do
end if
```

(b) Pseudocode Generated for Code Fragment

Figure 11
Code Fragment and Pseudocode Generated for Code Fragment

(Some poor implementations always introduce the temporary even when it is not needed.)

Unlike other HPF implementations, ours uses compiler-generated inlined expressions instead of function calls to determine local addressing values. Furthermore, our implementation does not introduce barrier synchronization, since the sends and receives generated by the transform phase will enforce any necessary synchronization. In general, this is much less expensive than a naive insertion of barriers. The reason this works can be seen as follows: first, any value needed by a processor is computed either locally or nonlocally. If the value is computed locally, the normal control flow guarantees correct access order for that value. If the value is computed nonlocally, the generated receive on the processor that needs the value causes the receiving processor to wait until the value arrives from the sending processor. The sending processor will not send the value until it has computed it, again because of normal control-flow. If the sending processor is ready to send data before the receiving processor is ready for it, the sending processor can continue without waiting for the data to be received. Digital's Parallel Software Environment (PSE) buffers the data until it is needed.[15]

Some Optimizations Performed by the Compiler

The GEM back end performs the following optimizations:

- Constant folding
- Optimizations of arithmetic IF, logical IF, and block IF-THEN-ELSE
- Global common subexpression elimination
- Removal of invariant expressions from loops
- Global allocation of general registers across program units
- In-line expansion of statement functions and routines
- Optimization of array addressing in loops
- Value propagation
- Deletion of redundant and unreachable code
- Loop unrolling
- Software pipelining to rearrange instructions between different unrolled loop iterations
- Array temporary elimination

In addition, the transform component performs some important optimizations, mainly devoted to improving interprocessor communication. We have implemented the following optimizations:

Message Vectorization

The compiler generates code to limit the communication to one SEND and one RECEIVE for each array being moved between any two processors. This is the most obvious and basic of all the optimizations that a compiler can perform for distributed-memory architectures and has been widely studied.[20–22]

If the arrays *A* and B are laid out as in Figure 12 and if *B* is to be assigned to *A*, then array elements *B*(4), *B*(5), and *B*(6), all of which live on processor 6, should be sent to processor 1. Clearly, we do not want to generate three distinct messages for this. Therefore, we collect these three elements and generate one message containing all three of them. This example involves full knowledge.

Communications involving partial knowledge are also vectorized, but they are much more expensive because the side of the message without initial knowledge has to be informed of the message. Although there are several ways to do this, all are costly, either in time or in space.

We use the same method, incidentally, to inline the HPF XXX_SCATTER routines. These new routines have been introduced to handle a parallel construct that could cause more than one value to be assigned to the same location. The outcome of such cases is determined by the routine being inlined. For instance, SUM_SCATTER simply adds all the values that arrive at each location and assigns the final result to that location. Although this is an example of interprocessor communication with partial knowledge, we can still build up messages so that only a minimum number of messages are sent.

In some cases, we can improve the handling of communications with partial knowledge, provided they occur more than once in a program. For more information, please see the section Run-time Preprocessing of Irregular Data Accesses.

Strip Mining and Loop Optimizations

Strip mining and loop optimizations have to do with generating efficient code on a per-processor basis, and so in some sense can be thought of as conventional. Generally, we follow the processing detailed in Reference 19 and summarized as:

- Strip mining obstacles are eliminated where possible by loop transformations (loop reversal or loop interchange).
- Temporaries, if introduced, are of minimal size; this is achieved by loop interchange.
- Exterior loop optimization is used to allow reused data to be kept in registers over consecutive iterations of the innermost loop.
- Loop fusion enables more efficient use of conventional optimizations and minimizes loop overhead.

Nearest-neighbor Computations

Nearest-neighbor computations are common in code written to discretize partial differential equations. See the example given in Figure 2.

If we have, for example, 16 processors, with the array *A* distributed in a (BLOCK, BLOCK) fashion over the processors, then conceptually, the array is distributed as in Figure 13, where the arrows indicate communication needed between neighboring processors. In fact, in this case, each processor needs to see values only from a narrow strip (or "shadow edge") in the memory of its neighboring processors, as shown in Figure 14.

The compiler identifies nearest-neighbor computations (the user does not have to tag them), and it alters the addressing of each array involved in these computations (throughout the compilation unit). As a result, each processor can store those array elements that are needed from the neighboring processors. Those array elements are moved in (using message vectorization) at the beginning of the computation, after which the entire computation is local.

Recognizing nearest-neighbor statements helps generate better code in several ways:

- Less run-time overhead. The compiler can easily identify the exact small portion of the array that needs to be moved. The communication for nearest-neighbor assignments is extremely regular: At each step, each processor is sending an entire shadow edge to precisely one of its neighbors. Therefore the communication processing overhead is greatly reduced. That is, we are able to generate

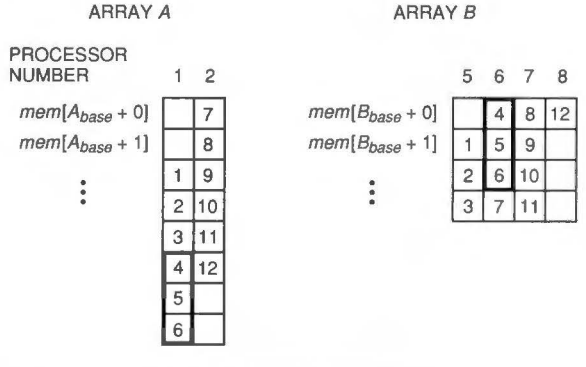


Figure 12
Two Arrays in Memory

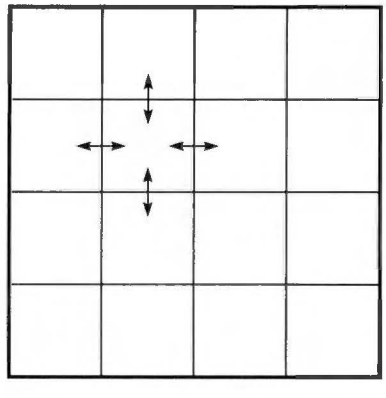


Figure 13
A Nearest-neighbor Communication Pattern

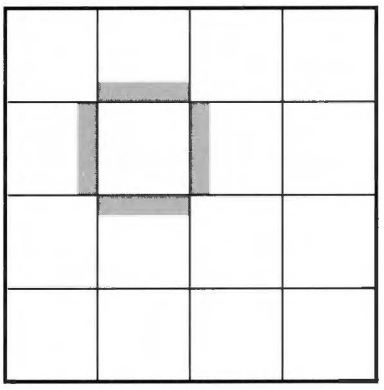


Figure 14
Shadow Edges for a Nearest-neighbor Computation

communication involving even less overhead than general communication involving full knowledge.

- No local copying. If shadow edges were not used, then the following standard processing would take place: For each shifted-array reference on the right-hand side of the assignment, shift the entire array; then identify that part of the shifted array that lives locally on each processor and create a local temporary to hold it. Some of that temporary (the part representing our shadow edge) would be moved in from a neighboring processor, and the rest of the temporary would be copied locally from the original array. Our processing eliminates the need for the local temporary and for the local copy, which is substantial for large arrays.
- Greater locality of reference. When the actual computation is performed, greater locality of reference is achieved because the shadow edges (i.e., the received values) are now part of the array, rather than being a temporary somewhere else in memory.
- Fewer messages. Finally, the optimization also makes it possible for the compiler to see that some messages may be combined into one message, thereby reducing the number of messages that must be sent. For instance, if the right-hand side of the assignment statement in the above example also contained a term $A(i+1, j+1)$, even though overlapping shadow edges and an additional shadow edge would now be in the diagonally adjacent processor, no additional communication would need to be generated.

Reductions

The SUM intrinsic function of Fortran 90 takes an array argument and returns the sum of all its elements. Alternatively, SUM can return an array whose rank is one less than the rank of its argument, and each of whose values is the sum of the elements in the argument along a line parallel to a specified dimension. In either case, the rank of the result is less than that of the argument; therefore, SUM is referred to as a reduction intrinsic. Fortran 90 includes a family of such reductions, and HPF adds more.

We inline these reduction intrinsics in such a way as to distribute the work as much as possible across the processors and to minimize the number of messages sent.

In general, the reduction is performed in three basic steps:

1. Each processor locally performs the reduction operation on its part of the reduction source into a buffer.
2. These partial reduction results are combined with those of the other processors in a "logarithmic" fashion (to reduce the number of messages sent).
3. The accumulated result is then locally copied to the target location.

Figure 15 shows how the computations and communications occur in a complete reduction of an array distributed over four processors. In this figure, each vertical column represents the memory of a single processor. The processors are thought of (in this case) as being arranged in a 2×2 square; this is purely for conceptual purposes—the actual processors are typically connected through a switch.

First, the reduction is performed locally in the memory of each processor. This is represented by the vertical arrows in the figure. Then the computations are accumulated over the four processors in two steps: the two parallel curved arrows indicate the interprocessor communication in the first step, followed by the communication indicated by the remaining curved arrow in the second step. Of course, for five to eight processors, three communication steps would be needed, and so on.

Although this basic idea never changes, the actual generated code must take into account various factors. These include (1) whether the object being reduced

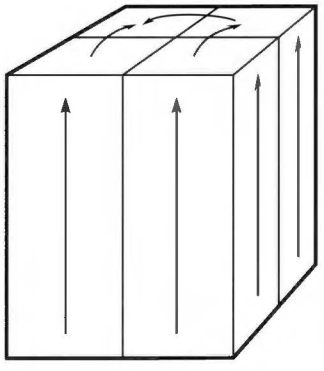


Figure 15
Computations and Communication for a Complete Reduction over Four Processors

is replicated or distributed, (2) the different distributions that each array dimension might have, and (3) whether the reduction is complete or partial (i.e., with a DIM argument).

Run-time Preprocessing of Irregular Data Accesses

Run-time preprocessing of irregular data accesses is a popular technique.[23] If an expression involving the same pattern of irregular data access is present more than once in a compilation unit, additional run-time preprocessing can be used to good effect. An abstract example would be code of the form:

```
call setup(U, V, W)
do i = 1, n_time_steps, 1
   do i = 1, n, 1
      A(V(i)) = A(V(i)) + B(W(i))
   enddo
   do i = 1, n, 1
      C(V(i)) = C(V(i)) + D(W(i))
   enddo
   do i = 1, n, 1
      E(V(i)) = E(V(i)) + F(W(i))
   enddo
enddo
```

which could be written in HPF as:

```
call setup(U, V, W)
do i = 1, n_time_steps, 1
   A = sum_scatter(B(W(1:n)), A, V(1:n))
   C = sum_scatter(D(W(1:n)), C, V(1:n))
   E = sum_scatter(F(W(1:n)), E, V(1:n))
enddo
```

To the compiler, the significant thing about this code is that the indirection vectors V and W are constant over iterations of the loop. Therefore, the compiler computes the source and target addresses of the data that has to be sent and received by each processor once at the top of the loop, thus paying this price one time. Each such statement then becomes a communication with full knowledge and is executed quite efficiently with message vectorization.

Other Communication Optimizations

The processing needed to set up communication of array assignments is fairly expensive. For each element of source data on a processor, the value of the data and the target processor number are computed. For each target data on a processor, the source processor number and the target memory address are computed. The compiler and run time also need to sort out local data that do not involve communication, as well as to vectorize the data that are to be communicated.

We try to optimize the communication processing by analyzing the iteration space and data space of the array sections involved. Examples of the patterns of operations that we optimize include the following:

- Contiguous data. When the source or target local array section on each processor is in contiguous memory addresses, the processing can be optimized to treat the section as a whole, instead of computing the value or memory address of each element in the section.

  In general, array sections belong to this category if the last vector dimension is distributed BLOCK or CYCLIC and the prior dimensions (if any) are all serial.

  If the source and target array sections satisfy even more restricted constraints, the processing overhead may be further reduced. For example, array operations that involve sending a contiguous section of BLOCK or CYCLIC distributed data to a single processor, or vice versa, belong to this category and result in very efficient communication processing.
- Unique source or target processor. When a processor only sends data to a unique processor, or a processor only receives data from a unique processor, the processing can be optimized to use that unique processor number instead of computing the processor number for each element in the section. This optimization also applies to target arrays that are fully replicated.
- Irregular data access. If all indirection vectors are fully replicated for an irregular data access, we can actually implement the array operation as a full-knowledge communication instead of a more expensive partial-knowledge communication.

  For example, the irregular data access statement

  ```
  A(V(:)) = B(:)
  ```

  can be turned into a regular remapping statement if V is fully replicated and A and B are both distributed.

  Furthermore, if B is also fully replicated, the statement is recognized as a local assignment, removing the communication processing overhead altogether.

Performance

In this section, we examine the performance of three HPF programs. One program applies the shallow-water equations, discretized using a finite difference scheme to a specific problem; another is a conjugate-gradient solver for the Poisson equation, and the third is a three-dimensional finite difference solver. These programs are not reproduced in this paper, but they can be obtained via the World Wide Web at http://www.digital.com/info/hpc/f90/.

The Shallow-water Benchmark

The shallow-water equations model atmospheric flows, tides, river and coastal flows, and other phenomena. The shallow-water benchmark program uses these equations to simulate a specific flow problem. It models variables related to the pressure, velocity, and vorticity at each point of a two-dimensional mesh that

is a slice through either the water or the atmosphere. Partial differential equations relate the variables. The model is implemented using a finite-difference method that approximates the partial differential equations at each of the mesh points.[24] Models based on partial differential equations are at the core of many simulations of physical phenomena; finite difference methods are commonly used for solving such models on computers.

The shallow-water program is a widely quoted benchmark, partly because the program is small enough to examine and tune carefully, yet it performs real computation representative of many scientific simulations. Unlike SPEC and other benchmarks, the source for the shallow-water program is not controlled.

The shallow-water benchmark was written in HPF and run in parallel on workstation farms using PSE. There is no explicit message-passing code in the program. We modified the Fortran 90 version that Applied Parallel Research used for its benchmark data. The F90/HPF version of the program takes advantage of the new features in Fortran 90 such as modules. The Fortran 77 version of the program is an unmodified version from Applied Parallel Research.

The resulting programs were run on two hardware configurations: as many as eight 275-megahertz (MHz) DEC 3000 Model 900 workstations connected by a GIGAswitch system, and an eight-processor AlphaServer 8400 (300-MHz) system using shared-memory as the messaging medium. Table 1 gives the speedups obtained for the 512 × 512-sized problem, with ITMAX set to 50.

The speedups in each line are relative to the DEC Fortran 77 code, compiled with the DEC Fortran version 3.6 compiler and run on one processor. The DEC Fortran 90 -wsf compiler is the DEC Fortran 90 version 1.3 compiler with the -wsf option ("parallelize HPF for a workstation farm") specified. Both compilers use version 3.58 of the Fortran RTL. The operating system used is Digital UNIX version 3.2.

Table 1 indicates that this HPF version of shallow water scales very well to eight processors. In fact, we are getting apparent superlinear speedup in some cases. This is due in part to optimizations that the DEC Fortran 90 compiler performs that the serial compiler does not, and in part to cache effects: with more processors, there is more cache. On the shared-memory machine, we are getting apparent superlinear speedups even when compared to the DEC Fortran 90 -wsf compiler's one-processor code; this is likely due to cache effects. The program appears to scale well beyond eight processors, though we have not made a benchmark-quality run on more than eight identical processors.

For purposes of comparison, Table 2 gives the published speedups from Applied Parallel Research on the shallow-water benchmark for the IBM SP2 and Intel Paragon parallel architectures. The speedups shown are relative to the one-processor version of the code. This table indicates that the scaling achieved by the DEC Fortran 90 compiler on Alpha workstation farms is comparable to that achieved by Applied Parallel Research on dedicated parallel systems with high-speed parallel interconnects.

A Conjugate-gradient Poisson Solver

The Poisson partial differential equation is a workhorse of mathematical physics, occurring in problems

Table 2
Speedups of HPF Shallow-water Code on IBM's and Intel's Parallel Architectures

	Number of Processors				
	8	4	3	2	1
IBM SP2	7.50	3.81	—	1.97	1.00
Intel Paragon	7.38	3.84	—	1.95	1.00

Table 1
Speedups of DEC Fortran 90/HPF Shallow-water Equation Code

	DEC Fortran 90 -wsf Compiler					DEC Fortran 77 Compiler
	Number of Processors					
	8	4	3	2	1	1
Eight 275-MHz, DEC 3000 Model 900 workstations in a GIGAswitch farm	8.57	3.13	2.19	1.59	1.00	1.00
Eight-processor, 300-MHz, shared-memory SMP AlphaServer 8400 systems	10.6	5.30	3.86	1.97	1.12	1.00

of heat flow and electrostatic or gravitational potential. We have investigated a Poisson solver using the conjugate-gradient algorithm. The code exercises both the nearest-neighbor optimizations and the inlining abilities of the DEC Fortran 90 compiler.[25]

Table 3 gives the timings and speedup obtained on a 1000 × 1000 array. The hardware and software configurations are identical to those used for the shallow-water timings.

Red-black Relaxation

A common method of solving partial differential equations is red-black relaxation.[26] We used this method to solve the Poisson equation in a three-dimensional cube. We compare the parallelization of this algorithm for a distributed-memory system (a cluster of Digital Alpha workstations) with Parallel Virtual Machine (PVM), which is an explicit message-passing library, and with HPF.[27] These algorithms are based on codes written by Klose, Wolton, and Lemke and made available as part of the suite of GENESIS distributed-memory benchmarks.[28]

Table 4 gives the speedups obtained for both the HPF and PVM versions of the program, which solves a 128 × 128 × 128 problem, on a cluster of DEC 3000 Model 900 workstations connected by an FDDI/GIGAswitch system. The speedups shown are relative to DEC Fortran 77 code written for and run on a single processor. This table shows that the HPF version performs somewhat better than the PVM version.

There is a significant difference in the complexity of the programs, however. The PVM code is quite intricate, because it requires that the user be responsible for the block partitioning of the volume, and then for explicitly copying boundary faces between processors. By contrast, the HPF code is intuitive and far more easily maintained. The reader is encouraged to obtain the codes (as described above) and compare them.

Table 4
Speedups of DEC Fortran 90/HPF and DEC Fortran 77/PVM on Red-black Code

	Number of Processors			
	8	4	2	1
DEC Fortran 77				1.00
DEC Fortran 77/PVM	7.01	3.73	1.79	—
DEC Fortran 90/HPF	8.04	4.10	1.95	1.05

In conclusion, we have shown that important algorithms familiar to the scientific and technical community can be written in HPF. HPF codes scale well to at least eight processors on farms of Alpha workstations with PSE and deliver speedups competitive with other vendors' dedicated parallel architectures.

Acknowledgments

Significant help from the following people has been essential to the success of this project: High Performance Computing Group engineering manager Jeff Reyer; the Parallel Software Environment Group led by Ed Benson and including Phil Cameron, Richard Warren, and Santa Wiryaman; the Parallel Tools Group managed by Tomas Lofgren and including David LaFrance-Linden and Chuck Wan; the Digital Fortran 90 Group led by Keith Kimball; David Loveman for discussions of language issues; Ned Anderson of the High Performance Computing Numerical Library Group for consulting on numerical issues; Brendan Boulter of Digital Galway for the conjugate-gradient code and help with benchmarking; Bill Celmaster, for writing the PVM version of the red-black benchmark and its related description; Roland Belanger for benchmarking assistance; and Marco Annaratone for useful technical discussions.

Table 3
Speedups of DEC Fortran 90/HPF on Conjugate-gradient Poisson Solver

	DEC Fortran 90 -wsf Compiler					DEC Fortran 77 Compiler
	Number of Processors					
	8	4	3	2	1	1
Eight 275-MHz, DEC 3000 Model 900 workstations in a GIGAswitch farm	14.1	8.38	5.20	2.52	1.07	1.00
Eight-processor, 300-MHz, shared-memory SMP AlphaServer 8400 systems	17.0	9.02	6.87	4.51	0.98	1.00

References and Notes

1. High Performance Fortran Forum, "High Performance Fortran Language Specification, Version 1.0," *Scientific Programming,* vol. 2, no. 1 (1993). Also available as Technical Report CRPC-TR93300, Center for Research on Parallel Computation, Rice University, Houston, Tex.; and via anonymous ftp from titan.cs.rice.edu in the directory public/HPFF/draft; version 1.1 is the file hpf_v11.ps.

2. C. Koelbel, D. Loveman, R. Schreiber, G. Steele, Jr., and M. Zosel, *The High Performance Fortran Handbook* (Cambridge, Mass.: MIT Press, 1994).

3. *Digital High Performance Fortran 90 HPF and PSE Manual* (Maynard, Mass.: Digital Equipment Corporation, 1995).

4. *DEC Fortran 90 Language Reference Manual* (Maynard, Mass.: Digital Equipment Corporation, 1994).

5. E. Albert, K. Knobe, J. Lukas, and G. Steele, Jr., "Compiling Fortran 8x Array Features for the Connection Machine Computer System," *Symposium on Parallel Programming: Experience with Applications, Languages, and Systems, ACM SIGPLAN,* July 1988.

6. K. Knobe, J. Lukas, and G. Steele, Jr., "Massively Parallel Data Optimization," *Frontiers '88: The Second Symposium on the Frontiers of Massively Parallel Computation, IEEE,* George Mason University, October 1988.

7. K. Knobe, J. Lukas, and G. Steele, Jr., "Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines," *Journal of Parallel and Distributed Computing,* vol. 8 (1990): 102–118.

8. K. Knobe and V. Natarajan, "Data Optimization: Minimizing Residual Interprocessor Data Motion on SIMD Machines," *Frontiers '90: The Third Symposium on the Frontiers of Massively Parallel Computation, IEEE,* University of Maryland, October 1990.

9. M. Gupta and P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers," *IEEE Transactions on Parallel and Distributed Systems,* vol. 3, no. 2 (1992): 179–193.

10. M. Gupta and P. Banerjee, "PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers," *ICS93: The Seventh ACM International Conference on Supercomputing,* Japan, 1993.

11. S. Chatterjee, J. Gilbert, and R. Schreiber, "The Alignment-distribution Graph," *Sixth Annual Workshop on Languages and Compilers for Parallel Computing,* 1993.

12. J. Anderson and M. Lam, "Global Optimizations for Parallelism and Locality on Scalable Parallel Machines," *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, ACM Press,* vol. 28 (1993): 1290–1317.

13. The seven values $A(9, 2)$, $A(9, 3)$, ... $A(9, 8)$ can be expressed concisely in Fortran 90 as $A(9, 2{:}8)$.

14. R. Souza et al., "GIGAswitch System: A High-performance Packet-switching Platform," *Digital Technical Journal,* vol. 6, no. 1 (1994): 9–22.

15. E. Benson, D. LaFrance-Linden, R. Warren, and S. Wiryaman, "Design of Digital's Parallel Software Environment," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 24–38.

16. D. Loveman, "The DEC High Performance Fortran 90 Compiler Front End," *Frontiers '95: The Fifth Symposium on the Frontiers of Massively Parallel Computation,* pages 46–53, McLean, Virginia, February 1995. IEEE.

17. D. Blickstein et al., "The GEM Optimizing Compiler System," *Digital Technical Journal,* vol. 4, no. 4 (Special Issue, 1992): 121–136.

18. C. Offner, "A Data Structure for Managing Parallel Operations," *Proceedings of the 27th Hawaii International Conference on System Sciences, Volume II: Software Technology* (IEEE Computer Society Press, 1994): 33–42.

19. J. Allen and K. Kennedy, "Vector Register Allocation," *IEEE Transactions on Computers,* vol. 41, no. 10 (1992): 1290–1317.

20. S. Amarasinghe and M. Lam, "Communication Optimization and Code Generation for Distributed Memory Machines," *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, ACM Press,* vol. 28 (1993): 126–138.

21. C.-W. Tseng, "An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines," Ph.D. thesis, Rice University, Houston, Tex., 1993. Available as Rice COMP TR93-199.

22. A. Rogers, "Compiling for Locality of Reference," Technical Report TR91-1195, Ph.D. thesis, Cornell University, Ithaca, N.Y., 1991.

23. J. Saltz, R. Mirchandaney, and K. Crowley, "Run-time Parallelization and Scheduling of Loops," *IEEE Transactions on Computers* (1991): 603–611.

24. R. Sadourney, "The Dynamics of Finite-difference Models of the Shallow-water Equations," *Journal of Atmospheric Sciences,* vol. 32, no. 4 (1975).

25. B. Boulter, "Performance Evaluation of HPF for Scientific Computing," *Proceedings of High Performance Computing and Networking, Lecture Notes in Computer Science 919* (Springer-Verlag, 1995).

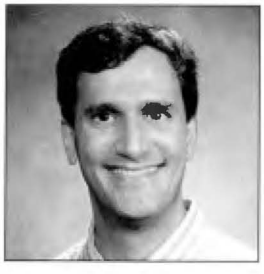
26. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in Fortran: The Art of Scientific Computing* (Cambridge: Cambridge University Press, 2d edition, 1992).

27. A. Geist, *PVM: Parallel Virtual Machine* (Cambridge, Mass.: MIT Press, 1994).

28. A. Hey, "The GENESIS Distributed Memory Benchmarks," *Parallel Computing,* vol. 17, no. 10–11 (1991): 1275–1283.

Biographies

Jonathan Harris
Jonathan Harris is a consulting engineer in the High Performance Computing Group and the project leader for the transform (HPF parallelization) component of the DEC Fortran 90 compiler. Prior to the High Performance Fortran project, he designed the instruction set for the DECmpp, a 16K processor machine that became operational in 1987. He also helped design a compiler and debugger for the machine, contributed to the processor design, and invented parallel algorithms, some of which were patented. He obtained an M.S. in computer science in 1985 as a Digital Resident at the University of Illinois; he has been with Digital since 1977.



John A. Bircsak
A principal software engineer in Digital's High Performance Computing Group, John Bircsak contributed to the design and development of the transform component of the DEC Fortran 90 compiler. Before joining Digital in 1991, he was involved in the design and development of compilers at Compass, Inc.; prior to that, he worked on compilers and software tools at Raytheon Corp. He holds a B.S.E. in computer science and engineering from the University of Pennsylvania (1984) and an M.S. in computer science from Boston University (1990).

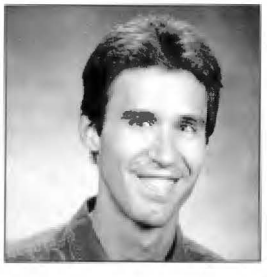
M. Regina Bolduc
Regina Bolduc joined Digital in 1991; she is a principal software engineer in the High Performance Computing Group. Regina was involved in the development of the transform and front end components of the DEC Fortran 90 compiler. Prior to this work, she was a senior member of the technical staff at Compass, Inc., where she worked on the design and development of compilers and compiler-generator tools. Regina received a B.A. in mathematics from Emmanuel College in 1957.

Jill Ann Diewald
Jill Diewald contributed to the design and implementation of the transform component of the DEC Fortran 90 compiler. She is a principal software engineer in the High Performance Computing Group. Before joining Digital in 1991, Jill was a technical coordinator at Compass, Inc., where she helped design and develop compilers and compiler-related tools. Prior to that position, she worked at Innovative Systems Techniques and Data Resources, Inc. on programming languages that provide economic analysis, modeling, and database capabilities for the financial marketplace. She has a B.S. in computer science from the University of Michigan.

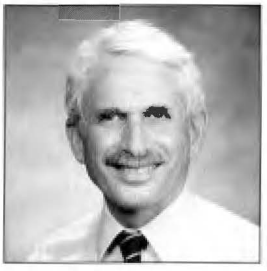
Israel Gale
Israel Gale is a principal writer in the High Performance Computing Group and the author of Digital's High Performance Fortran Tutorial. He joined Digital in 1994 after receiving an A.M. degree in Near Eastern Languages and Civilizations from Harvard University.

Neil W. Johnson
Before coming to Digital in 1991, Neil Johnson was a staff scientist at Compass, Inc. He has more than 30 years of experience in the development of compilers, including work on the vectorization and optimization phases and tools for compiler development. As a principal software engineer in Digital's High Performance Computing Group, he has worked on the development of the front-end phase for the DEC Fortran 90 compiler. He is a member of ACM and holds B.A. (magna cum laude) and M.A. degrees in mathematics from Concordia College and the University of Nebraska, respectively.

Shin Lee
Shin Lee is a principal software engineer in Digital's High Performance Computing Group. She contributed to the design and development of the transform component of the DEC Fortran 90 compiler. Before joining Digital in 1991, she worked on the design and development of compilers at Encore Computer Corporation and Wang Labs, Inc. She received a B.S. in chemistry from National Taiwan University and an M.S. in computer science from Michigan State University.



C. Alexander Nelson
In 1991, Alex Nelson came to Digital to work on the SIMD compiler for the MasPar machine. He is a principal software engineer in the High Performance Computing Group and helped design and implement the transform component of the DEC Fortran 90 compiler. Prior to this work, he was employed as a software engineer at Compass, Inc. and a systems architect at Incremental Systems. He received an M.S. in computer science from the University of North Carolina in 1987 and an M.S. in chemistry (cum laude) from Davidson College in 1985. He is a member of Phi Beta Kappa.



Carl D. Offner
As a principal software engineer in Digital's High Performance Computing Group, Carl Offner has primary responsibility for the high-level design of the transform component of the DEC Fortran 90 compiler. He is also a member of the Advanced Development Group working on issues of parallelizing DO loops. Before joining Digital in 1993, Carl worked at Intel and at Compass, Inc. on compiler development. Before that, he taught junior high and high school mathematics for 16 years. Carl represents Digital at the High Performance Fortran Forum. He is a member of ACM, AMS, and MAA and holds a Ph.D. in mathematics from Harvard University.

Design of Digital's Parallel Software Environment

Edward G. Benson
David C.P. LaFrance-Linden
Richard A. Warren
Santa Wiryaman

Digital's Parallel Software Environment was designed to support the development and execution of scalable parallel applications on clusters (farms) of distributed- and shared-memory Alpha processors running the Digital UNIX operating system. PSE supports the parallel execution of High Performance Fortran applications with message-passing libraries that meet the low-latency and high-bandwidth communication requirements of efficient parallel computing. It provides system management tools to create clusters for distributed parallel processing and development tools to debug and profile HPF programs. An extended version of dbx allows HPF-distributed arrays to be viewed, and a parallel profiler supports both program counter and interval sampling. PSE also supplies generic facilities required by other parallel languages and systems.

Digital's Parallel Software Environment (PSE) was designed to support the development and execution of scalable parallel applications on clusters (farms) of distributed- and shared-memory Alpha processors running the Digital UNIX operating system. PSE version 1.0 supports the High Performance Fortran (HPF) language; it also supplies generic facilities required by other parallel languages and systems. PSE provides tools to define a cluster of processors and to manage distributed parallel execution. It also contains development tools for debugging and profiling parallel HPF programs. PSE supports optimized message passing over multiple interconnect types, including fiber distributed data interface (FDDI), asynchronous transfer mode (ATM), and shared memory.[1]

In this paper, we present an overview of PSE version 1.0 and explain why it was designed and selected for use with HPF programs. We then discuss cluster definition and management, describe the PSE application model, and discuss PSE's message-passing communication options, including an optimized transport for message passing. We conclude with our performance results.

Overview of PSE

Many researchers and computer industry experts believe that to achieve cost-effective scalable parallel processing, systems must be built using off-the-shelf components and not specialized CPUs and interconnects.[2,3] In accordance with this view, we have designed Digital's PSE to support the building of a consistent yet flexible and easy-to-use parallel-processing environment across a networked collection of AlphaGeneration workstations, servers, and symmetric multiprocessors (SMPs). Layered on top of the Digital UNIX operating system, PSE provides the system software and tools needed to group collections of machines for parallel processing and to manage transparently the distribution and running of parallel applications. PSE is implemented as a set of run-time libraries and utilities and a daemon process.

PSE version 1.0 is designed to support clusters consisting of 1 to 256 machines interconnected with any networking fabric that Digital UNIX supports with the

transmission control protocol/internet protocol (TCP/IP). Networking technologies can range from simple Ethernet to FDDI, ATM, and MEMORY CHANNEL. Parallel execution is most efficient when the interconnect technology offers high-bandwidth and low-latency communications to the user at the process level. When building a cluster for parallel processing, the bisectional bandwidth of the communications fabric should scale with the number of processors in the cluster. In practice, such a configuration can be achieved by building clusters using Alpha processors and Digital's GIGAswitch/FDDI as components in a multistage switch configuration.[4,5] Figures 1 and 2 show two examples of PSE cluster configurations. Although the design center for PSE is a set of machines connected by a high-speed local area interconnect, a cluster can be constructed that includes remote machines connected by a wide area network.

PSE is a collection of many interrelated entities that support parallel processing. PSE's model is to collect machines (called *members*) into a set (called a *cluster*). The members are generally all the machines at a site or within an organization that have or might have PSE installed. One then subsets the cluster into named (*partitions*) that may overlap. The members of a partition usually share some common attribute, which could be administrative (e.g., the machines of the development group), geographic (e.g., connected to the same FDDI switch), or relevant to the configuration (e.g., large memory, SMP).

The members of a cluster, the partitions, and other related data form a configuration database that can be maintained in different ways, but preferably by a system administrator. The configuration database can be distributed using the Domain Name System (DNS) or as a simple file distributed by Network File System (NFS).[6] A daemon process **farmd** runs on each member to provide per-member dynamic information, such as availability and system load average. The static database plus the dynamic information allow applications to perform tasks such as load balancing.

HPF Program Support

PSE was designed to be largely language-independent; it currently supports the HPF programming language. HPF allows programmers to express data parallel computations easily using Fortran 90 array-operation syntax. As a result, users can obtain the benefits of parallel processing without becoming systems programmers and developing message passing or threads-based programs. The HPF language and compiler are discussed elsewhere in this issue of the *Digital Technical Journal*.[7]

Writing parallel applications in HPF is significantly less complex than decomposing a problem and coding a solution using explicit message passing, but good development tools are required. To allow the viewing of HPF distributed arrays, we developed an extended version of **dbx** and a parallel profiler that supports both program counter and interval sampling. These tools are discussed later in this paper.

High performance and efficient communication are essential to success in parallel processing. PSE includes a private message-passing library for use with compiler-generated code. Thus it avoids overhead such as buffer alignment and size checking that are required with user-visible programming interfaces, such as Parallel Virtual Machine (PVM).[8] The message-passing library supports shared memory and both TCP/IP and user datagram protocol (UDP)/IP protocols on many types of media, including FDDI and ATM. PSE also includes an optional subset implementation of the UDP, known as UDP_prime, that has been optimized to reduce latency and improve efficiency. This optimization is discussed later in this paper.

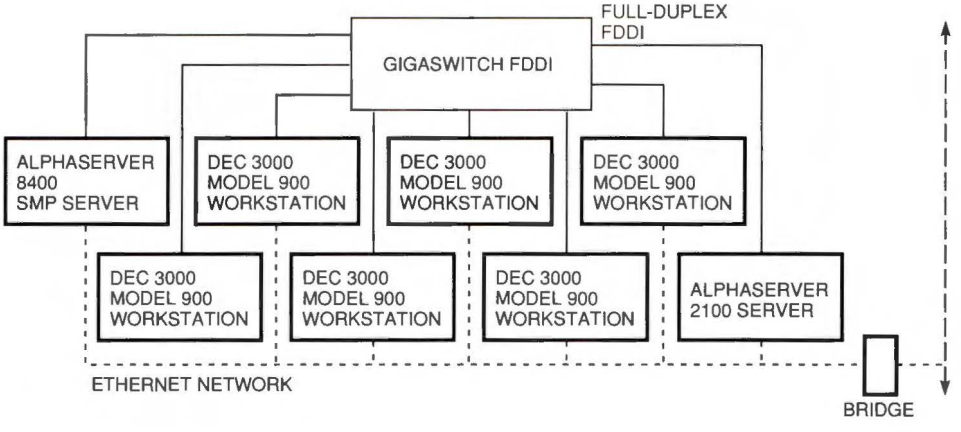


Figure 1
PSE Basic Configuration

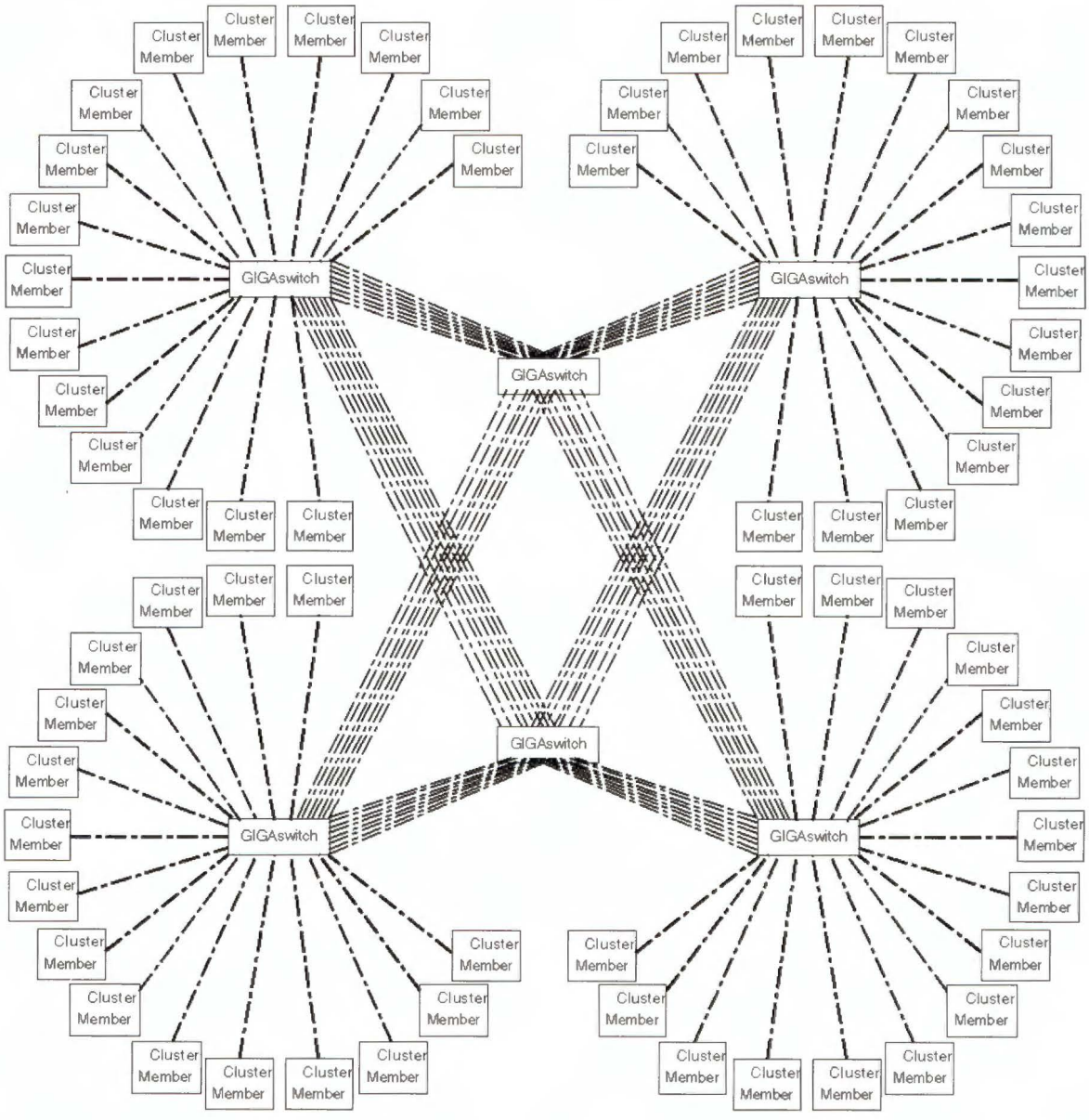


Figure 2
PSE Multistage Switch Configuration

Before developing PSE for use with HPF programs, Digital considered two major alternatives: the distributed computing environment (DCE) and PVM.[8,9] (At that time, the message-passing interface [MPI] standard effort was in progress.[10])

Although a good model for client-server application deployment, DCE is designed for use with remote CPU resources via procedure calls to libraries. This model is very different from the data-parallel and message-passing nature of distributed parallel processing. Its synchronous procedure call model requires the extensive use of threads. In addition, DCE contains a significant number of setup and management tasks. For these reasons, we rejected the DCE environment.

Three major considerations in our choice to develop PSE instead of using PVM were stability, performance, and transparency. At the start of the PSE project, the publicly available version of PVM did not meet the stability, performance, and transparency goals of the PSE project.

Cluster Definition and Management

PSE is designed to operate in a common system environment where systems are organized so that user access, file name space, host names, and so on are consistent. The ultimate goal for the systems in a distributed parallel-processing environment is to approach

the transparent usability of a symmetric multiprocessor. Facilities such as NFS (to mount/share file systems among machines, in particular working directories) and network information service (NIS) (also known as "yellow pages" and used to share password files) are frequently used to set up a common system environment. In such an environment, users can log into any machine and see the same environment. Other distributed environments such as Load Sharing Facility (LSF) make this same design assumption.[11]

A consistent file name space allows all processes that make up an application to have the same file system view by simply changing directory to the working directory of the invoking application. Consistent user access allows PSE to use the standard UNIX remote shell facility to start up peer processes with standard security checking.

Systems in a common system environment are candidates to become members of a cluster. A cluster is often the largest set of machines running PSE and sharing a common system environment within an organization or site. A cluster is divided into partitions that can overlap. A partition consists of a set of machines grouped together to meet the needs of an application or user. Although partitions may be defined in many ways, systems in a partition usually share common attributes.

Partitions

Parallel programs run most efficiently on a balanced hardware configuration. Typically, organizations have a varied collection of machines. Over time, organizations often acquire new hardware with different network adapters, faster CPUs, and more memory. Such situations can easily lead to increasing difficulty in predicting application performance if scheduling and load-balancing algorithms treat all machines in a cluster equivalently. In addition to hardware differences, individual machines can have different software installed that affects the ability to run applications.

The PSE engineering team recognized that the number of characteristics that users might want to manage for processor allocation and load-balancing purposes would be overwhelming. To limit the problem, a design was chosen that allows machines to be grouped arbitrarily into named partitions. A partition can be thought of as a parallel machine. Although a system can be a member of two different partitions and therefore cause overlap, PSE does not attempt to load balance or schedule processes beyond partition boundaries. Overlapping partitions can therefore create a complex and potentially conflicting scheduling situation. Well-defined and managed partitions allow for flexibility and predictability.

In addition to identifying machine membership, partition definition allows various execution-related characteristics to be set. Examples include the specification of a default communication type, the default execution priority, the upper bound on the execution priority, and access control to partition resources. Access control is enforced only on PSE-related activity and does not affect the use of the machine for other applications.

Configuration Database

PSE cluster configuration information is captured in a database. The database includes a list of cluster members, partitions, and partition members. Additional attributes such as the default partition of a cluster, user access lists for a partition, and preferred network addresses for members of a partition can be encoded in the database.

The PSE configuration database can be distributed to all cluster members in two ways: by storing it in a file that is accessible from all cluster members, or by storing it as a Domain Name System (DNS) database. The usage patterns of the cluster database fit well with the usage patterns of a DNS database. In particular, DNS provides central administrative control with version numbering to maintain consistency during updates. It is designed for query-often, update-seldom usage; it is distributed and allows secondary servers to increase availability. Applications linked with the PSE run-time libraries transparently access the database to obtain configuration information.

In the DNS database, each PSE configuration token-value pair is stored as DNS TXT records. The original specification for DNS did not have TXT records, but additional general information was attached to domain names at the request of MIT's Project Athena.[12] The list of the TXT records, along with DNS header information such as version number, forms a DNS domain whose name is the PSE cluster name. To facilitate the creation and setup of a PSE cluster, we built the psedbedit utility for editing and maintaining configuration databases.

A simple file that is available on all members of the cluster can also be used as the cluster configuration database. The file could be made available through NFS or copied to all nodes using rdist. This alternative might be appropriate for very simple clusters where the services of DNS are not warranted or in cases where local policy precludes the use of DNS.

Dynamic Information and Control

In addition to the static information of the configuration database, there are also several pieces of dynamic information that optimize usage of clusters and partitions. At the most fundamental level is availability, i.e., is a machine running? Other information includes the number of CPUs, load average, number of allowed PSE jobs, and number of active PSE jobs. All these

factors can help an application choose the best set of members for parallel execution. This dynamic information is collected by a daemon process (farmd). The farmd daemon process executes as a privileged (root) process on each cluster member and listens for requests on a well-known cluster-specific UDP/IP port.

Multiple cluster members defined in the configuration database are designated as load servers. The load servers are the central repository for the dynamic information for the entire cluster. Their farmd process periodically receives time-stamped updates from the individual daemons. Applications query the load servers for both static and dynamic information. Applications do not themselves parse the database nor query the individual farmd daemons running on each cluster member.

Once PSE is installed and configured, farmd is started each time the system is booted. The name of the cluster that farmd will service and the number of PSE jobs (job slots) that will be allowed to run are set. The inetd facility is used to restart farmd in response to UDP/IP connection requests, if farmd is not running.[13] Use of the inetd facility to start farmd improves the availability of machines to run PSE applications by transparently restarting farmd in the case of a failure.

As farmd daemons are started, they attempt to establish TCP/IP connections with their neighbors as defined by the PSE configuration database.[14] This process is undertaken by all cluster members and quickly results in a configuration ring whose purpose is the detection of node or network failures. We chose a simple ring of TCP/IP connections because the mechanism is passive, i.e., it relies on the loss of TCP/IP connectivity and does not impose any additional load on the system or network under normal conditions. When connectivity to a member is lost, neighboring cluster members report the member being unavailable. This prevents PSE from attempting to schedule new applications on the failed member.

Failures that do not break the configuration ring, but prevent updated load information from being sent to the load server, are detected by checking the time-stamps on previously received load information. As soon as a "time-to-live" period expires for a particular member's load information, the load servers disable further use of the suspect node. System managers are also able to set the number of job slots to zero at any time, thus disabling the host for new PSE-related activities. This has no effect on currently executing applications.

Pseudo-gang Scheduling

The start-up sequence for a PSE application includes the potential modification of execution priority and scheduling policy. These changes are made in accordance with the user command-line options and/or the default characteristics defined by the PSE configuration database. To allow nonroot UID processes to elevate scheduling priorities and/or alternate scheduling policies, farmd modifies the user process's scheduling priority or policy. Processes scheduled at a high real-time priority using a first in, first out (FIFO) queue with preemption policy achieve a pseudo-gang-scheduling effect. (Gang scheduling ensures that all processes associated with a job are scheduled simultaneously.) This effect occurs because of the scheduling preference given high-priority jobs and because PSE polls for messages for a period of time before giving up the CPU.

Using PSE

Parallel applications are developed for PSE using the Digital Fortran 90 compiler. When the Fortran 90 compiler is invoked with the -wsf N flag, HPF source codes are compiled and then linked with a PSE library for parallel execution on *N* processors. After defining a partition in which to run, a PSE application can be run simply by typing the name of the application. The following example shows the compilation and execution of a four-process program called *myprog* on a set of cluster members in the partition named fast.

