Massive Information Storage, Management, and Use

(NSF Institutional Infrastructure Proposal)

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720

January 1989
Report No. UCB/CSD 89/493

Abstract

This report is an edited version of a proposal submitted to the NSF Institutional Infrastructure program in September 1987 and funded in July 1988. The financial data and other supplementary information have been omitted, but the technical sections are unchanged.

The proposal builds on the current research of a broad spectrum of the Computer Science faculty and at the same time expands our computing paradigm in the direction of current and future changes in technology. The funded project,subsequently named the Mammoth project, provides the infrastructure (equipment, its maintenance, and support staff) to investigate many questions concerning the management of massive amounts of information, its efficient storage and fast retrieval on large capacity secondary storage media, its movement in high-storage high-capacity networks, and the facilitation by fast large-main-memory machines of its complex manipulation.

The Mammoth project both enhances and integrates various independently funded research projects that are summarized in the technical sections of this report. The reader can gain an overview of the Mammoth-related aspects of much, but certainly not all, of the research activities of the Computer Science faculty as they existed in the fall of 1987.

Massive Information Storage, Management, and Use

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California

September 1987

Project Director:	Susan L. Graham
Faculty Investigators:	David Anderson
	Brian A. Barsky
	Richard J. Fateman
	Domenico Ferrari
	Susan L. Graham
	Michael A. Harrison
	William Kahan
	Richard M. Karp
	Randy H. Katz
	Jitendra Malik
	John Ousterhout
	David A. Patterson
	C.V. Ramamoorthy
	Lawrence A. Rowe
	Stuart Russell
	Raimund Seidel
	Carlo H. Séquin
	Alan J. Smith
	Michael J. Stonebraker
	Robert Wilensky

Table of Contents

A. Introductory Material

A.1 Standard Cover Page

omitted

A.2 Form 1225 (*on original only*)

omitted

B. Executive Summary

Introduction

Recent advances in storage systems and high-speed data communication are changing the face of computer technology. Within the next few years it should be possible to fit a thousand megabytes (one gigabyte) of random-access memory within a single chassis. Compact units containing half-gigabyte magnetic disks will soon be available for under $1000, and cheap nonerasable optical disks can provide seemingly unlimited amounts of backup and archival storage. A single two-gigabyte optical disk can store two thousand color pictures, eight thousand engineering diagrams, a half-million pages of text, or twenty-four hours of high-quality sound; systems with several hundred such disks are expected to become available. And developments in fiber-optic technology will provide the bandwidth to allow a community of users to access massive amounts of storage and, in the process, share their data, their programs and their knowledge.

The Computer Science Division at Berkeley proposes to develop a massive hierarchical storage facility, to investigate a number of approaches to the design of systems software for such an environment, and to use the environment in a number of application areas, including:

i) The construction of large knowledge bases and their application in machine learning and natural language processing;

ii) The manipulation of complex objects such as designs, large programs, and multimedia documents;

iii) The processing of geometric and pictorial information such as images, surfaces and mathematical functions of several variables;

iv) The integrated storage of scientific program libraries, mathematical tables and procedures for the transformation and manipulation of symbolic information.

The Computer Science Division is a unit within the Department of Electrical Engineering and Computer Sciences. Our proposal has support from the Department and enjoys the strong backing of the campus administration. The University stands ready to provide substantial new resources, including three new faculty members over the next five years, faculty release time, staff positions, funds for space renovation and equipment maintenance, discretionary funds and a substantial overhead exemption. More information about the University's financial contribution is given in Section D.

The facility will be developed in stages. In the first year we plan to acquire a high speed fiber-optic network and two "information servers", each equipped with 128 megabytes of main memory, 10 magnetic disk drives and 6 optical disks. During this period we will also assemble some prototyping systems for development work. These systems will be obtained by upgrading existing equipment. We will acquire further information servers in the second year. Then, building on the experience of the first two years, we will acquire a "super information server" containing an 80 MIP processor, 100 disk drives, a gigabyte of primary memory and a terabyte of optical disk storage.

The development of such a system will require innovations in distributed operating systems, computer architecture, performance analysis, database system design and algorithm design. The availability of the system will enhance our research in artificial intelligence, text processing, programming systems, graphics, computer vision and scientific computation. The Division has the experience, the expertise and the management skills to carry out this project. We have strong research groups in all the relevant areas, and new recruiting will continue to strengthen our faculty. We have demonstrated the ability to successfully mount large, experimental research projects. The tangible results of such projects include Berkeley BSD UNIX, the INGRES relational database system and one of the first RISC processors. Most importantly, the discussions leading to this proposal have engaged the enthusiasm of the entire Computer Science faculty, and have led us to a shared vision of our goals and a consensus as to the infrastructure we need. The support we are requesting will allow us to create a shared facility that could not be obtained through individual research grants, and will weld our Division into a single community with a common research focus, much as the UNIX project did for our systems faculty in earlier times.

The following paragraphs summarize the specific research we propose to conduct:

Design and Analysis of a Massive Storage Facility

Operating System Research (David Anderson, Domenico Ferrari, John Ousterhout, Alan J. Smith)

We will explore several issues in operating system design that arise in connection with the new massive storage facility:

Performance Studies (Smith and Ousterhout) The proposed hierarchical mass storage system presents novel storage management problems because of the many levels and types of storage involved. We will investigate basic issues concerning the scheduling of file transfers so as to maximize performance, including user-transparent mechanisms for automatic file migration, together with the associated problems of providing access to previous file generations stored on nonerasable optical disks, ensuring the consistency of caches at various levels of the hierarchy, and developing suitable user interfaces to storage-related information.

Time Travel (Ousterhout) Exploiting the fact that the large capacity of optical disks makes it feasible to store several years of backup data, we will implement a "time-travel" mechanism, in which a user can issue a command that will cause the system to behave as if it had been restored to its state at a specified time in the past.

Replacing Magnetic Disks (Ousterhout) We will investigate whether, in certain application environments, optical disks can replace magnetic disks altogether. This study will be a continuation of current work to implement a file system without magnetic disks in connection with the Sprite operating system.

Communication Architectures for Access to Large Objects (Ferrari and Anderson) For most applications involving large read-write memories and a large optical store, large amounts of data will have to be communicated over high-speed fiber-optic links. A hierarchy of software protocols to provide the needed communication mechanisms will be developed as part of the DASH distributed operating system. The DASH communication facility will be integrated with the virtual memory and process scheduling components of the system so as to provide performance guarantees to the user.

Basic Operating System Enhancement To support the facility we will extend Berkeley UNIX to support very large memories and secondary storage, and will provide new kernel support for optical disks, which are not supported by our present system.

Research in Computer Architecture (Randy Katz and David Patterson)

Opportunities for innovation in I/O system design have arisen because of the drastic reduction in the size and cost of medium-capacity disk systems. It will now be possible to provide information servers with very large quantities of small-capacity inexpensive disks, and to gain large I/O bandwidth through parallel access to such disks. Because of its limited capacity, such a "disk farm" will have to be backed up by a very-large-capacity optical store. We propose to study the use of such an architecture for a transaction processing environment built on top of the POSTGRES database system. The fundamental technical problems will be to make such systems reliable and crash-proof, devise file migration strategies, overcome latency problems through buffering and parallel access schemes, and reduce bus traffic by attaching I/O devices directly to processor caches rather than main memory. Our goal is to achieve a low-cost system capable of handling 1000 transactions per second.

Network Event Management (C. V. Ramamoorthy)

We will build a network server that monitors events within a distributed computing system and triggers appropriate action when certain system states are entered. The transmission of status information between processors will require the high-speed network and communication architecture being developed; the large memory will be used to store the complex system state, and the optical disks will be used to store a history database for the evaluation of event management strategies.

Algorithmic Issues in Massive Information Storage (Richard Karp, Raimund Seidel)

The efficient use of very large memories obliges us to reexamine the fundamental algorithmic techniques and data structures that we employ for data storage and retrieval and the solution of combinatorial problems. We will explore search methods based on partitioning data into buckets, large-scale sorting methods based on distribution rather than merging, sorting methods suitable for nonerasable memories, data structure techniques for storing history data, the application of dynamic programming to DNA sequence analysis, and combinatorial algorithms based on dynamic programming and branch-and-bound techniques that require very large memories. The interactions with our colleagues fostered by the shared facility will be a source of both users of our methods and new algorithmic questions for us.

Use of a Massive Storage Facility

Research on Very Large Knowledge Bases (Stuart Russell, Robert Wilensky)

We plan to to construct a large, heterogeneous knowledge base that will enable progress in a number of different research projects in machine learning and natural language. The process of creating, managing and accessing this knowledge base will require fundamental research on the representation and use of meta-knowledge that describes the structure and content of the knowledge base, and on the goal selection and planning algorithms of an intelligent knowledge base manager and interface.

The breadth of a large common-sense knowledge base allows it to be used as a test bed for studies of the inductive acquisition, representation and use of abstract knowledge. The knowledge base will also contain a large amount of sensory data acquired in a simulated real-time environment; this data can be viewed as the life history of an autonomous agent with limited sensory powers, and we will use it to study how such agents can derive theories for dealing with the world from experience. Finally, the knowledge base will be used to study the process of making the common-sense inferences that must be used translate, summarize and understand natural language text.

Object Management in Data Bases (Michael J. Stonebraker, Lawrence A. Rowe)

We are developing a database system called POSTGRES that will handle a conventional relational presentation of data, but will also be capable of dealing with more complex data such as documents, CAD objects, computer programs, icons and bitmaps. POSTGRES employs abstract data types for the description of complex data and the construction of access methods. It supports the procedural description of complex data objects. For real-time applications it incorporates an integrated rules system that triggers a response whenever a particular event occurs. Because the POSTGRES storage manager never overwrites old data, history queries can be supported; this facility is only possible because of the optical disk technology. POSTGRES will require the high I/O bandwidth achievable through a large cluster of magnetic disks and a large main memory for buffering. Associated with POSTGRES is an object-oriented programming environment called OBJFADS, which will support the development of multimedia applications.