```
csh> setenv PSE_PARTITION fast
csh> f90 -wsf 4 myprog.f90 -o myprog
csh> myprog > myprog.out < myprog.dat &
```

Transparently, PSE starts up four processes on members of the partition fast; creates communications channels between the processes; supports redirected standard input, output, and error (standard I/O); and controls the execution and termination of the application. Several environment variables and run-time flags are available to control how an application executes. Figure 3 shows how to use PSE.

PSE Application Model

PSE implements an application as a collection of interconnected processes. The initial process created when a user runs an application is called the *controlling process.* It provides application distribution and start-up services and preserves UNIX user-interface semantics (i.e., standard I/O), but does not participate in the HPF parallel computation. The controlling process usually determines which partition members to use for the parallel computation by getting system load information from a load server and then distributing the new processes across the partition. As an alternative, users can direct computation onto specific partition members.

The controlling process starts a process called the *io_manager* on each partition member participating in the parallel execution. Each io_manager then starts one or more application peer processes that perform the user-specified computation. The use of an io_manager is necessary to create a parent-child

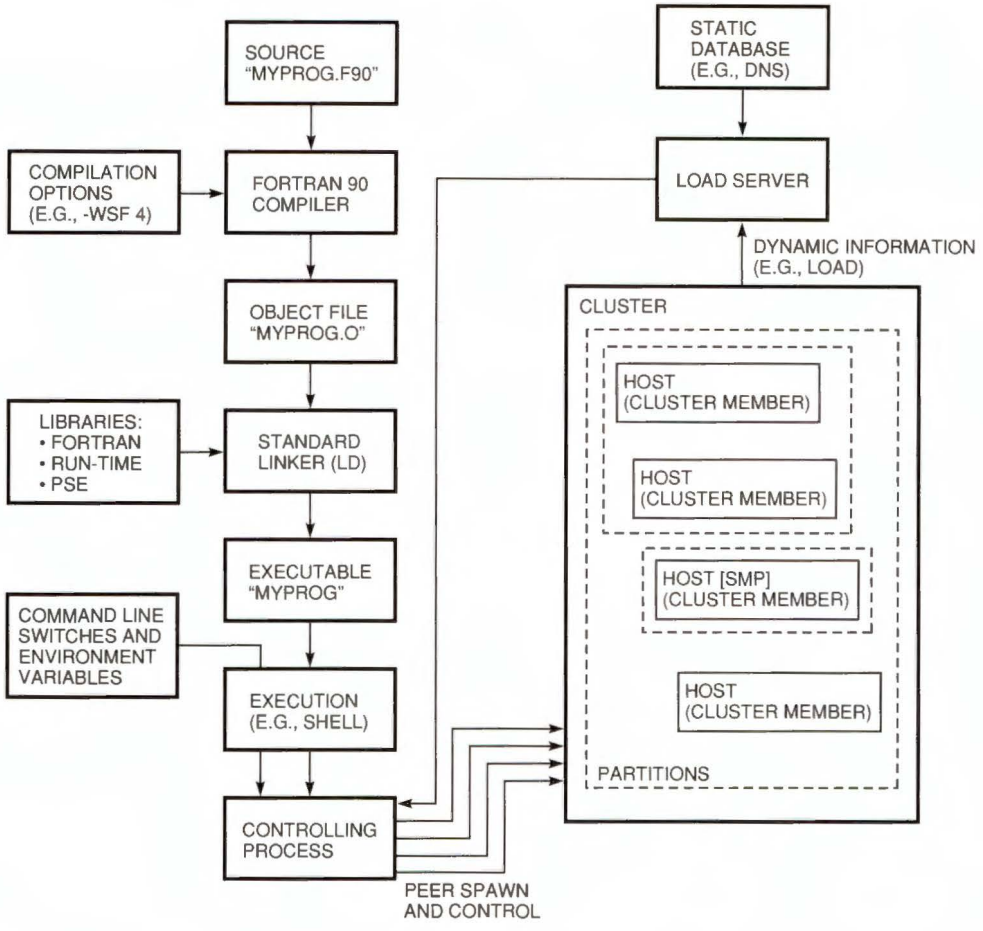


Figure 3
PSE Use

process relationship between the io_manager and peer processes. This relationship is used for exit status reporting and process control. It also enables or eases other activities, such as signal handling and propagation. Peer processes create communication channels between themselves and perform standard I/O through a designated peer. Standard I/O is forwarded to and from the controlling process through the io_manager. Figure 4 shows a PSE application structure.

Application Initialization

Prior to the execution of any user code, an initialization routine executes automatically through functionality provided by the linker and loader. The initialization routine implements both the controlling process functions and the HPF-specific peer initialization. Because no explicit call is required, parallel HPF procedures can be used within non-HPF main programs, and proper initialization will occur. A simple HPF main program can also be used with PSE to start up and manage a task-parallel application that uses PVM or MPI for message passing.

In general, the controlling process places peer processes onto members of a partition, although hand placement of individual peers onto selected members is possible. To achieve efficiency and fairness in mapping a set of peers, the controlling process consults with a load server for load-balancing information. Which members are used and the order in which they are used is based on each member's load average, number of CPUs, and number of available job slots.

As an alternative, PSE may map peer processes onto members based upon a user-selected mode of operation. In the default physical mode of operation, PSE maps one peer process per member. In virtual mode, PSE allows more than one peer process per member, thereby enabling large virtual clusters. This is useful for developing and debugging parallel programs on limited resources. Virtual clusters also improve application availability: when the requested number of peer processes is greater than the available set of partition members, applications continue to run; however, they may suffer performance degradation.

Application Peer Execution

Each application peer process has an io_manager parent process that provides it with environment initialization, exit value processing, I/O buffering, signal forwarding, and potential scheduling priority and policy modification. Rather than include the

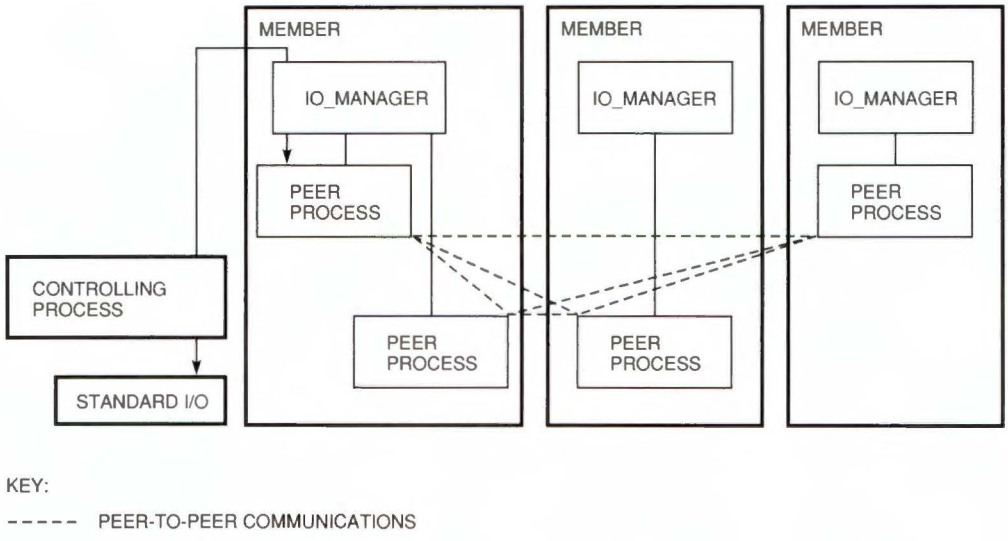


Figure 4
PSE Application Structure

io_manager's functions in each PSE executable, the io_manager is implemented as a simple utility.

Application peers run the same binary image as the controlling process. They inherit their current working directory, resource usage limits, and an augmented set of environment variables from their controlling process through their parent io_manager. When started, the initialization process described for the controlling process is repeated, but peers do not become controlling processes because they detect that a controlling process already exists. Instead, peer processes return from the initialization routines with communication links established and are ready to run user-application code. Figure 5 represents a controlling process, four application peers running on three members, and the communications between processes.

Application Exit

Multiple peer exits can have potentially conflicting exit values. Coordinating them into a single meaningful application exit value is the most challenging transparency issue faced by PSE. Under normal circumstances, all peer processes exit without error and at approximately the same time. The resulting exit values are reported to the application controlling process by the io_managers. The application (i.e., the controlling process) is allowed to exit without error only when all exit values are recorded and standard I/O connections are drained and closed. The HPF compiler generates synchronization code to guarantee the roughly synchronous exit for all nonerror conditions. This presumption allows PSE to implement a timely exit model, i.e., one by which we can reasonably assume

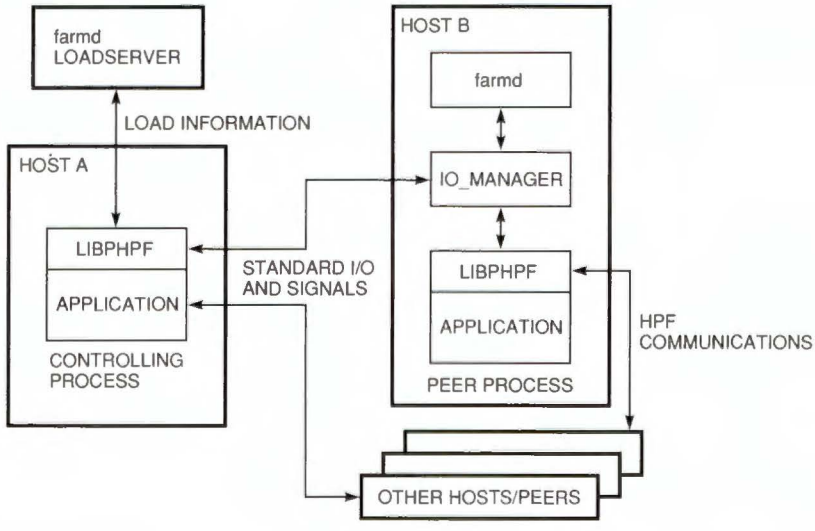


Figure 5
Communications between PSE Processes

normal activity will cease after receiving the last exit notification from an io_manager.

Peers that exit abnormally make it difficult to provide a meaningful exit value for the application. Consider one peer process that exits due to a segmentation fault and another that exits due to a floating-point exception. There is no single exit value possible for the application; PSE chooses the first abnormal value it sees. Furthermore, as a result of error detection in the communication library, the other peer processes will exit with lost network connections. It is possible that the controlling process will see an exit value for this effect before it sees an exit value for one of the causes, resulting in a misleading application exit value. To understand a faulting parallel application running under PSE, the core files associated with each peer process must be examined.

PSE includes support for capturing the entire application core state and for discriminating the multiple core files of a parallel application. Because peer processes share the same working directory, any core files generated would be inconsistent and overwrite one another due to *N* processes writing to the same core file name. PSE solves this problem by establishing a signal handler that catches core-generating signals, creates a peer-specific subdirectory, changes to the new directory, and resignals the signal to cause the writing of the core file. The root for the core directories can be set through an environment variable.

Issues

Although PSE achieves the standard UNIX look-and-feel for most application situations, complete transparency is not achieved. For example, timing an application-controlling process using the c-shell's built-in time command, does not time user code or provide meaningful statistics other than the elapsed wall clock time to start a parallel application and to tear it down. Another situation that highlights the parallel nature of PSE occurs during application debugging: multiple debug sessions are started by running the application with a debugger flag rather than by using dbx directly.

Tools for HPF Programming

The development model for HPF-based applications is a two-step process. First, a serial Fortran 90 program is written, debugged, and optimized. Then it is parallelized with HPF directives and again debugged and optimized. The development tools supplied with PSE address profiling and debugging. Unlike most of PSE, which is language-independent, both the pprof profiling facility and the "dbx in n windows" debugging facility are specific to HPF programming.

Profiling

Several issues in profiling parallel HPF programs do not apply to Fortran programs that execute serially. HPF execution occurs through multiple processes on multiple processors simultaneously and therefore produces multiple profiling data sets. The storage and analysis of these data sets must be coordinated to produce accurate and comprehensive program profiles. Unlike typical Fortran programs, significant time can be spent communicating in an HPF program. The Digital UNIX prof and pixie utilities do not handle either of these issues.[15] In addition, the prof utility has coarse-grained (1-millisecond resolution) program counter (PC) sampling and reports only down to the procedure level. To address these issues, Digital added profiling support to the Fortran 90 compiler and developed the pprof analysis tool.

Data Collecting The PSE parallel profiling facility handles profiling data collection in parallel by writing data to a set of files that are uniquely named. It encodes the application name, the type of data collection, and the peer number of the process. The analysis tool pprof merges the data in the file set when performing analysis and producing reports.

It supports two types of data collecting: nonintrusive traditional PC sampling and intrusive interval profiling. PC sampling simply records the program counter at each occurrence of the system clock interval interrupt. To achieve an accurate execution profile with PC sampling, programs must be long running to become statistically significant. Also, it is difficult to gather do-loop iteration data using PC sampling.

We developed interval profiling support to overcome the deficiencies of PC sampling. Interval profiling is achieved with compiler-inserted functions that record the entry and exit times for the execution of each event. This produces an accurate execution profile. Events include routines, array assignments, do loops, FORALL constructs, message sends, and message receives. Because the entry and exit times are recorded, time spent executing other events within an event is included, which gives a hierarchical profile. To achieve fine-resolution timings (single-digit nanoseconds), the Alpha process cycle counter is used to measure time.[16]

Analysis The pprof utility provides many different ways to examine and report on a large set of profiling data from a parallel program execution. Different approaches include focusing on routines, statements, or communications. In contrast, prof reports on procedures only. With pprof, the scope of the analysis can be limited to a single peer process or encompass all application processes. The range of reports generated can be comprehensive or limited to a number of events or

a percentage of time. Users can specify their reports from a combination of analysis, report format, and scoping options. By default, the pprof utility reports on routine-level activity averaged across all peer processes, which provides an overall view of application behavior.

Parallel programs execute most efficiently when there is minimum communication between processes. The high-level, data parallel nature of the HPF language reduces the visibility of communication to the programmer. To make tuning easier, pprof was designed with the ability to focus tuning on communication. Reports can be generated that help correlate the use of HPF data-distribution directives to observed communication activities.

Debugging

For PSE version 1.0, we are supplying a "dbx in n windows" capability. Each peer is controlled by a separate instance of dbx that has its own Xterm window. This capability gives users basic debugging functionality, including the ability to set breakpoints, get backtraces, and examine variables on an all-peer or a per-peer basis. We added a new command to dbx, hpfget, that allows the viewing of individual elements of a distributed array. We recognize it as far from meeting the challenges of an HPF debugger, and we are continuing the development of a new debugging technology.

Message-passing Model

One of the goals of PSE is to support high-performance, reliable message passing for parallel applications. At the start of the project, the HPF language and compiler technology were still in their infancy. Even though no HPF application code base existed, the PSE team needed to determine the messaging-passing requirements. To support message passing successfully, PSE had to be flexible enough to accommodate new interconnect technologies and network protocols, adapt to the message-passing characteristics of future HPF applications, and support the changing demands of the compiler. A need for high performance and efficiency with low latency was assumed.

The PSE message-passing facility provides primitives to initialize and terminate message-passing operations, to allocate and deallocate message buffers, and to send and receive messages. A PSE message contains a tag, a source peer number, and variable-length data. The higher layers fill in the tag, which is used as a message identifier on receive. The data is a stream of bytes without any data-type information. These primitives are not intended to be used in the application code. The HPF compiler implicitly generates calls to these primitives. Because the message-passing primitives are tightly coupled to the HPF compiler, overhead such as data-alignment restrictions and error checking can be eliminated.

The PSE message-passing model assumes that the application peers are running on systems with the same CPU architecture and networking capabilities. Each peer process can send or receive binary messages directly to or from any other peer. This is different from the PVM model, where messages might be routed to a pvmd daemon to be multiplexed to another peer, or messages might be converted to external data representation (XDR) to allow for data passing between machines with different architectures.[17]

Buffer allocation and deallocation routines are specific to each of the communication options that PSE supports. (These options are discussed in the following sections.) Before a message can be sent, a buffer must be allocated. The send primitive sends the message and implicitly deallocates the buffer. The receive primitive implicitly allocates a buffer containing the newly arrived message. Receive buffers have to be deallocated explicitly after they are used. Our initial design allowed a received message buffer to be reused for sending a new message, possibly to a different peer. This design was inefficient, especially when a communication option such as shared memory optimizes buffer allocation on a peer-by-peer basis. The current design uses a peer number as a parameter to the buffer allocation routine and does not allow reuse of the received message buffer.

The send primitive sends a message contained in a preallocated buffer to a specified peer. It guarantees reliable in-order delivery of messages. For underlying protocols, such as UDP/IP that do not provide this level of service, the message-passing library must provide it. A broadcast primitive is also provided to send a single message to all peers.

The receive primitive uses a particular message tag to receive a message with a matching tag from any peer. This allows the compiler to use functions that can perform calculations correctly when data is required from several peers, regardless of the order in which messages arrive. The normal operation for receive is to block the receiving peer until a matching tagged message arrives. A nonblocking receive is also provided to poll for messages.

Communication Options

PSE provides applications with several run-time selectable communication options. Within a single SMP system, PSE supports message passing over shared memory. On multiple system configurations, PSE supports network message passing using the TCP/IP or UDP/IP protocols over any network media that the Digital UNIX operating system supports. Currently, PSE supports a single communication option within an application execution, but the design supports multiple protocols and interconnects. Run-time selection of the communication options and media, which

is implemented using a vector of pointers to functions within a shared library, provides flexibility to introduce new protocols and media without having to recompile or relink existing applications.

Shared-memory Message Passing

The use of shared memory as a message-passing medium allows for very high performance because data does not have to be copied. When designing shared-memory messaging, we looked at a variety of interrelated issues, including coordination mechanisms, memory-sharing strategies, and memory consumption. The use of locks (i.e., semaphores) in the traditional manner to coordinate access to shared-memory segments proved problematic. For example, clients often request a message from any peer, not from a particular peer. This implies the use of a general receive semaphore that senders would unlock after delivering data. Contention for a single lock could be significant and could become a performance bottleneck. Instead of locks, a simple set of producer and consumer indexes is used to manage a ring buffer of messages. Senders read the consumer index and update the producer index, and receivers read the producer index and update the consumer index to synchronize. No locking is required.

Several memory-sharing strategies are possible: all peers may share a single large segment, each pair of peers may share a segment, and each pair of peers may have a pair of unidirectional segments. The use of unidirectional pairs of shared-memory segments offers several advantages: it simplifies the code by eliminating multiplexing; it fits in well with the design of MEMORY CHANNEL hardware, which is unidirectional; and by creating receive segments with read-only protection, it promotes robustness.[18] A disadvantage to the use of unidirectional segment pairs is increased memory use due to limited sharing. Because of its advantages and because the coordination of the producer/consumer index does not require segments to be shared between peers, we selected unidirectional pairs of shared-memory segments as our memory-sharing strategy.

To enhance performance, a receiver spins, waiting for a peer to produce a message. If there is no data after a number of spin iterations, the receiver voluntarily deschedules itself. The number of spin iterations was chosen to be small enough to be polite, but large enough to permit scheduling when a peer produced a message. An additional performance enhancement allows the user, via command line option, to prevent peers from migrating between processors, which results in better cache utilization.

TCP/IP Message Passing

TCP/IP is the default communication option. It provides full wire bandwidth for peer-to-peer communication with large message transfer sizes across a variety of network media. The implementation of the message-passing primitive operations is relatively straightforward since TCP/IP provides reliable, in-order, connection-oriented delivery of messages. The TCP/IP initialization routine sets up a vector of bound and connected socket descriptors, one for each peer. These sockets are used to send messages to other peers. The receive primitive uses a blocking select() system call on all sockets. Because TCP/IP is connection based, abnormal peer termination and network faults can be detected by connection loss.

Although TCP/IP provides acceptable bandwidth, latency-sensitive applications might suffer from the processing overhead of the TCP/IP protocol. The connection-oriented nature of TCP/IP also requires the application to maintain many socket descriptors, which reduces scalability and necessitates the use of expensive select() system calls on receive.

UDP/IP Message Passing

To address the latency and overhead of TCP/IP, PSE provides UDP/IP as an option that can be selected at run time. UDP/IP is a connectionless protocol that provides unordered, best-effort delivery of messages. Because UDP/IP is connectionless, the initialization function needs to set up a single locally bound socket description for all peer-to-peer communication. File descriptor use is not a scaling issue when UDP/IP is used for messaging.

Reliable in-order delivery of messages is implemented at the library level. Each peer maintains a set of send and receive ring buffers, one for each peer. The ring buffers have producer and consumer indexes to indicate positions in the ring where messages can be read or written. The buffer-allocation primitive allocates buffers from the send ring whenever possible, or from a pool of overflow buffers when the ring is full. The use of an overflow buffer eliminates the need for upper levels to provide flow control or to block sends. The send and receive primitives manipulate the producer and consumer indexes of the send and receive rings. In-order delivery of messages is guaranteed through the use of a sliding window protocol with sequentially numbered messages. For efficiency, piggybacked acknowledgments are used.

To improve scheduling synchronization among multiple peers, especially when a high-priority FIFO scheduling policy is used, the UDP/IP option uses a nonblocking socket. On receive, it loops calling the recvfrom() system call many times before calling the expensive select() system call to wait for a message to arrive. Abnormal peer termination and network faults cannot be detected since the socket layer does not maintain a connection state. The UDP/IP option contains a user-specifiable time-out value by which the peer application will exit when there is no socket activity.

The UDP/IP option provides better bandwidth than the TCP/IP with smaller messages and matches the TCP/IP bandwidth at large message size. The user-level latency reduction, however, was less than expected. The next two sections discuss our investigation into ways to optimize the latency of UDP/IP and the performance of the message-passing options.

Optimizing UDP/IP

Our initial approach to improve latency was to reexamine the standard UDP/IP code path within the Digital UNIX kernel for unnecessary overhead. Our idea was to create a faster path, optimized for a UDP/IP over a local area network (LAN) configuration by reducing numerous conditional checks in the path. Although this work yielded some improvement, it was not enough to justify supporting a deviation from the standard code path. An overhaul of the original code path would have been necessary for this approach to gain significant improvement in latency.

UDP/IP provides a general transport protocol, capable of running across a range of network interfaces. We realize the value in retaining the generality of UDP/IP. For optimal performance, however, we anticipate typical cluster configurations being constructed using a high-performance switched LAN technology such as the GIGAswitch/FDDI system.[5] In such configurations, the IP family of protocols presents unnecessary protocol-processing overhead. A messaging system using a lower-level protocol, such as native FDDI, would offer better latency, but its implementation requires the use of nonstandard mechanisms to access the data link layer directly, which is less general and portable than a UDP/IP implementation.

Based on the above observations, we designed a new protocol stack in the kernel, called UDP_prime, to coexist with the standard UDP/IP stack. UDP_prime packets conform to the UDP/IP specification.[19] To reduce the amount of per-packet processing and approach that of a lower-level protocol, UDP_prime imposes several restrictions on its use. These restrictions optimize the typical switched LAN cluster configurations. To retain the generality of UDP/IP, UDP_prime falls back to the standard UDP/IP stack when these restrictions are not applicable.

Restrictions on UDP_prime

The LAN nature of the cluster configuration imposes a restriction on UDP_prime. Each cluster member has to be within the same IP subnet, which is directly accessible from any other member. With this restriction, routing decision and internet-to-hardware address resolution can be done once for each peer-to-peer connection rather than on a per-packet basis. Per-packet UDP/IP checksum processing can also be eliminated, because intermediate routing is not involved and the data link cyclic redundancy check (CRC) is sufficient to guarantee error-free packets.

The next restriction is the maximum length of the message. PSE message passing uses fixed-size buffers. UDP_prime restricts the maximum buffer size to be the maximum transmission unit (MTU) of the underlying network interface. This eliminates per-message IP fragmentation and defragmentation overhead. Since the messaging clients have to fragment the messages into fixed-size buffers at the higher layer, there is no need for the IP layer to perform further fragmentation.

One complication in our current implementation occurs when multiple peers are running on a single system while others are on remote systems. The default behavior for peers within a single system is to communicate across the loopback interface. In this situation, there are two MTU values, one for the network interface and one for the loopback interface. Our current implementation of UDP_prime does not allow communication over the loopback interface so that a single-size MTU can be used. Further studies need to be done to find an optimal maximum buffer size, taking into account multiple MTU values, page alignment, and so forth.

Based on the above restrictions, UDP_prime optimizes the per-packet processing overhead of sending a packet by constructing a UDP, IP, and data link packet header template for each peer at initialization. Except for a few fields, the content of these headers is static with respect to a particular peer. UDP_prime defines a new IP option, IP_UDP_PRIME, for the setsockopt() system call, to allow the messaging system to define the set of peers and their Internet addresses involved in the application execution.[20] The IP option processing, done prior to sending any message, makes routing decisions, performs Internet-to-hardware address resolution, and fills in the static portion of the header fields. When sending a packet, UDP_prime simply copies the header template to the beginning of the packet, minimizing the per-packet processing overhead and increasing the likelihood of the templates being in the CPU cache. Several header fields, such as the IP identification, header checksum, and packet length fields, are then filled dynamically, and the complete packet is presented to the interface layer.

UDP_prime Packet Processing

Since a UDP_prime packet is a UDP/IP packet, the standard UDP/IP receive processing can handle the packet and deliver it to the messaging client. To trigger the use of UDP_prime optimized receive processing, the sending system uses the type of service (TOS) field within the IP header to specify priority delivery of the packet.[21] The priority delivery indication does not by itself uniquely differentiate between UDP_prime and UDP/IP packets, as any other IP packets can also have the TOS field set to priority. As a result, the

optimized receive processing has to check for the packet's adherence to the UDP_prime restrictions. Nonadherence to the restrictions reroutes the packet to the standard receive processing code.

When a packet arrives at a network interface, the interface posts a hardware interrupt, and the interface interrupt service routine processes the packet. The standard interrupt service routine deletes the data link header, and hands the packet over to the netisr kernel thread.[22] Netisr demultiplexes the packet based on the packet header contents and delivers it to the application's socket receive buffer. Netisr, designed to be a general-purpose packet demultiplexer, runs at a low-interrupt priority level. The main reason for a thread-based demultiplexer is extensibility. New protocol stacks can be registered to the thread. Since there is no a priori knowledge of the execution and SMP locking requirements of these stacks, a thread-based low-interrupt priority demultiplexer is needed so that the network interrupt processing time can be held to a minimum. The extensibility feature, however, introduces a context switch overhead.

For UDP_prime, the packet header processing time on the receive path is almost a small constant. We modified the interface service routine to demultiplex the packet by processing the data link, IP, and UDP headers, and deliver the packet to the socket receive buffer without handing it over to netisr. This short circuit path is used only when the packet is a UDP/IP packet with no IP fragmentation and with priority delivery indication. If these conditions are not met, the standard netisr path is chosen. The UDP_prime receive path eliminates the netisr context switch overhead. This is a significant advantage, especially when the receiving application runs with a real-time FIFO scheduling policy.

SMP Synchronization

One difficulty in designing the UDP_prime stack to run in parallel with the standard UDP/IP stack was in SMP synchronization.[23] The socket buffer structure is a critical section guarded by a complex lock. Requesting a complex lock in Digital UNIX blocks execution if the lock is taken. To prevent deadlocks, its use is prohibited at an elevated priority level, such as the case in the interrupt service routine. To work around this problem, a new spin lock was introduced in the short circuit path and in the socket layer where access to the socket buffer needs to be synchronized.

Performance

To measure message-passing performance, we used two DEC 3000 Model 700 workstations connected by a GIGAswitch/FDDI system using TURBOchannel-based DEFTA full-duplex FDDI adapters. Each workstation contained a 225-megahertz (MHz) Alpha 21064 microprocessor and was running the Digital UNIX version 3.0 operating system.

Figure 6 shows the message-passing bandwidth for TCP/IP, UDP/IP, and UDP_prime transports at different message sizes. The bandwidth was measured at the message-passing application programmer interface (API) level, taking into account allocation and deallocation of each message buffer in addition to the data transmission. TCP/IP, UDP/IP and UDP_prime bandwidth peaks at approximately 95 megabits per second at a 4,224-byte message, approaching the FDDI peak bandwidth. UDP/IP approaches the peak bandwidth at a 1,400-byte message, and UDP_prime at a 1,024-byte message. Reaching the peak bandwidth using small messages is a measure of protocol processing efficiency.

Figure 7 shows the minimum message-passing latency for TCP/IP, UDP/IP, and UDP_prime transports at different message sizes. The latency was measured as half of the minimum time to send a message and receive the same message looped by the receiver system over many iterations. The measurement made allowance for the allocation and deallocation of each message buffer, in addition to the round-trip transmission.

Compared to the TCP/IP option, UDP/IP has a slightly higher minimum latency. This is not expected, because the original goal of the UDP/IP option was to reduce TCP/IP processing overhead. It is, however, encouraging to see only a slight degradation in latency when the reliable in-order delivery protocol is implemented at the library level. This prompted us to use the same protocol engine in the library for UDP_prime. At a very small message size (4 bytes),

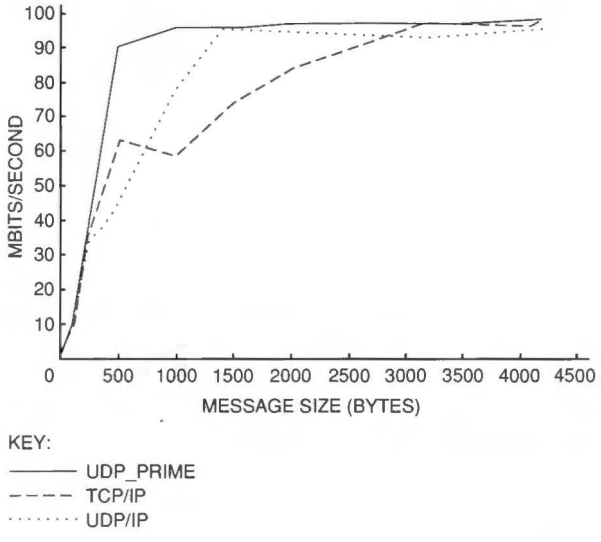


Figure 6
Peer-to-Peer Bandwidth

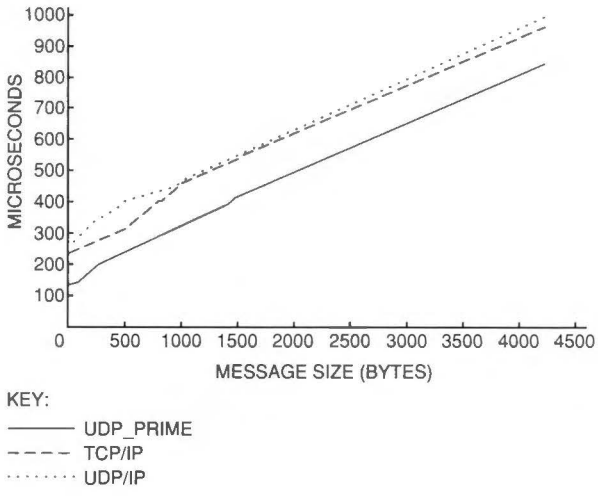


Figure 7
Minimum Peer-to-Peer Latency

protocol processing overhead dominates the latency. At this point, UDP_prime was 44 percent (103.5 microseconds) better than TCP/IP, even though UDP/IP and UDP_prime use the same mechanism.

As the message size increases, the protocol processing time remains constant, but the data copy time becomes dominant. Despite this, UDP_prime was approximately 12 percent better at a 4-kilobyte message.

Future Work

The current communication options along with the UDP_prime optimization provide good performance for HPF-style message passing on SMP systems and clusters. To remain competitive, however, we need to consider support for new high-performance communication media and configurations. We are working on support for MEMORY CHANNEL, the use of multiple interconnects and protocols within an application running on a cluster of SMPs, and lightweight protocols for use with ATM at speeds of 622 megabits per second and higher. The flexibility of the message-passing design will allow current applications to use future communication options without relinking.

We are also working on a new HPF debugger technology. Debugging a cluster-style HPF program is considerably harder than debugging a uniprocessing program. HPF's single-program multiple-data (SPMD) parallel programming model includes a single-threaded control structure, a global name space, and loosely synchronous parallel execution. HPF also supports the calling of extrinsic procedures that use other parallel programming styles or nonparallel computational kernels.

The goal of an HPF debugger is to present the application in source-level terms. Since HPF is roughly Fortran 90 with data-distribution directives, HPF is conceptually a single-threaded application with the compiler transforming pieces of the application to execute in parallel. As a result, an HPF debugger has to take the states from the actual peer processes and recreate a single source-level view of the application. It is not always possible to do this with complete precision. Consider the user interrupting the application, which interrupts the peer processes at different points within the computation. It is unlikely each peer is at the same place (e.g., the same program statement), and it is quite likely that the stack backtraces of the peers differ! Even if they are at the same place, they could be in different iterations of their local portions of a parallelized loop-like operation.

At the start of the HPF debugger project, we surveyed a variety of debuggers and disqualified all of them for logistical and/or technical reasons. Rather than modify an existing debugger technology so that it could debug cluster-style HPF programs, we initiated an effort to build a new debugger technology. As we continue to design the new HPF debugger to be general-purpose, portable, and extensible, we will be able to capitalize on modern programming concepts, paradigms, and techniques.

Summary

PSE contains the tools and execution environment to debug, tune, and deploy parallel applications written in the HPF language. From an end user's perspective, PSE provides transparency, flexibility, and compatibility with familiar tools. Using standard UNIX command syntax, the same executable can be run serially or in parallel on hardware ranging from a single-node system to a cluster of SMP systems. PSE supports several high-performance message-passing protocols running over a variety of network media. From a system administrator's perspective, PSE provides the flexibility to create a cluster from standard components and to control the cluster by assigning access controls and setting scheduling policy and priorities. Although it currently supports only the HPF language, PSE has the flexibility and generic infrastructure to support other parallel languages and programming models.

Acknowledgments

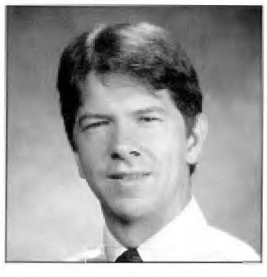
The PSE team would like to thank the members of the Fortran 90 and HPF compiler teams and to acknowledge the contributions of Chuck Wan, Rob Rodon, Phil Cameron, Israel Gale, Rishiyur Nikhil, Marco Annaratone, Bert Halstead, and George Surka.

References

1. *Digital High Performance Fortran 90: HPF and PSE Manual* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-2ATAA-Te, 1995).
2. G. Bell, "Scalable, Parallel Computers: Alternatives, Issues, and Challenges," *International Journal of Parallel Programming,* vol. 22, no. 1 (1994).
3. H. Kung et al., "Network-based Multicomputers: An Emerging Parallel Architecture," *Proceedings Super-Computing 91.*
4. W. Michel, *FDDI: An Introduction to Fiber Distributed Data Interface* (Newton, Mass.: Digital Press, 1992).
5. R. Souza et al., "GIGAswitch System: A High-performance Packet-switching Platform," *Digital Technical Journal,* vol. 6, no. 1 (Winter 1994): 9–22.
6. Internet Engineering Task Force, "Domain Name System," *RFC 883* (November 1983).
7. J. Harris et al., "Compiling High Performance Fortran for Distributed-memory Systems," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 5–23.
8. G. Geist et al., *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing* (Cambridge, Mass.: The MIT Press, 1994).
9. W. Rosenberry, *Understanding DCE* (Sebastopol, Calif.: O'Reilly & Associates, Inc., 1992).
10. W. Gropp et al., *Using MPI: Portable Parallel Programming with the Message Passing Interface* (Cambridge, Mass.: The MIT Press, 1994).
11. *LSF Administrator's Guide* (Toronto, Ont., Canada: Platform Computing Corporation, 1994).
12. G. Champine, *MIT Project Athena: A Model for Distributed Campus Computing* (Newton, Mass.: Digital Press, 1991).
13. W. Stevens, *UNIX Network Programming* (Englewood Cliffs, N.J.: Prentice-Hall, 1990).
14. D. Comer, *Internetworking with TCP/IP* (Englewood Cliffs, N.J.: Prentice-Hall, 1991).
15. *DEC OSF/1 Programmer's Guide* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-PS30C-TE, 1993).
16. R. Sites, ed., *Alpha Architecture Reference Manual* (Burlington, Mass.: Digital Press, Order No. EY-L520E-DP, 1992).
17. Internet Engineering Task Force, "XDR: External Data Representation," *RFC 1014* (June 1987).
18. R. Gillett, "Memory Channel Network for PCI: An Optimized Cluster Interconnect," *Hot Interconnects* (1995).
19. J. Postel, "User Datagram Protocol," *RFC 768* (Menlo Park, Calif.: SRI Network Information Center, 1980).
20. *DEC OSF/1 Reference Pages, Section 2: System Calls* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-PS30C-TE, 1993).
21. J. Postel, "Internet Protocol," *RFC 791* (Menlo Park, Calif.: SRI Network Information Center, 1981).
22. Open Software Foundation, *Design of the OSF/1 Operating System* (Englewood Cliffs, N.J.: Prentice-Hall, 1993).
23. J. Denham, P. Long, and J. Woodward, "DEC OSF/1 Version 3.0 Symmetric Multiprocessing Implementation," *Digital Technical Journal,* vol. 6, no. 3 (Summer 1994): 29–43.

Biographies

Edward G. Benson
Ed Benson is a principal engineer and the project leader for the parallel software environment product. Ed is a 1981 graduate of Tufts University. He joined Digital in 1984 after working at Harvard University and ADAC Corporation. In previous work at Digital, he led the DECmpp and VAXlab software projects and contributed to the design and development of the POSIX real-time extensions in Digital UNIX and OpenVMS.



David C. P. LaFrance-Linden
David LaFrance-Linden is a principal software engineer in Digital's High Performance Fortran Group. Since joining Digital in 1991, he has worked on tools for parallel processing and has developed a promising new debugger technology capable of debugging HPF. He has also contributed to the PSE implementation and compile-time performance of the HPF compiler. Before joining Digital, he worked at Symbolics, Inc. on front-end support, networks, operating system, performance, and CPU architecture. He received a B.S. in mathematics from M.I.T. in 1982.

Richard A. Warren
Richard Warren is a principal software engineer in the High Performance Computing Group, where his primary responsibility is the design and development of Digital's parallel software environment. Since joining Digital in 1977, Richard has contributed to PDP-11 systems development and VAX 32-bit shared-memory multiprocessor designs. He has also been a member of Corporate Research, first as an assignee in parallel processing to the Microelectronics and Computer Technology Corporation (MCC), and later as a researcher at the Digital Joint Project office at CERN, where he helped develop high-availability system software. Richard has a B.S. in electrical and computer engineering from the University of Massachusetts and is a co-inventor on several patents relating to coherent write-back cache design and high-performance bus/memory designs for SMPs.

Santa Wiryaman
A senior software engineer in the High Performance Computing Group, Santa Wiryaman develops enhancements to the Digital UNIX kernel and UDP/IP protocol stack to support optimal performance of message passing over FDDI and ATM networks. Since joining Digital's performance group in 1987, he has also contributed to many network-related performance characterizations, benchmarks, and the development of performance tools for UNIX and Windows NT. Santa received B.S. (1985) and M.S. (1987) degrees in computer science from Cornell University and Rensselaer Polytechnic Institute, respectively.

Michael Stonebraker

An Overview of the Sequoia 2000 Project

The Sequoia 2000 project is the joint effort of computer scientists, earth scientists, government agencies, and industry partners to build a better computing environment for global change researchers. The objectives of this widely distributed project are to support high-performance I/O on terabyte data sets, to put all data in a database management system, and to provide improved visualization tools and high-speed networking. The participants developed a four-level architecture to meet these objectives. Chief among the lessons learned is that the Sequoia 2000 system must be considered an end-to-end solution, with all pieces of the architecture working together. This paper describes the Sequoia 2000 project and its implementation efforts during the first three years. The research was sponsored by Digital Equipment Corporation.

The purpose of the Sequoia 2000 project is to build a better computing environment for global change researchers, hereafter referred to as Sequoia 2000 clients. These researchers investigate issues such as global warming, ozone depletion, environment toxification, and species extinction and are members of earth science departments at universities and national laboratories. A more detailed conception for the project appears in the Sequoia 2000 technical report "Large Capacity Object Servers to Support Global Change Research."[1]

The participants in the Sequoia 2000 project are investigators of four types: (1) computer science researchers, (2) earth science researchers, (3) government agencies, and (4) industry partners.

Computer science researchers are responsible for building a prototype environment that better serves the needs of the target clients. Participating in the Sequoia 2000 project are investigators from the Computer Science Division at the University of California, Berkeley; the Computer Science Department at the University of California, San Diego; the School of Library and Information Studies at the University of California, Berkeley; and the San Diego Supercomputer Center.

Earth science researchers must explain their needs to the computer science investigators and use the resulting prototype environment to perform better earth science research. The Sequoia 2000 project comprises earth science investigators from the Department of Geography at the University of California, Santa Barbara; the Atmospheric Science Department at the University of California, Los Angeles (UCLA); the Climate Research Division at the Scripps Institution of Oceanography; and the Department of Earth, Air, and Water at the University of California, Davis.

To ensure that the resulting computer environment addresses the needs of the Sequoia 2000 clients, government agencies that are affected by global change matters participate in the project. The responsibility of these agencies is to steer Sequoia 2000 research toward achieving solutions to their problems. The government agencies that participate are the State of California Department of Water Resources (DWR),

the State of California Department of Forestry, the Coordinated Environment Research Laboratory (CERL) of the United States Army, the National Aeronautics and Space Administration (NASA), the National Oceanic and Atmospheric Administration (NOAA), and the United States Geologic Survey (USGS).

The task of the industry participants is to use the Sequoia 2000 technology and to offer guidance and research direction. In addition, they are a source of free or discounted computing equipment. Digital Equipment Corporation was the original industry partner. Recently, Epoch Systems, Hewlett-Packard, Hughes, Illustra, MCI, Metrum Systems, PictureTel, RSI, SAIC, Siemens, and TRW have become participants.

The purpose of this paper is to present the goals of the Sequoia 2000 project and to discuss how we achieved these goals and the results we accomplished during the first three years. The paper describes the architecture that we decided to pursue and the state of the software efforts in the various areas. The most important lesson we have learned is that the Sequoia 2000 system must be considered an end-to-end solution. Hence, clients can be satisfied only if all pieces of the architecture work together in a harmonious fashion. Also, many services required by the clients must be provided by every element of the architecture, each working with the others. We illustrate this end-to-end characteristic of Sequoia 2000 by discussing three issues that cross all parts of the system: guaranteed delivery, abstracts, and compression. We then indicate other specific lessons that we learned during the first three years of the project. The paper concludes with the current state of the project and its future directions.

The Sequoia 2000 Architecture

The Sequoia 2000 architecture is motivated by four fundamental computer science objectives:

1. *Support high-performance I/O on terabyte data sets.* The Sequoia 2000 clients are frustrated by current computing environments because they cannot effectively store the massive amounts of data desired for research purposes. The four academic clients plus DWR collectively want to be able to store approximately 100 terabytes of information, much of which is common data sets used by multiple investigators. These clients would like high-performance system software that would allow sharing of assorted tertiary memory devices. Unlike the I/O activities of most other scientific computing users, their activity involves primarily random access. For example, DWR is digitizing the agency's library of 500,000 slides and is putting it on-line using the Sequoia 2000 system. This data set has some locality of reference but will have considerable random activity.
2. *Put all data in a database management system (DBMS).* To maintain the metadata that describe their data sets and thus aid in the retrieval of information, the Sequoia 2000 clients want to move all their data to a DBMS. More important, using a DBMS will facilitate the sharing of information. Because a DBMS insists on a common schema for shared information, it will allow the researchers to define a schema. Then all researchers must use a common notation for shared data. Such a system will be a big improvement over the current situation where every data set exists in a unique format and must be converted by every researcher who wishes to use it.
3. *Provide improved visualization tools.* Sequoia 2000 clients use popular scientific visualization tools such as Explorer, Khoros, AVS, and IDL and are eager to use a next-generation toolkit.
4. *Provide high-speed networking.* Sequoia 2000 clients realize that a 100-terabyte storage server (or 100-terabyte servers) will not be located on each of their desktops. Moreover, the storage is likely to be located at the other end of a wide area network (WAN), far from their client machines. Since the clients' visualization scenarios invariably involve animation, for example, showing the last 10 years of the ozone hole by playing time forward, the clients require ultrahigh-speed networking to move sequences of images from a server machine to a client machine.

To meet these objectives, we adopted the four-level architecture illustrated in Figure 1. The architecture comprises the footprint layer, the file system layer, the DBMS layer, and the application layer. This section discusses our efforts at each of the levels and then concludes with a discussion of the Sequoia 2000 networking that connects the elements of the architecture.

The Footprint Layer

The footprint layer is a software system that shields higher-level software, such as file systems, from device-specific characteristics of robotic devices. These characteristics include specific robot commands, block sizes, and media-specific issues. The footprint layer can be thought of as a common robot device driver. A footprint implementation exists for each of the four tertiary memory devices used by the project, namely, a Sony write once, read many (WORM) optical disk jukebox, an HP rewritable optical disk jukebox, a Metrum VHS tape jukebox, and an Exabyte 8-millimeter tape jukebox. Collectively, these four devices and the CPUs and disk storage systems in front of them were named Bigfoot, after the legendary, very tall recluse spotted occasionally in the Pacific Northwest.

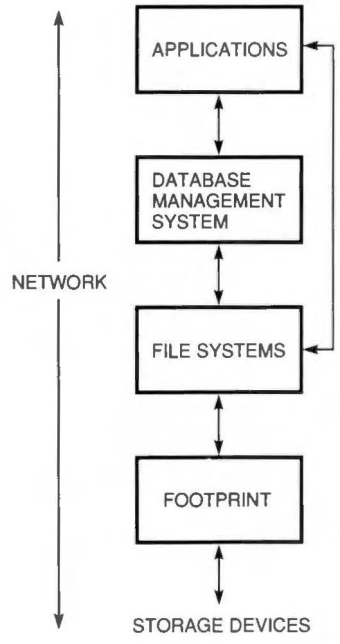


Figure 1
The Sequoia 2000 Architecture

The File System Layer

On top of the footprint layer is the file system layer. Two file systems manage data in the Bigfoot multilevel memory hierarchy. The first file system is Highlight, which extends the Log-structured File System (LFS) pioneered for disk devices by Ousterhout and Rosenblum to tertiary memory.[2,3] The original LFS treats a disk device as a single continuous log onto which newly written disk blocks are appended. Blocks are never overwritten, so a disk device can always be written sequentially. Hence, the LFS turns a random-write environment into a sequential-write environment. In particular problem areas, this may lead to much higher performance. Benchmark data support this conclusion.[4] In addition, the LFS can always identify the last few blocks that were written prior to a file system failure by finding the end of the log at recovery time. File system repair is then very fast, because potentially damaged blocks are easily found. This approach differs from conventional file system repair, where a laborious check of the disk must be performed to ascertain disk integrity.

Highlight extends the LFS to support tertiary memory by adding a second log-structured file system on top of the footprint layer. This file system also writes tertiary memory blocks sequentially, thereby obtaining the performance characteristics of the LFS. The Highlight file system adds migration and bookkeeping code that treats the disk LFS file system as a cache for the tertiary memory file system. In summary, Highlight should provide good performance for workloads that consist of mainly write operations. Since Sequoia 2000 clients want to archive vast amounts of data, the Highlight file system has the potential for good performance in the Sequoia 2000 environment.

The second file system is Inversion.[5] Most DBMSs, including the one used for the Sequoia 2000 project, support binary large objects (BLOBs), which are arbitrary-length byte strings of variable length. Like several commercial systems, Sequoia's data manager POSTGRES stores large objects in a customized storage system directly on a raw storage device.[6] As a result, it is a straightforward exercise to support conventional files on top of DBMS large objects. In this way, the front end turns every read or write operation into a query or an update, which is processed directly by the DBMS. Simulating files on top of DBMS large objects has several advantages. First, DBMS services such as transaction management and security are automatically supported for files. In addition, novel characteristics of our next-generation DBMS, including time travel and an extensible type system for all DBMS objects, are automatically available for files. Of course, the possible disadvantage of simulating files on top of a DBMS is poor performance. As reported by Olson, Inversion performance is exceedingly good when large blocks of data are read and written, as is characteristic of the Sequoia 2000 workload.[5]

At the present time, Highlight is operational but very buggy. Inversion, on the other hand, is used to manage production data on Sequoia's Sony WORM jukebox. Unfortunately, the reliability of the prototype system has not met user expectations. Sequoia 2000 clients have a strong desire for commercial off-the-shelf (COTS) software and are frustrated by documentation glitches, bugs, and crashes.

As a result, the Sequoia 2000 project team has also deployed two commercial file systems, Epoch and AMASS. The Epoch file system is quite reliable but does not support either of Sequoia's large-capacity robots. Hence, it is used heavily but only for small data sets. The AMASS file system is just coming into production use for Sequoia's Metrum robot and replaces an earlier COTS system, which was unreliable. Given the experience of the Sequoia 2000 team with tertiary memory support, tertiary memory users should carefully test all file system software.

The DBMS Layer

To meet Sequoia 2000 client needs, a DBMS must support spatial data such as points, lines, and polygons. In addition, the DBMS must support the large spatial arrays in which satellite imagery is naturally stored. These characteristics are not met by popular, general-purpose relational and object-oriented DBMSs.[7] The best fit to client needs is a special-purpose Geographic Information System (GIS) or a next-generation object-relational DBMS. Since it has one such object-relational system, namely

POSTGRES, the Sequoia 2000 project elected to focus its DBMS efforts on this system.

To make the POSTGRES DBMS suitable for Sequoia 2000 use, we require a schema for all Sequoia data. This database design process has evolved as a cooperative exercise between various database experts at Berkeley, the San Diego Supercomputer Center, CERL, and SAIC. The Sequoia schema is the collection of metadata that describes the data stored in the POSTGRES DBMS on Bigfoot. Specifically, these metadata comprise

- A standard vocabulary of terms with agreed-upon definitions that are used to describe the data
- A set of types, instances of which may store data values
- A hierarchical collection of classes that describe aggregations of the basic types
- Functions defined on the types and classes

The Sequoia 2000 schema accommodates four broad categories of data: scalar, vector, raster, and text. Scalar quantities are stored as POSTGRES types and assembled into classes in the usual way. Vector quantities are stored in special line and polygon types. Vectors are fully enumerated (as opposed to an arc-node representation) to take advantage of POSTGRES indexed searches. The advantages of this representation are discussed in more detail in "The Sequoia 2000 Benchmark."[7]

Raster data constitute the bulk of the Sequoia 2000 data. These data are stored in POSTGRES multi-dimensional arrays objects. The contents of textual objects (in PostScript or scanned page bitmaps) are stored in a POSTGRES document type. Both documents and arrays make use of a POSTGRES large object storage manager that can support arbitrary-length objects.

We have tuned the POSTGRES DBMS to meet the needs of the Sequoia 2000 clients. The interface to POSTGRES arrays has been improved, and a novel chunking strategy is now operational.[8] Instead of storing an array by ordering the array indexes from fastest changing to slowest changing, this system chooses a stride for each dimension and stores chunks of the correct stride sizes in each storage object. When user queries inspect the array in more than one way, this technique results in dramatically superior retrieval performance.

Sequoia 2000 clients typically run queries with user-defined functions in the predicate. Moreover, many of the predicates are very expensive in CPU time to compute. For example, the Santa Barbara group has written a function, SNOW, that recognizes the snow-covered regions in a satellite image. It is a user-defined POSTGRES function that accepts an image as an argument and returns a collection of polygons. A typical query using the SNOW function for the table IMAGES (id, date, content) would be to find the images that were more than 50 percent snow and that were observed subsequent to June 1992. In SQL, this query is expressed as follows:

```
select id
from IMAGES
where AREA (SNOW (content)) > 0.5
and date > "June 1, 1992"
```

The first clause in the predicate requires the CPU to evaluate millions of instructions, whereas the second clause requires only a few hundred instructions. The DBMS must be cognizant of the CPU cost of clauses when constructing a query plan, a cost component that has been ignored by most previous optimization work. We have extended the POSTGRES optimizer to deal intelligently with expensive functions.[9]

It is highly desirable to allow popular expensive functions to be precomputed. In this way, the CPU need only evaluate each such function once, rather than once for each query in which the function appears. Our approach to this issue is to allow databases to contain indexes on a function of the data and not on just the data object itself. Hence, the database administrator can specify that a B-tree index be built for the function AREA (SNOW(content)). Areas of images are arranged in sort order in a B-tree, so the first clause in the above query is now very inexpensive to compute. Using this technique, the function is computed at data entry or data update time and not at query evaluation time. A consequence of function indexing is that inserting a new image into the database may be very time-consuming, since function computation is now included in the load transaction. To deal with the undesirable lengthy response times for some loads, we have also explored lazy indexing and partial indexing. Thus, index building does not need to be synchronous with data loading.

The feedback from the Sequoia 2000 clients regarding POSTGRES is that it is not reliable enough to serve as a base for production work. Moreover, the documentation is inadequate, and no facility exists to train users. Our users want a COTS product and not a research prototype. Consequently, the Sequoia 2000 project has migrated to the commercial version of POSTGRES, namely the Illustra system, to obtain a COTS DBMS product. Migration to this system required reloading all project data, a task that is now nearly complete.

The Application Layer

The application layer of the Sequoia 2000 architecture contains five elements:

1. An off-the-shelf visualization tool
2. A visualization environment

3. A browsing capability for textual information
4. A facility to interface the UCLA General Circulation Model (GCM) to the POSTGRES/Illustra system
5. A desktop videoconferencing or "picturephone" facility

For the off-the-shelf visualization tool, we have converged around the use of AVS and IDL for project activities. AVS has an easy-to-use "boxes-and-arrows" user interface, whereas IDL has a more conventional linear programming notation. On the other hand, IDL has better two-dimensional (2-D) graphics features. Both AVS and IDL allow the user to read and write file data. To connect to the DBMS, we have written an AVS–POSTGRES bridge. This program allows the user to construct an ad hoc POSTGRES query and pipe the result into an AVS boxes-and-arrows network. Sequoia 2000 clients can use AVS for further processing on any data retrieved from the DBMS. IDL is being interfaced to AVS by the vendor. Consequently, data retrieved from the database can be moved into IDL using AVS as an intermediary. Now that we have migrated to the Illustra DBMS, we are considering porting this AVS bridge to the Illustra application programming interface (API).

AVS has some disadvantages as a visualization tool for Sequoia 2000 clients. First, its type system, which is different from the POSTGRES/Illustra type system, has no direct knowledge of the common Sequoia 2000 schema. In addition, AVS consumes significant amounts of main memory. Architecturally, AVS depends on virtual memory to pass results between various boxes. It also maintains the output of each box in virtual memory for the duration of an execution session. The user can thus change a run-time parameter somewhere in the network, and AVS will recompute only the downstream boxes by taking advantage of the previous output. As a result, Sequoia 2000 clients, who generally produce very large intermediate results, consume large amounts of both virtual and real memory. In fact, clients report that 64 megabytes of real memory on a workstation is often not enough to enable serious AVS use. Furthermore, AVS does not support zooming in to investigate data of interest to obtain higher resolution, nor does it keep track of the history of how any given data element was constructed, i.e., the so-called data lineage of an item. Lastly, AVS has a video player model for animation that is too primitive for many Sequoia 2000 clients.

Consequently, we have designed two new visualization environments. The first system, called Tecate, is being built at the San Diego Supercomputer Center. The Tecate infrastructure enables the creation of applications that allow end users to browse for and visualize data from networked data sources. This software platform capitalizes on the architectural strengths of current scientific visualization systems, network browsers, database management system front ends, and virtual reality systems, as discussed in a companion paper in this issue of the *Journal*.[10]

The other system, Tioga, is a boxes-and-arrows programming environment that is DBMS-centric, i.e., the environment type system is the same as the DBMS type system. The Tioga user interface gives the user a flight simulator paradigm for browsing the output of a network. In this way, the visualizer can navigate around data and then zoom in to obtain additional data on items of particular interest. The preliminary Tioga design was presented at the 1993 Very Large Databases Conference.[11] A first prototype, described by Woodruff, is currently running.[12]

A commercial version of the Tioga environment has also been implemented by Illustra. The Sequoia 2000 project is making considerable use of this tool, which is named Object-Knowledge. Early user experience with both Tioga and Object-Knowledge indicates that these systems are not easy to use. We are now exploring ways to improve the Tioga system. The objective is to build a system that a scientist with minimal training in the environment can use without a reference manual.

The third element of the application layer is a browsing capability for textual information of interest to our clients. This capability is a cornerstone of the Sequoia 2000 architecture. Initially, we converted a stand-alone text retrieval system called Lassen to our DBMS-centric view. The first part of the Lassen system is a facility for constructing weighted keyword indexes for the words in a POSTGRES document. This indexing system, Cheshire, builds on the pioneering work of the Cornell Smart system and operates as the action part of a POSTGRES rule, which is triggered on each document insertion, update, or removal.[1,13] The second part of the Lassen system is a front-end query tool that understands natural language. This tool allows a user to request all documents that satisfy a collection of keywords by using a natural language interface. The Lassen system has been operational for more than a year, and retrievals can be requested against the currently loaded collection of Sequoia 2000 documents.

In addition, we have moved Lassen to Z39.50, a popular protocol oriented toward information interchange and information retrieval.[14] The client portion of Lassen has been changed to emit Z39.50, and we have written a Z39.50-to-POSTGRES translator on the server side. In this way, the Lassen client code can access non–Sequoia 2000 information and the Sequoia 2000 server can be accessed by text-retrieval front ends other than the Cheshire system.

With our move to the Illustra DBMS, we have converted the client side of Lassen to work with Illustra.

Illustra has an integrated document data type with capabilities similar to the extensions we made to POSTGRES.

A related Berkeley project is focused on digitizing all the Berkeley Computer Science Technical Reports. This project uses a Mosaic client to access a custom World Wide Web server called Dienst, which stores technical report objects in a UNIX file system. In a few months, we expect to convert Dienst to store objects in the Sequoia 2000 database, rather than in files. When this system, nicknamed Database Dienst, is operational, Mosaic/Dienst service will be available for all textual objects in the Sequoia schema.

Our fourth thrust in the application layer is a facility to interface the UCLA General Circulation Model (GCM) to the POSTGRES/Illustra system. This program is a "data pump" because it pumps data out of the simulation model and into the DBMS. We named the program "the big lift" after the DWR pumping station that raises Northern California water over the Tehachapi Mountains into Southern California.

Basically, the UCLA GCM produces a vector of simulation output variables for each time step of a lengthy run for each tile in a three-dimensional (3-D) grid of the atmosphere and ocean. Depending on the scale of the model, its resolution, and the capability of the serial or parallel machine on which the model is running, the UCLA GCM can produce from 0.1 to 10.0 megabytes per second (MB/s) output. The purpose of the big lift is to install the output data into a database in real time. UCLA scientists can then use ObjectKnowledge, Tioga, Tecate, AVS, or IDL to visualize their simulation output. The big lift will likely have to exploit parallelism in the data manager, if it is required to keep up with the execution of the model on a massively parallel architecture.

The fifth application system is a conferencing system. Since Sequoia 2000 is a distributed project, we learned early that face-to-face meetings that required participants to travel to other sites and electronic mail were not sufficient to keep project members working as a team. Consequently, we purchased conference room videoconferencing equipment for each project site. This technology costs approximately $50,000 per site and allows multiway videoconferences over integrated services digital network (ISDN) lines.

Although the conference room equipment has helped project communication immensely, it must be set up and taken down at each use because the rooms it occupies at the various sites are normally used as classrooms. Therefore, videoconferencing tends to be used for arranged conferences and not for spur-of-the-moment interactions. To alleviate this shortcoming, Sequoia 2000 has also invested in desktop videoconferencing. A video compression board, a microphone, speakers, a network connection, a video camera, and the appropriate software can turn a conventional workstation into a desktop videoconferencing facility. In addition, video can be easily transmitted over the network interface that is present in virtually all Sequoia 2000 client machines. We are using the Mbone software suite to connect about 30 of our client machines in this fashion and are migrating most of our videoconferencing activities to desktop technology. This effort, which is called Hollywood, strives to further improve the ability of Sequoia 2000 researchers to communicate.

Note that the Sequoia 2000 researchers do not need groupware, i.e., the ability to have common windows on multiple client machines separated by a WAN, in which common code can be run, updated, and inspected. Rather, our researchers need a way to hold impromptu discussions on project business. They want a low-cost multicast picturephone capability, and our desktop videoconferencing efforts are focused in this direction.

Sequoia 2000 Networking

The last topic of this section on the Sequoia 2000 architecture is the networking agenda. Regarding Figure 1, it is possible for the implementation of each layer to exist on a different machine. Specifically, the application can be remote from the DBMS, which can be remote from the file system, which can be remote from the storage device. Each layer of the Sequoia 2000 architecture assumes a local UNIX socket connection or a local area network (LAN) or WAN connection using the transmission control protocol/internet protocol (TCP/IP). Actual connections among Sequoia 2000 sites use either the Internet or a dedicated T3 network, which the University of California provides as part of its contribution to the project.

The networking team judged Digital's Alpha processors to be fast enough to route T3 packets. Hence, the project uses conventional workstations as routers; custom machines are not required. Furthermore, the Sequoia 2000 network has installed a unique guaranteed delivery service through which an application can make a contract with the network that will guarantee a specific bandwidth and latency if the client sends information at a rate that does not exceed the rate specified in the contract. These algorithms, which are based on the work of Ferrari, require a setup phase for a connection that allocates bandwidth on all the lines and in all the switches.[15]

Lastly, the network researchers are concerned that the Digital UNIX (formerly DEC OSF/1) operating system copies every byte four times in between retrieving it from the disk and sending it out over a network connection. The efficient integration of networking services into the operating system is the topic of a companion paper by Pasquale et al. in this issue.[16]

Sequoia 2000 as an End-to-End Problem

The major lesson we have learned from the Sequoia 2000 project is that many issues facing our clients cannot be isolated to a single layer of the Sequoia 2000 architecture. This section describes three such end-to-end problems: guaranteed delivery, abstracts, and compression.

Guaranteed Delivery

Clearly, guaranteed delivery must be an end-to-end contract. Suppose a Sequoia 2000 client wishes to visualize a specific computation; for example, the client wants to observe Hurricane Andrew as it moves from the Bahamas to Florida to Louisiana. Specifically, the client wishes to visualize appropriate satellite imagery at a resolution of 500 × 500 in 8-bit color at 10 frames per second. Hence, the client requires 2.5 MB/s of bandwidth to his screen. The following scenario might be the computation steps that take place.

The DBMS must run a query to fetch the satellite imagery. The query might require returning a 16-bit data value for each pixel that will ultimately appear on the screen. The DBMS must therefore agree to execute the query in such a way that it guarantees output at a rate of 5.0 MB/s.

The storage system at the server will fetch some number of I/O blocks from secondary and/or tertiary memory. DBMS query optimizers can accurately guess how many blocks they need to read to satisfy the query. The DBMS can then easily generate a guaranteed delivery contract that the storage manager must satisfy, thus allowing the DBMS to satisfy its contract.

The network must agree to deliver 5.0 MB/s over the network link that connects the client to the server. The Sequoia 2000 network software expects exactly this type of contract request.

The visualization package must agree to translate the 16-bit data element into an 8-bit color and render the result onto the screen at 2.5 MB/s.

In short, guaranteed delivery is a collection of contracts that must be adhered to by the DBMS, the storage system, the network, and the visualization package. One approach to architecting these contracts was presented at the 1993 Very Large Databases Conference.[11]

Abstracts

One aspect of the Sequoia 2000 visualization process is the necessity of abstracts. Consider the Hurricane Andrew example. The client might initially want to browse the hurricane at 100 × 100 resolution. Then, on finding something of interest, the client would probably like to zoom in and increase the resolution, usually to the maximum available in the original data. This ability to dynamically change the amount of resolution in an image is supported by abstracts.

Note that providing abstracts is a much more powerful construct than merely providing for resolution adjustment. Specifically, obtaining more detail may entail moving from one representation to another. For example, one could have an icon for a document, zoom in to see the abstract, and then zoom in further to see the entire document. Hence, zooming can change from iconic to textual representation. This use of abstracts was popularized in the DBMS community by an early DBMS visualization system called the Spatial Data Management System (SDMS).[17]

Sequoia 2000 clients wish to have abstracts; however, it is clear that they can be managed by the visualization tool, the DBMS, the network, or the file system. In the former case, abstracts are defined for boxes-and-arrows networks.[11] In the DBMS, abstracts would be defined for individual data elements or for data classes. If the network manages abstracts, it will use them to automatically lower resolution to eliminate congestion. Much research on the optimization of network abstracts (called hierarchical encoding of data in that community) is available. In the file system, abstracts would be defined for files. Sequoia 2000 researchers are pursuing all four possibilities, and it is expected that this notion will be one of the powerful constructs to be used by Sequoia 2000 software, perhaps in multiple ways.

Compression

The Sequoia 2000 clients are adamant on the issue of compression—they are open to any compression scheme as long as it is lossless. As scientists, they believe that ultimate resolution may be required to understand future phenomena. Since it is not possible to predict what these phenomena might be or where they might occur, the Sequoia 2000 scientists want access to all data at full resolution.

Some Sequoia 2000 data cannot be compressed economically and should be stored in uncompressed form. The inclusion of abstracts offers a mechanism to lower the bandwidth required between the storage device and the visualization program. No saving of tertiary memory through compression is available for such data.

Other data ought to be stored in compressed form. The question of when compression and decompression should occur can be handled by using a just-in-time decompression strategy. For example, if the storage manager compresses data as they are written and then decompresses them on a read operation, the network manager may then recompress the data for transmission over a WAN to a remote site where they will be decompressed a second time. Obviously, data should be moved in compressed form and decompressed only when necessary. In general, decompression will occur in the visualization system on the client machine. If search criteria are performed on the data,

then the DBMS may have to decompress the data to perform the search. If an application resides on the same machine as the storage manager, the file system must be in charge of decompressing the data. All software modules in the Sequoia 2000 architecture must cooperate to perform just-in-time decompression and as-early-as-possible compression. Like guaranteed delivery, compression is a task that requires all software modules to cooperate.

Specific Lessons Learned

In addition to the end-to-end issues, we learned other lessons from the first three years of the Sequoia 2000 experience, as discussed in this section.

Lesson 1: Infrastructure is necessary, time-consuming, and very expensive.

We learned early in the project that electronic mail and travel between sites would not result in the desired degree of cooperation from geographically dispersed researchers from different disciplines. Consequently, we made a significant investment in infrastructure. This included meetings for all the Sequoia 2000 participants, which are now held twice a year, and videoconferencing equipment at each site. Through this video link, project members interact by holding a weekly distributed seminar, semimonthly operations committee meetings, occasional steering committee meetings, and meetings between researchers with common interests. The video quality of the project's current videoconferencing equipment is not high, and to achieve success when participants are located far apart, specially trained individuals must operate the equipment. Nevertheless, the equipment has proven to be valuable in generating cohesion in the dispersed project. We have installed desktop videoconferencing systems on 30 Sequoia 2000 workstations and expect to replace our current conference room equipment with next-generation desktop technology.

In addition, we conducted a learning experiment in which a course taught by one of the Sequoia 2000 faculty members at the Santa Barbara campus was broadcast over our videoconferencing equipment to four other sites. Students could take the course for credit at their respective campuses. Of course, the overhead of setting up such a course was substantial. A new course had to be added at each campus, and every step in the approval process required briefings on the fact that the instructor was from a different campus and on the way everything was going to work. This experiment was popular, and students have requested additional courses taught in this manner.

On the other hand, we also tried to run a computer science colloquium using this technology. We broadcast from various sites to six computer science departments around the U.S. Initial student interest was high because of the lineup of eminent speakers. Such speakers could be recruited easily, because they only had to locate the nearest compatible equipment and then get to that site. No air travel was required. The experiment failed, however, because attendance decreased throughout the semester and ended at an extremely low level.

The basic problem was that, typically, speakers were not skilled in using the medium—they would put too much information on slides and then flip though the slides before remote sites could get a complete transmission. Also, the question-and-answer period could not be very interactive because of the many sites involved. The experiment ended after one semester and will not be repeated.

Lesson 2: There was often a mismatch between the expectations of the earth scientists and those of the computer scientists.

The computer scientists on the Sequoia 2000 team wanted access to knowledgeable application specialists who could describe their problems in terms understandable to the computer scientist. The computer scientists then wanted to think through elegant solutions, verify with the earth scientists that the solutions were appropriate, and then prototype the results. The earth scientists wanted final COTS solutions to their problems; they were unsympathetic about poor documentation, bugs, and crashes.

With considerable effort, the expectations are converging. The ultimate solution is to move to COTS software modules as they become available for portions of the system and augment the modules with in-house prototype code.

We have found that the best way to make forward progress was to ensure that each earth science group using Sequoia 2000 prototype code had one or more sophisticated staff programmers who could deal successfully with the quirks of prototype code. With computer science expertise surrounding the earth scientists, the problems in this area became much more manageable. We also discovered that we could distribute such expertise. In fact, support programmers for Sequoia 2000 clients are often not at the same physical location as the client.

Lesson 3: Interdisciplinary research is fundamentally difficult.

One lengthy discussion on the construction of a Sequoia 2000 benchmark eventually led to the discussion presented in the 1993 ACM SIGMOD conference paper entitled "The Sequoia 2000 Benchmark," which we referred to previously.[7] The computer science researchers were arguing strongly for a representative abstract example of earth science data access, i.e., the "specmark of earth science." On the other hand, the earth scientists were equally adamant that the benchmark convey the exact data accesses.

Finally, the computer scientists and the earth scientists realized that the word "benchmark" has a different meaning for each of the two groups of researchers. To earth scientists, a benchmark is a scenario, whereas to computer scientists, a benchmark is an abstract example. This vignette was typical of the experience these two disciplines had trying to understand one another. Fundamentally, this process is time-consuming, and ample interaction time should be planned for any project that must deal with multiple disciplines.

The Sequoia 2000 project participants made effective use of "converters." A converter is a person of one discipline who is planted directly in the research group of another discipline. Through informal communication, this person serves as an interpreter and translator for the other discipline. Converters are encouraged by the existence of a formal exchange program, whereby central Sequoia 2000 resources pay the living expenses of the exchange personnel.

Lesson 4: Database technology is a major advance for earth scientists.

Our initial plan was to introduce database technology into the project with the expectation that the earth scientists would pick it up and use it. Unfortunately, they are accustomed to data being in files and found it very difficult to make the transition to a database view. The earth scientists are becoming increasingly aware of the inherent advantages of DBMS technology.

In addition, we appointed the earth scientist with the most computer science knowledge as leader of the database design effort. This person chaired a committee of mainly computer scientists who were charged with producing a schema.

This technique failed for several reasons. First, the computer scientists disagreed about whether we were designing an interchange format, by which sites could reliably exchange data sets (i.e., an on-the-wire representation), or a schema for stored data at a site. Most earth science standards, such as the Hierarchical Data Format (HDF) and the network Common Data Form (netCDF), are of the first form, and there was substantial enthusiasm for simply choosing one of these formats.[18,19] On the other hand, some computer scientists argued that an on-the-wire representation mixes the data (e.g., a satellite image) and the metadata that describe it (e.g., the frequency of the sensor, the date of the data collection, and the name of the satellite) into a single, highly encoded bit string. A better design would separate the two kinds of data and construct a good stored schema for it.

A second problem was that numerous legacy formats are currently in use, and some earth scientists did not want to change the formats they were using. This led to many arguments about the merits of one legacy format over another, which in turn caused the opposing sides to conclude that both formats under discussion should be supported in addition to a neutral representation.

A third problem was that earth science data are fundamentally quite complex. For example, earth scientists store geographic points, which are 3-D positions on the earth's surface. There are approximately 20 popular projections of 3-D space onto 2-D space, including (latitude, longitude), Mercator projection, and Lambert Equal Azimuthal projection. With every instance of a geographic point, it is necessary to associate the projection system that is being used. Another data problem is related to units. Some geographic data are represented as integers, with miles as the fundamental unit; other data are represented as floating-point numbers, with meters as the underlying unit. In addition, satellite imagery must be massaged in a variety of ways to "cook" it from raw data into a usable form. Cooking includes converting imagery from a one-dimensional stream of data recorded in satellite flight order into a 2-D representation. Many details of this cooking process must be recorded for all imagery. This dramatically expands the metadata about imagery as well as forces the earth scientist to write down all the extra data elements.

Schema design turned out to be laborious and very difficult. The earth scientists did not have a good understanding of database design and thus were not prepared to take on the extreme complexity of the task. As a result, we have reconstructed our database design effort. Now, two computer scientists are responsible for producing a schema. They interact with the earth scientists when such action helps to accomplish the task.

Lesson 5: Project management is a substantial problem.

Sequoia 2000 is a large project. About 110 people attended the last general meeting. The attendees included approximately 30 computer scientists, 40 earth scientists, and 40 visitors from industry. Multiple efforts on multiple campuses must "plug and play." Synchronizing distributed development is an extreme challenge. Furthermore, the skill of project management is not fostered in a university environment, nor is it rewarded in a university faculty evaluation.

The principal investigators viewed the time spent on project management as time that could be better invested in research activities. An obvious solution would be for the Sequoia 2000 project to hire a professional project manager. Unfortunately, it is impossible to pay a nonfaculty person the market rates normally received by such skilled persons. One strategy we attempted to use was to solicit a visitor with the desired skill mix from one of our industrial sponsors. Our efforts in this direction failed, and we were never able to recruit project management expertise for

the Sequoia 2000 effort. As a result, project management was performed poorly at best. In any future large project, this component should be addressed satisfactorily up front by project personnel.

Lesson 6: Multicampus projects are extremely difficult to implement.

Sequoia 2000 work is taking place in seven different organizations within the University of California educational system. There is a constant need to transfer money and people among these organizations. Accomplishing such moves is a difficult and slow process, however, because of the bureaucracy within the system. In addition, the personnel rules of the University are often in conflict with the needs of the Sequoia 2000 project. As a result, multi-institution projects, where participants are in different and often distant locations, are extremely difficult to carry out.

Status and Future Plans

The Sequoia 2000 project is more than three years old and has nearly accomplished its objectives. We have a common schema in place for all Santa Barbara and UCLA data, and all participants have agreed to use the schema. This schema serves as leverage for the standards efforts under way in the spatial arena.[20] The infrastructure is in place to enable this schema to evolve as more data types, user-defined functions, and operators are included in the future.

The combination of Object-Knowledge, Illustra, Epoch, and AMASS is proving robust and meets our clients' needs. Lastly, we have ample resources to move our prototype into production use at UCLA and Santa Barbara during the next several months.

We are also extending the scope of the prototype in two different directions. First, we will recruit additional earth scientists to utilize our system. This will require extending our common schema to meet their needs and then installing our suite of software at their site. We expect to recruit two to three new groups during the next year.

Second, a companion project, the Electronic Repository, has as one of its objectives to use the Sequoia 2000 technology to support an environmental digital library of aerial photography, polygonal data, and text for the Resources Agency of the State of California.[21] This electronic library project is extending the reach of Sequoia 2000 technology from earth scientists toward a broader community.

Our research activities are also very active. As noted earlier, we are continuing our visualization activities and anticipate an improved Tioga system. The Sequoia 2000 clients have made it clear that they want seamless access to distributed data, and we have evolved POSTGRES to a wide-area distributed DBMS that makes decisions based on an economic paradigm. This system is called Mariposa.[22] In our COTS system, a bad impedance mismatch exists between the DBMS and the tertiary memory file systems. We have therefore shifted our research focus to constructing an intelligent mass storage interface that properly supports a DBMS.

Finally, the Sequoia 2000 network currently supports service guarantees, but there is no economic framework in which to place multiple levels of service. As a result, our networking research is focused on construction of this type of framework.

We anticipate a robust production environment for earth science researchers by the end of 1995. In addition, we expect to continue to improve the Sequoia 2000 environment with future research results in the above areas.

References and Notes

1. M. Stonebraker and J. Dozier, "Large Capacity Object Servers to Support Global Change Research," Sequoia 2000 Technical Report 91/1, Berkeley, California (July 1991).
2. J. Kohl et al., "Highlight: Using a Log-structured File System for Tertiary Storage Management," *Proceedings of the 1993 Winter USENIX Meeting,* San Diego, California (January 1993).
3. M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-structured File System," *ACM Transactions on Computing Systems (TOCS)* (February 1992).
4. M. Seltzer et al., "An Implementation of a Log-structured File System for UNIX," *Proceedings of the 1993 Winter USENIX Meeting,* San Diego, California (January 1993).
5. M. Olson, "The Design and Implementation of the Inversion File System," *Proceedings of the 1993 Winter USENIX Meeting,* San Diego, California (January 1993).
6. M. Stonebraker et al., "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (March 1990).
7. M. Stonebraker et al., "The Sequoia 2000 Benchmark," *Proceedings of the 1993 ACM SIGMOD Conference,* Washington, D.C. (May 1993).
8. S. Sarawagi and M. Stonebraker, "Efficient Organization of Large Multidimensional Arrays," *Proceedings of the 1993 IEEE Data Engineering Conference,* Houston, Texas (February 1993).
9. J. Hellerstein and M. Stonebraker, "Predicate Migration: Optimizing Queries with Expensive Predicates," *Proceedings of the 1993 ACM SIGMOD Conference,* Washington, D.C. (May 1993).

10. P. Kochevar and L. Wanger, "Tecate: A Software Platform for Browsing and Visualizing Data from Networked Data Sources," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 66–83.

11. M. Stonebraker et al., "Tioga: Providing Data Management for Scientific Visualization Applications," *Proceedings of the 1993 VLDB Conference,* Dublin, Ireland (August 1993).

12. A. Woodruff et al., "Zooming and Tunneling in Tioga: Supporting Navigation in Multidimensional Space," Sequoia 2000 Technical Report 94/48, Berkeley, California (March 1994).

13. R. Larson, "Classification, Clustering, Probabilistic Information Retrieval and the On-Line Catalog," *Library Quarterly* (April 1991).

14. *Information Retrieval Application Service Definition and Protocol Specification for Open Systems Interconnection,* ANSI/NISO Z39.50-1992 (revision and redesignation of ANSI Z39.50-1988) (New York: American National Standards Institute/National Information Standards Organization, 1992).

15. D. Ferrari, "Client Requirements for Real-time Communication Services," *IEEE Communications* (November 1990).

16. J. Pasquale et al., "High-performance I/O and Networking Software in Sequoia 2000," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 84–96.

17. C. Herot, "SDMS: A Spatial Data Base System," *ACM Transactions on Computing Systems (TOCS)* (June 1980).

18. The National Center for Supercomputing Applications (NCSA) at the University of Illinois developed the Hierarchical Data Format (HDF) as a multiobject file format.

19. Network Common Data Form (netCDF) is an interface for scientific data access and a freely distributed software library that provides an implementation of the interface. netCDF was developed by Glenn Davis, Russ Rew, and Steve Emmerson at the Unidata Program Center in Boulder, Colorado. The netCDF library defines a machine-independent format for representing scientific data. Together, the interface, the library, and the format support the creation, access, and sharing of scientific data.

20. J. Anderson and M. Stonebraker, "Sequoia 2000 Metadata Schema for Satellite Images," *SIGMOD Record,* Vol. 23, No. 4 (December 1994).

21. R. Larson et al., "The Sequoia 2000 Electronic Repository," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 50–65.

22. M. Stonebraker et al., "An Economic Paradigm for Query Processing and Data Migration in Mariposa," *Proceedings of IEEE Parallel and Distributed Information Systems Conference,* Austin, Texas (September 1994).

Biography

Michael Stonebraker
Michael Stonebraker is a professor of electrical engineering and computer science at the University of California, Berkeley, where he has been employed since 1971. He was one of the principal architects of the INGRES relational database management system and subsequently constructed Distributed INGRES. For the last six years, Michael has been developing POSTGRES, a next-generation DBMS that can manage objects and rules, as well as data. Michael is a founder of INGRES Corporation, the founder of Illustra Information Technologies, a past chairman of ACM SIGMOD, and the author of many papers on DBMS technology. He lectures widely and was the winner of the first ACM SIGMOD innovations award in 1992.

The Sequoia 2000 Electronic Repository

Ray R. Larson
Christian Plaunt
Allison G. Woodruff
Marti A. Hearst

A major effort in the Sequoia 2000 project was to build a very large database of earth science information. Without providing the means for scientists to efficiently and effectively locate required information and to browse its contents, however, this vast database would rapidly become unmanageable and eventually unusable. The Sequoia 2000 Electronic Repository addresses these problems through indexing and retrieval software that is incorporated into the POSTGRES database management system. The Electronic Repository effort involved the design of probabilistic indexing and retrieval for text documents in POSTGRES, and the development of algorithms for automatic georeferencing of text documents and segmentation of full texts into topically coherent segments for improved retrieval. Various graphical interfaces support these retrieval features.

Global change researchers, who study phenomena that include the Greenhouse Effect, ozone depletion, global climate modeling, and ocean dynamics, have found serious problems in attempting to use current information systems to manage and manipulate the diverse information sources crucial to their research.[1] These information sources include remote sensing data and images from satellites and aircraft, databases of measurements (e.g., temperature, wind speed, salinity, and snow depth) from specific geographic locations, complex vector information such as topographic maps, and large amounts of text from a variety of sources. These textual documents range from environmental impact reports on various regions to journal articles and technical reports documenting research results.

The Sequoia 2000 project brought together computer and information scientists from the University of California (UC), Digital Equipment Corporation, and the San Diego Supercomputer Center (SDSC), and global change researchers from UC campuses to develop practical solutions to some of these problems.[2] One goal of this collaboration was the development of a large-scale (i.e., multiterabyte) storage system that would be available to the researchers over high-speed network links. In addition to storing massive amounts of data in this system, global change researchers needed to be able to share its contents, to search for specific known items in it, and to retrieve relevant unknown items based on various criteria. This sharing, searching, and retrieving had to be done efficiently and effectively, even when the scale of the database reached the multiterabyte range.

The goal of the Electronic Repository portion of the Sequoia 2000 project was to design and evaluate methods to meet these needs for sharing, searching, and retrieving database objects (primarily text documents). The Sequoia 2000 Electronic Repository is the precursor of several ongoing projects at the University of California, Berkeley, that address the development of digital libraries.

For repository objects to be effectively shared and retrieved, they must be indexed by content. User interfaces must allow researchers to both search for items based on specific characteristics and browse the repository for desired information. This paper summarizes

the research conducted in these areas by the Sequoia 2000 project participants. In particular, the paper describes the Lassen text indexing and retrieval methods developed for the POSTGRES database system, the GIPSY system for automatic indexing of texts using geographic coordinates based on locations mentioned in the text, and the TextTiling method for improving access to full-text documents.

Indexing and Retrieval in the Electronic Repository

The primary engine for information storage and retrieval in the Sequoia 2000 Electronic Repository is the POSTGRES next-generation database management system (DBMS).[3] POSTGRES is the core of the DBMS-centric Sequoia 2000 system design. All the data used in the project was stored in POSTGRES, including complex multidimensional arrays of data, spatial objects such as raster and vector maps, satellite images, and sets of measurements, as well as all the full-text documents available. The POSTGRES DBMS supports user-defined abstract data types, user-defined functions, a rules system, and many features of object-oriented DBMSs, including inheritance and methods, through functions in both the query language, called POSTQUEL, and conventional programming languages. The POSTQUEL query language provides all the features found in relational query languages like SQL and also supports the nonrelational features of POSTGRES. These features give POSTGRES the ability to support advanced information retrieval methods.

We used these features of POSTGRES to develop prototype versions of advanced indexing and retrieval techniques for the Electronic Repository. We chose this approach rather than adopting a separate retrieval system for full-text indexing and retrieval for the following reasons:

1. Text elements are pervasive in the database, ranging in size from short descriptions or comments on other data items to the complete text of large documents, such as environmental impact reports.
2. Text elements are often associated with other data items (e.g., maps, remote sensing measurements, and aerial photographs), and the system must support complex queries involving multiple data types and functions on data.
3. Many text-only systems lack support for concurrent access, crash recovery, data integrity, and security of the database, which are features of the DBMS.
4. Unlike many text retrieval systems, DBMSs permit ad hoc querying of any element of the database, whether or not a predefined index exists for that element.

Moreover, there are a number of interesting research issues involved in the integration of methods of text retrieval derived from information retrieval research with the access methods and facilities of a DBMS. Information retrieval has dealt primarily with imprecise queries and results that require human interpretation to determine success or failure based on some specified notion of relevance. Database systems have dealt with precise queries and exact matching of the query specification. Proposals exist to add probabilistic weights to tuples in relations and to extend the relational model and query language to deal with the characteristics of text databases.[4,5] Our approach to designing this prototype was to use the features of the POSTGRES DBMS to add information retrieval methods to the existing functionality of the DBMS. This section describes the processes used in the prototype version of the Lassen indexing and retrieval system and also discusses some of the ongoing development work directed toward generalizing the inclusion of advanced information retrieval methods in the DBMS.[6]

Indexing

The Lassen indexing method operates as a daemon invoked whenever a new text item is appended to the database. Several POSTGRES database relations (i.e., classes, in POSTGRES terminology) provide support for the indexing and retrieval processes. Figure 1 shows these classes and their logical linkages. These classes are intended to be treated as system-level classes, which are usually not seen by users.

The wn_index class contains the complete WordNet dictionary and thesaurus.[7] It provides the normalizing basis for terms used in indexing text elements of the database. That is, all terms extracted from data elements in the database are converted to the word form used in this class. The POSTQUEL statement defining the class is