POSTGRES and OBJFADS will be used to develop the IC-CIM software system to manage the Microfabrication Laboratory at Berkeley. Three subsystems are currently under development: a facility management system to maintain information about facility layout, equipment locations and utility connections; a work-in-progress system to schedule the facility, guide operators through processing steps, perform automated measurements and store the results of measurements; a wafer checking system based on visual scanning of wafers; and an equipment maintenance system. The use of optical disks and large memories will be crucial for this application because of the large volume of data, much of it in the form of images.

Integrated Interactive Development of Complex Objects (Susan L. Graham, Michael A. Harrison)

Pan is an interactive editing system that combines a text editing capability with support for high-level manipulation of structured language-based objects such as programs, designs, process descriptions, interface definitions and document specifications. It supports syntactic editing, logic-based and free-form semantic annotation, browsing and viewing of such objects. It has potential application to software modification and reusability. Its continued development entails research on semantic specification and analysis, incremental updating of semantic information, and the management of a combination of stable and ephemeral information within a storage hierarchy.

VorTeX is an integrated document preparation environment for the interactive development of high quality technical documents that involve mathematics, text and graphics. Future work on the system entails research on integrated user interfaces, multiple representations of documents, support for composite objects, animation and bibliographic assistance.

Pan and VorTeX are being integrated to use a common text editing capability, a common window manager and other shared utilities. Further research will enable the power of both systems to be brought to bear on user-constructed objects, by treating them simultaneously as linguistic objects understood by Pan, and as documents understood by VorTeX.

Storage and Interactive Display of Pictorial Realistic and Dynamically Varying Images (Carlo H. Sequin, Brian A. Barsky, and Jitendra Malik)

In problems such as optimizing multi-parameter analog integrated circuits or understanding the convergence behavior of a neural network, there is a need to visualize a cost function or energy surface, and to display lower-dimensional views of that surface. A related problem, arising in Sequin's UNIGRAFIX geometric modeling and rendering system, is to display interactively a complex 3-dimensional object such as a building or a mechanical part, as surface modeling parameters such as color, reflectivity and index of refraction change in real time. We are developing computer graphics techniques to satisfy these needs. Our methods will depend heavily on the use of a massive memory hierarchy.

The Beta-spline was introduced by Barsky as a tool for the representation of curves and surfaces. It has two shape parameters, called bias and tension. Users of graphics systems readily are able to adjust these parameters to achieve the desired representation. The Beta-spline has been generalized by allowing these shape parameters to vary locally along a curve or surface, and the use of further control parameters is under investigation. We will develop a library of images produced with various specifications of these parameters, together with a facility for the interactive animated display of such images, so that we can determine which control parameters are the most useful, and can provide users with the support they need to be able to adjust these parameters.

Even with gigabyte memories these problems cannot be solved by the brute-force method of storing all the possible pictures that might be displayed; it is the goal of our research to find practical methods of generating such images. Data compression techniques will be essential in this connection.

The storage of a color image of a scene typically requires a megabyte of memory. There are significant problems in computer vision that require the storage of hundreds of such images simultaneously in main memory, and thousands more on optical disks. One such problem, studied by Malik, is the analysis of time-varying images; relative motion permits a scene to be segmented and objects to be recognized much more reliably than is possible with a few isolated views. A library of several thousand images would also be used to test computer vision systems, to determine the parameters, such as specularities and sharpness of shadow edges, that best describe the world, and to provide training sets for learning algorithms.

Scientific Computing (Richard J. Fateman, William Kahan)

The availability of gigabyte storage systems will enable symbolic manipulation systems to perform new classes of optimizations and program transformations. One simple technique is to store known input-output pairs for a complex function, and to determine the output by table-look-up when an input is repeated. When a function of several variables is to be tabulated with some arguments fixed and the others varying, it should be possible to simplify the computation by symbolic transformations exploiting the fixed parameters. In the case of a function capable of running on diverse data types, it may be useful to store specialized algorithms for particular, frequently occurring data types such as "truncated power series in z with coefficients in the quotient field of polynomials in x and y over the integers".

With very large memories it should be possible to store large bodies of mathematical knowledge such as tables of integrals. Such knowledge is often difficult to index; it is not clear how to determine rapidly which table entries might be relevant to a particular integration problem. The problems of organizing and accessing such hard-to-index data will be a major component of our research.

We intend to develop tools for maintaining the integrity of very large scientific software libraries. We shall focus on documenting the bugs that arise because of the implicit limitations of numerical computation on finite-capacity machines, and developing retrospective diagnostics to enable users of multiple layers of software to trace the effects of software modifications and determine what went wrong.

C. Equipment

C.1 Computing Facilities Currently Available for Research

omitted

C.2 Description of Equipment and Accessories Requested

In keeping with our theme of massive memory systems, we propose to assemble several configurations of "information servers", by a combination of new systems and upgrades to existing equipment. Our equipment purchases will take place during the first four years of the grant, in order to obtain the latest equipment for each stage of the effort. Section C.3 gives the rationale for this equipment plan.

Year One

(a) Two "information servers" consisting of the following:

> Sun 4 class machine with 128 megabytes memory
> 10 magnetic disk drives (5 1/4 inch)
> 6 optical disks (2 gigabyte capacity each)
> 60 platters

(b) Three prototype systems for development efforts needing dedicated hardware. We expect to assemble these systems by upgrading existing equipment with needed components such as optical disks.

(c) High speed network. We intend to replace much of our ethernet based LAN by a faster, higher bandwidth fiber-based network.

Year Two

(a) Two additional information servers. We assume the cost is the same as for Year One. If prices drop, we will use the savings to acquire larger amounts of storage or faster machines.

(b) Additional upgrades to our existing equipment, for compatibility with the emerging configuration. Again, we assume that the cost is the same as for Year One and that any savings will be used to increase capacity or speed.

Years Three and Four

(a) One "super information server" consisting of

> high-speed machine (80 mips)
> 100 magnetic disk drives (5 1/4 inch)
> optical disk system (1 terabyte capacity)
> and platters

We can only estimate the cost of the system, since the equipment we want is not yet on the market. We will determine the system configuration and the vendors during Year Two of the grant. The decision will be determined by the results of our research in the first two years and by availability at that time. In addition to speed, the requirements for the machine will include massive primary memory and, most likely, a shared memory multiprocessor architecture. The system must run 4BSD or whatever operating system is standard in our environment at the time.

(b) Additional upgrades to existing equipment, for compatibility with the emerging configuration. We again assume that the cost is the same as for Year One. However, by this time, we will be able to upgrade to larger and more powerful systems than in Year One.

C.3 Rationale for Selected Equipment

Computer science research at Berkeley has always taken place in a shared computing environment. Initially, that environment consisted of terminal access to time-shared mainframes, mostly PDP/11's and VAXes. More recently, we have migrated to a workstation/server environment in which Sun and MicroVax workstations (and a few other varieties from time to time) have been connected via local networks to compute servers and print servers.

The proposed facility will be a major enhancement to that environment and will change its character. The use of a shared facility will force us to develop compatible solutions to the various storage-intensive problems we are pursuing. We will continue to maintain network access to shared equipment and to use a common operating system at the client level. In addition, we will continue to use high-performance workstations of various kinds as our primary interface to the computing environment. New workstations are not part of the budget for this proposal. We anticipate acquiring new workstations and upgrading the ones we have through the other grants and contracts that support our research.

The proposed facility must meet various requirements:

a) The configuration must provide a massive storage system organized in a memory hierarchy for use in various application domains. Here, a usable, accessible system is the goal.

b) We must provide an environment for software developers who require dedicated systems with prototype quantities of the components of interest. These systems will be particularly important for operating systems work and for some aspects of data base software development.

c) We need an environment in which several approaches to object management can be explored. Both distributed solutions and hierarchical organizations should be supportable.

d) We are striving for an environment where high I/O throughput is possible. Experiments with disk striping and parallel file systems will require a substantial number of disk arms.

e) Rapid high-bandwidth communication between the workstations and the servers is essential.

The researchers building experimental systems software will sometimes run systems that have bugs and that cause system crashes. Consequently it is important that they do their experiments on separate hardware from those groups whose software is at the application level. The latter group needs reliable system software and steady predictable access to computing. Therefore we plan to have multiple smaller hardware systems in the early years.

We predict that many projects will migrate to a large-primary-memory environment in the early years, but that the projects that are primarily clients of secondary storage subsystems will move to the massive memory environment only after some initial work has been done by the systems researchers to provide the fundamental capabilities. Consequently, we plan our "super information server" for the third year of the proposal, not the first. The information servers purchased during the first two years will enable the projects to do the experimental software development that will position them to exploit the large information server when it arrives.

We anticipate additional years of use of our more recently acquired existing equipment; notably many of the workstations and the upgradeable file servers. However, our existing mainframes and some of our peripherals are old and no longer "state-of-the art". In addition, our ethernet-based local network is both overloaded and significantly less powerful than the emerging fiber-based technologies. We propose to replace them with the new equipment.

C.4 Maintenance Costs per Year and Method of Computation

omitted

C.5 Installation Costs per Year and Method of Computation

omitted

D. Budget

D.1 Summary of Project Costs

omitted

D.2 - University Contribution

omitted

D.3 Project Budget - NSF form 1030

omitted

D.4 Management Structure and Plan for Operating the Facility

Professor Susan L. Graham will serve as the Project Director. She has experience as Project Director for the DARPA tasking contract for "Productivity Engineering in the UNIX Environment", which has provided research and/or infrastructure support to most members of the Computer Science Division at various times. She will chair a steering committee consisting of Professors Fateman, Karp, Ousterhout, Russell, Sequin, and Stonebraker. This group is representative of the faculty – each faculty investigator has a close colleague on the committee. Professors Karp and Sequin are former departmental Associate Chairs for Computer Science; Professor Fateman is the current Associate Chair. Professor Fateman is also a former departmental Vice-Chair for Computing. Professor Stonebraker has considerable experience directing the former INGRES project and the current POSTGRES project. Professor Graham serves on the departmental Computer Needs and Resources committee (as do Professors Wilensky and Patterson). All of the steering committee members are actively involved in research supported by the proposed infrastructure.

The steering committee will be responsible for setting policy concerning sharing the facility, for making specific purchasing decisions, and for supervising the programmers. It will also serve as a focal point for technical interaction among the research groups.

The staff for the proposed facility will consist of four technical people, a business manager, and a grant administrator. The technical people, whose salaries will be paid either directly by the campus, or indirectly by virtue of the overhead exemption, will consist of a hardware engineer, a system manager, and two professional programmers. Initially the programmers will carry out the UNIX development needed to make the facility usable. In later years they will transfer the research technology we develop to the systems used by our "user-level" projects. One of the programmers will also have some system management responsibility. The business manager (the AAIII in the budget) will be Ms. Joan Slobin, who currently has the same responsibility for the DARPA project mentioned above. She was formerly on the staff of the campus Sponsored Projects Office and is familiar with the various campus and government procedures. She will supervise a full-time grant administrator (the AAII in the budget) who will take care of the day-to-day business and clerical needs of the facility. We have used a similar structure in all of our large multi-investigator projects in the department and have found it to be very successful.

We will also have the help of the Computer Systems Support Group (CSSG), which provides hardware and software services to the department on a recharge basis. Professor Richard Newton of EECS ably directs CSSG, and has been instrumental in establishing effective recharge procedures, budgetary controls, and work scheduling. The existence of CSSG has enabled the department to benefit from a pooled collection of spare parts, from shared maintenance arrangements, and from the flexibility to move staff around as needed. CSSG also provides senior staff supervision of hardware and software technicians employed by a particular project or research group, if requested. The faculty sets direction and policy in all cases.

D.5 Certification Statement

omitted

E. Staff Credentials

E.1 Curriculum Vitae

omitted

E.2 Current and Pending Research Support

omitted

F. Research

1. Introduction

Advances in computer technology are providing not only smaller and faster processors and faster communication among them, but also significantly denser and less expensive storage media. Particularly in the realm of storage, we are on the threshold of some very important developments.

The first 4-megabit random access memory chips have been demonstrated in a few development labs and will soon be available on the market. These components will make it possible to fit a gigabyte (8 billion bits) of fast random access memory into a standard chassis.

While the storage density of magnetic disks is increasing at a steady rate, optical disks are finally emerging as a highly competitive product. Systems having large disks at the cost of $10-15,000 are already available. Compact units with small disks with storage capacities of about 0.5 gigabytes, based on organic coatings will soon be available for less than $1000.

While progress in high-performance magnetic disks has been slow but steady, costs of small magnetic disks have been dropping rapidly because of the volume demand from personal computers and workstations. With inexpensive hard disks and single chip disk controllers now becoming commercially available, it will soon be possible to package a large number of "personal computer" magnetic disks in a small space and at a small cost. Since the small disks are almost as fast as the expensive ones, it may be possible to provide high bandwidth I/O, by taking advantage of the potential parallelism of many small disks, with compact disks covering any shortfall in storage capacity of these smaller disks. This presents the opportunity to dramatically increase the system I/O bandwidth while simultaneously reducing system cost.

The availability of these new devices will have a significant impact on computing systems of the future and on the ways that the users will interact with them. The effects of these technological advances will be felt throughout the breadth of computing research.

The Computer Science faculty at Berkeley has a wide variety of research interests and accomplishments in the areas of software, architecture, theory, and applications. In the early years of our program, one of the important factors in fostering research collaboration and informal technical interactions was the commonality of our involvement with UNIX, which was then at the forefront of systems innovation. We propose to strengthen our internal research ties and infrastructure by again pursuing a common emphasis in our research. Our theme is the design, analysis, and use of a hierarchical massive storage facility based on the emerging storage technologies. Sharing of the storage components will be facilitated by a high speed network.

Our attention to storage and data management issues encompasses very large main memories, significant quantities of magnetic disks, and vast quantities of write-once-read-many storage (WORM devices). Our intent is not simply to increase file space or to decrease swapping, but rather to investigate a major paradigm shift inspired by the forthcoming storage components.

The discussion that follows is divided into two major sections. The first summarizes those projects whose major emphasis is on the technology needed to provide the computing environment that supports massive storage; the second describes those projects that rely on use of the facility.

Nine faculty will be involved in projects that primarily concern the design and analysis of the massive storage facility. These studies include the design of an archiving and version-management system, an investigation of trade-offs among the various kinds of storage, the development of communication mechanisms in such a distributed system, the design of fast I/O architectures, and the study of fundamental algorithms used in such systems. Performance studies will provide essential feedback for later phases of these projects. In addition to the research activities, some development work by technical staff will be needed to support the facility.

A group of about a dozen additional faculty will profit from the enhanced environment as users, developing a wide variety of approaches, algorithms, and techniques that depend on the availability of massive storage. Large, reasonably priced secondary storage systems will permit the creation of knowledge bases of significant sizes for research in artificial intelligence, natural language understanding, and machine learning. The storage of large numbers of readily accessible high-resolution raster images for use in computer vision, computer graphics and geometric modeling will also be exploited. Rich representations and powerful manipulations of complex objects will be developed, supported by a sophisticated next-generation data base system and a variety of new user interface capabilities.

2. Design and Analysis of a Massive Storage Facility

2.1. Operating System Research (David Anderson, Domenico Ferrari, John Ousterhout, Alan J. Smith)

The operating system component of the proposed work has two major aspects: basic support and research. The first portion consists of a collection of relatively mundane tasks that must be completed to provide basic access to the massive memories. These include extensions to existing systems to support memories and disk farms of the envisioned size, plus new kernel support for optical disks, which are not currently supported by our systems at all. This portion will be carried out primarily by the project support staff, except where use of the system exposes interesting new research problems.

The second (and more interesting) portion of the operating system work consists of new research areas that the envisioned system will open up. The paragraphs below describe some of the projects we currently plan to pursue, which center around the file system and the communication system. We first describe the research in performance evaluation that will be necessary to obtain the desired (basic) properties from the system; following that are outlines of planned work on more radical designs which include the ability to reset the system state to some earlier period, and the possibility of such a system without magnetic disks as staging devices. Communications aspects of the system are described last.

System Overview

As described elsewhere in this proposal, we envision a massive multi-level file system, consisting of a large number of smaller (3.5", 5", 8") high density magnetic disks, backed by a very large optical storage system. Each disk will hold on the order of .5 x 10^9 bytes, and the optical storage system will hold between 10^{11} and 10^{12} bytes. The optical storage media are particularly inexpensive, and can be written to as an almost free resource, although eventually not all optical storage will be kept on line. We refer to each optical disk as a "WORM" (write once, read many).

In normal operation, information will be read from and written to magnetic storage, and will be migrated, either automatically or by user request, to optical archival storage. Optical disk will also be used as a replacement for magnetic tape for the purpose of incremental (daily) and full (periodic) disk backup. A WORM is considerably more convenient than tape, and backup and retrieval can be accomplished largely without an operator, by using a large on-line juke-box for the WORM.

Performance Studies (Smith, Ousterhout)

A hierarchical distributed storage system, as described in this proposal, is effective to the extent that

(a) The capacity appears to be that of the largest level,

(b) the average access time is close to that of the fastest level,

(c) reference to data is (almost) transparent as far as its storage location or storage level, and

(d) all levels and location have sufficient bandwidth to satisfy average transaction loads and adequately handle peak loads. Ideally, such a system would also have reliability mechanisms such that data is never lost or destroyed, and integrity mechanisms, such that multiple, inconsistent versions of the same information were not permitted.

In this sub-section, we discuss the management of the distributed hierarchical mass storage system, with regard to performance evaluation studies and the implementation of effective management algorithms. The research proposed here is unique because of the availability of a large mass storage system from which measurements may be taken, experiments run, and algorithms and mechanisms implemented.

Our proposed research on the development of high performance algorithms for the management of such a file migration system will address questions such as:

(a) When do we (automatically) push files from fast to slow storage? (see[Blac87a, Smit81a] for earlier research on this topic.)

(b) When a file is referenced, should a group of files be fetched instead?

(c) No file system is infinite, and the existence of a huge "bit bucket" will simply result in everyone saving everything. A mechanism is needed to compact the file system around those files actually still in use, as opposed to earlier versions of files. How and when to compact the file system, and how to physically organize the files after compaction is an interesting problem for study. To the extent that the system supports the "time travel" described below, it will be necessary to preserve almost all file versions, but the likely rarity of such time travel suggests that we emphasize access to current and recent versions.

(d) In line with (a) and (c) above, we note that files will be periodically migrated from "on-line" to "off-line" as the mass store fills up - we do not believe that the storage will be "infinite". The 10^{12} bit photostore formerly in use at Lawrence Livermore National Laboratory represented only a 9 month on-line buffer, before data was moved to the "shelf." The problem is to select the files for movement to the shelf and to reorganize the file system afterward so as to maximize performance. (We do not just want to migrate "platters" offline, treating the store as a FIFO buffer, as did LLNL.)

(e) We will look at the data structure and organization issues in a file system of the envisioned size and structure. Some of the interesting research issues derive from the novel nature of the media: for example, WORMs will require new approaches to managing file maps and directory structures, since they cannot be rewritten in place. Other issues stem from the sheer size of the system: for example, are there efficient ways of managing generation data sets (such as are used in MVS), where all previous file generations remain because of the nonerasable nature of optical disk? Finally, there are user interface issues, since we not only want to avoid cluttering memory with inefficient structures; we want to avoid confusing the user with the display of undesired information. (See [Smit81b] for a discussion of I/O and file system optimization techniques.)