```
create wn_index (
termid = int4,      /* unique term ID */
word = text,        /* the term or phrase */
pos = char,         /* WordNet part of speech
                       information */
sense_cnt = int2,   /* number of senses of word */
ptruse_cnt = int2,  /* types and locations of */
offset_cnt = int2,  /* related terms in WordNet*/
ptruse = int2[] ,   /* database are stored in */
offset = int4[])    /* these arrays
```

All other references to terms in the indexing process are actually references to the unique term identifiers (*termid*) assigned to words in this class. The wn_index dictionary contains individual words and common phrases, although in the prototype implementation, only single words are used for indexing purposes. The other parts of the record include WordNet database information such as the part of speech (*pos*) and an array of pointers to the different senses of the word.

The kw_term_doc_rel class provides a linkage between a particular text item in any class or text large object (we will refer to either as documents) and

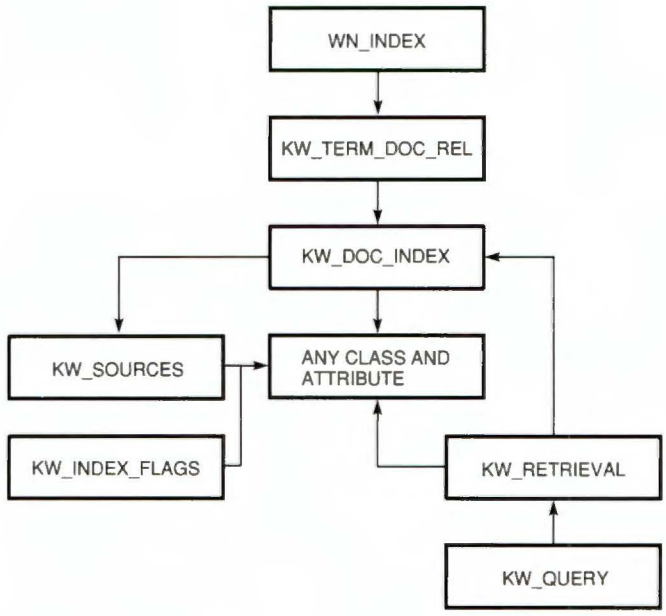


Figure 1
The Lassen POSTGRES Classes for Indexing and Their Linkages

a particular term from the wn_index class. The POSTQUEL definition of this class is

```
create kw_term_doc_rel (
termid = int4,     /* WordNet termid number */
synset = int4,     /* WordNet sense number */
docid = int4,      /* document ID */
termfreq = int4)   /* term frequency within
                      the document */
```

The raw frequency of occurrence of the term in the document (*termfreq*) is included in the kw_term_doc_rel tuple. This information is used in the retrieval process for calculating the probability of relevance for each document that contains the term. The kw_doc_index class stores information on individual documents in the database. This information includes a unique document identifier (*docid*), the location of the document (the class, the attribute, and the tuple in which it is contained), and whether it is a simple attribute or a large object (with effectively unlimited size). The kw_doc_index class also maintains additional statistical information, such as the number of unique terms found in the document. The POSTQUEL definition is as follows:

```
create kw_doc_index (
docid = int4,       /* document ID */
reloid = oid,       /* oid of relation
                       containing it */
attroid = oid,      /* attribute definition of
                       attr containing it */
attrnum = int2,     /* attribute number of attr
                       containing it */
tupleid = oid,      /* tuple oid of tuple
                       containing it */
sourcetype = int4,  /* type of object -- attribute
                       or large object */
doc_len = int4,     /* document length in words */
doc_ulen = int4)    /* number of unique words in
                       document */
```

The kw_sources class contains information about the classes and attributes indexed at the class level, as well as statistics such as the number of items indexed from any given class. The following POSTQUEL statement defines this class:

```
create kw_sources (
relname = char16,      /* name of indexed
                          relation */
reloid = oid,          /* oid of indexed
                          relation */
attrname = char16,     /* name of indexed
                          attribute */
attroid = oid,         /* object ID of indexed
                          attribute */
attrnum = int2,        /* number of indexed
                          attribute */
attrtype = int4,       /* attribute type -- large
                          object or otherwise */
num_indexed = int4,    /* number of items
                          indexed */
last_tid = oid,        /* oid and time for last */
last_time = abstime,   /* tuple added */
tot_terms = int4,      /* total terms from all
                          items */
tot_uterms = int4,     /* total unique terms from
                          all items */
include_pat = text,    /* simple patterns to */
exclude_pat = text)    /* match for indexable
                          /* items */
```

The other classes shown in Figure 1 relate to the indexing and retrieval processes. The Lassen prototype uses the POSTGRES rules system to perform such tasks as storing the elements of the bibliographic records in an appropriate normalized form and to trigger the indexing daemon.

Defining an attribute in the database as indexable for information retrieval purposes (i.e., by appending a new tuple to the kw_sources definition) creates a rule that appends the class name and attribute name to the

kw_index_flags class whenever a new tuple is appended to the class. Another rule then starts the indexing process for the newly appended data. Figure 2 shows this trigger process.

The indexing process extracts each unique keyword from the indexed attributes of the database and stores it along with pointers to its source document and its frequency of occurrence in kw_term_doc_rel. This process is shown in Figure 3. The indexing daemon and the rules system maintain other global frequency information. For example, the overall frequency of occurrence of terms in the database and the total number of indexed items are maintained for retrieval processing. The indexing daemon attempts to perform any outstanding indexing tasks before it shuts down. It also updates the kw_doc_index tuple for a given indexable class and attribute with a time stamp for the last item indexed (*last_tid* and *last_time*). This permits ongoing incremental indexing without having to reindex older tuples.

Retrieval

The prototype version of Lassen provides ranked retrieval of the documents indexed by the indexing daemon using a probabilistic retrieval algorithm. This algorithm estimates the probability of relevance for each document based on statistical information on term usage in a user's natural language query and in the database. The algorithm used in the prototype is based on the staged logistic regression method.[8]

A POSTGRES user-defined function invokes ranked retrieval processing. That is, from a user's perspective, ranked retrieval is performed by a simple function call (kwsearch) in a POSTQUEL query language statement. Information from the classes created and maintained by the indexing daemon are used to estimate the probability of relevance for each indexed document. (Note that the full power of the POSTQUEL query language can also be used to perform conventional Boolean retrieval using the classes created by the indexing process and to combine the results of ranked retrieval with other search criteria.) Figure 4 shows the process involved in the probabilistic ranked retrieval from the repository database.

The actual query to the Lassen ranked retrieval process consists simply of a natural language statement of the searcher's interests. The query goes through the

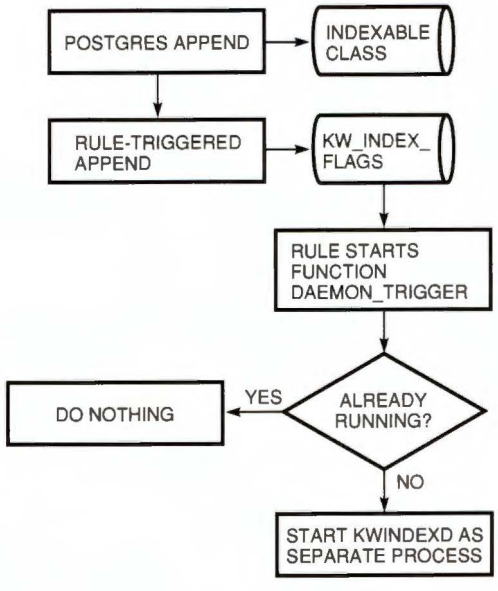


Figure 2
The Lassen Indexing Trigger Process

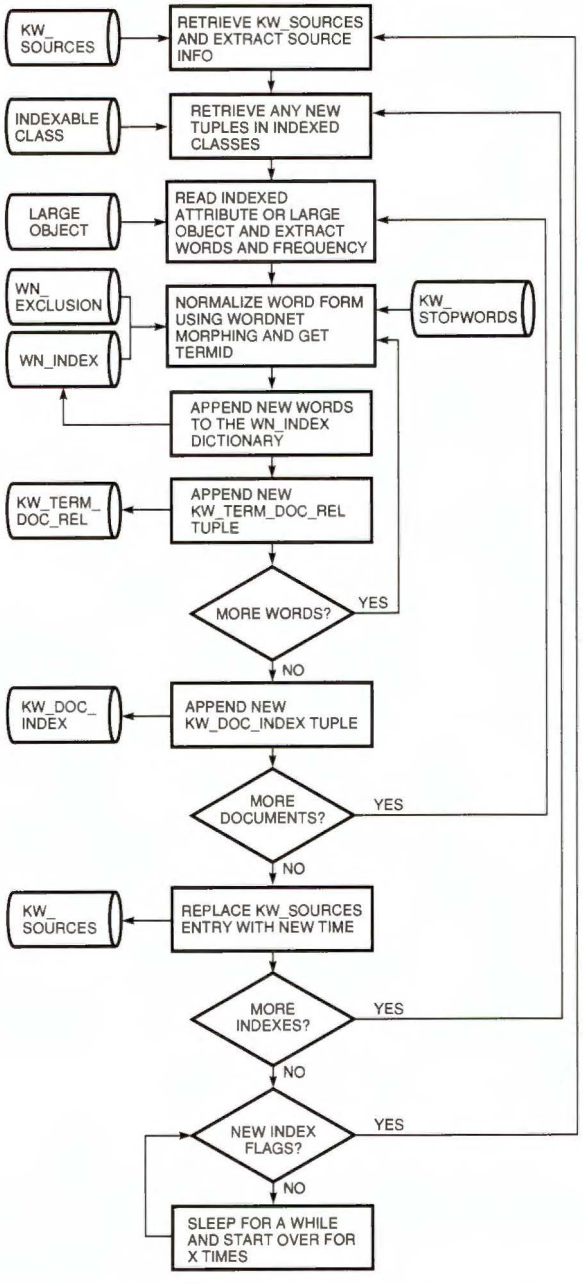


Figure 3
The Lassen Indexing Daemon Process

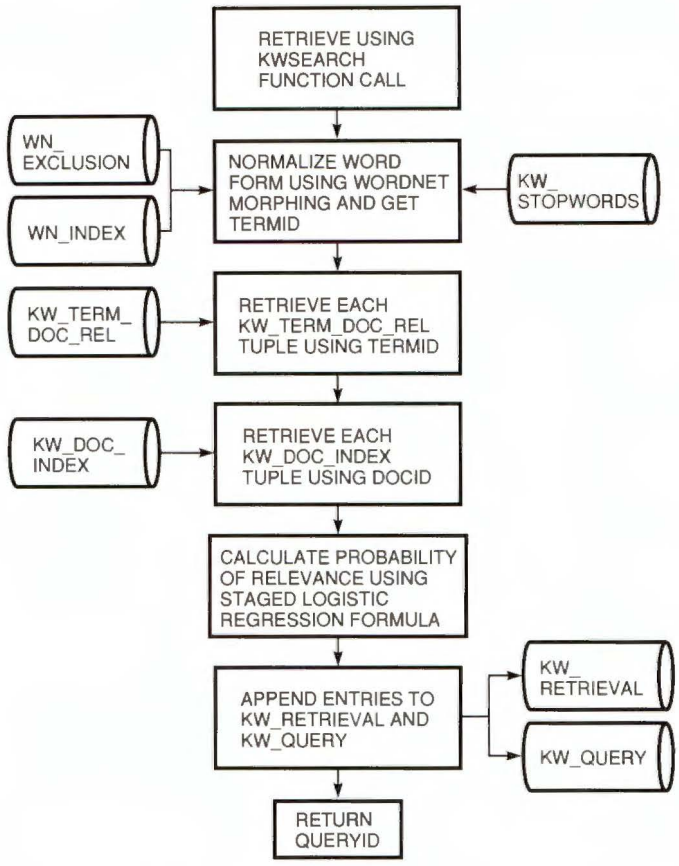


Figure 4
The Lassen Retrieval Process

same processing steps as documents in the indexing process. The individual words of the query are extracted and located in the wn_index dictionary (after removing common words or "stopwords"). The termids for matching words from wn_index are then used to retrieve all the tuples in kw_term_doc_rel that contain the term. For each unique document identifier in this list of tuples, the matching kw_doc_index tuple is retrieved. With the frequency information contained in kw_term_doc_rel and kw_doc_index, the estimated probability of relevance is calculated for each document that contains at least one term in common with the query. The formulae used in the calculation are based on experiments with full-text retrieval.[8] The basic equation for the probabilistic model used in Lassen states the following: The probability of the event that a document is relevant R, given that there is a set of N "clues" associated with that document, A_i for $i = 1, 2, \ldots, N$, is

$$\log O(R|A_1,\ldots,A_N) = \log O(R) + \sum_{i=1}^{N} [\log O(R|A_i) - \log O(R)], \quad (1)$$

where for any events E and E', the odds $O(E|E')$ is $P(E|E')/P(\bar{E}|E')$, i.e., a simple transformation of the probabilities. Because there is not enough information to compute the exact probability of relevance for any user and any document, an estimation is derived based on logistic regression of a set of clues (usually terms or words) contained in some sample of queries and the documents previously judged to be relevant to those queries. For a set of M terms that occur in both a query and a given document, the regression equation is of the form

$$\log O(R|A_1,\ldots,A_M) \approx c_0 + c_1 \cdot f(M) \sum_{1}^{M} X_{m,1} + \cdots + c_K \cdot f(M) \sum_{1}^{M} X_{m,K} + c_{K+1} M + c_{K+2} M^2, \quad (2)$$

where there are K retrieval variables $X_{m,K}$ used to characterize each term or clue, and the c_i coefficients are constant for a given training set of queries and documents. The coefficients used in the prototype were derived from analysis of full-text documents

and queries (with relevance judgments) from the TIPSTER information retrieval test collection.[9] The derivation of this formula is given in "Probabilistic Retrieval Based on Staged Logistic Regression."[8] The full retrieval equation used for the prototype version of retrieval described in this section is

$$\log O(R|A_1,\ldots,A_M) \approx -3.51 + \frac{1}{\sqrt{M}+1}\left[37.4\sum_{1}^{M} X_{m,1} + 0.330\sum_{1}^{M} X_{m,2} - 0.1937\sum_{1}^{M} X_{m,3}\right] + 0.0929M, \quad (3)$$

where

$X_{m,1}$ is the quotient of the number of times the mth term occurs in the query and the sum of the total number of terms in the query plus 35;

$X_{m,2}$ is the logarithm of the quotient arrived at by dividing the number of times the mth term occurs in the document by the sum of the total number of terms in the document plus 80;

$X_{m,3}$ is the logarithm of the quotient arrived at by dividing the number of times the mth term occurs in the database (i.e., in all documents) by the total number of terms in the collection;

M is the number of terms held in common by the query and the document.

Note that the M^2 term called for in Equation 2 was not found to provide any significant difference in the results and was omitted from Equation 3. The constants 35 and 80, which were used in $X_{m,1}$ and $X_{m,2}$, are arbitrary but appear to offer the best results when set to the average size of a query and the average size of a document for the particular database. The sequence of operations performed to calculate the probability of relevance is shown in Figure 5. Note that in the figure, $k1$, ..., $k5$ represent the constants of Equation 3.

The probability of relevance is calculated for each document (by converting the logarithmic odds to a probability) and is stored along with a unique query identifier, the document identifier, and some location information in the kw_retrieval class. The query itself

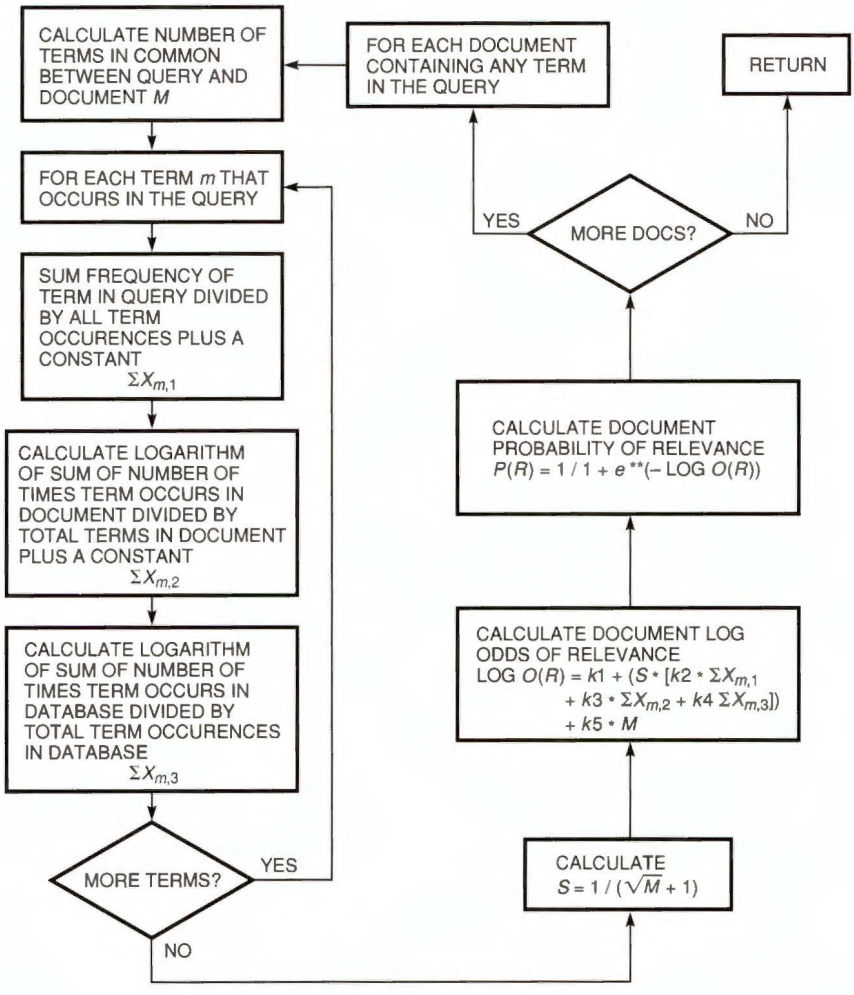


Figure 5
The Calculation for the Staged Logistic Regression Probabilistic Ranking Process

and its unique identifier are stored in the kw_query class. To see the results of the retrieval operation, the query identifier is used to retrieve the appropriate kw_retrieval tuples, ranked in order according to the estimated probability of relevance. The kw_retrieval and kw_query classes have the following POSTQUEL definitions:

```
create kw_query (
query_id = int4,          /* ID number */
query_user = char16,      /* POSTGRES user name */
query_text = text)        /* the actual query */