(f) The file system described will be a distributed system with caching and file migration throughout. There are performance issues in selecting when and where to move files when they are in use by multiple users or by a sequence of different users[Kure87a, Porc82a]. Closely related is the subject of multiple cache consistency, which has been (is being) studied in the context of shared main memory [Swea86a] ; we also have several projects underway to explore cache consistency in the very different environment of a shared file system (preliminary results already appear in[Nels87a] and[Thom87a]).

(g) Such a system will also implement disk caches in main memory (see[Smit85a] and[Nels87a]), providing another level to the hierarchy. We will study algorithms for the design and management of disk caches (fetch and replacement algorithms, block sizes, cache size determination, consistency and locking considerations, etc.). We will consider in this study the issue of multiple caches, both horizontally (multiple machines) and vertically (caches at the file server, and caches at the worker machines).

Time Travel (Ousterhout)

The role of optical disks depicted in the previous section is a relatively conventional one, where WORMs replace tapes as the slowest layer in a storage hierarchy. This section and the next describe two unconventional roles that optical disks might play in a file system.

One of the most interesting aspects of using optical disks instead of tapes for backup and file migration is that they allow the backed-up information to be kept on-line and easily accessible. For example, recent measurements of usage patterns suggest that a jukebox holding 50-100 WORMs can hold several years of backup data for a community of substantial size. This offers the possibility of providing efficient version management facilities as an integral part of the file system. We propose to implement automatic version management in the Sprite operating system[Oust87a] through the notion of ***time travel.*** Time travel will restore a user's view of the file system to what it was at a particular time in the past. For example, there might be a "ct" command analogous to the UNIX "cd" command except that it changes the current time instead of the current directory. "Ct 30-Jan-85" would cause the entire file system to appear to the user as it did on January 30, 1985. The operating system would automatically select files and directories from optical archive or magnetic disk in order to sustain the illusion of backing up time. This facility could be used in a variety of ways, from restoring a software project to the time of a previous release in order to chase a bug in that release, to recovering mail received several years ago but since deleted. In comparison to traditional version control systems, time travel has the advantages of being automatic (users need not specify when to archive new versions), and providing easy access to consistent groups of files (as long as there is a time when such a consistent group existed).

Three of the most important questions we hope to answer are: can time travel be implemented in a transparent and efficient fashion? are there sufficient uses for it to justify the costs associated with its implementation? and does time travel provide a superior form of version control in comparison to existing version control mechanisms?

The massive amounts of archival storage available will also permit a "version" file system such as DEC's VMS to be used as it was originally intended. In such a system, whenever a file is stored, a new version is created. In most systems, those old versions are rapidly erased, since disk space is at a premium. We expect to be able to keep versions indefinitely, until explicitly erased for reasons other than lack of storage. Conversely, we also expect our implementation to include a mechanism for truly "erasing" a file, most easily by zeroing the WORM region (if the physical write mechanism so permits), or by recopying and compacting the disk. It must be possible to destroy information irrevocably.

Replacing Magnetic Disks (Ousterhout)

Another innovative study will address the question of whether optical disks can replace magnetic disks altogether. A study of file accesses on timeshared UNIX systems[Oust85a] suggests that file system working sets are small: if a few tens of megabytes of main memory are dedicated to a cache of recently-used file data, it will rarely be necessary to access disk (assuming a few tens of typical users). In such a system, the primary purpose of magnetic disk would be as backup in case of a crash or power failure. Yet magnetic disks are themselves subject to catastrophic failures (head crashes) so they also have to be backed up. Why not eliminate the magnetic disk entirely, and replace it with a large main-memory cache for fast access and an optical archival store for backup and versioning?

The most important issue in this approach is the rate at which file data would have to be written to optical store in order to provide an adequate degree of crash protection. Would optical disks fill at a rate that would be economically or administratively infeasible? Fortunately, the file access study suggests that with current file access patterns and a write-through policy, only about .01 optical disk would be consumed per user per day (hence a jukebox of 50 optical disks could satisfy the needs of about 15 users for about a year before some of the disks would have to be replaced).

We propose to implement a file system without magnetic disks as part of the Sprite operating system and measure whether it can support a user community of reasonable size with acceptable performance and resilience. This project will have to address several interesting research problems, such as how to organize the optical disks, what writing policy to use for the main-memory caches, and how to selectively copy information from full disks to new empty ones in order to keep the most useful information most accessible. For very large files being slowly modified, such as large databases, frequent copying may be impractical; fortunately, our measurements in[Oust85a] indicate that the great majority of files are small. One of the goals of the research will be to identify the range of file sizes and access patterns where an optical-disk-only approach is most useful, and to investigate techniques for handling even the largest files (e.g. permit updates to part of a file without copying the entire file). We will also consider the interaction between this work and the time travel research discussed in the previous section.

Communication Architectures for Access to Large Objects (Ferrari and Anderson)

Another operating system research project will address the communication issues associated with very large memories. The proposed hierarchy of very large memories will be used for a range of experimental applications. For many (perhaps most) of these applications, the following will hold:

(1) Data will be communicated over a network, either because the information manager and object data are on different machines, because the client is on a remote machine, or because the object data itself is distributed.

(2) Object interfaces will require the efficient communication of very large amounts of data. We anticipate that very large memories must be accessible by correspondingly high speed communication in order to be generally useful.

(3) In many cases, the object interface will have real-time constraints. This will hold for digitized audio and video data.

(4) Eventually, access to large objects will often occur through long-distance high-delay networks.

Current communication architectures (such as the TCP/IP networking system in 4.2 BSD UNIX) probably cannot meet the communication performance needs of many potential applications of the proposed large information facility. The implementation of TCP/IP in 4.2BSD, for instance, cannot transmit data at a sustained rate greater than 1.5 Mb's when running on 1 MIPS machines circa 1984. Also, it has no real-time capabilities. We therefore propose to conduct research in communication architectures at both the network and operating systems levels to address these needs. This research will be undertaken as part of the DASH project[Ande87a], and will involve the development of prototype operating system kernels on both the "prototype information servers" and the client workstations.

The central component of this new communication architecture is the hierarchy of software protocols. Such protocols have a range of reliability, performance and complexity properties. In some applications request/reply protocols are appropriate, while in others stream-oriented protocols are preferable. Bandwidth, delay, buffering, caching, flow control, and communication security requirements will be dependent on the application. In addition, more sophisticated protocols will be needed for use over high-delay networks.

For performance reasons, it is generally necessary for protocols to run in the operating system kernel. Therefore, the operating system used to access the large memory system (at both the client and server ends) must provide a flexible and efficient framework for protocols.

The DASH project is investigating a new general-purpose communication architecture for distributed systems. This architecture is based on "real-time message streams" having guaranteed performance properties (delay and bandwidth). Streams can be bound together into "bundles", and can be dynamically bound to "protocol processes" and "filter processes". The DASH communication facility is integrated with the virtual memory and process scheduling components of the system to remove performance bottlenecks and provide performance guarantees at the user level. It can also utilize shared-memory multiprocessors for faster communication.

We believe that a new approach to communication is crucial for the proposed large-object facility, and that the DASH approach is a viable approach. The availability of the large-object facility will allow us to experiment with the new communication architecture, using it to construct protocols suitable for the major applications. We will therefore be able to evaluate its flexibility, its adequacy for supporting such a large variety of requirements, and the effectiveness of the resulting protocols in the local-area environment as well as in its extension to high-performance wide-area networks.

2.2. Research in Computer Architecture (Randy H. Katz, David A. Patterson)

The architectural research community has made great strides forward in the last few years in the areas of processor architecture and memory system design. Considerably less effort has been directed towards the issues of I/O system design. The proposed research is concerned with the general questions of how to achieve very high performance I/O subsystems. I/O is often viewed as an arcane art by the academic community, and the opportunity is ripe to reexamine I/O subsystems in the light of advances in VLSI and central processor architecture. In particular, we are interested in the technological issues of how to dramatically increase the available I/O bandwidth. We plan to use the proposed equipment to perform experiments to explore new organizations of I/O systems.

For many years disks have been getting bigger but not faster. Is I/O therefore destined to be the bottleneck that prevents us taking advantage of exponential growth in CPU speed and memory size? We think not, for while disks are not much faster, the push from the personal computer marketplace has created much cheaper disks. With inexpensive hard disks and single chip disk controllers now becoming commercially available, it is possible to build a high bandwidth disk system from several personal computer disks, if parallelism can be exploited.

We do not believe that it is possible to study hardware problems in a vacuum. We believe the recent success Berkeley has had in the systems area may be due to our emphasis in building and measuring real hardware and software systems. To drive our experiments, we must choose a well-understood and well-instrumented application that requires a large I/O bandwidth. Because of its need for large I/O capabilities and the local expertise in database technology at Berkeley, we have chosen high-performance database transaction processing as our target application. In many other environments, one can simply provide massive amounts of main memory to reduce I/O bandwidth requirements through buffering. However, this tactic usually fails in transaction processing systems. In addition, so called "hot spots" provide a significant stress test of the data manager's ability to remove performance bottlenecks. Consequently, this proposal focuses on transaction processing as the ultimate client of our efforts. The transaction processing environment will be built on top of an extension of POSTGRES, an object-oriented data manager currently under development (see Section 3.2). A proposal for this study is being submitted separately.

An extremely challenging performance target, requiring significant CPU and I/O activity, is 1000 transactions per second, where each transaction is a debit/credit style transaction. By relying on many small/inexpensive disks instead of a few large/expensive ones, system I/O bandwidth can be increased *while simultaneously reducing* the system cost.

Assuming for now that the processing power exists for the 1000 transactions per second, the question is then whether the I/O power exists. For the style of debit/credit transaction we are considering, estimates have been made of 2 to 8 I/O's per transaction. If we vary the number as well as the speed of the disks, we can estimate the number of disks that must be actively servicing I/O requests to achieve 1000 transactions per second (TPS):

Number of active disks for 1000 TPS			
	2 I/O per trans	5 I/O per trans	8 I/O per trans
"Minicomputer" Magnetic Disk	50	126	201
"Personal Computer" Magnetic Disk	86	214	343

Between 50 and 350 active disks are needed to service 1000 TPS. If restricted to conventional mainframe disk technology, it is beyond reasonable cost or physical limits to build in this fashion. Because of the small size of the personal computer disks and because of the single chip disk controllers, 6 or 8 disks could be packed per board, thus making it feasible to include these disks on or near the CPU boards. Using current prices, the system cost is estimated to vary from $150,000 to $980,000 using today's technology:

Cost for 1000 TPS system			
	2 I/O per trans	5 I/O per trans	8 I/O per trans
"Minicomputer" Magnetic Disk	$280K	$630K	$980K
"Personal Computer" Magnetic Disk	$150K	$300K	$460K

This "back-of-the-envelope" calculation suggests the potential of fast computers and cheap disks, as comparable performance systems are quadruple processor mainframes with large disk farms. This leads us to the experiments and questions we will explore to determine the practicality of these ideas:

(1) What is the reliability of a rack of inexpensive disks? What error correction techniques are appropriate in this environment? How many extra inexpensive disks are needed to exceed the reliability of mainframe disks?

(2) While a farm of inexpensive disks has high bandwidth, the individual latency is much worse than that of a mainframe disk. What is the latency of queries for data spread over dozens of disks? Can full-track-buffer disk controllers overcome the latency problems?

(3) Inexpensive disks also have much lower capacity than mainframe disks. Can databases separate slowly changing data so that a WORM jukebox can inexpensively make up the loss in storage capacity? What will be the impact on performance of relying on WORMs?

(4) Not surprisingly, reducing I/O's per transaction reduces the number of required active disks in the system and this is the most significant way to reduce cost. A major source of the I/O activity in a database (or file system) is just putting data on disk in case of a power failure. We propose to utilize a battery backup system for the gigabyte main memory. Can the database and operating system software treat main memory as stable storage? If so, how is I/O traffic reduced? Will a one-gigabyte main memory need better error correction schemes if data are not touched for several days?

(5) If the CPU performance and I/O bandwidth seems adequate, the key issue is whether the bus can support that amount of I/O activity. We will investigate several schemes to reduce the I/O bus bandwidth demands by an order of magnitude. This includes what we believe is a promising organization of attaching I/O devices directly to the processor caches in a multiprocessor rather than the more conventional approach of connecting them to shared memory. This allows the caches to act as multiple I/O ports to the main memory. This in turn opens another set of interesting issues: How will I/O into the cache affect the performance of each processor since I/O displaces data from its cache? Do caches need to be enhanced with software options to prevent such interference? Can smaller I/O blocks be loaded from full-track-buffer controllers and acted upon directly by a program in the local CPU, so I/O never goes to main memory?

In summary, this equipment gives us the opportunity to examine a long neglected area of computer architecture. By using POSTGRES as the driving force, we have a challenging I/O intensive application to drive this new study. As our first example of past neglect and new opportunities, we see that promising new levels of high performance and low cost can be achieved by observing that: (1) the largest disks are not that much faster than the slower disks, and (2) the price of the smaller disks has been reduced because of the volume demand from personal computers and workstations. The proposed equipment provides a testbed that will uncover the strengths and weaknesses of new approaches to I/O.

2.3. Network Event Management (C. V. Ramamoorthy)

In a distributed computing environment, events occur asynchronously on all computing nodes. Each event brings the computing environment from one global state to another. In order for network clients to properly respond to the changing network state, it is desirable to have the following services available in the network environment:

(1) *Answering queries about the system state*: Availability of the state information allows clients to evaluate the current situation before taking actions.

(2) *Notifying clients of state changes*: A client may wish to fire actions when certain predicates on the system state are true. Instead of having all clients monitor the system status, it is desirable to have a distinguished agent that notifies interested clients when

the system enters certain states so that the status collection effort is not replicated.

(3) *Maintaining desirable states by invoking proper actions*: A system may occasionally enter undesirable states due to design errors, poor distribution of resources, or hostile attacks. It is crucial that the system be able to reconfigure itself in some way to maintain the system goals or to bring the system back to desirable states.

We intend to build a network server that maintains a network status database and fires actions when certain predicates on the network state or its history become true. Such a network service would support applications that require fast responses to rapidly changing network state and protection against design errors, such as air traffic control, industrial process control, stock exchange markets and battle management.

Our group has been working on several research issues that are relevant to solutions to the global state management problem, including Reconfiguration Control in Dynamic Networks[Rama82a] and Global Information Management[Rama82b, Gane84a]. Our previous work will be integrated into our framework for event management, which is described next.

Proposed System Layering

We will build the target system using a layered approach. Each layer presents an abstraction and reduces the complexity visible to upper layers. Our research effort will concentrate on the third, fourth, and fifth layers.

The first layer is the communication subsystem layer. It provides primitives for setting up reliable communication channels between processes on the same host or different hosts. This layer is well supported by the Berkeley 4.3 BSD UNIX as it implements the DARPA Internet protocols and provides a socket facility [Leff83a] for processes to establish communication among each other in a convenient way.

The second layer is the clock synchronization layer. It guarantees that clocks of all non-faulty nodes are *well* synchronized. We intend to adopt the Berkeley TEMPO clock synchronization algorithm[Guse83a, Guse85a], which satisfies many nice properties such as close synchrony among clocks, closeness to real time, local causality, and fault tolerance.

The third layer is the status-maintenance layer. A status maintainer is responsible for maintaining a network status database by collecting status reports from local status maintainers on individual nodes. The history of status changes is also recorded for detecting event sequences and various inference needs. Each status report is time-stamped using the local clock time. The status collected must be presented in various abstractions to network clients to expedite the decision making process. Derived abstractions must be updated continuously since the network state changes dynamically. To solve this problem, knowledge of the system status must be structured in a way that would facilitate real-time processing, reasoning and planning.

The fourth layer is the event management layer. A network event manager accepts service requests from network clients (processes or users). A request is of the form {event, action}. When a registered event occurs, the corresponding action is taken on behalf of the client. The action can be as simple as sending a notification to the client, or as complex as initiating a planning process to construct the detailed action sequence.

The event manager can be extended from a request-driven system to a goal-driven system. Instead of registering a set of specific requests at the network event manager, a client can specify only its goal. The event manager can continuously verify that the goal is achieved in the network environment; if not, the event manager can generate plans automatically to achieve that goal. However, in a dynamic environment, a time-consuming plan generation process would not be able to respond effectively. Real-time distributed planning, which might include learning from experience or situation matching, must be developed to meet the response time requirement.

The fifth layer is the distributed computation and application layer. Based on the information provided by the event manager, distributed computations or other applications can adapt to the changing network status. Initially, we would like to use the event manager to assist in certain resource management functions and provide event notification services to network clients. Our future plan is to use the event manager in three research areas: (1) as an on-line monitor for distributed simulations, (2) as an interactive debugger for distributed programs, and (3) as a project coordinator in a distributed software development environment.

The proposed facility will provide essential support for our research. We propose to have individual local status maintainers maintain a shared network status database. This approach eliminates the need for a global status maintainer and the processing of status report packets at the network event manager. Moreover, multiple event managers with various access control policies, accuracy requirements, and caching strategies can easily share the network status database and provide proper services to different classes of clients. Sufficient memory, accessible over the network, is essential. With a high speed network, status reports can be transmitted more frequently and the network status database can be kept more accurate. Currency of the status information is critical for making real-time decisions on resource management such as process migration or process allocation.

The optical disks will be used to record the history of the network status database. The history database can be utilized in many ways: (a) Statistics of critical system parameters can be calculated from the history database. (b) A system administrator may trace the origin of system failures or degraded system performance by studying event sequences recorded in the history database. (c) Ample data would be available for carrying out trace-driven simulations. All the above information would facilitate the studies of the behavior of distributed systems and give birth to better algorithms for resource management.

Implementation Summary

Although certain research issues still require further investigation, to gain some concrete experience, we have already implemented a prototype network event manager on VAX 780 machines and Sun-3/50 workstation clusters; both systems run Berkeley UNIX[Chen86a]. We adopted the entity-relationship-attribute model proposed by Chen[Chen83a], with procedural attachment to model the objects and relations in the network environment. On our Sun workstation clusters, the network status database is kept by individual local status maintainers at a shared file server, which simulates a huge shared memory accessible to all network nodes. To avoid performance degradation, the network event manager automatically migrates from a heavily loaded host to a lightly loaded host when necessary. The event manager has been used to provide event notification services, support allocation of dynamically created processes, and assist in migration and rebooting of transaction-based servers. Our preliminary implementation experience has provided many insights to the event management problem and proved the feasibility of the approach.

2.4. Algorithmic Issues in Massive Information Storage (Richard Karp, Raimund Seidel)

The efficient use of very large memories obliges us to reexamine the fundamental algorithmic techniques and data structures that we employ for data storage and retrieval and the solution of combinatorial problems, and to pay serious attention to new regions of the time-storage tradeoff curves that arise in these problems. The following paragraphs outline some of the issues that we will investigate.

Searching

The fundamental problem here is to access those parts of a large knowledge base or data base that are relevant to a particular task or query. Novel issues will arise because of the complexity, longevity and heterogeneity of the data objects being stored and the sheer size of the files being searched.

In one class of problems we can assume that each data object is described by many different access keys or attributes, and that the relevance of a given data object to a query will depend on several of these attributes. Important special cases will include partial-match queries, in which it is desired to access all those objects possessing a certain set of attributes, as well as close-match queries, in which we seek all objects whose vector of attributes is close, in some metric, to a given vector. The efficient implementation of such searches can be approached by bucket techniques, in which the collection of data objects is partitioned into subsets, or buckets, only a few of which will be relevant to a given query. Some work has been done on the construction of such partitioning schemes based on various kinds of combinatorial designs, but much more work remains to be done.