create kw_retrieval (
query_id = int4,          /* link to the query */
doc_id = int4,            /* document ID number */
rel_oid = oid,            /* location of doc */
attr_oid = oid,
attr_num = int2,
tuple_id = oid,
doc_len = int4,           /* size of document */
doc_match_terms = int4,   /* number of query terms
                             in the document */
doc_prob_rel = float8)    /* estimated probability
                             of relevance */
```

The algorithm used for ranked retrieval in the Lassen prototype was tested against a number of other systems and algorithms as part of the TREC competition and provided excellent retrieval performance.[10] We have found that the retrieval coefficients used in the formula derived from analysis of the TIPSTER collection appear to work well for a variety of document types. In principle, the staged logistic regression retrieval coefficients should be adapted to the particular characteristics of the database by collecting relevance judgments from actual users and reapplying the staged logistic regression analysis to derive new coefficients. This activity has not been performed for this prototype implementation.

The primary contribution of the Lassen prototype has been as a proof-of-concept for the integration of full-text indexing and ranked retrieval operations in a relational database management system. The prototype implementation that we have described in this section has a number of problems. For example, in the prototype design for indexing and retrieval operations, all the information used is visible in user-accessible classes in the database. Also, the overhead is fairly high, in terms of storage and processing time, for maintaining the indexing and retrieval information in this way. For example, POSTGRES allocates 40 bytes of system information for each tuple in a class, and indexing can take several seconds per document.

Currently, we are investigating a class of new access methods to support indexing and retrieval in a more efficient fashion. The class of methods involves declaring some POSTGRES functions that can extract subelements of a given type of attribute (such as words in a text document) and generate indexes for each of the subelements extracted. Other types of data might also benefit from this class of access methods. For example, functions that extract subelements like geometric shapes from images might be used to generate subelement indexes of image collections. Particular index element extraction methods can be of great value in providing access to the sort of information stored in the Sequoia 2000 Electronic Repository. The following section describes one such index extraction method developed for the special needs of Sequoia 2000 data.

GIPSY: Automatic Georeferencing of Text

Environmental Impact Reports (EIRs), journal articles, technical reports, and myriad other text items related to global change research that might be included in the Sequoia 2000 database are examples of a class of documents that discuss or refer to particular places or regions. A common retrieval task is to find the items that refer to or concentrate on a specific geographic region. Although it is possible to have a human catalog each item for location, one goal of the Electronic Repository was to make all indexing and retrieval automatic, thus eliminating the requirement for human analysis and classification of documents in the database. Therefore, part of our research involved developing methods to perform automatic georeferencing of text documents, that is, to automatically index and retrieve a document according to the geographic locations discussed or displayed in or otherwise associated with its content.

In Lassen and most other full-text information retrieval systems, searches with a geographical component, such as "Find all documents whose contents pertain to location X," are not supported directly by indexing, query, or display functions. Instead, these searches work only by references to named places, essentially as side effects of keyword indexing. Whereas human indexers are usually able to understand and apply correct references to a document, the costs in time and money of using geographically trained human indexers to read and index the entire contents of a large full-text collection are prohibitive. Even in cases where a document is meticulously indexed manually, geographic index terms consisting of keywords (text strings) have several well-documented problems with ambiguity, synonymy, and name changes over time.[11,12]

Advantages of the GIPSY Model

To deal with these problems, we developed a new model for supporting geographically based access to text.[13] In this model, words and phrases that contain geographic place names or geographic characteristics are extracted from documents and used as input to certain database functions. These functions use spatial reasoning and statistical methods to approximate the

geographic position being referenced in the text. The actual index terms assigned to a document are a set of coordinate polygons that describe an area on the earth's surface in a standard geographical projection system. Using coordinates instead of names for the place or geographic characteristic offers a number of advantages.

- Uniqueness. Place names are not unique, e.g., Venice, California, and Venice, Italy, are not apparently different without the qualifying larger region to differentiate them. Using coordinates removes this ambiguity.
- Immunity to spatial boundary changes. Political boundaries change over time, leading to confusion about the precise area being referred to. Coordinates do not depend on political boundaries.
- Immunity to name changes. Geographic names change over time, making it difficult for a user to retrieve all information that has been written about an area during any extended time period. Coordinates remove this ambiguity.
- Immunity to spatial, naming, and spelling variation. Names and terms vary not only over time but also in contemporary usage. Geographic names vary in spelling over time and by language. Areas of interest to the user will often be given place names designated only in the context of a specific document or project. Such variations occur frequently for studies done in oceanic locations. Names associated with these studies are unknown to most users. Coordinates are not subject to these kinds of verbal variations.

Indexing texts and other objects (e.g., photographs, videos, and remote sensing data sets) by coordinates also permits the use of a graphical interface to the information in the database, where representations of the objects are plotted on a map. A map-based graphical interface has several advantages over one that uses text terms or one that simply uses numerical access to coordinates. As Furnas suggests, humans use different cognitive structures for graphical information than for verbal information, and spatial queries cannot be fully simulated by verbal queries.[14] Because many geographical queries are inherently spatial, a graphical model is more intuitive. This is supported by Morris' observation that users given the choice between menu and graphical interfaces to a geographic database preferred the graphical mode.[15] A graphical interface, such as a map, also allows for a dense presentation of information.[16]

To address the needs of global change scientists, the Sequoia 2000 project team proposed a new browser paradigm.[17] This system, called Tioga, displays information topologically according to continuous characteristics that are attributes of the data.[18] For example, documents may be displayed on a map according to their latitude and longitude. Documents may also be displayed according to the time at which they were generated and the time to which they refer, as well as by more abstract functions such as the reading level of the document and the author's attitudes as expressed in the document. A prototype of the geographical browsing component was included in the Lassen Geographic Browser, which is shown in Figure 6.

This browser allows any georeferenced object in the database to be indicated by an icon on the map. The user employs the mouse to center the map on any location and to zoom in or out for more or less map detail. Icons can be made to appear at any coordinates and for any range of magnification values. When an icon is selected by the user, a menu of the objects georeferenced at the icon coordinates and detail level are displayed for selection.

An Algorithm to Georeference Text

The advantages of georeferencing are apparent. Not so apparent is how to perform such a task automatically. We developed the following three-part thesaurus-based algorithm to explore this task; the algorithm provides the basis for georeferencing in GIPSY.[19]

1. Identify geographic place names and phrases. This step attempts to recognize all relevant content-bearing geographic words and phrases. The parser for this step must "understand" how to identify geographic terminology of two types:
   a. Terms that match objects or attributes in the data set. This step requires a large thesaurus of geographic names and terms, partially hand built and partially automatically generated.
   b. Lexical constructs that contain spatial information, e.g., "adjacent to the coast," "south of the delta," and "between the river and the highway."

   To implement this part of the algorithm, a list of the most commonly occurring constructs must be created and integrated into a thesaurus.
2. Locate pertinent data. The output of the parser is passed to a function that retrieves geographic coordinate data pertinent to the extracted terms and phrases. Spatially indexed data used in this step can include, for example, name, size, and location of cities and states; name and location of endangered species; and name, location, and bioregional characteristics of different climatic regions. The system must identify the spatial locations that most closely match the geographic terms extracted by the parser and, when geographic modifiers are used, heuristically modify the area of coverage. For example, the phrase "south of Lake Tahoe" will map to the area south of Lake Tahoe, covering approximately the same volume. This spatial representation is, by

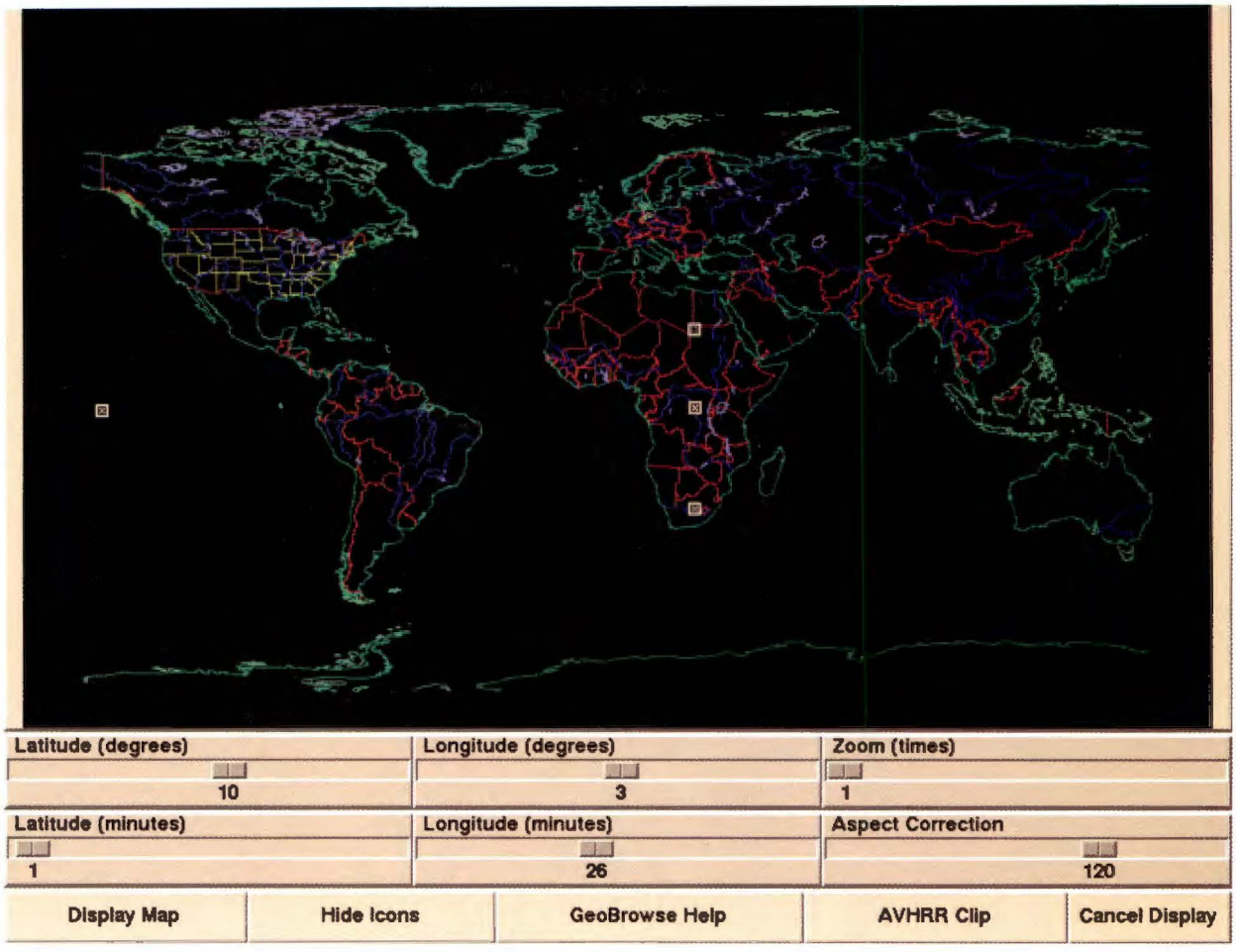


Figure 6
Screen from the Lassen Geographic Browser

necessity, the result of an arbitrary assumption of size, but its purpose is to provide only partial evidence to be used in determining locations as described below.

Since geopositional data for land use (e.g., cities, schools, and industrial areas) and habitats (e.g., wetlands, rivers, forests, and indigenous species) is also available, extracted keywords and phrases for these types of data must be recognized. The thesaurus entries for this data should incorporate several other types of information, such as synonymy (e.g., Latin and common names of species) and membership (e.g., wetlands contain cattails, but geopositional data on cattails may not exist, so we must use their mention as weak evidence of a discussion of wetlands and use that data instead).

For our implementation of GIPSY, we used two primary data sets to construct the thesaurus. The first was a subset of the United States Geological Survey's Geographic Names Information System (GNIS).[20] This data set contains latitude/longitude point coordinates associated with over 60,000 geographic place names in California. To facilitate comparison with other data sets, the GNIS latitude/longitude coordinates were converted to the Lambert-Azimuthal projection. Examples of place names with associated points include

University of California Davis: –1867878 –471379

Redding: –1863339 –234894

Data for land use and habitat data was derived in the United States Geological Survey's Geographic Information Retrieval and Analysis System (GIRAS).[21]

Each identified name, phrase, or region description is associated with one or more polygons that may be the place discussed in the text. Weights can be assigned to each of these polygons based on the frequency of use of its associated term or phrase in the text being indexed and in the thesaurus. Many relevant terms do not exactly match place names or the feature and land use types listed above. For example, alfalfa is a crop grown in California and should be associated with the crop data from the GIRAS land use data set. The thesaurus was therefore extended, both manually and by extraction of

relationships from the WordNet thesaurus, to include the following types of terms:[7]

synonymy

= : = synonym

kind-of relationships

~ : = hyponym (maple is a ~ of tree)

@ : = hypernym (tree is a @ of maple)

part-of relationships

: = meronym (finger is a # of hand)

% : = holonym (hand is a % of finger)

& : = evidonym (pine is a & of shortleaf pine)

3. Overlay polygons to estimate approximate locations. The objective of this step is to combine the evidence accumulated in the preceding step and infer a set of polygons that provides a reasonable approximation of the geographical locations mentioned in the text. Each *geophrase, weight, polygon* tuple can be represented as a three-dimensional "extruded" polygon whose base is in the plane of the x- and z-axes and whose height extends upward on the y-axis a distance proportional to its weight (see Figure 7a). As new polygons are added, several cases may arise.

   a. If the base of a polygon being added does not intersect with the base of any other polygons, it is simply laid on the base map beginning at $y = 0$ (see Figure 7b).

   b. If the polygon being added is completely contained within a polygon that already exists on the geopositional skyline, it is laid on top of that extruded polygon, i.e., its base plane is positioned higher on the y-axis (see Figure 7c).

   c. If the polygon being added intersects but is not wholly contained by one or more polygons, the polygon being added is split. The intersecting portion is laid on top of the existing polygon and the nonintersecting portion is positioned at a lower level (i.e., at $y = 0$). To minimize fragmentation in this case, polygons are sorted by size prior to being positioned on the skyline (see Figure 7d).

In effect, the extruded polygons, when laid together, are "summed" by weight to form a geopositional skyline whose peaks approximate the geographical locations being referenced in the text. The geographic coordinates assigned to the text segment being indexed are determined by choosing a threshold of elevation z in the skyline, taking the x-z plane at z, and using the polygons at the selected elevation. Raising the elevation of the threshold will tend to increase the accuracy of the retrieval, whereas lowering the elevation tends to include other similar regions.

To see the results of this process in the GIPSY prototype, consider the following text from a publication of the California Department of Water Resources:

> The proposed project is the construction of a new State Water Project (SWP) facility, the Coastal Branch, Phase II, by the Department of Water Resources (DWR) and a local distribution facility, the Mission Hills Extension, by water purveyors of northern Santa Barbara County. This proposed buried pipeline would deliver 25,000 acre-feet per year (AF/YR) of SWP water to San Luis Obispo County Flood Control and Water Conservation District (SLOCFCWCD) and 27,723 AF/YR to Santa Barbara County Flood Control and Water Conservation District (SBCFCWCD).... This extension would serve the South Coast and Upper Santa Ynez Valley. DWR and the Santa Barbara Water Purveyors Agency are jointly producing an EIR for the Santa Ynez Extension. The Santa Ynez Extension Draft EIR is scheduled for release in spring 1991.[22]

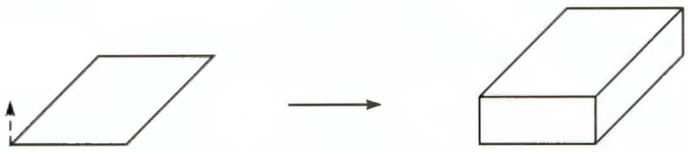
The resulting surface plot appears in Figure 8. The figure contains a gridded representation of the state of California, which is elevated to distinguish it from the base of the grid. The northern part of the state is on the left-hand side of the image. The towers rising over the state's shape represent polygons in the skyline generated by GIPSY's interpretation of the text. The largest towers occur in the area referred to by the text, primarily centered on Santa Barbara County, San Luis Obispo, and the Santa Ynez Valley area.

The surface plots generated in this fashion can also be used for browsing and retrieval. For example, the two-dimensional base of a polygon with a thickness above a certain threshold can be assigned as a coordinate index to a document. These two-dimensional polygons might then be displayed as icons on a map browser such as the one shown in Figure 6.

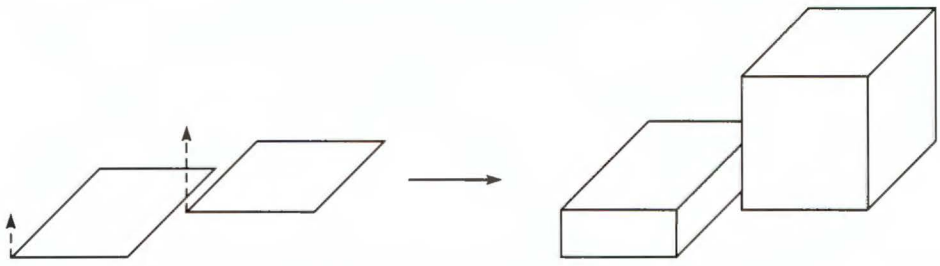
Future Work

Research remains to be done on several extensions to the existing GIPSY implementation. Because a geographic knowledge base and spatial reasoning are fundamental to the georeferencing process, they have been the focus of initial research efforts.

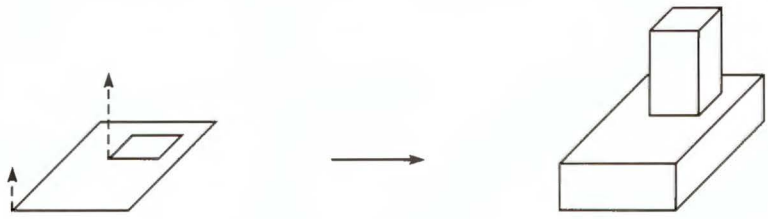
The existing prototype can be complemented by the addition of more sophisticated natural language processing. For example, spatial reasoning and geographic data could be combined with parsing techniques to develop semantic representations of the text. Adjacency indicators, such as "south of" or "between," should be recognized by a parser. Also, the work on document segmentation described below could be used to explore the locality of reference to geographic entities within full-text documents. GIPSY's technique may be most effective when applied to a paragraph or section level of a text instead of to the entire document.



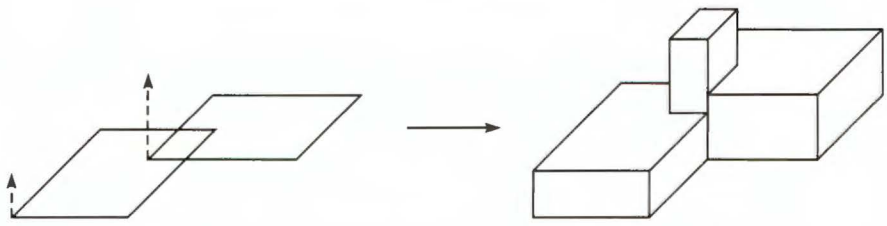
(a) The "weight" of a polygon, indicated by the vertical arrow, is interpreted as "thickness."



(b) Two adjacent polygons do not affect each other; each is merely assigned its appropriate "thickness."



(c) When one polygon subsumes another, their "thicknesses" in the area of overlap are summed.



(d) When two polygons intersect, their "thicknesses" are summed in the area of overlap.

Figure 7
Overlaying Polygons to Estimate Approximate Locations

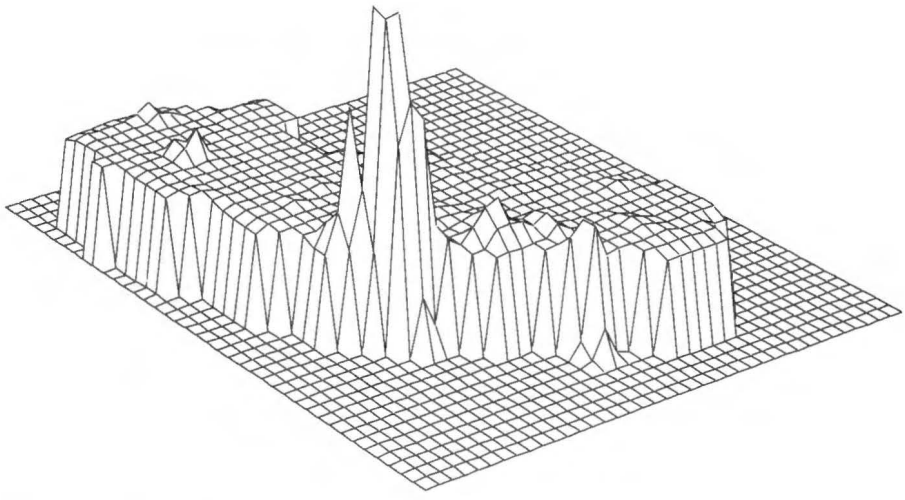


Figure 8
Surface Plot Produced from the State Water Project Text

TextTiling: Enhancing Retrieval through Automatic Subtopic Identification

Full-length documents have only recently become available on-line in large quantities, although technical abstracts, short newswire texts, and legal documents have been accessible for many years.[23] The large majority of on-line information has been bibliographic (e.g., authors, titles, and abstracts) instead of the full text of the document. For this reason, most information retrieval methods are better suited for accessing abstracts than for accessing longer documents. Part of the repository research was an exploration of new approaches to information retrieval particularly suited to full-length texts, such as those expected in the Sequoia 2000 database.

A problem with applying traditional information retrieval methods to full-length text documents is that the structure of full-length documents is quite different from that of abstracts. (In this paper, "full-length document" refers to expository text of any length. Typical examples are a short magazine article and a 50-page technical report. We exclude documents composed of headlines, short advertisements, and any other disjointed texts of whatever length. We also assume that the document does not have detailed orthographically marked structure. Croft, Krovetz, and Turtle describe work that takes advantage of this kind of information.[24]) One way to view an expository text is as a sequence of subtopics set against a backdrop of one or two main topics. A long text comprises many different subtopics that may be related to one another and to the backdrop in many different ways. The main topics of a text are discussed in its abstract, if one exists, but subtopics are usually not mentioned. Therefore, instead of querying against the entire content of a document, a user should be able to issue a query about a coherent subpart, or subtopic, of a full-length document, and that subtopic should be specifiable with respect to the document's main topic(s).

Consider a *Discover* magazine article about the Magellan space probe's exploration of Venus.[25] A reader divided this 23-paragraph article into the following segments with the labels shown, where the numbers indicate paragraph numbers:

1–2 Intro to Magellan space probe
3–4 Intro to Venus
5–7 Lack of craters
8–11 Evidence of volcanic action
12–15 River Styx
16–18 Crustal spreading
19–21 Recent volcanism
22–23 Future of Magellan

Assume that the topic of volcanic activity is of interest to a user. Crucial to a system's decision to retrieve this document is the knowledge that a dense discussion of volcanic activity, rather than a passing reference, appears. Since volcanism is not one of the text's two main topics, the number of references to this term will probably not dominate the statistics of term frequency. On the other hand, document selection should not necessarily be based on the number of references to the target terms.

The goal should be to determine whether or not a relevant discussion of a concept or topic appears. A simple approach to distinguishing between a true discussion and a passing reference is to determine the locality of the references. In the computer science operating systems literature, locality refers to the fact that over time, memory access patterns tend to concentrate in localized clusters rather than be distributed evenly throughout memory. Similarly, in full-length texts, the close proximity of members of a set of

references to a particular concept is a good indicator of topicality. For example, the term *volcanism* occurs 5 times in the Magellan article, the first four instances of which occur in four adjacent paragraphs, along with accompanying discussion. In contrast, the term *scientists,* which is not a valid subtopic, occurs 13 times, distributed somewhat evenly throughout. By its very nature, a subtopic will not be discussed throughout an entire text. Similarly, true subtopics are not indicated by only passing references. The term *belly dancer* occurs only once, and its related terms are confined to the one sentence it appears in. As its usage is only a passing reference, belly dancing is not a true subtopic of this text.

Our solution to the problem of retaining valid subtopical discussions while at the same time avoiding being fooled by passing references is to make use of locality information and to partition documents according to their subtopical structure. This approach's capacity for improving a standard information retrieval task has been verified by information retrieval experiments using full-text test collections from the TIPSTER database.[26,27]

One way to get an approximation of the subtopic structure is to break the document into paragraphs, or for very long documents, sections. In both cases, this entails using the orthographic marking supplied by the author to determine topic boundaries.

Another way to approximate local structure in long documents is to divide the documents into even-sized pieces, without regard for any boundaries. This approach is not practical, however, because we are interested in exploring the performance of motivated segmentation, i.e., segmentation that reflects the text's true underlying subtopic structure, which often spans paragraph boundaries.

Toward this end, we have developed TextTiling, a method for partitioning full-length text documents into coherent multiparagraph units called tiles.[26,28,29] TextTiling approximates the subtopic structure of a document by using patterns of lexical connectivity to find coherent subdiscussions. The layout of the tiles is meant to reflect the pattern of subtopics contained in an expository text. The approach uses quantitative lexical analyses to determine the extent of the tiles and to classify them with respect to a general knowledge base. The tiles have been found to correspond well to human judgments of the major subtopic boundaries of science magazine articles.

The algorithm is a two-step process. First, all pairs of adjacent blocks of text (where blocks are usually three to five sentences long) are compared and assigned a similarity value. Second, the resulting sequence of similarity values, after being graphed and smoothed, is examined for peaks and valleys. High similarity values, which imply that the adjacent blocks cohere well, tend to form peaks, whereas low similarity values, which indicate a potential boundary between tiles, create valleys. Figure 9 shows such a graph for the *Discover* magazine article mentioned earlier. The vertical lines indicate where human judges thought the topic boundaries should be placed. The graph shows the computed similarity of adjacent blocks of text. Peaks indicate coherency, and valleys indicate potential breaks between tiles.

The one adjustable parameter is the size of the block used for comparison. This value, *k,* varies slightly from text to text. As a heuristic, it is assigned the average paragraph length (in sentences), although the block size that best matches the human judgment data is sometimes one sentence greater or smaller. Actual paragraphs are not used because their lengths can be highly irregular, leading to unbalanced comparisons.

Similarity is measured by using a variation of the tf.idf (term frequency times inverse document frequency) measurement.[30] In standard tf.idf, terms that are frequent in an individual document but relatively infrequent throughout the corpus are considered to be good distinguishers of the contents of the individual document. In TextTiling, each block of *k* sentences is treated as a unit, and the frequency of a term within each block is compared to its frequency in the entire document. (Note that the algorithm uses a large stop list; i.e., closed class words and other very frequent terms are omitted from the calculation.) This approach helps bring out a distinction between local and global extent of terms. A term that is discussed frequently within a localized cluster (thus indicating a cohesive passage) will be weighted more heavily than a term that appears frequently but scattered evenly throughout the entire document, or infrequently within one block. Thus if adjacent blocks share many terms, and those shared terms are weighted heavily, there is strong evidence that the adjacent blocks cohere with one another.

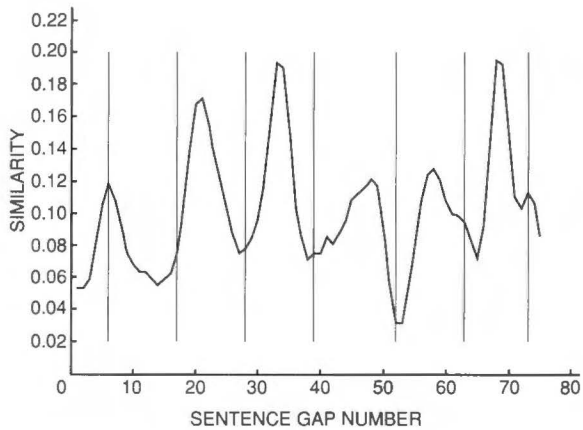


Figure 9
Results of TextTiling a 77-sentence Science Article

Similarity between blocks is calculated by the following cosine measure: Given two text blocks $b1$ and $b2$,

$$\cos(b1,b2) = \frac{\sum_{t=1}^{n} w_{t,b1}\, w_{t,b2}}{\sqrt{\sum_{t=1}^{n} w_{t,b1}^2 \sum_{t=1}^{n} w_{t,b2}^2}},$$

where t ranges over all the terms in the document, and $w_{t,b1}$ is the tf.idf weight assigned to term t in block $b1$. Thus, if the similarity score between two blocks is high, then not only do the blocks have terms in common, but the common terms are relatively rare with respect to the rest of the document. The evidence in the reverse is not as conclusive. If adjacent blocks have a low similarity measure, this does not necessarily mean that the blocks cohere. In practice, however, this negative evidence is often justified.

The graph is then smoothed using a discrete convolution[31] of the similarity function with the function $h_k(.)$, where

$$h_k(i) \equiv \begin{cases} \frac{1}{k^2}(k - |i|), & |i| \leq k-1 \\ 0, & \text{otherwise.} \end{cases}$$

The result is smoothed further with a simple median smoothing algorithm to eliminate small local minima.[32] Tile boundaries are determined by locating the lowermost portions of valleys in the resulting plot. The actual values of the similarity measures are not taken into account; the relative differences are what are of consequence.

Retrieval processing should reflect the assumption that full-length text is meaningfully different in structure from abstracts and short articles. We have conducted retrieval experiments that demonstrate that taking text structure into account can produce better results than using full-length documents in the standard way.[26,28,29] By working within this paradigm, we have developed an approach to vector-space-based retrieval that appears to work better than retrieving against entire documents or against segments or paragraphs alone.

The resulting retrieval method matches a query against motivated segments and then sums the scores from the top segments for each document. The highest resulting sums indicate which documents should be retrieved. In our test set, this method produced higher precision and recall than retrieving against entire documents or against segments or paragraphs alone.[26] Although the vector-space model of retrieval was used for these experiments, probabilistic models such as the one used in Lassen are equally applicable, and the method should provide similar improvement in retrieval performance.

We believe that recognizing the structure of full-length text for the purposes of information retrieval is very important and will produce considerable improvement in retrieval effectiveness over most existing similarity-based techniques.

Conclusion

The Sequoia 2000 Electronic Repository project has provided a test bed for developing and evaluating technologies required for effective and efficient access to the digital libraries of the future. We can expect that as digital libraries proliferate and include vast databases of information linked together by high-bandwidth networks, they must support all current and future media in an easily accessible and content-addressable fashion.

The work begun on the Sequoia 2000 Electronic Repository is continuing under UC Berkeley's digital library project sponsored jointly by the National Science Foundation (NSF), the National Aeronautics and Space Administration (NASA), and the Defense Advanced Research Projects Agency (DARPA). Digital libraries are a fledgling technology with no firm standards, architectures, or even consensus notions of what they are and how they are to work. Our goal in this ongoing research is to develop the means of placing the contents of this developing global virtual library at the fingertips of a worldwide clientele. Achieving this goal will require the application of advanced techniques for information retrieval, information filtering, resource discovery, and the application of new techniques for automatically analyzing and characterizing data sources ranging from texts to videos. Much of the work needed to enable our vision of these new technologies was pioneered in the Sequoia 2000 Electronic Repository project.

References

1. J. Dozier, "How Sequoia 2000 Addresses Issues in Data and Information Systems for Global Change," Sequoia 2000 Technical Report 92/14 (S2K-92-14) (Berkeley, Calif.: University of California, Berkeley, 1992) (ftp://s2k-ftp.cs.berkeley.edu/pub/sequoia/tech-reports/s2k-9 2-14/s2k-92-14.ps).
2. M. Stonebraker, "An Overview of the Sequoia 2000 Project," *Digital Technical Journal,* vol. 7, no. 3 (1995, this issue): 39–49.
3. M. Stonebraker and G. Kemnitz, "The POSTGRES Next-generation Database Management System," *Communications of the ACM,* vol. 34, no. 10 (1991): 78–92.
4. N. Fuhr, "A Probabilistic Relational Model for the Integration of IR and Databases," *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '93),* Pittsburgh, June 27–July 1, 1993 (New York: Association for Computing Machinery, 1993): 309–317.

5. D. Blair, "An Extended Relational Document Retrieval Model," *Information Processing and Management,* vol. 24 (1988): 349–371.

6. R. Larson, "Design and Development of a Network-Based Electronic Library," *Navigating the Networks: Proceedings of the ASIS Midyear Meeting,* Portland, Oregon, May 21–25, 1994 (Medford, N.J.: Learned Information, Inc., 1994): 95–114. Also available as Sequoia 2000 Technical Report 94/54, July 1994 (ftp://s2k-ftp.cs.berkeley.edu/pub/sequoia/tech-reports/s2k-9 4-54/s2k-94-54.ps).

7. G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller, "Five Papers on WordNet," CSL Report 43 (Princeton, N.J.: Princeton University: Cognitive Science Laboratory, 1990).

8. W. Cooper, F. Gey, and D. Dabney, "Probabilistic Retrieval Based on Staged Logistic Regression," *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '92),* Copenhagen, Denmark, June 21–24, 1992 (New York: Association for Computing Machinery, 1992): 198–210.

9. D. Harman, "The DARPA TIPSTER Project," *SIGIR Forum,* vol. 26, no. 2 (1992): 26–28.

10. W. Cooper, A. Chen, and F. Gey, "Experiments in the Probabilistic Retrieval of Full Text Documents," *Text Retrieval Conference (TREC-3) Draft Conference Papers* (Gaithersburg, Md.: National Institute of Standards and Technology, 1994).

11. A. Griffiths, "SAGIS: A Proposal for a Sardinian Geographical Information System and an Assessment of Alternative Implementation Strategies," *Journal of Information Science,* vol. 15 (1989): 261–267.

12. D. Holmes, "Computers and Geographic Information Access," *Meridian,* vol. 4 (1990): 37–49.

13. A. Woodruff and C. Plaunt, "GIPSY: Georeferenced Information Processing SYstem," *Journal of the American Society for Information Science,* vol. 45, no. 9 (1994): 645–655.

14. G. Furnas, "New Graphic Reasoning Models for Understanding Graphical Interfaces," *Human Factors in Computing Systems: Reaching Through Technology Proceedings (CHI '91 Conference),* New Orleans, April-May 1991 (New York: Association for Computing Machinery, 1991): 71–78.

15. B. Morris, "CARTO-NET: Graphic Retrieval and Management in an Automated Map Library," *Special Libraries Association, Geography and Map Division Bulletin,* vol. 152 (1988): 19–35.

16. C. McCann, M. Taylor, and M. Tuori, "ISIS: The Interactive Spatial Information System," *International Journal of Man-Machine Studies,* vol. 28 (1988): 101–138.

17. J. Chen, R. Larson, and M. Stonebraker, "Sequoia 2000 Object Browser," *Digest of Papers, Thirty-seventh IEEE Computer Society International Conference (COMPCON Spring 1992),* San Francisco, February 24-28, 1992 (Los Alamitos, Calif.: Computer Society Press, February 1992): 389–394.

18. M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu, "Tioga: Providing Data Management Support for Scientific Visualization Applications," *Proceedings of the Nineteenth International Conference on Very Large Data Bases,* Dublin, Ireland (August 1993): 25–38.

19. A. Woodruff and C. Plaunt, "Automated Geographic Indexing of Text Documents," Sequoia 2000 Technical Report 94/41 (S2K-94-41) (Berkeley, Calif.: University of California, Berkeley, 1994) (ftp://s2k-ftp.cs.berkeley.edu/pub/sequoia/tech-reports/s2k-9 4-41/s2k-94-41.ps).

20. Geographic Names Information System/United States Department of the Interior, United States Geological Survey, rev. ed., Data User's Guide, vol. 6 (Reston, Va.: United States Geological Survey, 1987).

21. J. Anderson, E. Hardy, J. Roach, and R. Witmer, "A Land Use and Land Cover Classification System for Use with Remote Sensor Data," United States Geological Survey Professional Paper #964 (Washington, D.C.: United States Government Printing Office, 1976).

22. State Water Project, Coastal Branch, Phase II, and Mission Hills Extension (Sacramento, Calif.: California Department of Water Resources, 1991).

23. C. Tenopir and J. Ro, *Full Text Databases* (New York: Greenwood Press, 1990).

24. W. Croft, R. Krovetz, and H. Turtle, "Interactive Retrieval of Complex Documents," *Information Processing and Management,* vol. 26, no. 5 (1990): 593–616.

25. A. Chaikin, "Magellan Pierces the Venusian Veil," *Discover,* vol. 13, no. 1 (January 1992).

26. M. Hearst and C. Plaunt, "Subtopic Structuring for Full-Length Document Access," *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '93),* Pittsburgh, June 1993 (New York: Association for Computing Machinery, 1993): 59–68.

27. M. Hearst, "Context and Structure in Automated Full-Text Information Access," Ph.D. dissertation, Report No. UCB/CSD-94/836 (Berkeley, Calif.: University of California, Berkeley, Computer Science Division, 1994).

28. M. Hearst, "TextTiling: A Quantitative Approach to Discourse Segmentation," Sequoia 2000 Technical Report 93/24 (S2K-93-24) (Berkeley, Calif.: University of California, Berkeley, 1993) (ftp://s2k-ftp.cs.berkeley.edu/pub/sequoia/tech-reports/s2k-9 3-24/s2k-93-24.ps).

29. M. Hearst, "Multi-Paragraph Segmentation of Expository Text," *Proceedings of the Thirty-second Meeting of the Association for Computational Linguistics,* Los Cruces, New Mexico, June 1994.

30. G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer* (Reading, Mass.: Addison-Wesley, 1989).

31. The authors are grateful to Michael Braverman for proving that the smoothing algorithm is equivalent to this convolution.

32. L. Rabiner and R. Schafer, *Digital Processing of Speech Signals* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1978).

Biographies

Ray R. Larson
Ray Larson is an Associate Professor at the University of California, Berkeley, in the School of Information Management and Systems (formerly the School of Library and Information Studies). He teaches courses and conducts research on the design and evaluation of information retrieval systems. Ray received his Ph.D. from the University of California. He is a member of the American Society for Information Science (ASIS), the Association for Computing Machinery (ACM), the IEEE Computer Society, the American Association for the Advancement of Science, and the American Library Association. He is the Associate Editor for *ACM Transactions on Information Systems* and received the *ASIS Journal* Best Paper Award in 1993.

Christian Plaunt
Christian Plaunt is a doctoral student and graduate research assistant at the University of California, Berkeley, School of Information Management and Systems. His interests include experimental information retrieval system modeling, simulation, design, and evaluation; artificial intelligence techniques for information retrieval; multistage retrieval techniques; information filtering; and music. Chris holds master's degrees in library and information studies and in music (composition). In his spare time, he composes, plays the piano, and works in the Music Library at California State University, Fresno, near which he lives with his wife and their three Siamese cats.

Allison G. Woodruff
Allison Woodruff is a Ph.D. student in the Electrical Engineering and Computer Science Department at the University of California, Berkeley. Her research interests include spatial information systems, multimedia databases, visual programming languages, and user interfaces. Previously, she worked as a geographic information systems specialist for the California Department of Water Resources. Allison holds a B.A. in English from California State University, Chico, and an M.A. in linguistics and an M.S. in computer science from the University of California, Davis.

Marti A. Hearst
Currently a member of the research staff at Xerox Palo Alto Research Center, Marti Hearst completed her Ph.D. in computer science at the University of California, Berkeley, in April 1994. Her dissertation examined context and structure of full-text documents for information access. Her current research interests include intelligent information access, corpus-based computational linguistics, user interfaces, and psycholinguistics.

Peter D. Kochevar
Leonard R. Wanger

Tecate: A Software Platform for Browsing and Visualizing Data from Networked Data Sources

Tecate is a new infrastructure on which applications can be constructed that allow end users to browse for and then visualize data within networked data sources. This software platform capitalizes on the architectural strengths of current scientific visualization systems, network browsers like Netscape, database management system front ends, and virtual reality systems. Applications layered on top of Tecate are able to browse for information in databases managed by database management systems and for information contained in the World Wide Web. In addition, Tecate dynamically crafts user interfaces and interactive visualizations of selected data sets with the aid of an intelligent system. This system automatically maps many kinds of data sets into a virtual world that can be explored directly by end users. In describing these virtual worlds, Tecate uses an interpretive language that is also capable of performing arbitrary computations and mediating communications among different processes.

All people share the need to find and assimilate information. Data from which information is created is increasingly available electronically, and that data is becoming more and more accessible with the proliferation of computer networks. Therefore, the world is quickly becoming abstracted as a collection of networked data spaces, where a data space is a data source or repository whose access is controlled by means of a well-defined software interface. Some examples of data spaces are a database managed by a database management system, the World Wide Web (WWW or Web), and any data object that resides in a computer's main memory and whose components are accessible through the object's methods.

The need to locate data and then map it to a form that is readily understood lies at the core of learning, conducting commerce, and being entertained. To address this need, interactive tools are required for exploring data spaces. These tools should allow any end user to browse the contents of data spaces and to inspect, measure, compare, and identify patterns in selected data sets. Combining both tasks into one tool is both elegant and utile in that end users need to learn only one system to seamlessly switch back and forth between browsing for data and assimilating it. Before such applications can be constructed, however, a firm foundation must be defined that provides an interface to data spaces, helps map data into a visual representation, and manages user interactions with elements in the visualizations.

This paper describes one such software platform, called Tecate, which has been implemented as a research prototype to help understand the issues involved in exploring data spaces. With Tecate, the emphasis has been on developing the tools needed to build end-to-end applications. Such applications can access data spaces, automatically create virtual worlds that represent data found in data spaces, and give end users the ability to navigate and interact with those worlds as the mechanism for exploring data spaces. Because of this emphasis, Tecate's development concentrated on understanding what system components are needed to create end-to-end applications and how those components interact rather than on the functionality of individual components. As a consequence,

the tools provided by Tecate can be used to build applications of only modest capabilities.

Historically, Tecate grew out of the Sequoia 2000 project, which was initiated jointly by Digital Equipment Corporation and the University of California in 1991. The primary purpose of the Sequoia 2000 project was to develop information systems that would allow earth scientists to better study global environmental change. Sequoia 2000 participants needed to browse for data sets on which to test scientific hypotheses and then to interactively visualize the data sets once found. The data can be quite varied in content and structure, ranging from text and images to time-varying, multidimensional, gridded or polyhedral data sets. Such data may stream from many different sources, e.g., databases managed by a database management system, a running simulation of some physical process, or the WWW. Therefore, a tool was required that could interface to any such source. To be of maximum use, though, the tool had to be easy to use so that the scientists themselves could make sophisticated data queries and then experiment with the query results using a wide variety of data visualization techniques.

Generalizing from its Sequoia 2000 roots, the design of Tecate is intended to achieve four goals:

1. Interface to general data spaces wherever they may reside.
2. Saliently visualize most kinds of data, e.g., scientific data and the listings in a telephone book.
3. Dynamically craft user interfaces and interactive visualizations based on what data is selected, who is doing the visualizing, and why the user is exploring the data.
4. Allow end users to interact with elements in visualizations as a means to query data spaces, to explore alternate ways of presenting information, and to make annotations.

There are systems available today that have some of these capabilities, but no one system possesses all four. Data visualization systems such as AVS, Khoros, or Data Explorer are capable of visualizing scientific data; however, they are poor at interfacing to general data spaces, they provide only limited interactivity within visualizations themselves, and they require visualizations to be crafted by hand by knowledgeable end users.[1,2,3] Network browsers such as Netscape are good at fetching data from certain types of data spaces but are limited in the variety of data they can directly visualize without having to rely on external viewer programs. Moreover, most network browsers offer a restricted type of interactivity where only hyperlinks can be followed and text can be submitted through forms. Finally, front ends to database management systems provide elaborate querying mechanisms for selecting data from a database, but they lack a sophisticated means for visualizing and further exploring query results.

The Tecate architecture borrows from that of visualization systems, network browsers, and database management systems as well as from virtual reality systems like Alice and the Minimal Reality Toolkit/Object Modeling Language (MR/OML).[4,5] One major contribution of the Tecate system is that it incorporates the architectural strengths of these systems into a coherent whole. In addition, Tecate possesses at least two novel features that are not found in other data visualization systems. One feature is Tecate's use of an interpretive language that can describe three-dimensional (3-D) virtual worlds. This language is more than a markup language in that it is capable of performing arbitrary computations and facilitating communication among different processes. The second novel component of Tecate is the presence of an expert system that automatically crafts interactive visualizations of data. This system is intended to make data space exploration easier to perform by having end users simply state their goals while leaving the details of implementing a visualization to attain those goals to the expert system.

The remainder of the paper outlines Tecate's system model and architecture and then identifies and describes Tecate's major components. Finally, the paper sketches Tecate's capabilities by discussing two simple applications that have been implemented on top of the Tecate software framework. The first application is a tool for visualizing earth science data residing in a database managed by a database management system. The second application is a Web browser that uses 3-D graphics as an underlying browsing paradigm rather than depending solely on the medium of hypertext.

Tecate's System Model

After presenting an overview of Tecate's system model, this section provides details of the object model and the interpretive, object-oriented language used to describe virtual world objects.

Overview

From the standpoint of an applications programmer, Tecate is a distributed, object-oriented system. All major components of Tecate, as well as entities appearing in virtual worlds created by Tecate, are objects that communicate with one another by means of message passing. The main focus within Tecate is on object-object interactions. These interactions occur primarily when objects send messages to one another. An object can also send a message to itself, which has the effect of making a local function call. Unlike with graphics systems such as Open Inventor, rendering is not a central activity within Tecate; rather it is just a side effect

of object-object interactions.[6] In this sense, Tecate is like virtual reality programming systems such as Alice and MR/OML, although Tecate is far more flexible.

In the Tecate system, objects can create and destroy other objects and can alter the properties of existing objects on-the-fly. Such capabilities make Tecate very extensible and give it great power and flexibility. These capabilities can also cause problems for applications programmers, however, if care is not taken when writing programs. Presently, all of an object's properties are visible to all other objects, and hence those properties can be manipulated from outside the object. In the future, some form of selective property hiding needs to be added so that designated properties of an object cannot be altered by other objects.

A powerful feature of Tecate is its ability to dynamically establish object-subobject relationships. This feature provides a mechanism for building assemblies of parts similar to the mechanisms in classical hierarchical graphics systems like Doré or Open Inventor.[7] This feature also provides the capability of creating sets or aggregates of objects that share some trait, such as being highlighted. Tecate allows all objects within a set to be treated en masse by providing a means of selectively broadcasting messages to groups of objects. A message that is sent to an object can be forwarded to all the object's subobjects. Thus, for example, one object can serve as a container for all other objects that are highlighted; the highlighted objects are merely subobjects of the container. To unhighlight all highlighted objects, a single unhighlight message can be sent to the container object, which then forwards the message to all its subobjects. In general, an object can be the subobject of any number of other objects and thus simultaneously be a member of many different sets.

The handling of user input within Tecate is intended to appear the same as ordinary object-object interactions. All physical input devices that are known to Tecate have an agent object associated with them that acts as a device handler. All objects that wish to be informed of a particular input event register with the appropriate agent. When an input event occurs, the agent sends all registered objects a message notifying them of the event. Complex events, such as the occurrence of event A and event B within a specified time period, can easily be defined by creating new handler objects. These handlers register to be informed of separate events but then, in turn, inform other objects of the events' conjunction.

The Object Model

Tecate uses an object model in which no distinction is made between classes and instances, as is done in languages like C++.[8] In Tecate, there is a single object creation operation called cloning. Any object in the system can serve as a prototype from which a copy can be made through the clone operation. A clone inherits properties from its prototype by copying the prototype's properties, but any such property can be altered or removed, either by another object or by the clone itself, so that a clone can take on an identity of its own.

The object model is based on delegation. When Tecate clones an object to produce a new object, the prototype's properties are not explicitly copied. Instead, the new object retains a reference to the object from which it was cloned. When a reference to a property is made within an object, the system looks for the property value locally within the object. If no property value is found locally, then the object's prototype is searched to associate a value with the reference. If the prototype is itself a clone, the prototype's prototype is recursively searched to resolve the reference, and so on. This type of "lazy" evaluation of property references is called delegation.

Note that with delegation, a change in value for a property in an object may affect the values of all other objects that can trace their ancestry through prototype-clone relationships to the original object. This type of semantics is useful for establishing class-instance-like relationships between objects. For example, one object may represent a particular class of automobile tire, and all clones of the object would represent class instances. If a class-level change is needed that would affect all instances, e.g., a new tread pattern is to be introduced, only the object representing the tire class needs to change.

The clone-prototype chaining implied by delegation can be overridden by changing the property values locally. Thus, if one particular tire instance is to have a new tread pattern, then the pattern is altered in that instance only. References to the tread pattern for that object will use the local tread value rather than chain back to the tire class object. All other instances will continue to reference the value present in the tire class object.

All Tecate objects possess four classes of properties:

1. Appearance—attributes that affect an object's visual appearance, such as geometric and topological structure, color, texture, and material properties
2. Behaviors—a set of methods that are invoked upon receipt of messages from other objects
3. State—a collection of variables whose values represent an object's state
4. Subobjects—a list of objects that are parts of a given object, just as a wheel is part of a car

Although most users of the system uniformly see communicating objects, a distinction is actually made between two kinds of objects based on how they are implemented by applications programmers. Resource objects are implemented primarily as external processes using some compilable, general-purpose programming language such as C or Fortran. Objects that have

compute-intensive behaviors or whose behavior executions are time-critical are generally implemented as resource objects. For instance, most Tecate objects that provide system services, such as rendering or database management, are implemented as resource objects.

Objects populating virtual worlds that represent data features are implemented differently than resource objects by using an interpretive programming language called the Abstract Visualization Language (AVL). Such objects are called dynamic objects because they may be created, destroyed, and altered on-the-fly as a Tecate session unfolds. Nonetheless, the ability to dynamically add, remove, and alter object properties is not solely endemic to dynamic objects. Resource properties may also be changed on-the-fly because resources are actually implemented with a dynamic object that interfaces to the portion of the resource that is implemented as an external process.

The Abstract Visualization Language

AVL is essential to the Tecate system; it is through AVL that applications programmers write applications that use Tecate's features.[9] AVL is an interpretive, object-oriented programming language that is capable of performing arbitrary computations and facilitating communication among different processes. Through this language, applications programmers specify and manipulate object properties and invoke object behaviors by sending messages from one object to another.

AVL is a typeless language that manipulates character strings; it is based on the Tcl embeddable command language.[10] AVL extends Tcl by adding object-oriented programming support, 3-D graphics, and a more sophisticated event-handling mechanism. Although AVL is a proper superset of Tcl, the relationship between AVL and Tcl is much like that between C and C++. By adding a small set of new constructs to Tcl, the way applications programmers structure AVL programs differs markedly from how they structure Tcl programs, just as the C++ language extensions to C greatly alter the C programming style.

One use of AVL is to describe virtual worlds that represent data sets. Through AVL, objects that populate these worlds can be assigned behaviors that are elicited through user interaction. For instance, selecting a 3-D icon can cause a Universal Resource Locator (URL) to be followed out into the WWW. In this sense, AVL is somewhat like the Hypertext Markup Language (HTML) that underlies all Web browsers today, or, more fitting, it is similar to the Virtual Reality Modeling Language (VRML) that has been proposed as a 3-D analog of HTML.[11] AVL does, however, differ markedly from HTML and VRML, which are only markup languages. Because AVL is a full-fledged programming language that has sophisticated interaction handling built in, it is philosophically more similar to interpretive languages like Telescript, NewtonScript, and PostScript.[12,13,14] Like Telescript, for instance, AVL programs can encode "smart agents" that can be sent across a network to perform user tasks at a remote machine, if an AVL interpreter resides there. Note, however, that in the present version of Tecate, there is no notion of security when arbitrary AVL code runs on a remote machine.

AVL includes some additional commands that augment the Tcl instruction set, for instance, **clone** and **delete**. The **clone** command is the object creation command within AVL, and the **delete** command is the complementary operation to delete objects from the system. Object properties are specified and manipulated using the **add** command and deleted using the **remove** command. Behaviors in one object are initiated by another object using the **send** command, which specifies the behavior to invoke and the arguments to be passed. Queries about object properties can be made using the **inquire** command. The **which** command is used to determine where an object's properties are actually defined in light of Tecate's use of delegation to resolve property references. Finally, AVL provides a rich set of matrix and vector operators that are useful when positioning objects within 3-D scenes.

As an example of how AVL is used in practice, Figure 1 depicts a code fragment similar to one that appears in the WWW application described later in the paper. The code fragment creates a 3-D Web site icon that is positioned on a world map. The code begins with the definition of the *Hyperlink* object from which all Web site icons are cloned. The *Hyperlink* object is itself cloned from the *Visual* object that is predefined by Tecate at system start-up. The *Visual* object contains properties that relate to the viewing of objects within scenes. For instance, objects that are cloned from the *Visual* object inherit behaviors to rotate themselves and to change their color. To the properties that are inherited from the *Visual* object, the *Hyperlink* object adds the state variables *url* and *desc,* which will be used to store respectively a URL and its textual description. In addition, objects cloned from the *Hyperlink* object will inherit the default appearance of a solid blue sphere having unit radius.

The specification for the *Hyperlink* object also defines three behaviors: *init, openUrl,* and *showDesc*. The *init* behavior replaces the *init* method inherited from the *Visual* object. When an object cloned from the *Hyperlink* object receives an *init* message, it sets its *url* and *desc* state variables, positions itself within the scene whose name is given by the argument *scene,* and registers itself with the mouse handler agent to receive two events. When mouse button 1 is depressed, the agent sends the object the *openUrl* message, which in turn requests the WWW Interface to fetch the data pointed to by the object's URL. Depressing button 2 invokes the *showDesc* message, causing the Web site URL and description to be displayed by a previously