There are other retrieval problems in which the relevance of a data object cannot be specified in terms of a match on a finite preassigned list of attributes. For example, the data objects might be rules in an expert system, and relevance might be determined by a term matching algorithm; or the data objects might represent scenes or pictures, and relevance might be determined by the presence of some particular object in the scene. Typically, very few of the objects in the data base will match, and the keys to rapid search are (i) to partition the data base so that the potentially relevant data objects are localized to a few regions or buckets and (ii) to devise an algorithm for term or picture matching that terminates very rapidly in the typical case, where there is no match.

Sorting

Although there is an immense literature on sorting, the selection of a sorting method in a given situation is not a routine matter. The method of choice depends on a multitude of factors: the structure, size and number of the records to be sorted, the nature of the key on which sorting is done, the statistical distribution of the keys, the size and nature of the mass storage medium, the size of main memory, the transfer rate between mass storage and main memory, the speed of the central processing unit or units being used, and so forth. In practice, merge sorting is the most commonly used method for large data sets, but it is our belief that distribution, or bucket sorting methods, will prevail in the realm of extremely large problems. In such a method the range of values of the keys is partitioned into intervals, and the keys are distributed into buckets according to the intervals in which they lie. Then, in later passes through the data, the keys in each bucket are sorted, and, finally, the contents of the several buckets are reassembled into a single sorted file. In order for this method to compete favorably with more conventional methods, two preconditions must be met: i) The distribution of the keys must be well understood, so that interval boundaries can be chosen that will ensure an even distribution of keys among the intervals; this will require the development of suitable on-line techniques for sampling and estimation. ii) The memory system must support fast random access to many buckets; a large disk farm would be suitable for this purpose.

Algorithms for Write-Once Memories

The use of a large write-once memory for sorting and searching poses novel questions of algorithm design. When standard read-write memories are used, methods such as Quicksort or Heapsort enable sorting to be done efficiently using very little storage other than the array that holds the keys being sorted. However, these methods may rewrite the contents of a given memory cell many times. Similarly, fast algorithms for implementing abstract data types such as dictionaries, priority queues and double-ended queues achieve space efficiency only at the cost of rewriting the same pointer field many times as the structure evolves.

We have begun to investigate how sorting and searching might be performed in a write-once memory. If unlimited storage is available we can implement the dictionary data type using a kind of balanced search tree, in such a way that each insertion into a n-element dictionary requires time O(log n), but may annex as many as log n new storage locations. In the case of sorting there appears to be a time-space trade-off. One can sort in linear space and quadratic time, but the fastest sorting methods seem to require space n log n. We will attempt to prove lower bounds showing that this trade-off is inherently necessary.

The Maintenance of Histories

The problem of recording the history of a complex data object over time arises in many contexts. In another section, John Ousterhout proposes to maintain the history of an entire file system, so that it may be backed up to its state at any past time, for purposes of debugging or error recovery. On a more limited scale, scan-line algorithms in computational geometry pose similar problems; in that case, time corresponds to the position of the scan line as it sweeps across a scene, and one needs the ability to recover a snapshot of the data that was recorded by the scan-line algorithm at any point during the sweep. The recreation of past versions of complex, changing documents poses similar problems. Driscoll, Sarnak, Sleator, and Tarjan[Dris86a] have initiated the study of persistent data structures, enabling the retrieval of data from past incarnations of dictionaries or priority queues. We propose to continue the study of efficient data structures for maintaining histories; we expect that there will be an inherent trade-off between time and space, so that the availability of a massive storage facility will lead to a great reduction in the time required to perform operations on past versions of a data structure.

The Maintenance of Views

Applications of massive storage systems to problems in vision, graphics and multidimensional optimization often require the storage and retrieval of a set of closely related data objects. Examples of such sets are the various lower-dimensional projections of a multidimensional surface, or successive snapshots of a time-varying image or scene. A fundamental problem in data structure design is to represent such an ensemble of data objects so that any one of them can easily be retrieved or regenerated, and then displayed. Often, the number of objects in the ensemble is infinite, as in the case of the projections of a multidimensional surface, or else extremely large, so that it is not possible to represent each data object explicitly in storage. One possible solution is to store a selected set of reference objects explicitly, and then to generate other objects by a process of interpolation. This approach has been used in computational geometry in connection with the hidden-surface problem that arises in generating two-dimensional views of three-dimensional scenes. It is also possible to use data compression techniques to reduce the space needed to store a reference view, and thus increase the number of reference views stored, at the cost of greater complexity in the regeneration and display of any given view. We intend to study the time-storage trade-offs that occur in connection with such problems.

DNA Sequence Analysis

The revolution in DNA sequencing techniques has made available a massive amount of data whose analysis may unlock some of the secrets of heredity and evolution and enable us to understand the nature of certain genetic malfunctions and hereditary diseases. The analysis requires string-matching routines that reveal similarities among the various sequences. A typical problem is that of string-matching with errors: given a short pattern x and a much longer string y, find all those blocks of y that coincide almost perfectly with x. Problems of this kind can be solved by dynamic programming algorithms that tend to be storage-limited. The combination of an optical disk to store a database of DNA sequences, together with a very large read-write memory to provide the large amounts of workspace required for dynamic programming algorithms, might enable significant advances in this field. New algorithms are needed to support more ambitious efforts to detect patterns and similarities in these sequences. There is a possibility for cooperation in this endeavor with the Genome Project proposed by the Lawrence Berkeley Laboratory.

Combinatorial Optimization

The availability of massive memories will enable new approaches to the solution of large combinatorial optimization problems. One example is the use of dynamic programming methods which have, up to now, been considered impractical because of their large storage requirements.

A second example is the implementation of the branch-and-bound method. In this method the possible sequences of choices in the solution to a combinatorial problem are represented by a tree, and the leaves of the tree represent possible solutions. In addition, each solution has a cost, and, associated with each node, there is a lower bound on the cost of any solution compatible with the choices made in reaching that node. The object is to find a leaf of minimum cost. The primitive operation is node expansion, in which the children of a node are created and their cost bounds evaluated.

Two contrasting approaches to implementing the branch-and-bound method are depth-first search and best-first search. In depth-first search, a single path through the tree is under consideration at any time, and the deepest node in that path is the one to be expanded next. In best-first search, the next node to be expanded is the one which has the lowest cost bound. Depth-first search minimizes storage requirements, while best-first search minimizes time, at the cost of a great increase in storage requirements. In today's practice depth-first search is the method of choice, because of its modest storage requirements. With the advent of massive read-write memories it will be possible to consider a whole spectrum of methods intermediate between depth-first and best-first search, and to choose an optimum point along the resulting time-storage tradeoff curve.

3. Use of a Massive Storage Facility

3.1. Research on Very Large Knowledge Bases (Stuart Russell, Robert Wilensky)

Several areas of Artificial Intelligence research require large fast memories and backup storage, mainly because of the need for large amounts of knowledge. The proposed facility will make possible important new research efforts in the creation, management and use of very large knowledge bases, and allow investigation of methods in an unexplored region of time/space tradeoffs.

Creating and Managing Very Large Knowledge Bases

The major barrier to further progress in many areas of Artificial Intelligence is the lack of a sufficiently large body of common-sense knowledge about the world[Lena87a]. This is particularly true in the areas of natural language processing, knowledge-based learning, computer-aided instruction, and planning.

Until recently, there have been no attempts to create large, heterogeneous knowledge bases. The CYC project [Lena86a] , currently underway at MCC, is a long-term attempt to create just such an encyclopaedic knowledge base. Despite a major commitment of resources, it is already clear that current hardware and software technologies are inadequate for dealing with the amount of information involved, especially since it forms a highly interconnected network requiring large parts of the knowledge base to be in fast memory at one time. A skeletal knowledge base containing less than 1% of the projected contents strains the limits of a 24-megabyte machine with two gigabytes of magnetic disk storage. Maintaining consistency and rapid accessibility will also require a large increase in processing power. The proposed configuration offers a possibility for progress in solving these problems. We plan to use the facility in order to construct a large, unified knowledge base which will enable progress in a number of different research projects in machine learning and natural language, as well as involving several research efforts in the creation, management and accessing of the knowledge base itself.

1) Meta-knowledge

In order to maintain consistency and correctness in a large knowledge base, the system itself requires knowledge about what the data is (meta-knowledge). The system must ensure consistency of vocabulary, usage and structure of knowledge even when it is being entered by many users simultaneously. We will therefore study the design of a knowledge base description language capable of supporting all of the processes for which the knowledge can be used.

Persons adding knowledge to the system must be given strong guidance for the task to be humanly feasible. We will therefore prime the system with knowledge sufficient to make it an expert on the methods of its own augmentation, as in[Davi79a]. The knowledge base will contain a representation of the knowledge entry process, involving the transfer of facts from humans to the system; a representation of the interface, to enable the system to take appropriate actions to guide the user in entering facts; and a representation of the system's goals and specific information needs, to enable it to generate strategies for speeding up the knowledge entry process.

The meta-knowledge itself, which underlies these abilities, will come from three sources:

a) Explicit provision by the user; this is only possible for sophisticated users.

b) Inductive abstraction from the base-level data by the system; this is possible, but difficult, for 'machine-readable' databases.

c) The appropriate solution may be a cooperative process, involving automatic production during the knowledge *generation* phase; we will design and build knowledge entry tools with highly structured generation 'contexts' that produce information about the data being generated, as well as producing the data itself. A trivial, existing example is a document processor which tags its output with the appropriate filetype. Less trivial examples might include symbolic annotations associated with the output of a graphics scene and animation generator, a symbolic mathematics manipulation system, and an expanded version of the data fields used by a mail system, including 'transaction type'[Wino86a]. The research described subsequently by Graham and Harrison will be important for this aspect of the problem.

2) Intelligent manager and interface

During the creation phase, the system's goals for self-augmentation are paramount. For the knowledge base to be useful, for instance for users to obtain any leverage from a massive database containing such things as the Library of Congress, advances will have to be made in intelligent interfaces that respond to the user's needs.