```
# Define a prototype for all Web icons
clone Hyperlink Visual

add Hyperlink {
   state {
      url ""
      desc ""
   }
   appearance {
      shape {sphere}
      diffuseColor {0.0 0.0 1.0}
      repType {surface}
   }
   behavior {
      # Initialize hyperlink
      init {url desc pos scene window} {
         addstate url $url
         addstate desc $desc
         send [getself] move "add $pos"
         add $scene "subobject [getself]"
         send $window addEvent "[getself] {Button-1 {openUrl {}}} {Button-2 {showDesc {}}}"
      }

      # Open the URL
      openUrl {} {send www fetch "[getstate url]"}

      # Display the description
      showDesc {} {send metaViewer display "[getstate desc]"}
   }
}

# Initialize an informational landscape
clone scene Visual
clone window Viewer
send window init {scene}

# Create a Web site icon
clone hlink Hyperlink
send hlink init {"http://www.sdsc.edu/Home.html"
      "SDSC home page" "-2.3 -2.0 1.0" scene window}

# Use the SDSC model geometry
add hlink {appearance {shape {box}}}
```

Figure 1
An Implementation of a World Wide Web Icon in the Abstract Visualization Language

defined interface widget called the *metaViewer.* The AVL command **getself**, which is used within the *init* behavior body, returns the name of the object on which the behavior was called, thus allowing applications programmers to write generic behaviors. The other AVL commands, **getstate** and **addstate**, are shorthand for "**get [getself] state ...**" and "**add [getself] {state ...}.**"

Once the *Hyperlink* object is defined, a *scene,* a display window, and a Web site icon are created. The Tecate *scene* object is cloned from the *Visual* object. The *window* object, cloned from the predefined *Viewer* object, is the viewport into which the scene is to be rendered. Finally, *hlink* is a Web site icon whose appearance differs from that which is inherited from the *Hyperlink* object. Rather than being spherical, the shape of the *hlink* icon is a unit cube.

Tecate's Architecture

The general structure of Tecate and how it relates to application programs is depicted in Figure 2. Tecate consists of a kernel, a set of basic system services, and a toolkit of predefined objects. The Tecate kernel, which is shown in Figure 3, is an object management system called the Abstract Visualization Machine; AVL is its native language. The Abstract Visualization Machine is responsible for creating, destroying, altering, rendering, and mediating communication between objects. The two major components of the Abstract Visualization Machine are the Object Manager and the Rendering Engine.

The Object Manager is the primary component of the Abstract Visualization Machine. It is responsible for interpreting AVL programs, managing a database of

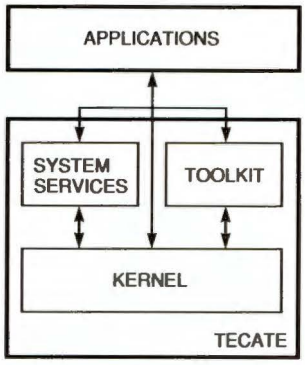


Figure 2
The Tecate System and Its Relationship to Application Programs

objects, mediating communication between objects, and interfacing with input devices. The Object Manager is itself a resource object that is distinguished by the fact that all other resource objects are spawned from this one object. In addition, the Object Manager is responsible for creating a distinguished dynamic object, called *Root,* from which all other dynamic objects can trace their heritage through prototype-clone relationships.

The Object Manager is implemented on a simple, custom-built thread package. Each object within Tecate can be thought of as a process that has its own thread of control. Each thread can be implemented either as a lightweight process that shares the same machine context as the Object Manager's operating system process or as its own operating system process separate from that of the Object Manager. Lightweight processes are so named because their use requires little system overhead, which enables thousands of such processes to be active at any given time. Within Tecate, dynamic objects are implemented as lightweight processes, whereas resource objects are implemented as heavyweight operating system processes, which may or may not be paired with a lightweight, adjunct process. A low-level function library is provided to handle the creation and destruction of threads and to handle interthread communication regardless of how the threads are implemented.

Closely allied with the Object Manager is the Rendering Engine, which is a special resource object wholly contained within the Abstract Visualization Machine. The Rendering Engine is responsible for creating a graphical rendition of a virtual world that is specified by AVL programs interpreted by the Object Manager. When interpreting an AVL program, the Object Manager strips off appearance attributes of objects and sends appropriate messages to the Rendering Engine so that it can maintain a separate display list that represents a virtual world. Display lists are represented as directed, acyclic graphs whose connectivity is determined by object-subobject relationships that are specified within AVL programs.

The present Rendering Engine implementation uses the Doré graphics package running on a DEC 3000 Model 500 workstation.[7] The display lists that are created by invoking behaviors within the Rendering Engine are actually built up and maintained through Doré. The set of messages that the Rendering Engine responds to represents an interface to a platform's graphics hardware that is independent of both the graphics package and the display device.

Layered on top of the Abstract Visualization Machine are Tecate's system services and the object toolkit. The system services consist of a collection of resource objects that are automatically instantiated at system start-up. These resources include an expert system called the Intelligent Visualization System, the Database Interface, the WWW Interface, and a

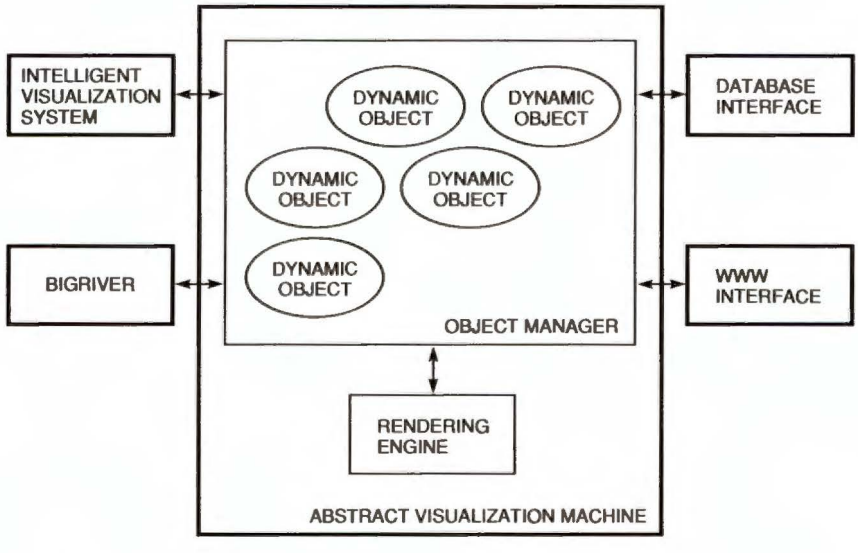


Figure 3
Detail of Tecate's Kernel (the Abstract Visualization Machine) and the System Services Provided by Tecate

visualization programming system called BigRiver. Figure 3 shows these resources in relationship to Tecate's kernel. Each resource is a Tecate object that has a number of predefined behaviors that can be useful to applications programmers. For instance, the WWW Interface has a behavior that fetches a data file referred to by a URL and then translates the file's contents into an appropriate AVL program.

The toolkit within Tecate is a set of predefined dynamic objects that programmers can use to develop applications. These objects are considered abstract objects in the sense that they are not intended to be used directly. Rather, they serve as prototypes from which clones can be created. The toolkit consists of objects such as viewports, lights, and cameras that are used to illuminate and render virtual worlds. The toolkit also contains a modest collection of 3-D user interface widgets that can be used within virtual worlds created by an applications programmer. These widgets include sliders, menus, icons, legends, and coordinate axes.

One useful object in the toolkit that aids in simulating physical processes and helps in performing animations is a clock. This object is an event generator that signals every clock tick. If objects wish to be informed of a clock pulse, those objects register themselves with the clock object just like objects register themselves with input device agent objects. The default clock object can be cloned, and each clone can be instantiated with a different clock period down to a resolution of one millisecond. Any number of clocks can be ticking simultaneously during a Tecate session. Since new clocks can be created dynamically, and objects can register and unregister to be informed of clock pulses on-the-fly, clocks can be used as timers and triggers, and as pacesetters.

Application Resources

Tecate's system services are predefined application resources that aid in interactively visualizing data. As mentioned previously, these objects include the Intelligent Visualization System, the Database Interface, the WWW Interface, and the BigRiver visualization programming system. In addition, an applications programmer can easily add new application resources using tools provided with the base Tecate system. Such new resources can be built around either user-written programs or commercial off-the-shelf applications. To create a new application resource, a programmer needs to provide a set of functions that can be invoked by other Tecate objects. These functions correspond to behaviors that are called when the resource receives a message from other objects. Tools are provided to register the behaviors with Tecate and to manage the communication between a resource and other Tecate objects.[15]

The Intelligent Visualization System

The Intelligent Visualization System allows Tecate to dynamically build interactive visualizations and user interfaces that aid nonexpert end users in exploring data spaces. This knowledge-based system is similar in concept to other expert visualization systems, as the literature describes.[16–21] The Intelligent Visualization System differs from other expert visualization systems in two important ways. First, the Intelligent Visualization System does not merely create a presentation of information as do most other systems. Instead, the Intelligent Visualization System creates virtual worlds with which end users can interact to alter the way data is presented, to make queries for additional data, and to store new data back into data spaces.

The second way the Intelligent Visualization System differs from expert visualization systems is that it takes a holistic approach to fashioning a visualization. Most systems decompose data into elementary components, determine how to visualize each component separately, and then recompose the individual visualizations into a final presentation. In contrast, Tecate's Intelligent Visualization System analyzes the full structure of data by relying on a sophisticated data model based on the mathematical notion of fiber bundles.[22–24] One way to view fiber bundles is as a generalization of the concept of graphs of mathematical functions. Depending on the character of a fiber bundle's independent and dependent variables, certain visualization techniques are more applicable than others.

In general, the Intelligent Visualization System automatically crafts virtual worlds based on a task specification and a description of the data that is to be visualized. A task specification represents a high-level data analysis goal of what an end user hopes to understand from the data. For instance, an end user may wish to determine if there is any correlation between temperature and the density of liquid water in a climatology data set. Usually, task specifications must be input by an end user, although at times they can be inferred automatically by the system. Tecate provides a simple task language from which task specifications can be built, and it provides a point-and-click tool for end users to create these specifications when needed. Data descriptions, on the other hand, do not require any end-user input because they are provided automatically by a data-space interface when data is imported into the system.

From the data description and task specification, a Planner within the Intelligent Visualization System produces a dataflow program that when executed builds an appropriate virtual world that represents a selected data set. The Planner uses a collection of rules, definitions, and relationships that are stored in a knowledge base when building a visualization that addresses a given task specification. Contents of the knowledge base include knowledge about data

models, user tasks, and visualization techniques. The Planner functions by constructing a sentence within a dataflow language defined by a context-sensitive graph grammar. At each step in the construction of the sentence, rules in the knowledge base dictate which productions in the grammar are to be applied and when. Presently, the knowledge base is implemented using the Classic knowledge representation system; the Planner is implemented in CLOS.[25,26]

BigRiver

The dataflow program produced by the Intelligent Visualization System is written in a scripting language that is interpreted by BigRiver, a visualization programming system similar to AVS and Khoros.[1,2] From a technical standpoint, BigRiver is not particularly innovative and will eventually be reimplemented using some existing visualization system that has more functionality. The reason that BigRiver was created from scratch was to better understand how existing visualization programming systems work and to overcome limitations within those systems. These limitations are their inability to be embedded within other applications, their lack of comprehensive data models, and their inability to work with user-supplied renderers. The latest generation of visualization programming systems, such as Data Explorer and AVS/Express, overcome many of these limitations.[3,27]

Like most of the existing visualization systems, BigRiver consists of a collection of procedures called modules, each of which has a well-defined set of inputs and outputs. Functional specifications for these modules represent some of the knowledge contained in the Intelligent Visualization System's knowledge base. Visualization scripts that are interpreted by BigRiver specify module parameter values and dictate how the outputs of chosen modules are to be channeled into the inputs of others.

BigRiver modules come in three varieties: I/O, data manipulators, and glyph generators. All modules use self-describing data formats based on fiber bundles. One format is used for manipulation within memory; the other is an on-the-wire encoding intended for transporting data across a network. An input module is responsible for converting data stored in the on-the-wire encoding into the in-memory format. The data manipulator modules transform fiber bundles of one in-memory format into those of another. The glyph generators take as input fiber bundles in the in-memory format and produce AVL programs that when executed build virtual worlds containing objects that represent features of selected data sets. A single display module takes as input AVL code and passes it to the Abstract Visualization Machine. By means of the Rendering Engine, the Abstract Visualization Machine uses the appearance attributes of objects to create an image of a virtual world that contains the objects.

The Database Interface

The Database Interface provides the means to interact with a database management system, which in the current version of Tecate can be either POSTGRES or Illustra.[28,29] Database queries, written in POSTQUEL for POSTGRES-managed databases or in SQL for Illustra databases, are sent to the Database Interface by Tecate objects where they are passed to a database management system server for execution. The server returns the query results to the Database Interface, which then attempts to package them up as an on-the-wire encoding of a fiber bundle buffered on local disk. If the result is a set of tuples in the standard format returned by POSTGRES or Illustra, the Database Interface performs the fiber bundle translation. For most other nonstandard results, the so-called binary large objects (BLOBs) of the database realm, the Database Interface cannot yet arbitrarily perform the translation into the on-the-wire fiber bundle encoding. The only BLOBs that the Database Interface can deal with presently are those that are already encoded as on-the-wire fiber bundles. The difficult problem of automated data format translation was not addressed during Tecate's initial development, although the intent is to address this issue in the future.

Once query results are buffered on disk, a description of the fiber bundle and the location of the buffer are sent back to the object that made the query request of the Database Interface. That object might then request the Intelligent Visualization System to structure a virtual world whose image would appear on the display screen by way of BigRiver and the Rendering Engine. Objects in the virtual world can be given behaviors that are elicited by user interactions. These behaviors might then result in further database queries and so on. Chains of events such as these provide a means for browsing databases through direct manipulation of objects within a virtual world.

The World Wide Web Interface

The WWW Interface functions similarly to the Database Interface but instead of accessing data in a database, the WWW Interface provides access to data stored on the World Wide Web. Messages that contain URLs are passed to the WWW Interface, which then fetches the data pointed to by the URLs. In retrieving data from the Web, the WWW Interface uses the same CERN software libraries used by Web browsers like Netscape.

Once a data file is fetched, the WWW Interface attempts to translate its contents into an AVL program, which is then passed to the Object Manager for interpretation. AVL either specifies the creation of a new virtual world that represents the data file's contents or specifies new objects that are to populate the current world being viewed. If the fetched data file contains a stream of AVL code, the WWW Interface

merely forwards the file to the Object Manager. If the file contains general data in the form of an on-the-wire encoding of a fiber bundle, the WWW Interface appeals to the Intelligent Visualization System to structure an appropriate virtual world. If the data file contains a stream of HTML code, the WWW Interface invokes an internal translator that translates HTML code into an equivalent AVL program, which is then interpreted by the Object Manager. This interpreter actually understands an extended version of HTML that supports the direct embedding of AVL within HTML documents. Through this mechanism, 3-D objects with which users can interact can be embedded directly into a hypertext Web page—something that few if any other Web browsers can do today.

Example Applications

Applications that browse the contents of data spaces and then interactively visualize selected results have the same overall structure. One browser application component acts as a data space interface, and through this interface queries are posed, query results are imported into the application, and data generated by the application is stored back into a data space. Once data has been imported into the application, a second component must map the data into some appropriate virtual world. Finally, a third component must manage any interactions that may take place between an end user and elements that populate the virtual worlds that are created.

In creating an application using Tecate, the Database Interface and the WWW Interface represent resources that can be used to form the application's data space interface. The mapping of data into a representative virtual world can utilize Tecate's Intelligent Visualization System and the BigRiver visualization programming system. Finally, the management of these worlds can take place through AVL programs that exercise the features of Tecate's Abstract Visualization Machine. The following two examples that were implemented in AVL illustrate how Tecate can be used to create applications that browse data spaces.

Visualizing Data in a Database

A simple example of an application that exploits Tecate's features is one that browses for earth science data in a database and then provides visualizations of that data. The initial user interface for this application is built using a collection of user interface widgets, where each widget is a Tecate dynamic object. Because the Tecate system does not yet have a comprehensive 3-D widget set, some widgets still rely on two-dimensional (2-D) constructs provided by the Tk widget set that is implemented on top of the Tcl language.[30]

Figure 4 depicts the flow of messages between some of the more important objects that are used within the application. One object is the Map Query Tool that is used to make certain graphical queries for earth science data sets whose geographical extents and time stamps fall within user-specified constraints. The tool is built around a world map on which regions of interest can be specified (see Figure 5). When a user marks a region of interest on the map and selects a temporal range, a query message is sent to the Database Interface. The result of the query is returned to the Map Query Tool, which then forwards a description of the result to the Intelligent Visualization System. To structure an appropriate visualization, an inferred select task directive accompanies the result. The ensuing script produced by the Intelligent Visualization System is executed by BigRiver, which produces a stream of AVL code that is sent to the Abstract Visualization Machine for interpretation.

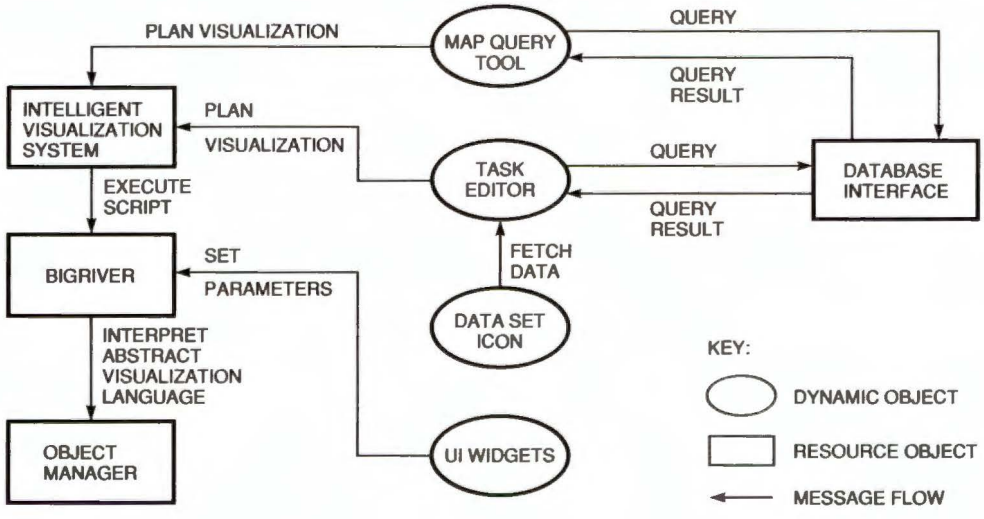


Figure 4
Message Flow between Important Objects in the Earth Science Application

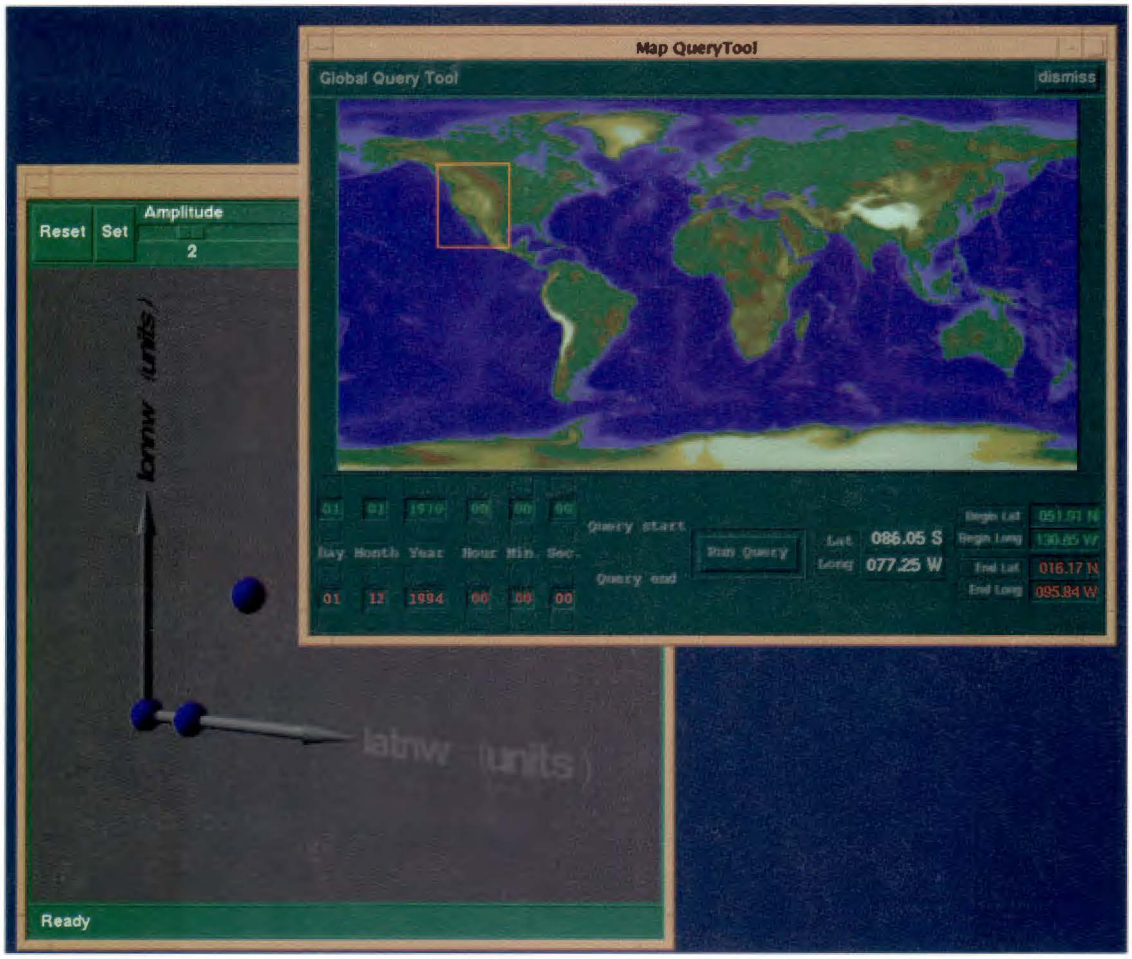


Figure 5
The Map Query Tool Showing a Visualization of a Query Result

This AVL program creates a new virtual world that consists of a collection of 3-D objects. Each object acts as an icon that corresponds to one data set that was returned as the result of the initial query (see Figure 5). The Intelligent Visualization System also builds in two behaviors for each icon. Depending on how a user selects an icon, either the metadata associated with the data set represented by the icon is displayed in a separate window or a query message is sent to the Database Interface requesting the actual data. In the latter case, the Map Query Tool again forwards the query result to the Intelligent Visualization System, and another virtual world containing objects representing data features is created and displayed with the aid of BigRiver and the Abstract Visualization Machine. In general, data exploration proceeds this way by creating and discarding virtual worlds based on interactions with objects that populate prior worlds.

After selecting an icon to actually view the data associated with it, an end user is asked by the Intelligent Visualization System to input a task specification using a Task Editor. Generally, data sets can be visualized in many different ways. The Intelligent Visualization System uses the task specification to select the one visualization that best satisfies the stated task. After a task specification is entered, a visualization of the selected data set appears on the screen. The BigRiver dataflow program that the Intelligent Visualization System creates to do that visualization can be edited by hand by knowledgeable end users to override the decisions made by the system.

Figure 6 shows a Task Editor and a visualization crafted by the Intelligent Visualization System after an end user selected a data-set icon. The visualization represents hydrological data that consists of a collection of tuples, each corresponding to a set of measurements made at discrete geographical locations. Based on the task specification that the end user entered, the Intelligent Visualization System chose to map the data into a coordinate system that has axes that represent latitude, longitude, and elevation. Each sphere represents an individual measurement site, whose color is a function of the mean temperature. When an end user selects a sphere, the actual data values associated with

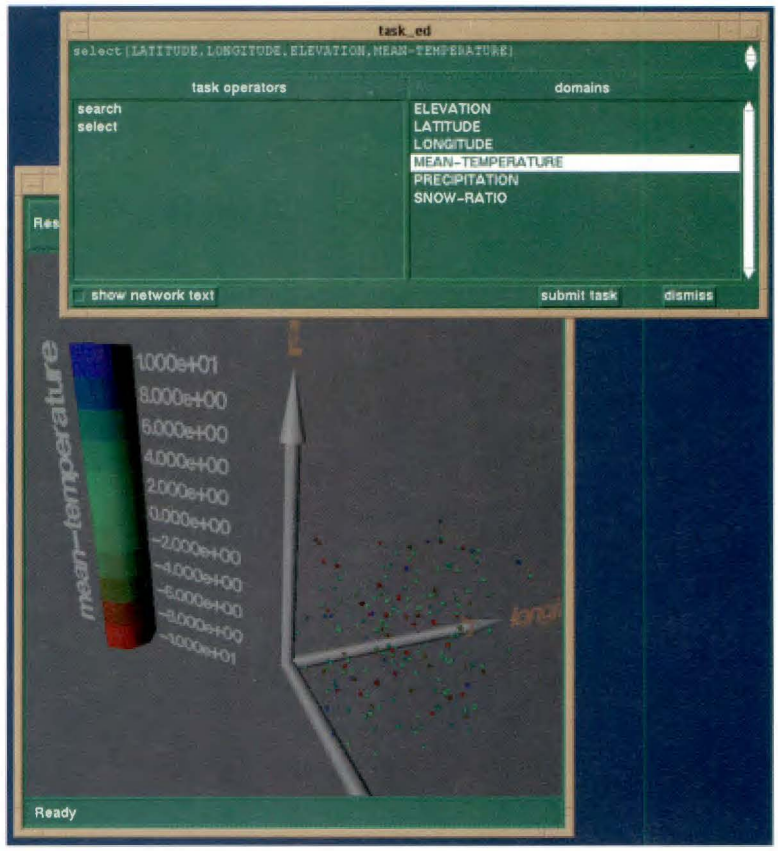


Figure 6
Task Editor Showing a Visualization of Hydrological Data

the location represented by the sphere are displayed. In addition, the Intelligent Visualization System automatically places into the virtual world of the visualization a color legend to help relate sphere colors to mean temperature values.

Figure 7 depicts another virtual world showing a visualization of data-set output from a regional climate model program. The data set is a 3-D array indexed by latitude, longitude, and elevation. Each array element is a tuple that contains cloud density, water content, and temperature values. In this instance, the end user entered a task specification that stated that the spatial variation in temperature was of primary importance. The Intelligent Visualization System responded by specifying a visualization that represented the temperature data as an isosurface, i.e., a surface whose points all have the same value for the temperature. Included in the virtual world is a widget that can be used to change the isosurface value and the field variable that is being studied.

The isosurface widget that appears in the visualization shown in Figure 7 is of special interest because of the way that it is implemented. Embedded in the tool is a slider that is used to change the isosurface value. As with most sliders, the slider value indicator automatically moves when a mouse button is held down while pointing at one of the slider ends. To achieve this simple animation, Tecate's clock object is used. When the mouse button is first depressed while the cursor is over a slider end, the slider indicator registers itself to be informed of clock ticks. From then on, at every clock tick, the indicator receives an update message from the clock, at which time the indicator repositions itself and increments or decrements the current slider value. When the mouse button is released, the slider sends a message to BigRiver indicating that a new isosurface is to be calculated and displayed. In addition, the slider indicator unregisters itself from the clock signaling that it no longer is to receive the update messages. In general, applications can use this same clock mechanism to perform more elaborate animations.

A 3-D World Wide Web Browser

In the Tecate Web browser, exploration of the World Wide Web and its contents occurs by placing an end user onto an informational landscape. This landscape is a 3-D virtual world whose appearance reflects the content and the structure of a designated subset of the entire Web. Upon application start-up, an end user is presented with an initial informational landscape that consists of a planar map of the earth embedded in a 3-D space, as shown in Figure 8. In general, the

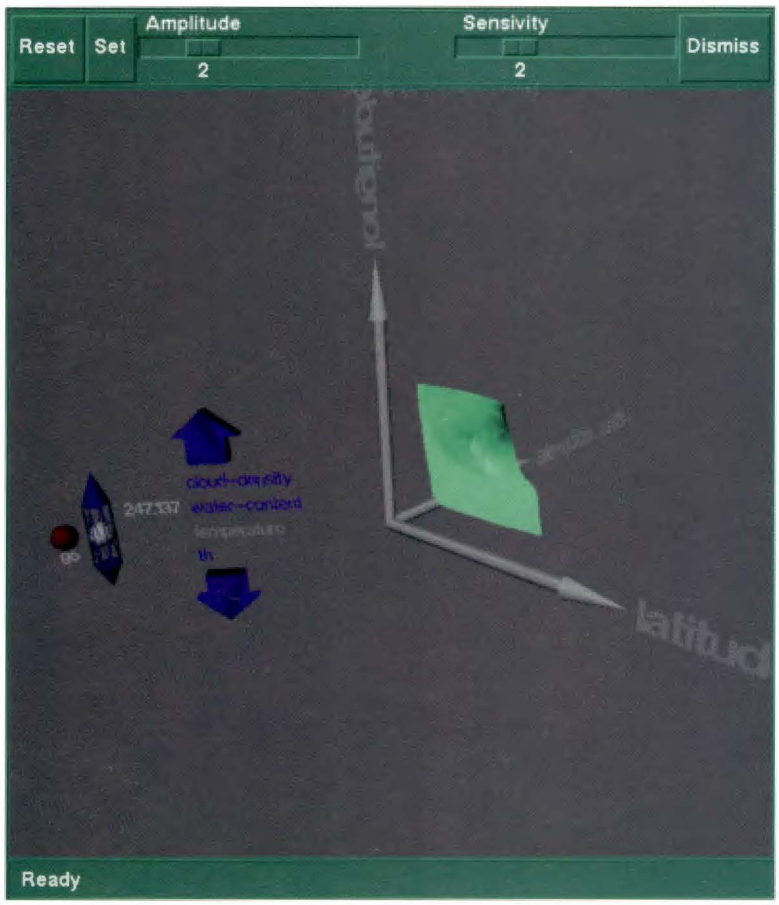


Figure 7
Task Editor Showing a Visualization of Regional Climate Data, Including an Isosurface and a User Interface Widget

initial informational landscape can be any 3-D scene and does not have to be geographically based. For instance, an informational landscape might be a virtual library where books on shelves serve as anchors for hyperlinks to different Web sites.

In the present browser application, selected Web sites appear as 3-D icons on the world map. These icons are positioned either in locations where Web servers physically reside or in locations referenced within Web documents (see Figure 8). A user places information that describes these sites into a database that serves as an elaboration of the hot list of current hypertext-based browsers. When the browser application is first started, it sends a query for the initial complement of Web sites to the Database Interface. The browser application then invokes a BigRiver script that visualizes the results by placing icons representing each site onto the world map.

Suspended above the world map is a 3-D user interface widget that is used to query a database of Web sites that are of interest to an end user (see Figure 8). This database, where the initial set of Web sites is stored, includes information such as URLs, keywords, geographical locations, and Web site types. Currently,

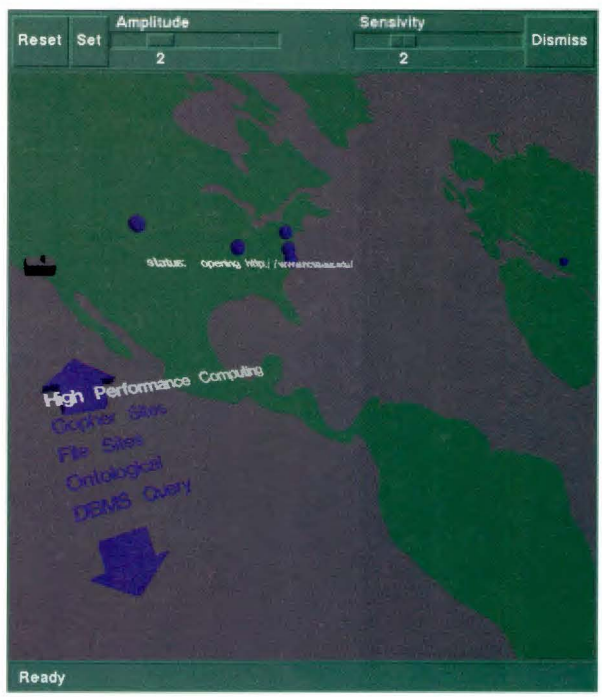


Figure 8
Tecate Web Browser Informational Landscape Showing WWW Sites Depicted as 3-D Icons on a Map of the World

individual users are responsible for maintaining their own databases by adding or removing Web site entries by hand. An automated means for building these databases can be easily added to the browser application so that Web information could be accumulated based on where and when an end user travels on the Web.

During a browsing session, the Web Query Tool allows arbitrary SQL queries to be posed to the database by an end user. In addition, the Web Query Tool has provisions to allow packaged queries to be initiated by a simple click of a mouse button. In both cases, queries are sent to the Database Interface for forwarding to the appropriate database server. The Database Interface packages up the query results as on-the-wire fiber bundles which are returned to the Web Query Tool. The Web Query Tool then invokes a BigRiver script, which converts the fiber bundle data into AVL code. This code, when interpreted by the Object Manager, creates a visualization of the Web sites that satisfies the query. Generally, a visualization such as this consists of placing on the world map a set of 3-D icons whose appearances are a function of the Web site type. However, query result visualizations need not be limited to an organization based on geographical position. For instance, a query for the contents of an end user's own file directory results in a new informational landscape that consists of an evenly spaced grid of icons suspended within a room, as shown in Figure 9.

Each icon that appears within an informational landscape is cloned from an AVL *Hyperlink* abstract object that stores its URL in a state variable. Each Web site icon inherits from the Hyperlink prototype a behavior that causes data pointed to by its URL state variable to be fetched by means of the WWW Interface when the icon is selected. When the data is drawn across the Web, Tecate's WWW Interface attempts to structure a visualization of it. Figure 10 summarizes the message flow between the more important objects within the Web browser application.

If an end user selects an icon and a Web server returns a stream of HTML, the WWW Interface translates the stream into AVL and displays the result on the base of an inverted pyramid whose apex is centered on the chosen icon (see Figure 11). The text and imagery resulting from the HTML appear similarly as they would when visualized using a hypertext-based browser like Netscape. Hyperlinks are represented as highlighted text, which the user can follow by selecting the text. These hyperlinks are Tecate objects that

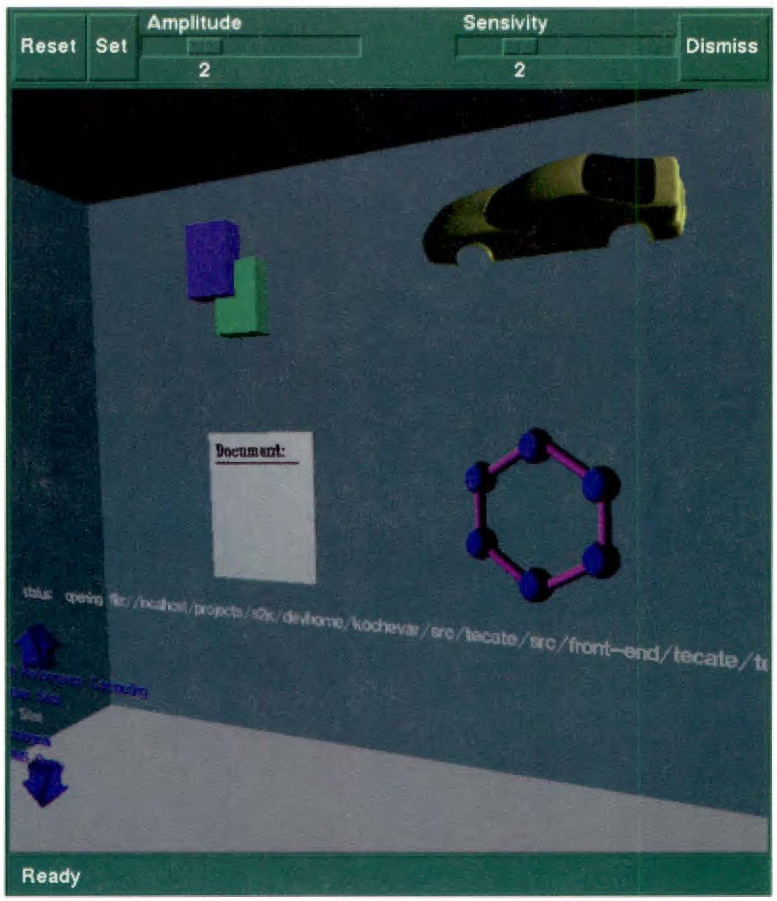


Figure 9
Sample End-user Nongeographical Informational Landscape

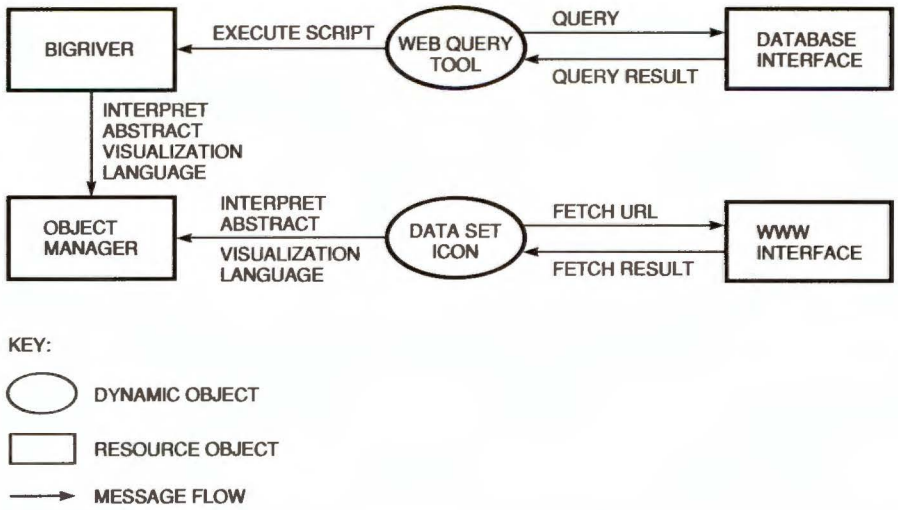


Figure 10
Message Flow between Important Objects in the Web Browser

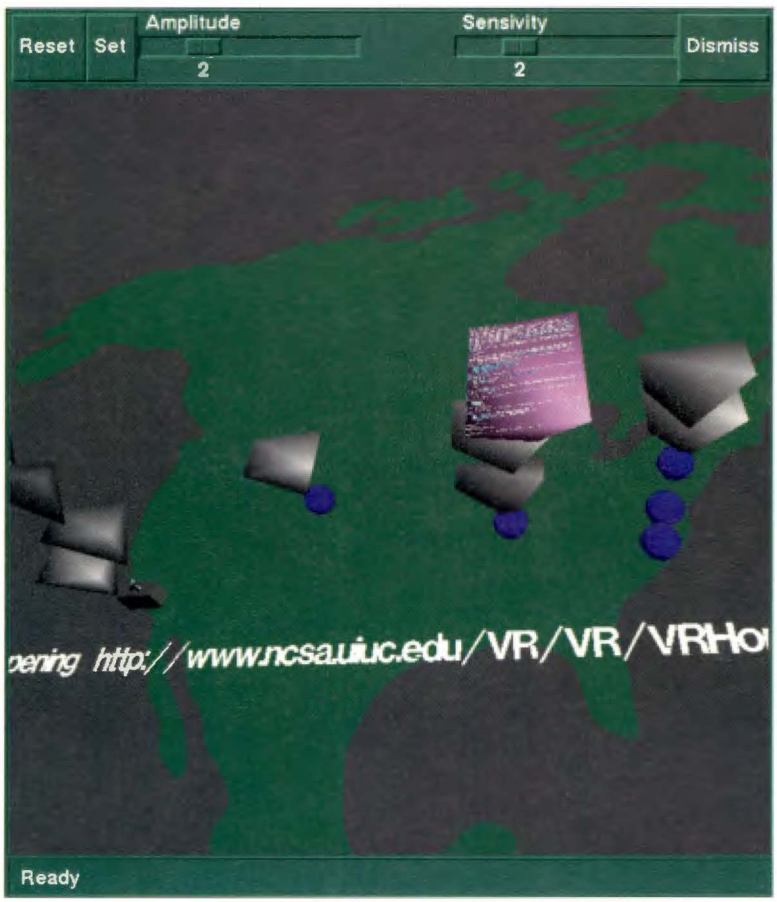


Figure 11
Results of a Tecate Browsing Session Showing a Hyperlink and a Forest of Pyramids That Represents the User's Travels on the Web

are cloned from the same Hyperlink prototype as the Web site icons. If another HTML document is retrieved by following a hyperlink, that document is viewed on the base of another inverted pyramid whose apex rests on the selected text and so on (see Figure 11). Rather than having to page back and forth between hypertext documents as with most hypertext-based browsers, in Tecate, an end user needs only to move about the virtual world to gain an appropriate viewpoint from which to examine a desired document. Overall, as shown in Figure 11, a browsing session with Tecate's Web browser results in a forest of pyramidal structures that represent a pictorial history of an end user's travels on the Web.

Although Tecate's Web browser is capable of viewing HTML documents, its main purpose is not to emulate what can currently be done using hypertext-based browsers, albeit using 3-D. Rather, the new browser is intended to visualize primarily more complex types of data. When data does not consist of a stream of HTML code, the WWW Interface attempts to visualize what was returned from the Web. These visualizations can take place in virtual worlds separate from the informational landscape from where the data request was initiated, or they can be placed within the original informational landscape. Figure 12 depicts an example of a Web document that has embedded within it a miniature virtual world containing a model of a car. An end user can freely interact with this model to initiate any behavior defined for objects populating the subworld. For instance, selecting the car with the mouse causes the car wheels to spin. Figure 13 shows the AVL code embedded in the HTML page for the Web document shown in Figure 12.

Conclusions

Tecate provides the infrastructure on which applications can be created for browsing and visualizing data from networked data sources. Architecturally, Tecate seeks to bring together into one package useful features found in visualization systems, network browsers, database front ends, and virtual reality systems. As a first prototype, Tecate was created using a breadth-first development strategy. That is, developers deemed it essential to first understand what components were needed to build a general data space exploration utility and then determine how those components interact.

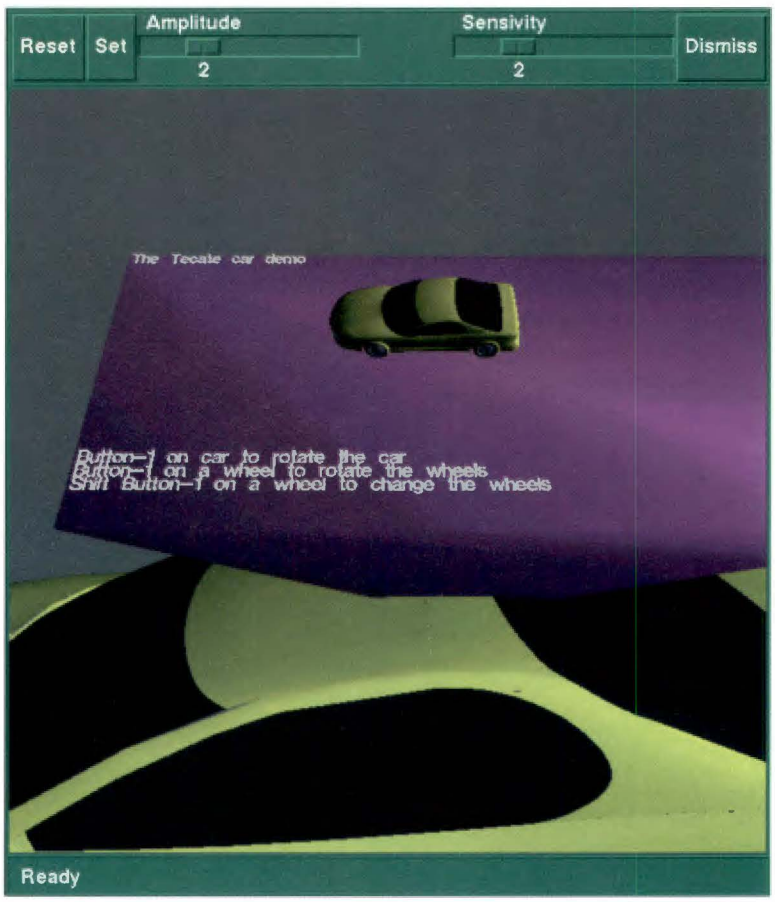


Figure 12
Example of a Web Document with Embedded 3-D Virtual World

```
<HEAD>
<TITLE>The Tecate car demo</TITLE>
</HEAD>

<BODY>

<H1>The Tecate car demo</H1>

<AVL>
# Global variables
global TEC_WEB_PARENT TEC_WEB_WIN
set path "/projects/s2k/sharedata"

# Define car part prototype
clone CarPart Visual
add CarPart {
   state {angle 10}
   appearance {
      repType surface
      interpType surface
   }
   behavior {
      around {args} {
         for {set i 0} {$i < 360} {incr i [getstate angle]} {
            send [getself] rotate "add 0 [getstate angle] 0"
         }
      }
   }
}

# Define car body
clone car_body CarPart
add car_body {
   appearance {
      replacematrix {rotate {0.0 0.0 90.0}}
      shape {AliasObj "$path/car_body.tri"}
   }
}

# Define generic wheel
clone wheel CarPart
        .
        .
        .

# Define car's four wheels
clone back_right CarPart
        .
        .
        .

# Assemble car
clone wheels CarPart
add wheels {subobject {back_right back_left front_left front_right}}
clone car CarPart
add car {
  appearance {replacematrix {translate {28.0 -8.0 3.0} rotate {90.0 90.0 0.0}}}
  subobject {car_body wheels}
}
add $TEC_WEB_PARENT {subobject {car}}

# Bind pick events to car
send $TEC_WEB_WIN addEvent {wheel {Pick-Shift-Button-1 {rot_wheels {}}}}
send $TEC_WEB_WIN addEvent {wheel {Pick-Button-1 {around {}}}}
send $TEC_WEB_WIN addEvent {car {Pick-Button-1 {around {}}}}
</AVL>

<PRE>
Button-1 on car to rotate the car <BR>
Button-1 on a wheel to rotate the wheels <BR>
Shift Button-1 on a wheel to change the wheels <BR>
</PRE>

<HR>
<P>
</BODY>
```

Figure 13
AVL Code Embedded in the HTML Page for the Web Document Example

This development strategy traded off the functionality of individual components for the completeness of a fully running visualization system.

In terms of achieving its design goals, the Tecate effort has been moderately successful. Tecate can now provide interfaces to two kind of data spaces: the World Wide Web and databases managed by the POSTGRES and Illustra database management systems. In addition, interfaces to other data spaces can be implemented easily by creating new resource objects using the tools provided by Tecate. Much work still needs to be done, however. For example, the attendant data translation problem must be satisfactorily solved; data passing through an interface that is stored in one format should be automatically converted into Tecate's favored format and vice versa.

When building visualizations of data, Tecate now understands data that has a specific conceptual structure, in particular, arbitrary sets of tuples and multidimensional arrays where array elements may be tuples. Although data types from many different disciplines possess such a structure, some types remain that do not, for instance, data that has a lattice-like or polyhedral structure. Furthermore, Tecate can now construct only crude visualizations of the data types that it does understand. The primary reason for this shortcoming is that the basic module set within the BigRiver resource is incomplete, and the knowledge base within the Intelligent Visualization System contains limited knowledge of visualization techniques that can be used to transform data into virtual worlds.

At present, Tecate does dynamically craft simple user interfaces and interactive visualizations using its Intelligent Visualization System. This expert system takes into account how data is conceptually structured and end-user tasks regarding what is to be understood from the data. Still, the Intelligent Visualization System does not yet consider data semantics, end-user preferences, or display system characteristics when building visualizations. Nonetheless, Tecate does provide the capabilities to create highly interactive applications. Sophisticated event handling constructs are built into AVL, and the Intelligent Visualization System uses those features to automatically place user interface widgets into the virtual worlds it specifies.

Regarding future work, hopefully, succeeding generations of the Tecate system will include many new features and enhancements. The management of objects needs to be reworked so that thousands of objects can be efficiently handled simultaneously. Although Tecate now builds virtual worlds, virtual reality gadgetry has yet to be integrated into the system. The Abstract Visualization Language needs new features, and it needs to be streamlined. Tecate can also benefit greatly from a more complete toolkit of 3-D widgets that can be used to interact with objects within virtual worlds. Finally, the Doré graphics system that Tecate uses should be replaced with a more mainstream system like OpenGL, which will allow Tecate to run on a wide variety of hardware platforms.

Tecate is an exciting system to use and an excellent foundation from which to pursue further research and development in the exploration of general data spaces. Tecate advances the state of the art by demonstrating a comprehensive means to graphically browse for data and then interactively visualize data sets that are selected. Tecate accomplishes these tasks by using an expert system that automatically builds virtual worlds and by exploiting the flexibility of an interpretive, object-oriented language that describes those worlds.

Acknowledgments

The work described in this paper was supported by Digital Equipment Corporation, the University of California, and the San Diego Supercomputer Center as part of the Sequoia 2000 project. We would like to give special thanks to Frank Araullo, Mike Kelley, Jonathan Shade, and Colin Sharp for their help in constructing the Tecate prototype.

References

1. *AVS User's Guide* (Waltham, Mass.: Advanced Visual Systems Inc., May 1992).
2. *Khoros User's Manual* (Albuquerque, N. Mex.: The Khoros Group, Department of Electrical and Computer Engineering, University of New Mexico, 1992).
3. *IBM Visualization Data Explorer: User's Guide* (Armonk, N.Y.: International Business Machines Corporation, 1992).
4. R. Pausch et al., "Alice: A Rapid Prototyping System for Virtual Reality," Course Notes #2: Developing Advanced Virtual Reality Applications, *Proceedings of the ACM SIGGRAPH '94 Conference* (1994).
5. *Object Modeling Language (OML) Programmer's Manual* (Edmonton, Alberta, Canada: Department of Computing Science, University of Alberta, 1992).
6. P. Strauss and R. Carey, "An Object-oriented 3D Graphics Toolkit," *Proceedings of the ACM SIGGRAPH '92 Conference* (1992).
7. *Doré Programmer's Guide* (Santa Clara, Calif.: Kubota Graphics Corporation, 1994).
8. D. Ungar and R. Smith, "Self: The Power of Simplicity," *SIGPLAN Notices,* vol. 22, no. 12 (December 1987): 227–241.
9. P. Kochevar, "Programming in Tecate," available on the Internet at http://www.sdsc.edu/SDSC/Research/Visualization/Tecate/tecate.html (May 1995).
10. J. Ousterhout, "Tcl: An Embeddable Command Language," *Proceedings of the 1990 Winter USENIX Conference* (1990).

11. G. Bell, A. Parisi, and M. Pesce, "The Virtual Reality Modeling Language Specification," available on the Internet at http://vrml.wired.com (November 1994).

12. J. White, "Telescript Technology: The Foundation for the Electronic Marketplace," General Magic white paper (Sunnyvale, Calif.: General Magic, Inc., 1994).

13. J. McKeehan and N. Rhodes, *Programming for the Newton: Software Development with NewtonScript* (Cambridge, Mass.: Academic Press Professional, 1994).

14. Adobe Systems Incorporated, *PostScript Language Reference Manual* (Reading, Mass.: Addison-Wesley Publishing Company, 1990).

15. L. Wanger, "Writing Tecate Resources," available on the Internet at http://www.sdsc.edu/SDSC/Research/Visualization/Tecate/tecate.html (May 1995).

16. S. Casner, "A Task-analytic Approach to the Automated Design of Graphic Presentations," *ACM Transactions on Graphics,* vol. 10, no. 2 (April 1991): 111–151.

17. E. Ignatius and H. Senay, "Visualization Assistant," *Proceedings of the IEEE Visualization Workshop on Intelligent Visualization Systems* (October 1993).

18. J. Mackinlay, "Automating the Design of Graphical Presentations of Relational Information," *ACM Transactions on Graphics,* vol. 5, no. 2 (1986): 110–141.

19. H. Senay and E. Ignatius, "VISTA: A Knowledge-based System for Scientific Data Visualization," Technical Report GWU-IIST-92-10 (Washington, D.C.: George Washington University, March 1992).

20. Z. Ahmed et al., "An Intelligent Visualization System for Earth Science Data Analysis," *Journal of Visual Languages and Computing* (December 1994).

21. P. Kochevar et al., "An Intelligent Assistant for Creating Data Flow Visualization Networks," *Proceedings of the AVS '94 Conference* (1994).

22. D. Butler and M. Pendley, "A Visualization Model Based on the Mathematics of Fiber Bundles," *Computers in Physics,* vol. 3, no. 5 (September/October 1989).

23. D. Butler and S. Bryson, "Vector-bundle Classes Form Powerful Tool for Scientific Visualization," *Computers in Physics,* vol. 6, no. 6 (November/December 1992): 576–584.

24. R. Haber, B. Lucas, and N. Collins, "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids," *Proceedings of the Visualization '91 Conference* (1991).

25. L. Resnick et al., *CLASSIC Description and Reference Manual for the Common LISP Implementation* (Murray Hill, N.J.: AT&T Bell Laboratories, 1993).

26. G. Steele, Jr., *Common LISP: The Language, Second Edition* (Bedford, Mass.: Digital Press, 1990).

27. *AVS/Express Developer's Reference* (Waltham, Mass.: Advanced Visual Systems Inc., June 1994).

28. M. Stonebraker and G. Kemnitz, "The POSTGRES Next-generation Database Management System," *Communications of the ACM* (October 1991): 78–92.

29. *Using Illustra* (Oakland, Calif.: Illustra Information Technologies, Inc., June 1994).

30. J. Ousterhout, "An X11 Toolkit Based on the Tcl Language," *Proceedings of the 1991 Winter USENIX Conference* (1991).

Biographies

Peter D. Kochevar
Peter Kochevar is a principal software engineer in Digital's External Research Program. From 1992 to 1994, he led the data visualization research efforts of the Sequoia 2000 project, which were undertaken at the San Diego Supercomputer Center (SDSC). Currently, Peter is a visiting scientist at the SDSC, where he leads researchers in developing interactive data visualization systems. Peter joined Digital in 1990 as a member of the Workstations Engineering Group. In earlier work, he was a software engineer for the Boeing Commercial Airplane Company. Peter received a B.S. (1976) in mathematics from the University of Michigan and an M.S. (1982) in mathematics from the University of Utah. He also holds M.S. and Ph.D. degrees in computer science from Cornell University.

Leonard R. Wanger
Len Wanger is the head of development at Interactive Simulations, Inc., working on interactive molecular modeling tools. He is also a member of the Computer Science Department staff at the University of California, San Diego, where he researches next-generation visualization systems at the San Diego Supercomputer Center. He received a B.S. in computer science from the University of Iowa in 1987 and an M.S. in architectural science from Cornell University in 1991. His research interests include visual front ends to simulations, database support for visualization systems, navigation in virtual environments, and the perception of complex data spaces.

High-performance I/O and Networking Software in Sequoia 2000

Joseph Pasquale
Eric W. Anderson
Kevin Fall
Jonathan S. Kay

The Sequoia 2000 project requires a high-speed network and I/O software for the support of global change research. In addition, Sequoia distributed applications require the efficient movement of very large objects, from tens to hundreds of megabytes in size. The network architecture incorporates new designs and implementations of operating system I/O software. New methods provide significant performance improvements for transfers among devices and processes and between the two. These techniques reduce or eliminate costly memory accesses, avoid unnecessary processing, and bypass system overheads to improve throughput and reduce latency.

In the Sequoia 2000 project, we addressed the problem of designing a distributed computer system that can efficiently retrieve, store, and transfer the very large data objects contained in earth science applications. By very large, we mean data objects in excess of tens or even hundreds of megabytes (MB). Earth science research has massive computational requirements, in large part due to the large data objects often found in its applications. There are many examples: an advanced very high-resolution radiometer (AVHRR) image cube requires 300 MB, an advanced visible and infrared imaging spectrometer (AVIRIS) image requires 140 MB, and the common land satellite (LANDSAT) image requires 278 MB. Any throughput bottleneck in a distributed computer system becomes greatly magnified when dealing with such large objects. In addition, Sequoia 2000 was an experiment in distributed collaboration; thus, collaboration tools such as videoconferencing were also important applications to support.

Our efforts in the project focused on operating system I/O and the network. We designed the Sequoia 2000 wide area network (WAN) test bed, and we explored new designs in operating system I/O and network software. The contributions of this paper are twofold: (1) it surveys the main results of this work and puts them in perspective by relating them to the general data transfer problem, and (2) it presents a new design for container shipping. (For a complete discussion of container shipping, see Reference 1.) Since container shipping is a new design, this paper devotes more space to it in relation to the other surveyed work (whose detailed descriptions may be found in References 2 to 9). In addition to this work, we conducted other network studies as part of the Sequoia 2000 project. These include research on protocols to provide performance guarantees and multicasting.[10–17]

To support a high-performance distributed computing environment in which applications can effectively manipulate large data objects, we were concerned with achieving high throughput during the transfer of these objects. The processes or devices representing the data sources and sinks may all reside on the same workstation (single node case), or they may be distributed over many workstations connected by the network

(multiple node case). In either case, we wanted applications, be they earth science distributed computations or collaboration tools involving multipoint video, to make full use of the raw bandwidth provided by the underlying communication system.

In the multiple node case, the raw bandwidth is from 45 to 100 megabits per second (Mb/s), because the Sequoia 2000 network used T3 links for long-distance communication and a fiber distributed data interface (FDDI) for local area communication. In the single node case, the raw bandwidth is approximately 100 megabytes per second, since the workstation of choice was one of the DECstation 5000 series or the Alpha-powered DEC 3000 series, both of which use the TURBOchannel as the system bus.

Our work focused only on software improvements, in particular how to achieve maximum system software performance given the hardware we selected. In fact, we found that the throughput bottlenecks in the Sequoia distributed computing environment were indeed in the workstation's operating system software, and not in the underlying communication system hardware (e.g., network links or the system bus). This problem is not limited to the Sequoia environment: given modern high-speed workstations (100+ millions of instructions per second [mips]) and fast networks (100+ Mb/s), performance bottlenecks are often caused by software, especially operating system software. System software throughput has not kept up with the throughputs of I/O devices, especially network adapters, which have improved tremendously in recent years. These technology improvements are being driven by a new generation of applications, such as interactive multimedia involving digital video and high-resolution graphics, that have high I/O throughput requirements. Supporting these applications and controlling these devices have taxed operating system technology, much of which was designed during times when intensive I/O was not an issue.

In the next section of this paper, we describe the Sequoia 2000 network, which served as an experimental test bed for our work. Following that, we analyze the data transfer problem, which serves as the context for the three subsequent sections. There we describe our solutions to the data transfer problem. Finally, we present our conclusions.

The Sequoia 2000 Network Test Bed

The Sequoia 2000 network is a private WAN that we designed to span five campuses at the University of California: Berkeley, Davis, Los Angeles, San Diego, and Santa Barbara. The topology is shown in Figure 1. The backbone link speeds are 45 Mb/s (T3) with the exception of the Berkeley-Davis link, which is 1.5 Mb/s (T1). At each campus, one or more FDDI

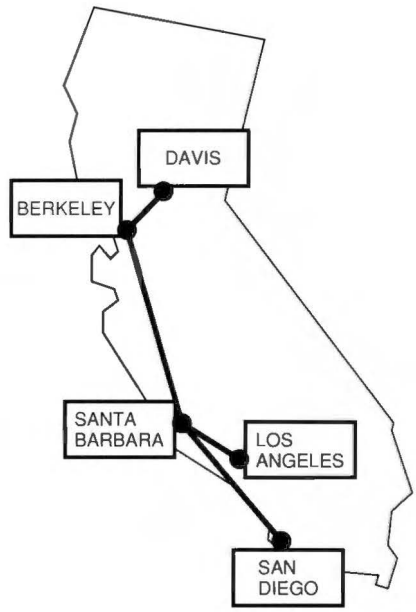


Figure 1
Sequoia 2000 Research Network

local area networks (LANs) that operate at 100 Mb/s are used for local distribution. At some campuses, the configuration is a hierarchical set of rings. For example, at UC San Diego, one FDDI ring covered the campus and joined three separate rings: one at the Computer Systems Lab (our laboratory) in the Department of Computer Science and Engineering, one at the Scripps Institution of Oceanography, and one at the San Diego Supercomputer Center.

We used high-performance general-purpose computers as routers, originally DECstation 5000 series and later DEC 3000 series (Alpha-powered) workstations. Using workstations as routers running the ULTRIX or the DEC OSF/1 (now Digital UNIX) operating system provided us with a modifiable software platform for experimentation. The T3 (and T1) interface boards were specially built by David Boggs at Digital. We used off-the-shelf Digital products for FDDI boards, both models DEFTA, which supports both send and receive direct memory access (DMA), and DEFZA, which supports only receive DMA.

The Data Transfer Problem

Since a data source or sink may be either a process or device, and the operating system generally performs the function of transferring data between processes and devices, understanding the bottlenecks in these operating system data paths is key to improving performance. These data paths generally involve traversing numerous layers of operating system software. In the case of network transfers, the data paths are extended by layers of network protocol software.

To understand the performance problem we were trying to solve, consider a common client-server interaction in which a client has requested data from a server. The data resides on some source device, e.g., a disk, and must be read by the server so that it may send the data to the client over a network. At the client, the data is written to some sink device, e.g., a frame buffer for display.

Figure 2 shows a typical end-to-end data path where the source and sink end-point workstations are running protected operating system kernels such as UNIX. The source device generates data into the memory of its connected workstation. This memory is generally only addressable by the kernel; to allow the server process to access the data, it is physically copied into memory addressable via the server process's address space, i.e., user space. Physically copying data from one memory location to another (or more generally, touching the data for any reason) is a major bottleneck in modern workstations.

In travelling through the kernel, the data generally travels over a device layer and an abstraction layer. The device layer is part of the kernel's I/O subsystem and manages the I/O devices by buffering data between the device and the kernel. The abstraction layer comprises other kernel subsystems that support abstractions of devices, providing more convenient services for user-level processes. Examples of kernel abstraction layer software include file systems and communication protocol stacks: a file system converts disk blocks into files, and a communication protocol stack converts network packets into datagrams or stream segments. Sometimes, a kernel implementation may cause physical copying of data between the device layer and the abstraction layer; in fact, copying may even occur within these layers.

From kernel space, the data may travel across several more layers in user space, such as the standard I/O layer and the application layer. The standard I/O layer buffers I/O data in large chunks to minimize the number of I/O system calls. The application layer generally has its own buffers where I/O data is copied.

From the server process in user space, the data is then given to the network adapter; this may cause transfers across user process layers and then across the kernel layers. The data is then transferred over the network, which generally consists of a set of links connected by routers. If the routers have kernels whose software structure is like that described above, a similar (but typically simpler) intramachine data transfer path will apply.

Finally, the data arrives at the client's workstation. There, the data travels in a similar way as was described for the server's workstation: from the network adapter, across the kernel, through the client process's address space, and across the kernel again, finally reaching the sink device.

From this analysis, one can surmise why throughput bottlenecks often occur at the end points of the end-to-end data transfer path, assuming sufficiently fast hardware devices and communication links. At the end points, there may be significant data copying as the data traverses the various software layers, and there is protection-domain crossing (kernel to user to kernel), among other functions. The overheads caused by these functions, directly and indirectly, can be significant.

Consequently, we focused on improving operating system I/O and network software, including optimizations for the four possible process/device data transfer scenarios: process to process, process to device, device to process, and device to device, with special care in addressing cases where either source or sink

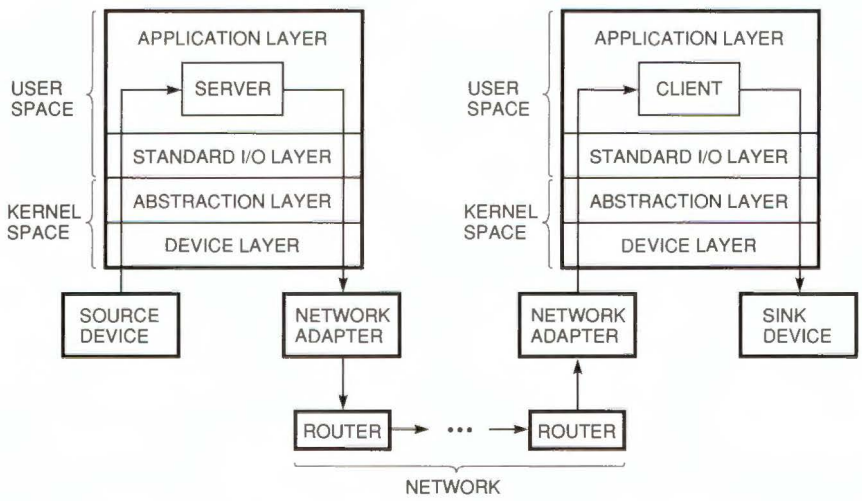


Figure 2
An End-to-End Data Path from a Source Device on One Workstation to a Sink Device on Another Workstation

device is a network adapter. In this paper, we use the term *data transfer problem* to refer to the problem of reducing these overheads to achieve high throughput between a source device and a sink device, either of which can be a network adapter within a single workstation.

Although the data transfer problem may also exist in intermediate routers, it does so to a much lesser degree than with end-user workstations (assuming modern router software and hardware technology). This is because of a router's simplified execution environment and its reduced needs for transfers across multiple protected domains. However, there is nothing that precludes the application of the techniques discussed in this paper to router software. In fact, since we used general-purpose workstations for routers to support a flexible, modifiable test bed for experimentation with new protocols, our work was also applied to router software.

In the next three sections, we describe various approaches to solving the data transfer problem. Since data copying/touching is a major software limitation in achieving high throughput, avoiding data copying/touching is a constant theme. Much of our work involves finding ways to avoid or limit touching the data without sacrificing the flexibility or protection commonly provided by most modern operating systems.

We describe two solutions to the data transfer problem that avoid all physical copying and are based on the principle of providing separate mechanisms for I/O control and data transfer.[18–21] The reader will see that while these two solutions are based on different approaches (indeed, they can even be viewed as competing), they fill different niches based on differing assumptions of how I/O is structured. In other words, each is appropriate and optimal for different situations. In addition to the data transfer problem, we address a special problem—the bottleneck created by the checksum computation for I/O on a network using the transmission control protocol/internet protocol (TCP/IP).

Container Shipping

Container shipping is a kernel service that provides I/O operations for user processes. High performance is obtained by eliminating the in-memory data copies traditionally associated with I/O. Additional gains are achieved by permitting the selective accessing (mapping) of data. Finally, the design we present makes possible specific optimizations that further improve performance.

The goals of the container shipping model of data transfer for I/O are to provide high performance without sacrificing protection and to fully support the principle of general-purpose computing. Full access to I/O data by user-level processes has long been a standard feature of operating systems. This ability has traditionally been provided by copying data to and from process memory at each instance when data is transferred. The divergence of CPU and dynamic random access memory (DRAM) speeds makes this in-memory copying more inefficient and costly every year. This problem is often attacked with application-specific silicon or kernel modifications. A less-costly and longer-lasting solution is to redesign the I/O subsystem to provide copy-free I/O. Container shipping provides this ability, as well as additional performance gains, in a uniform, general, and practical way.

Containers

A container is one or more pages of memory. In these pages, it may contain a single block of data, whose location is identified by an offset and a length. When a container is mapped into an address space, the pages form a contiguous region of memory, where the data can be manipulated. A container can be owned by one and only one domain, e.g., some user process or the kernel itself, at any single point in time. The owning domain may map the container for access. When access is not required, mapping can be avoided, which saves time.

User-level processes use container shipping system calls to perform the following functions:

- Allocation: cs_alloc and cs_free allocate and deallocate containers and their resources (e.g., physical pages).
- Transfer: cs_read and cs_write perform I/O using containers.
- Mapping: cs_map and cs_unmap allow a process to access the data in a container.

The cs_read and cs_write calls take as parameters an I/O path identifier (such as a UNIX file descriptor), a data size, and parameters describing a list of containers, or a return area for such a list. Several options are also available, such as one for cs_read that immediately maps all the resulting containers. Data is never copied within memory to satisfy cs_read and cs_write, so all I/O performed this way is copy-free.

Because the mapping of containers is always optional, a process can move data from one device to another without mapping it at all. When containers of data flow through a pipeline of several processes, substantial additional savings can be obtained if several of the processes do not map the containers, or if they map only some of the containers.

Although container shipping has six different system calls versus the two of conventional I/O, read and write, the actual number of calls a process issues with container I/O may be no greater than with conventional I/O. When data is not mapped, only cs_read and cs_write calls are required. Even if data is mapped, it may be possible to perform the mapping through

flags to cs_read, without calling cs_map. Unmapping is automatic in cs_write, so if cs_unmap is not used, two system calls are still sufficient.

As shown in Figure 3, a process reads data in a container from one device and writes it to another device. Three pages of memory form one container that stores two and one-half pages of data. On input (cs_read), the source device deposits data into physical memory pages forming the container. The process that owns the container may then map (cs_map) it so that the data can be manipulated in its address space. The data is then output (cs_write) to the sink device. Output can occur without having mapped the container. Mapping can also occur automatically on cs_read.

Eliminating In-Memory Copying

Unconditionally avoiding the copying of data within memory during I/O leads to the first of several performance gains from container shipping. Other solutions exist that avoid copies only in limited cases. To be uniform and general, copy-free I/O must be possible without restrictions due to the devices used, the order of operations, or the availability of special device hardware.

In many I/O operations, the data requested by a user-level process is already in system memory when the request is made. This situation can arise when data is moving between two processes via the I/O system, such as is done with pipes. Many optimized file systems perform read-ahead and in-memory caching to improve performance, so file I/O requests may also be satisfied with data that is already in memory. Finally, conventional network adapters transfer entire packets into memory before they are examined by protocol layers in the kernel. Only after protocol processing can this data be delivered to the correct user-level process. When requested data is already in memory, the only possible copy-free transfer mechanism that allows full read/write access in the address space of a process is virtual memory remapping. Techniques that rely on device-specific characteristics such as programmable DMA or outboard protocol processors cannot provide uniform, device-independent copy-free I/O, because these mechanisms cannot transfer data that is already in memory.

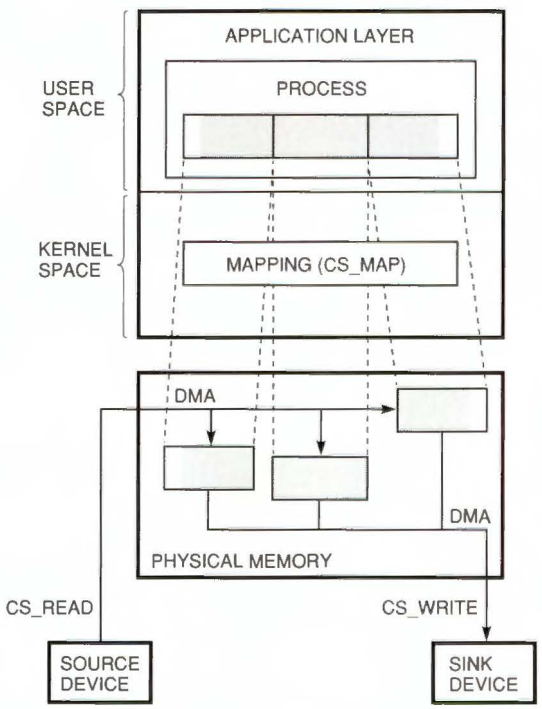


Figure 3
Container Shipping Transfer and Mapping

Using virtual memory remapping, container shipping can perform copy-free I/O regardless of when or where data arrives in memory, and with or without any special device hardware that might be available. Virtual memory hardware is employed to control the ownership of, and access to, memory that contains I/O data. Ownership and access rights are transferred between domains when container I/O is performed, while data sits motionless in memory. This technique requires no special assistance from devices and applies to interprocess communication as well as all physical I/O. Because user-level processes retain complete access to I/O data with no in-memory copying, user-level programming remains a practical solution for high-performance systems.

The Gain/Lose Model

In container I/O, reading and writing are coupled with the gain and loss of memory. We chose the gain/lose model because it is simple and provides higher performance without sacrificing protection. Shared memory is a more complicated alternative to the gain/lose model, which also avoids data copying. The use of shared memory to allow a set of processes to efficiently communicate, however, reduces the protection between domains. Shared-memory I/O schemes also tend to be complicated because of the close coordination required between a user process and the kernel when they both manipulate a shared data pool. Since data is never shared under the gain/lose model, protection domains need not be compromised, and less user/kernel cooperation and trust is required.

The gain/lose model has three major implications for programmers. First, a process must dispose of I/O data that it gains, or memory consumption may grow rapidly. One way to dispose of data is to perform a cs_write operation on it, so a process performing matched reads and writes on a stream of data will not accumulate any extra memory. Second, to avoid seriously complicating conventional memory models, not all memory is eligible for use in write operations. For example, writing data from the stack would leave an inconvenient hole in that part of the virtual memory, so this is not allowed. Finally, because data that is

written is lost, a writing process must copy any data that will still be needed after the write. Fortunately, applications that move great volumes of data often have no further need for it after a write is completed.

Implications of Virtual Memory Remapping

In addition to the use of the gain/lose model, the decision to use virtual memory remapping has substantial implications for the design and use of an I/O system. Several changes are unavoidably visible to programmers. For example, data can no longer be placed exactly at any requested location in an address space. Virtual memory remapping can change the virtual page in which a physical page of memory appears, but it cannot realign data within a page. Furthermore, mapping can rearrange memory only at page boundaries. The exact location where incoming I/O data is placed is determined by the kernel. After a read operation is complete, a process can discover the address of the data and access it at that address.

Some kinds of I/O place data in memory in a form that differs from the way it is presented to user-level processes. For example, network packets may arrive with media-level headers that are not seen by higher levels. These packets may also arrive out of order, or in fragments that collectively form a single message. Without help from an outboard protocol processor or the use of in-memory copying, these packets cannot be linearized. With container shipping, a process may be required to accept a message that consists of multiple fragments in memory. The semantics of the communication do not change, but the data representation differs. This issue is less troublesome for writes, because kernels typically use internal structures to reorganize network data without copying it. The mbufs found in UNIX are an example of such a kernel structure.

Virtual memory remapping is not a simple technique, and it must be used with care to achieve high performance. Although remapping a page is almost always faster than copying it, remapping also consumes time. This time comes from kernel virtual memory bookkeeping and from side effects (such as translation lookaside buffer [TLB] flushes) of address space changes. For these reasons, container shipping makes all mapping optional. Some operating systems such as Mach perform lazy mapping, using the page fault mechanism to map pages when they are first accessed.[22] This technique avoids unnecessary map operations but incurs the extra penalty of having to map on demand while a program waits for access to data. Taking one page fault for every page in a large region, as is common in modern systems, is particularly expensive. Furthermore, lazy mapping still requires the setting of page table entries (and possibly other data structures) to prepare for the possibility of page faults, which can be costly for very large data objects. This cost is avoided in container shipping.

Optimizations

The container shipping design makes possible optimizations beyond copy and map elimination. Some make use of the fact that I/O often flows through pathways that are predictable. Other optimizations are possible on a per-container basis.

High-speed I/O is often generated by long-running processes, such as multimedia applications, real-time data processing, or processes that run for a long time merely by virtue of processing a very large data object (common in Sequoia applications). This I/O typically flows through pathways in the system that are essentially static. Data enters through one device, moves through a fixed set of domains, and leaves through another device. Kernel awareness of this locality can be used to optimize some container operations.

An I/O path through which same-sized containers move repeatedly offers the opportunity to recycle containers and their associated data structures. Per-transfer cost can be reduced by reusing the same set of pages and reusing page tables and address space. To perform recycling, the kernel can keep track of which containers were given to which processes, or the kernel can match up recycled containers by size or by device type.

In a system with a large secondary cache, promptly recycling a just-written container may allow its reuse while its data is still in the cache. In the best case, all data may be automatically cached because of this recycling. For example, DMA operations in DEC 3000-series systems update the secondary cache. Because this cache is much faster than main memory, the data can now be accessed more quickly.

Even without identifying an I/O pathway, careful tracking of the contents of container memory pages can allow savings in security-driven zero fills. A just-freed page consists entirely of sensitive data; the entire page must be cleaned before it can be given to any other user. But if this page is used as the target of a data-generating operation such as a DMA, only the part not overwritten needs to be zeroed. Furthermore, this zeroing can be postponed until the data is mapped; thus it may be avoided completely. If filling memory with zeroes causes it to be loaded in the cache, zeroing immediately before the map offers a cache benefit, because the data may be used shortly after it is mapped.

Container Shipping Implementation and Performance

Container shipping has been implemented in DEC OSF/1 version 2.0 (now Digital UNIX) on Alpha-powered DEC 3000-series workstations. All six system calls are supported, and container I/O can be measured in a variety of situations. Conventional UNIX I/O remains, so a system can boot and run normally, using container I/O only for specific experiments.

In our early paper, we showed significant throughput improvements for container-based interprocess

communication (IPC) within the ULTRIX version 4.2a operating system on a DECstation 5000/200 system.[1] With the new DEC OSF/1 implementation on Alpha workstations, we compared the I/O performance of conventional UNIX I/O to that of container shipping for a variety of I/O devices as well as IPC. These experiments are described in detail elsewhere.[23] Large improvements in throughput were observed, from 40 percent for FDDI network I/O (despite large non–data-touching protocol and device-driver overheads) to 700 percent for socket-based IPC.

We devised an experiment that exercises both the IPC and I/O capabilities of container shipping. Images (640 × 480 pixels, 1 byte per pixel) are sent by one process and received by a second process using socket IPC. The receiver process then does output to a frame buffer to display the images on the screen. This is a common application in the Sequoia project: viewing an animation composed of images displayed at a rate of up to 30 frames per second (fps). In fact, scientists often want to view as many simultaneously displayed animations as possible.

We carried out this experiment first using conventional UNIX I/O (i.e., read and write) and then using container shipping (i.e., cs_read and cs_write). Figure 4 shows the throughput obtained for a sender process transferring data to a receiver process, which then outputs the data to a frame buffer. The improvement of container shipping over UNIX I/O is almost 400 percent. Assuming the maximum 30 fps rate, conventional I/O supports the full display of one animation and container I/O supports six. In general, the greater the relative speed between an I/O device and memory, the greater the relative throughput of container shipping versus UNIX I/O will be.

Related Work

The use of virtual transfer techniques to avoid the performance penalty of physical copying goes back to TENEX.[24] Mach (like TENEX) uses virtual copying, i.e., transferring a data object by mapping it in the new address space, and then physically copying if the data is modified (copy-on-write).[22] This differs from container shipping, which uses virtual moving; i.e., the data object leaves the source domain and appears in the destination domain, where it can be read and written without causing fault handling, which is expensive. If the original domain wants to keep a copy, it may do so explicitly. Thus, container shipping places a greater burden on the programmer in return for improved performance.

The two systems that are most similar to container shipping are DASH and Fbufs.[25,26] Containers are similar to the IPC pages used in DASH and the fast buffers used by Fbufs. DASH provides high-performance interprocess communication: it achieves fast, local IPC by means of page remapping, which allows processes to own regions of a restricted area of a shared address space. The Fbufs system uses a similar technique, enhanced by caching the previous owners of a buffer, allowing reuse among trusted processes and eliminating memory management unit (MMU) updates associated with changing buffer ownership. The differences between these two systems and container shipping are examined in detail elsewhere.[23]

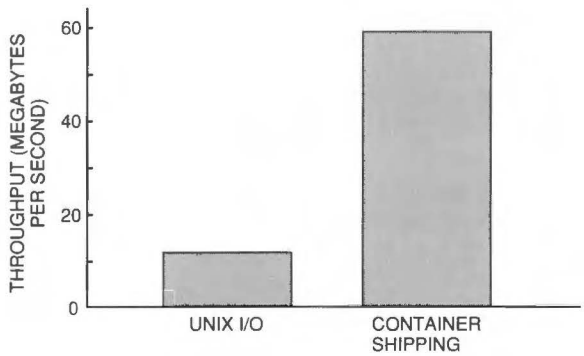


Figure 4
Throughput of IPC and Frame Buffer Output

Peer-to-Peer I/O

In addition to container shipping, we have investigated an alternative I/O system software model called peer-to-peer I/O (PPIO). As a direct result of the structure of this model, its implementation avoids the well-known overheads associated with data copying. Unlike other solutions, PPIO also reduces the number of context-switch operations required to perform I/O operations. In contrast to container shipping, PPIO is based on a streaming approach, where data is permitted to flow between a producer and consumer (these may be devices, files, etc.) without passing through a controlling process' address space. In PPIO, processes use the splice system call to create kernel-maintained associations between producer and consumer. Splice represents an addition to the conventional operating system I/O interfaces and is not a replacement for the read and write system functions.

The Splice Mechanism

The splice mechanism is a system function used to establish a kernel-managed data path directly between I/O device peers.[2,3] It is the primary mechanism that processes invoke to use PPIO. With splice, an application expresses an association between a data source and sink directly to the operating system through the use of file descriptors. These descriptors do not refer to memory addresses (i.e., they are not buffers):