The Unix Consultant (UC) system [Wile84a] is designed to ascertain user goals from natural language dialogue, and to form plans which the user can execute to achieve those goals. It thus provides a useful prototype for an intelligent knowledge system aide. We will continue research into approaches to modelling the user (also requiring a large common-sense knowledge base) to ascertain the user's goals and abilities, not just his or her explicit commands, and into planning methods that can achieve those goals. The knowledge base manager will also perform inductive reasoning on user session records and data characteristics (including usage records) in order to schedule resources and perform appropriate data migration.

Managing a large knowledge base therefore requires research on the representation and use of meta-knowledge that describes the contents of the knowledge base, and on the goals and planning algorithms of an intelligent knowledge base manager and interface. For small, uniform knowledge bases with a single creator, consistency and retrieval/presentation problems are manageable using *ad hoc* techniques, but experience on the CYC system has shown that a more principled approach is necessary in order to scale up to real-world knowledge bases. We plan to work in collaboration with MCC on these problems, and to implement a system on the proposed facility, concentrating primarily on the issues involved in creation and management of knowledge, leaving the generation of content to the MCC group.

Using Very Large Knowledge Bases

1) Inductive reasoning over large knowledge bases

Inductive reasoning involves finding regularities in observations. Small knowledge bases allow the possibilities of finding simple regularities at a slightly higher level of generality than the observations themselves. However, work in philosophy and AI [Good83a] [Russ86a] has shown that much more general, abstract kinds of knowledge are possible, and indeed necessary for good performance in complex worlds, and their acquisition is only possible from large bodies of knowledge, either heterogeneous (as above) or specialized, as in NIH medical databases. For example, from a database of Spaniards we can induce the rule that, generally speaking, people born in Spain speak Spanish. It would require a much larger database to discover that people born in a country generally speak the same language as their fellow-citizens. A still larger database is needed to learn the general knowledge necessary to actually explain this observation, the knowledge being at an abstract level concerning social groups, communication and life histories; but this knowledge would enable sense to be made of thousands of other, more specific observations, and represents exactly the kind of unexplicated common sense that AI systems currently lack. The computational problems we will tackle include the use of knowledge to reduce the combinatorics of the search for regularities, and the generation of goal-directed foci for the search.

The ability to store and rapidly access large knowledge bases will therefore enable, and motivate, fundamental research in machine learning, continuing the theoretical approach begun in[Russ86b]. We will study the inductive acquisition, representation and use of abstract knowledge from large knowledge bases, both general and domain-specific. (Negotiations are underway for a joint project with NASA Ames involving inductive inference on a set of 29,000 aircraft accident reports.)

2) Life histories for autonomous intelligent agents

The RALPH project (Rational Agents with Limited Performance Hardware) is a long-term project recently initiated at Berkeley to study software architectures and algorithms for situated, autonomous agents in a simulated, real-time environment, in which the agents derive their theories for dealing with the world from experience. This requires storing away enormous amounts of sensory data for subsequent analysis, essentially a complete 'life history'. Current hardware restrictions limit the length of simulation runs and the complexity of the sense data that can be used; unlimited and easily accessible storage would eliminate this problem. We plan to use the facility to do research on the abstraction of high-level representations and regularities from a large stream of sensory snapshots.

Other AI methods, particularly truth maintenance[Doyl79a] and uncertainty management have been inadequately investigated because of their large storage requirements. We will use the facility to allow incorporation of these methods in the architecture of the autonomous agents, and to investigate the storage, retrieval and usage problems that actually arise when reasoning traces are kept for a significant proportion of the system's activities.

Initial funding for this research is already in the pipeline from Lockheed; Rockwell Science Center intends to support this work when money becomes available; a larger proposal will be submitted to NSF later this year.

3) Inference for natural language understanding

To build a system capable of understanding an interesting amount of text, an enormous knowledge base needs to be created. This knowledge is needed for making the common-sense inferences that readers of natural language texts must make in order to understand those texts. Applications include machine translation, automatic summarization of texts, and creation of knowledge bases from free text (for example, scientific articles or medical studies). Natural language dialogue (for example, as part of an intelligent interface) requires additional knowledge, beyond the domain itself, concerning conversational conventions and user goals.

The principal research area concerning the use of large amounts of common-sense knowledge in natural language understanding is its role in making inferences beyond the facts stated explicitly in the text, thereby allowing the meaning of the text to be correctly ascertained. We have developed techniques for making such inferences. Our approach involves a form of parallel search through the knowledge base[Norv87a]. Since the connectivity of the data is high, a very large fragment of knowledge needs to be readily accessible to perform such searches. Thus, in addition to coping with the volume of knowledge already discussed above, the proposed facility will be crucial in allowing real-time access to the knowledge for natural language dialogue. To make this practical, we will study algorithms for efficient network traversal when the size of the knowledge base is substantial.

3.2. Object Management in Databases (Michael J. Stonebraker, Lawrence A. Rowe)

There is common consensus that relational database systems are appropriate for business data-processing applications where there is a large volume of fixed format, relatively simple data. However, they fail badly when asked to manage data that is more complex in nature. Examples of non-business data with this property include documents, CAD objects, computer programs, and spatial objects such as icons or bitmaps.

We are developing a next-generation database system and applications programming environment whose objective is to manage conventional business data as well as to manage efficiently the sorts of objects that are used in non-business applications. This database system, POSTGRES, builds on the notions of abstract data types, extendible access methods, and procedural representation of objects to provide required support and is described in the next subsection. The construction of POSTGRES has been in progress for a year, and we hope to have a working prototype by the end of 1987.

The POSTGRES system will be enhanced by an object based programming environment for database applications called OBJFADS. The OBJFADS system will impose performance and functionality requirements on POSTGRES that will provide a good test of the successful use of the proposed facility. An overview of OBJFADS follows the description of POSTGRES.

POSTGRES

POSTGRES is designed to leverage four major ideas: abstract data types, procedures, rules, and a no-overwrite storage manager. Our abstract data type system borrows heavily from the programming language community, where the same idea has been extensively investigated. However, in a database environment, it is also crucial to provide fast access paths to new kinds of objects. For example, spatial objects such as boxes, lines, and polygons require specific spatial access methods such as R-trees [Gutm84a] and KDB-trees [Robi81a] in order to be efficiently searched. Hence, POSTGRES is designed so that normal access methods such as B-trees and hashing can be included, as well as any access methods that a knowledgeable user wishes to write. In this way specific optimized access paths can be provided by the implementor of a new type.

Procedures have been widely used in frame-based languages in the AI community such as FRL[Robe77a] and SRL[Fox84a]. In POSTGRES, procedures are used as a representation tool to model objects with a complex internal structure, as well as a vehicle to support transactions consisting of several query language statements, such as TP1[Tand85a]. Lastly, they can be used to model objects with unpredictable composition. In POSTGRES, the query language has been extended with constructs to allow the efficient exploitation of procedural objects.

The third major concept that POSTGRES is exploiting is the notion of an integrated rules system. There are many applications, such as real time control of physical plants and automated trading of stocks on an exchange, where human expertise must be integrated into data management capabilities. For example, an expert system that assisted a human plant operator would want to be alerted when specific events happen (for example the reading on meter A is greater than twice meter B). Such a program wishes to store rules in a database system and have the data manager automatically fire them. POSTGRES rules are, in fact, query language commands which provide the illusion that they run indefinitely. Hence, they are easy for a user to grasp, since he or she must already know the query language. Also, they are powerful enough to perform most rule management functions[Ston87a].

Lastly, the POSTGRES storage manager is designed so that data is never overwritten. In this way, previous data is available for retrieval, and is automatically archived to a large capacity storage system. This allows a user to ask historical queries and also supports a much simplified crash recovery scheme[Ston86a].

POSTGRES is explicitly designed to use the proposed large optical disk system. This facility will allow us to run POSTGRES in a realistic setting and to experiment with optimization of the subsystem which spools historical records onto the archive. It will also allow users to run historical queries on large databases, so we can ascertain the value of that construct.

POSTGRES is also coded to make use of a very large main memory buffer pool. Hence, we expect to use the main memory of the proposed facility to accelerate database access to modest-size databases. Again, a realistic environment will allow users (and us) to ascertain the utility of POSTGRES constructs.

Lastly, we are excited about a large number of magnetic disks, because it will allow us to construct a file system which spreads blocks in a single file across several disks. Such a parallel file system allows POSTGRES to decompose individual commands into subpieces that can be executed in parallel on a tightly coupled multiprocessor.

OBJFADS

We are developing an object-oriented programming environment for POSTGRES, called OBJFADS, that supports the development of multimedia applications (i. e., applications that involve text, graphical, image, and voice data). The system supports the development of "what you see is what you get" (WYSIWYG) application interfaces that make users more productive. The same benefits are extended to application developers because the programming environment uses the same kind of interface. In fact, the program development environment is an OBJFADS application. The system also provides support for interface extensibility to match the extensibility in POSTGRES and for asynchronously triggered events generated by rules in the database.

A portion of the object hierarchy (data and procedures) will be stored in the database and shared by other applications. All instances of a class defined with metaclass *dbclass* are automatically stored in the database. Database objects that are referenced in an application are implicitly retrieved from the database into an "object cache" in the application program. Updates to *dbclass* objects are automatically propagated to the database and other applications that have the object in their "object cache." The OBJFADS run-time system will implement concurrency control and crash recovery protocols to control sharing and to protect the data from system failures.

Our future plans include developing application program support for executing rules stored in a POSTGRES database. We expect that an application "rule cache" similar to the "object cache" will be required to support complex expert systems efficiently. This system will be an important part of any applications developed at Berkeley where high-quality, flexible user-interfaces are required.

3.3. IC-CIM – An Application for POSTGRES and OBJFADS (Lawrence A. Rowe)

We are developing applications to manage the Microfabrication Laboratory at Berkeley[Hodg87a] using the OBJFADS and POSTGRES systems discussed previously. This research tests the effectiveness of these two systems when applied to a complex manufacturing information system. The long-term goal is to completely automate the laboratory. First we describe the system that we are building; then we discuss the way that the proposed hardware will assist us.