```
sd = splice (fd1, fd2);
```

As shown in Figure 5, the call establishes an in-kernel data path, i.e., a splice, between a data source and sink

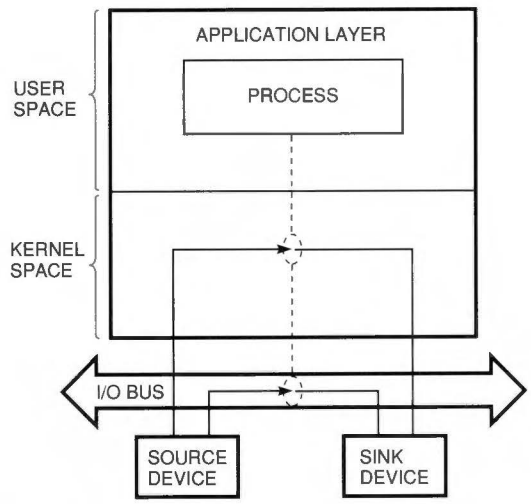


Figure 5
A Splice Connecting a Source to a Sink Device

device. If the I/O bus and the devices support hardware streaming, the data path is directly over the bus, avoiding system memory altogether. Although the process does not necessarily manipulate the data, it controls the size and timing of the dataflow. To manipulate the data, a processing module can be downloaded either into the kernel or directly on the devices if they support processing.

The data source and sink device are specified by the references fd1 and fd2, respectively. The splice descriptor sd is used in subsequent calls to read or write to control the flow of data across the splice. For example, the following call causes size bytes of data to flow from the source to the sink:

```
splice_ctrl_msg sc;
sc.op = SPLICE_OP_STARTFLOW;
sc.increment = size;
write (sd, &sc, sizeof(sc));
```

Data produced by the devices referenced by fd1 is automatically routed to fd2 without user process intervention, until size bytes have been produced at the source. The increment field specifies the number of bytes to transfer across a splice before returning control to the calling user application. When control is returned, dataflow is stopped. A SPLICE_OP_STARTFLOW must be executed to restart dataflow. The increment represents an important concept in PPIO and refers to the amount of data the user process is willing to have transferred by the operating system on its behalf. In effect, it specifies the level of delegation the user process is willing to give to the system. Specifying SPLICE_INCREMENT_DEFAULT indicates the system should choose an appropriate increment. This is generally a buffer size deemed convenient by the operating system.

The splice mechanism eliminates copy operations to user space by not relying on buffer interfaces such as those present in the conventional I/O functions read and write. By eliminating the user-level buffering, kernel buffer sharing is possible. More specifically, when block alignment is not required by an I/O device, a kernel-level buffer used for data input may be used subsequently for data output.

In addition to removing the buffering interfaces, splice also combines the read/write functionality together in one call. The splice call indicates to the operating system the source and sink of a dataflow, providing sufficient information for the kernel to manage the data transfer by itself without requiring user-process execution. Thus, context-switch operations for data transfer are eliminated. This is important: context switches consume CPU resources, degrade cache performance by reducing locality of reference, and affect the performance of virtual memory by requiring TLB invalidations.[27,28]

For applications making no direct manipulation of I/O data (or for those allowing the kernel to make such manipulations), splice relegates the issues of managing the dataflow (e.g., buffering and flow control) to the kernel. Data movement may be accomplished by a kernel-level thread, possibly activated by completion events (e.g., device interrupt) or operating in a more synchronous fashion. Flow control may be achieved by selective scheduling of kernel threads or simply by posting reads only to data-producing devices when data-consuming peers complete I/O operations. A kernel-level implementation provides much flexibility in choosing which control abstraction is most appropriate.

One criticism of streaming-based data transfer mechanisms is that they inhibit innovation in application development by disallowing applications direct access to I/O data.[29] However, many applications that do not require direct manipulation of I/O data can benefit from streaming (e.g., data-retrieving servers that do not need to inspect the data they have been requested to deliver to a client). Furthermore, for applications requiring well-known data manipulations, kernel-resident processing modules (e.g., Ritchie's Streams) or outboard dedicated processors are more easily exploited within the kernel operating environment than in user processes.[30,31] In fact, PPIO supports processing modules.[4]

PPIO Implementation and Performance

The PPIO design was conceived to support large data transfers. The decoupling of I/O data from process address space reduces cache interference and eliminates most data copies and process manipulation. PPIO and the accompanying splice system call have

been implemented within the ULTRIX version 4.2a operating system for the DEC 5000 series workstations, and within DEC OSF/1 version 2.0 for DEC 3000 series (Alpha-powered) workstations, each for a limited number of devices.

Three performance evaluation studies of PPIO have been carried out and are described in our early papers.[2,3,4] They indicate CPU availability improves by 30 percent or more; and throughput and latency improve by a factor of two to three, depending on the speed of I/O devices. Generally, the latency and throughput performance improvements offered by PPIO improve with faster I/O devices, indicating that PPIO scales well with new I/O device technology.

Improving Network Software Throughput

Network I/O presents a special problem in that the complexity of the abstraction layer (see Figure 2), a stack of network protocols, is generally much greater than that for other types of I/O. In this section, we summarize the results of an analysis of overheads for an implementation of TCP/IP we used in the Sequoia 2000 project. The primary bottleneck in achieving high throughput communication for TCP/IP is due to data-touching operations: one expected culprit is data copying (from kernel to user space, and vice versa); another is the checksum computation. Since we have already focused on how to avoid data copying in the previous two sections, we discuss how one can safely avoid computing checksums for a common case in network communication.

Overhead Analysis

We undertook a study to determine what bottlenecks might exist in TCP/IP implementations to direct us in our goal of optimizing throughput. The full study is described elsewhere.[9]

First, we categorized various generic functions commonly executed by TCP/IP (and UDP/IP) protocol stacks:

- Checksum: the checksum computation for UDP (user datagram protocol) and TCP
- DataMove: any operations that involve moving data from one memory location to another
- Mbuf: the message-buffering scheme used by Berkeley UNIX-based network subsystems
- ProtSpec: all protocol-specific operations, such as setting header fields and maintaining protocol state
- DataStruct: the manipulation of various data structures other than mbufs or those accounted for in the ProtSpec category
- OpSys: operating system overhead
- ErrorChk: The category of checks for user and system errors, such as parameter checking on socket system calls
- Other: This final category of overhead includes all the operations that are too small to measure. Its time was computed by taking the difference between the total processing time and the sum of the times of all the other categories listed above.

Other studies have shown some of these overheads to be expensive.[32-34]

We measured the total amount of execution time spent in the TCP/IP and UDP/IP protocol stacks as implemented in the DEC ULTRIX version 4.2a kernel, to send and receive IP packets of a wide range of sizes, broken down according to the categories listed above. All measurements were taken using a logic analyzer attached to a DECstation 5000/200 workstation connected to another similar workstation by an FDDI LAN attached through a Digital DEFZA FDDI adapter.

Figure 6 shows the per-packet processing times versus packet size for the various overheads for UDP packets. These are for a large range of packet sizes, from 1 to 8,192 bytes. One can distinguish two different types of overheads: those due to data-touching operations (i.e., data move and checksum) and those due to non–data-touching operations (all other categories). Data-touching overheads dominate the processing time for large packets that typically contain application data, whereas non–data-touching operations dominate the processing time for small packets that typically contain control information. Generally, data-touching overhead times scale linearly with packet size, whereas non–data-touching overhead times are comparatively constant. Thus, data-touching overheads present the major limitations to achieving maximum throughput.

Data-touching operations, which do identical work in the TCP and UDP software, also dominate processing times for large TCP packets.[9]

Minimizing the Checksum Overhead

As can be seen in Figure 6, the largest bottleneck to achieving maximum throughput (i.e., which one achieves by sending large packets) is the checksum computation. We applied two optimizations to minimize this overhead: improving the implementation of the checksum computation, and avoiding the checksum altogether in a special but common case where we felt we were not compromising data integrity.

We improved the checksum computation implementation by applying some fairly standard techniques: operating on 32-bit rather than 16-bit words, loop unrolling, and reordering of instructions to maximize pipelining. With these modifications, we

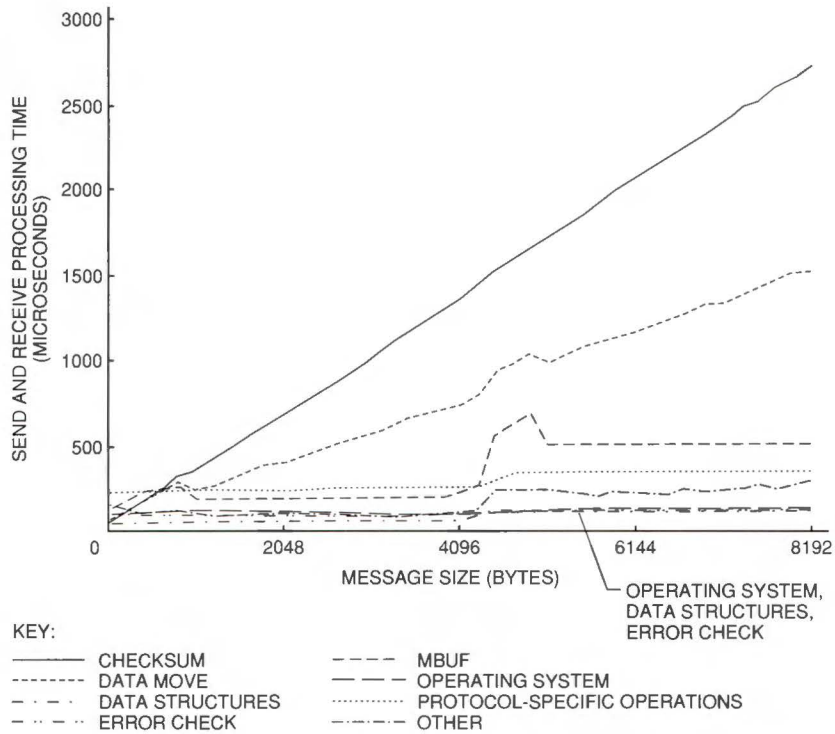


Figure 6
UDP Processing Overhead Times

reduced the checksum computation time by more than a factor of two. Figure 7 shows that the overall throughput improvement is 37 percent. The throughput measurements were made between two DECstation 5000/200 systems communicating over an FDDI network. Overall throughput is still a fraction of the maximum FDDI network bandwidth (100 Mb/s) because of data-copying overheads and machine-speed limitations. See Reference 6 for detailed results.

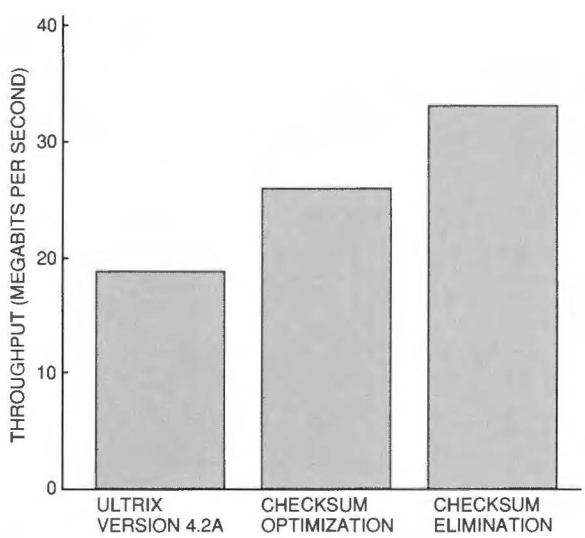


Figure 7
UDP/IP End-to-End Throughput

A very easy way of significantly raising TCP and UDP throughput is to simply avoid computing checksums; in fact, many systems provide options to do just this. The Internet checksum, however, exists for a good reason: packets are occasionally corrupted during transmission, and the checksum is needed to detect corrupted data. In fact, the Internet Engineering Task Force (IETF) recommends that systems not be shipped with checksumming disabled by default.[35]

Ethernet and FDDI networks, however, implement their own cyclic redundancy checksum (CRC). Thus, packets sent directly over an Ethernet or FDDI network are already protected from data corruption, at least at the level provided by the CRC. One can argue that for LAN communication, the Internet checksum computation does not significantly add to the machinery for error detection already provided in hardware.

Thus, our second optimization was simply to eliminate the software checksum computation altogether when computing the checksum would make little difference. Consequently, as part of the implementation of the protocol, when the source and destination are determined to be on the same LAN, the software checksum computation is avoided. Figure 7 shows the resulting 74 percent improvement in throughput over the unmodified ULTRIX version

4.2a operating system, and a 27 percent improvement over the implementation with the optimized checksum computation algorithm.

Of course, one must be very careful about deciding when the Internet checksum is of minimal value. We believe it is reasonable to turn off checksums when crossing a single network that implements its own CRC, especially when one considers the performance benefits of doing so. In addition, since the destinations of most TCP and UDP packets are within the same LAN on which they are sent, this policy eliminates the software checksum computation for most packets.

Our checksum elimination policy differs somewhat from traditional TCP/IP design in one aspect of protection against corruption. In addition to the protection between network interfaces given by the Ethernet and FDDI checksums, we require a software checksum in host memory as a protection from errors in data transfer over the I/O bus. For common devices such as disks, however, data transfers over the I/O bus are routinely assumed to be correct and are not checked in software. Therefore, a reduction in protection against I/O bus transfer errors for network devices does not seem unreasonable.

Turning off the Internet checksum protection in any wider area context seems unwise without considerable justification. Not all networks are protected by CRCs, and it is difficult to see how one might check that an entire routed path is protected by CRCs without undue complications involving IP extensions. A more fundamental problem is that network CRCs protect a packet only between network interfaces; errors may arise while a packet is in a gateway machine. Although such corruption is unlikely for a single machine, the chance of data corruption occurring increases exponentially with the number of gateways a packet crosses.

Summary and Conclusions

We described various solutions to achieving high performance in operating system I/O and network software, with a particular emphasis on throughput. Two of the solutions, container shipping and peer-to-peer I/O, focused on changes in the I/O system software structure to avoid data copying and other overheads. The third solution focused on the avoidance of additional data-touching overheads in TCP/IP network software.

Container shipping is a kernel service that provides I/O operations for user processes. High performance is obtained by eliminating the in-memory data copies traditionally associated with I/O, without sacrificing safety or relying on devices with special-purpose functionality. Further gains are achieved by permitting the selective accessing (mapping) of data. We measured performance improvements over UNIX of 40 percent (network I/O) to 700 percent (socket IPC).

PPIO is based on the hypothesis that the memory-oriented model of I/O present in most operating systems presents a bottleneck that adversely affects overall performance. PPIO decouples user-process execution from interdevice dataflow and can achieve improvements in both latency and throughput over conventional systems by a factor of 2 to 3.

Finally, we considered the special case of network I/O where data moving/copying is not the only major overhead. We showed that the checksum computation is a major source of TCP/IP network processing overhead. We improved performance by optimizing the checksum computation algorithm and eliminating the checksum computation when communicating over a single LAN that supports its own CRC, improving throughput by 37 percent to 74 percent for UDP/IP.

Acknowledgments

We are indebted to David Boggs, who built the T1 and T3 boards which worked like a charm. We appreciate the efforts of Richard Bartholomaus and Ira Machefsky who helped us get our first DECstation 5000/200 workstations. As our interface to Digital, Ira Machefsky helped in many other ways; we sincerely thank him. We thank Fred Templin for the technical expertise he provided us on Digital networking equipment. We thank Mike Stonebraker and Jeff Dozier for their leadership of the Sequoia 2000 project. We thank our network research colleagues, Professors Domenico Ferrari and George Polyzos, and their students, with whom we enjoyed collaborating; we have benefited from their advice. Finally, thanks to Jean Bonney for supporting our project from start to end.

References

1. J. Pasquale, E. Anderson, and K. Muller, "Container Shipping: Operating System Support for Intensive I/O Applications," *IEEE Computer,* vol. 27, no. 3 (1994): 84–93.

2. K. Fall and J. Pasquale, "Exploiting In-kernel Data Paths to Improve I/O Throughput and CPU Availability," *Proceedings of the USENIX Winter Technology Conference,* San Diego (January 1993), pp. 327–333.

3. K. Fall and J. Pasquale, "Improving Continuous-media Playback Performance with In-kernel Data Paths," *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS),* Boston, Mass. (June 1994), pp. 100–109.

4. K. Fall, "A Peer-to-Peer I/O System in Support of I/O Intensive Workloads," Ph.D. dissertation, University of California, San Diego, 1994.

5. J. Kay and J. Pasquale, "The Importance of Non–Data-Touching Processing Overheads in TCP/IP," *Proceedings of the ACM Communications Architectures and Protocols Conference (SIGCOMM),* San Francisco (September 1993), pp. 259–269.

6. J. Kay and J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *Proceedings of the USENIX Winter Technology Conference,* San Diego (January 1993), pp. 249–258.

7. J. Kay and J. Pasquale, "A Summary of TCP/IP Networking Software Performance for the DECstation 5000," *Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS),* Santa Clara, Calif. (May 1993), pp. 266–267.

8. J. Kay, "PathIDs: Reducing Latency in Network Software," Ph.D. dissertation, University of California, San Diego, 1995.

9. J. Kay and J. Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP," *IEEE/ACM Transactions on Networking* (accepted for publication).

10. D. Ferrari, A. Banerjea, and H. Zhang, *Network Support for Multimedia: A Discussion of the Tenet Approach* (Berkeley, Calif.: International Computer Science Institute, Technical Report TR-92-072, 1992).

11. H. Zhang, D. Verma, and D. Ferrari, "Design and Implementation of the Real-Time Internet Protocol," *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems,* Tucson, Ariz. (February 1992).

12. A. Banerjea, E. Knightly, F. Templin, and H. Zhang, "Experiments with the Tenet Real-time Protocol Suite on the Sequoia 2000 Wide Area Network," *Proceedings of the ACM Multimedia,* San Francisco (October 1994).

13. V. Kompella, J. Pasquale, and G. Polyzos, "Multicast Routing for Multimedia Applications," *IEEE/ACM Transactions on Networking,* vol. 1, no. 3 (1993): 286–292.

14. V. Kompella, J. Pasquale, and G. Polyzos, "Two Distributed Algorithms for Multicasting Multimedia Information," *Proceedings of the Second International Conference on Computer Communications and Networks (ICCCN),* San Diego (June 1993), pp. 343–349.

15. J. Pasquale, G. Polyzos, E. Anderson, and V. Kompella, "Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks," *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV),* D. Shepherd, G. Blair, G. Coulson, N. Davies, and F. Garcia (eds.), *Lecture Notes in Computer Science,* vol. 846 (Springer-Verlag, forthcoming).

16. J. Pasquale, G. Polyzos, E. Anderson, and V. Kompella, "The Multimedia Multicast Channel," *Journal of Internetworking: Research and Experience* (in press).

17. J. Pasquale, G. Polyzos, and V. Kompella, "Real-time Dissemination of Continuous Media in Packet-switched Networks," *Proceedings of the 38th IEEE Computer Society International Conference (COMPCON),* San Francisco (February 93), pp. 47–48.

18. K. Muller and J. Pasquale, "A High-Performance Multi-Structured File System Design," *Proceedings of the 13th ACM Symposium on Operating System Principles (SOSP),* Asilomar, Calif. (October 1991), pp. 56–67.

19. J. Pasquale, "I/O System Design for Intensive Multimedia I/O," *Proceedings of the Third IEEE Workshop Workstation Operation Systems (WWOS),* Key Biscayne, Fla. (April 1992), pp. 29–33.

20. J. Pasquale, "System Software and Hardware Support Considerations for Digital Video and Audio Computing," *Proceedings of the 26th Hawaii International Conference on System Sciences (HICSS),* Maui, IEEE Computer Society Press (January 1993), pp. 15–20.

21. C. Thekkath, H. Levy, and E. Lazowska, "Separating Data and Control Transfer in Distributed Operating Systems," *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS),* San Jose, Calif. (October 1994), pp. 2–11.

22. R. Rashid et al., "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," *IEEE Transactions on Computers,* vol. 37, no. 8 (1988): 896–908.

23. E. Anderson, "Container Shipping: A Uniform Interface for Fast, Efficient, High-bandwidth I/O," Ph.D. dissertation, University of California, San Diego, 1995.

24. D. Bobrow, J. Burchfiel, D. Murphy, and R. Tomlinson, "TENEX, a Paged Time-Sharing System for the PDP-10," *Communications of the ACM,* vol. 15, no. 3 (1972): 135–143.

25. S.-Y. Tzou and D. Anderson, "The Performance of Message-Passing Using Restricted Virtual Memory Remapping," *Software—Practice and Experience,* vol. 21, no. 3 (1991): 251–267.

26. P. Druschel and L. Peterson, "Fbufs: a High-bandwidth Cross-Domain Transfer Facility," *Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP),* Asheville, N.C. (December 1993), pp. 189–202.

27. J. Mogul and A. Borg, "The Effect of Context Switches on Cache Performance," *Proceedings of the ASPLOS-IV* (April 1991), pp. 75–84.

28. B. Bershad, T. Anderson, E. Lazowska, and H. Levy, "Lightweight Remote Procedure Call," *ACM Transactions on Computer Systems,* vol. 8, no. 1 (1990): 37–55.

29. P. Druschel, M. Abbott, M. Pagels, and L. Peterson, "Analysis of I/O Subsystem Design for Multimedia Workstations," *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV),* November 1992.

30. D. Ritchie, "A Stream Input-Output System," *AT&T Bell Laboratories Technical Journal,* vol. 63, no. 8 (1984): 1897–1910.

31. D. Presotto and D. Ritchie, "Interprocess Communication in the Eighth Edition UNIX System," *Proceedings of the USENIX Winter Conference* (January 1985), pp. 309–316.

32. L.-F. Cabrera, E. Hunter, M. Karels, and D. Mosher, "User-Process Communication Performance in Networks of Computers," *IEEE Transactions on Software Engineering,* vol. 14, no. 1 (1988): 38–53.

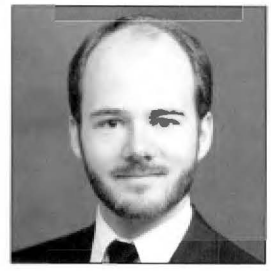
33. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications* (1989): 23–29.

34. R. Watson and S. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," *ACM Transactions on Computer Systems,* vol. 5, no. 2 (1987): 97–120.

35. R. Braden, "Requirements for Internet Hosts—Communication Layers," *Internet Request for Comments 1122,* (Network Information Center, 1989).

Biographies

Joseph Pasquale
Joseph Pasquale is an associate professor in the Department of Computer Science and Engineering at the University of California at San Diego. He has a B.S. and an M.S. from the Massachusetts Institute of Technology and a Ph.D. from the University of California at Berkeley, all in computer science. In 1989, he established the UCSD Computer Systems Laboratory, where he and his students do research in network and operating system software design, especially to support I/O-intensive applications such as distributed multimedia (digital video and audio) and scientific computing. He also investigates issues of coordination and decentralized control in large distributed systems. He has published more than 40 refereed conference and journal articles in these areas and received the NSF Presidential Young Investigator Award in 1989.



Eric W. Anderson
Eric Anderson received B.A. (1989), M.S. (1991), and Ph.D. (1995) degrees from the University of California at San Diego. His dissertation is on the development of a uniform interface for fast, efficient, high-bandwidth I/O. As a research assistant at UCSD, he contributed to the planning and installation of the Sequoia 2000 network and conducted research in operating system I/O and high-speed networking. He is currently a postgraduate researcher with the Computer Systems Laboratory at UCSD, where he is involved in further studies of high-performance I/O techniques. He is a member of ACM and has coauthored papers on the multimedia multicast channel, operating system support for I/O-intensive applications, and filter propagation in dissemination trees in continuous media networks.

Kevin R. Fall
Kevin Fall received a Ph.D. in computer science from the University of California at San Diego in 1994 and a B.A. in computer science from the University of California at Berkeley in 1988. He held concurrent postdoctoral positions with UCSD and MIT before joining the Lawrence Berkeley National Laboratory in September 1995, where he is a staff computer scientist in the Network Research Group. While at UC Berkeley, he was responsible for the integration of security software into Berkeley UNIX and protocol development for the campus' supercomputer. While at UC San Diego, Kevin developed a high-performance I/O architecture designed to the support the large I/O demands of the Sequoia 2000 database and multimedia applications. He was also responsible for the routing architecture and system configuration of the Sequoia 2000 network.

Jonathan S. Kay
Jon Kay received a Ph.D. in computer science from the University of California at San Diego. While working toward his doctorate, he was involved in the Sequoia 2000 networking project. He joined Isis Distributed Systems of Ithaca, N.Y. in 1994 to work on distributed computing toolkits. He also holds a B.S. and an M.S. in computer science from Johns Hopkins University.

Call for Authors from Digital Press

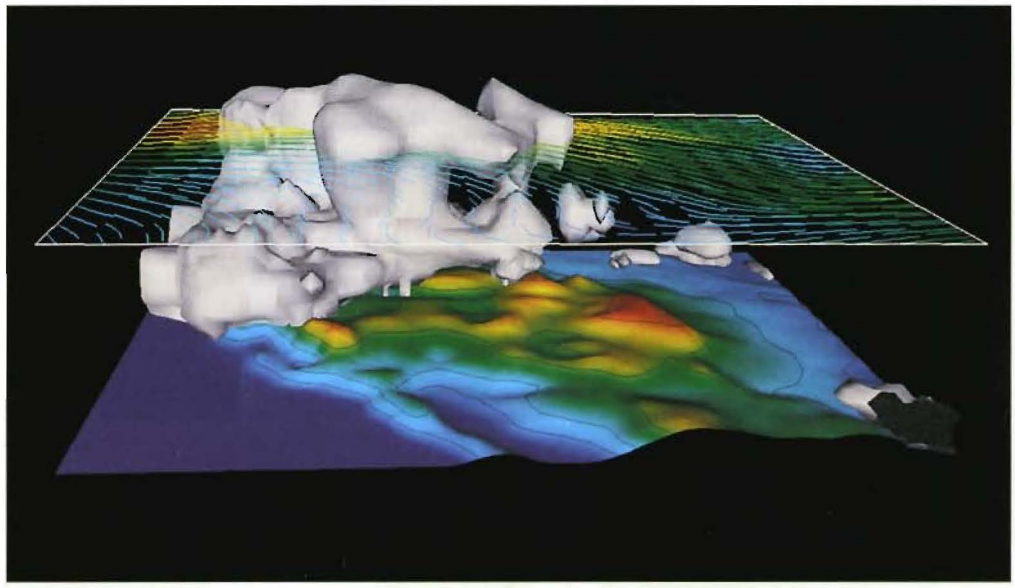
Digital Press is an imprint of Butterworth-Heinemann, a major international publisher of professional books and a member of the Reed Elsevier group. Digital Press is the authorized publisher for Digital Equipment Corporation: The two companies are working in partnership to identify and publish new books under the Digital Press imprint and create opportunities for authors to publish their work.

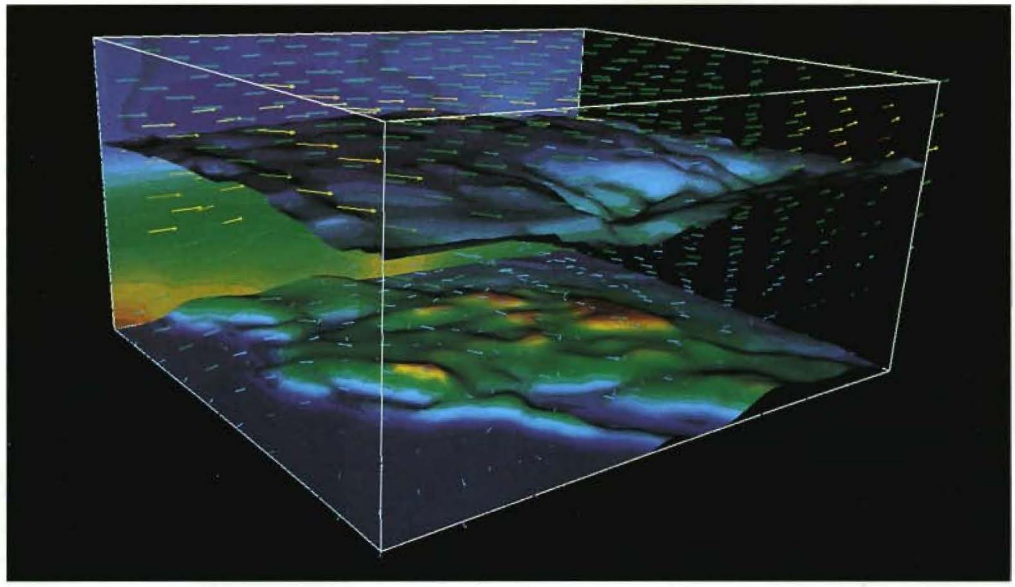
Digital Press is committed to publishing high-quality books on a wide variety of subjects. We would like to hear from you if you are writing or thinking about writing a book.

Contact: Mike Cash, Digital Press Manager, or
Liz McCarthy, Assistant Editor

DIGITAL PRESS
313 Washington Street
Newton, MA 02158-1626
U.S.A.
Tel: (617) 928-2649, Fax: (617) 928-2640
E-mail: Mike.Cash@BHein.rel.co.uk or
LizMc@world.std.com

digital™





ISSN 0898-901X