The systems we are currently developing as part of an automation facility are:

(1) A facility management system that will maintain information about facility layout, equipment locations, and utility connections. The data representing the facility will be stored in the database and a WYSIWYG interface will be developed that allows laboratory personnel to determine the location, connections, spatial relationships between the various entities in the facility, equipment and utility dependencies, and status information

on the environment. The database representation will use the graphics representation developed as part of the UNIGRAFIX project[Sequ85a].

(2) A "work in progress" (WIP) system that will keep track of all lots processed by the fabrication facility, guide operators through the processing steps to be performed, automate the execution of steps where the equipment is directly connected to the system, collect and store measurements taken during the process, and permit the scheduling of work to maximize throughput. This system includes several different applications including a WIP executer, a scheduler, a process-flow editor, and a laboratory manager tool. The database representation for process and equipment descriptions, the WIP executer state, and measurement data uses an object-oriented data representation that is being implemented on top of POSTGRES[Rowe86a].

(3) A wafer checking system that will visually scan a wafer and determine whether it is being correctly processed. For example, wafers can be checked after a photolithography step to determine if an error has occurred during processing. This application can institute corrective action (for example, asking the operator to reset the equipment, execute an equipment diagnostic, or schedule equipment maintenance).

(4) An equipment maintenance system that will keep track of equipment operations, qualification, and repair. This system will include interfaces to display statistic summaries of equipment histories and rules to recognize potential problems before they occur.

These applications have been chosen because they represent a variety of applications that involve many different types of data (business and engineering)[Rowe87a]. Moreover, the applications are related in that data from one application can be used by another application. For example, the facility management data will be used by the equipment maintenance system to determine what other equipment might be impacted by a problem traced to a particular utility network (for example, water quality) and the wafer checking system is called from the WIP system.

The proposed facility will have a major impact on this research. All of the IC-CIM applications will generate considerable amounts of data that must be analyzed and studied. Moreover, the wafer checking system and facility management system will store images to represent various wafer states and pictures of pieces of equipment. The optical disks and large memories will be crucial elements in these systems.

3.4. Integrated Interactive Development of Complex Objects (Susan L. Graham, Michael A. Harrison)

As the hardware and software aspects of computing have advanced, increasing attention has been given to the nature of the assistance that computing systems can give to their users in performing complex computing tasks. Two of these tasks are document preparation and software development. We have been designing and implementing two systems to support these activities – the Pan language editing system and the VorTeX document preparation system. In the period ahead, we will integrate and extend the two systems to provide flexible user access to structured information through the use of multimedia presentation techniques, persistent object bases, and enhanced analysis techniques.

The proposed facility plays an important role in our research. The information representations used during interactive sessions are very large and require rapid access and update. Transparent access to large primary memory is essential. The knowledge that is built up over time about the programs, documents, and other structured language-based objects developed in this environment will be captured, maintained, and retrieved for use in subsequent sessions. That capability requires both the secondary storage components of the proposed facility and the research results envisioned in other sections of this proposal by many of our colleagues.

The subsections that follow describe the Pan and VorTeX systems and their potential integration.

The Pan Editing System

Pan is an interactive editing system that combines a text editing capability with support for high-level manipulation of structured objects such as programs[Ball86a, Ball87a]. The components of the system are description-driven, so that multiple language definitions and presentation styles can be supported. In addition to syntactic editing, browsing and viewing, Pan will have a capability for rich semantic annotation, both formally, by means of incrementally evaluated logic-based semantic specification, and by means of commentary. Semantic analysis is used not only for checking of well-formedness, but also for browsing and presentation.

We intend to use this richness not only for incremental checking during program construction but also to explore ideas for program understanding, for checking and enforcing extralingual program properties (for example, conformance with project standards) and for recording a variety of other information. Our notion of language extends beyond traditional programming languages. It also includes such examples as design languages, interface definition languages, process description languages, and the language of documents (paragraphs, sections, and so on). Especially in the earlier stages of our experimentation, we are maintaining large amounts of data associated with structured objects and their interrelationships. A small piece of language text (be it program, design, or whatever) may have a very large amount of associated information.

Since the system is used interactively, it must update the semantic information incrementally. We are currently developing the necessary algorithms. Our previous experience developing complex algorithms to support code generation[Aigr84a] was that until we achieved a deep enough understanding of the issues, we needed considerably more space for our computations.

We also intend to study some granularity and persistence issues. Our semantic information constitutes a small object base. In principle, all the information resides in a POSTGRES-like store; in practice, much of the information is too ephemeral during development to be worth storing (much less keeping). However, once it becomes reasonably stable, it becomes important persistent information. We have the following research questions. Is the management of that information application specific, or will the techniques proposed for managing the information storage facility suffice? Can there be a transparent continuum between local data and 'stored' data? Will it be transparent with respect to performance?

One of the purposes of our approach to program development is its applicability for software reuse. Successful software reuse has several aspects - effective retrieval of reusable components, incorporation of those components in a new environment, and the ability to modify components in a reliable way. (See Richard Fateman's example in his discussion of binding time later in this proposal). Each of these aspects requires augmentation of the traditional descriptions of computations and related objects solely as commented program text. Language-based mechanisms for effective storage and retrieval of information in massive repositories will be essential. Object oriented paradigms and late binding will be important but will probably not suffice. Particularly in the period before we understand this problem, we will need to keep large amounts of information about software objects. (Design decisions, development history, test data ...) As much as possible, this information will be structured and formal, but it will be very diverse.

The VorTeX Document Preparation System

VorTeX is an integrated document preparation environment capable of producing high quality technical documents which involve mathematics, text, and graphics. The major research goals of the VorTeX project are the following:

(1) To produce an integrated user interface to support a complete technical document preparation system

(2) To provide dynamically consistent multiple representations of documents

(3) To establish links with other related and relevant software systems.

Multiple representation systems are discussed in [Perk84a] and [Chen87a]. In VorTeX, both source and target representations of a document are maintained and presented. The source representation refers to a TeX document in its original unformatted form; the target representation presents its formatted result. The user can edit both representations using a text editor and what is called a proof editor, respectively. Changes made to one representation propagate to the other version automatically. It is easy to support source modifications that cause the target representation to change, but one of our principal research issues is to go from a target version back to the source version.

The system reformats a document and redisplays it on the screen incrementally. Only the part of the document or the subregion of the screen that is affected by recent changes is reprocessed. The environment has support for automatic production of indexes for books, support for bibliographic material, spelling checkers, and other document-related utilities. The system runs on, but is not restricted to, workstations with a high resolution bit-mapped display. There is a high degree of interaction with the user[Chen86b]. Our current system uses Sun workstations and the X window system. In order to provide device independent high quality graphics, we use PostScript[Adob85a].

Future research topics include the following:

(1) Support for Composite Objects
We want to support not only text, mathematics, tables, and graphics, but also non-textual objects such as raster images, audio, and functional objects. For example, we want to be able to integrate a complex object from another system, such as a spreadsheet, into a document without losing its particular functionality. It is not too hard to incorporate these kinds of objects with special-purpose mechanisms. What we seek is a coherent general integration with the base system.

VorTeX, and many other similar systems, achieve their functionality through using complex internal representations[Chen86c, Sand78a]. Then sophisticated transformations are applied to give the different views. Thus the internal representations become extremely large. To solve the problems involved in managing and accessing those representations, we need access to large amounts of primary and secondary storage, as well as new structuring mechanisms.

(2) Animation
Our implementation of graphics allows primitive animation so that it is possible to integrate animation into compound documents. We intend to enhance that aspect of the system. There is great potential for applications of such compound documents in education, as well as in other domains where it is important to display dynamically changing

situations.

(3) Bibliographic Assistance
In future versions of the system, we will extend our bibliographic interface. We can now "click" on a citation and a new window provides information on the reference. In the future, we will be able also to display the actual source reference material. Further examples of browsing are easy to imagine. For example, a reader of an electronic version of this proposal might want to click on reference [VorTeX] and then see the formatted bibliography entry. A further click might open to a window with the text of the paper in it. Another menu choice might be to open another window and run a canned demonstration of the VorTeX system in it.

Integration Issues

An obvious and relatively easy form of integration of Pan and VorTeX is to use a common text editing capability, a common window manager, and other shared user interface utilities. However, the deeper and more interesting research issues come up in adding new aspects to the two systems in a common way and in treating objects constructed by users as composite objects "understood" simultaneously by both systems. Here are some examples of the kinds of issues we will investigate:

(1) A Pan object must be known to the system both as a document, with all the formatting power of VorTeX, including audible and animated annotation (documentation), and as a linguistic object with the syntactic and semantic structure and manipulations supported by Pan. It is not yet clear to what extent multiple coordinated representations will be used, as opposed to a single very rich representation.

(2) A VorTeX object will probably require a language-based formatting description, together with the linguistic-based editing and manipulation capabilities of Pan, rather than the LaTeX macro enhancements currently used with TeX.

(3) The secondary storage of these complex objects in an object oriented persistent data base must be worked out. One problem is how to store and maintain histories of document revisions both as documents and as linguistic objects. This problem has some similarities with the traditional problem of designing source code support systems, but the complexity of the representations and the introduction of semantic information will force the development of new techniques.

(4) It should be possible to work with the documents and other linguistic objects in a shared environment. To do so raises issues of authorization and access control. For example, certain users may or may not have permission to change a document. Independently, some users may annotate it, while less privileged readers may not be able to do that. We intend to support users in maintaining variants of a document or program with a common base that can change over time.

3.5. Computer Graphics for the Visualization of Complex Objects (Carlo H. Séquin)

Interactive computer graphics can provide powerful tools to understand the structure of complex mathematical objects such as the high-dimensional "cost-functions" or "energy-surfaces" of multi-parameter optimization problems. One aspect of our research is to develop advanced tools for the visualization of such objects. In two recent research projects in which we wanted to use computer graphics for this purpose we have encountered significant limitations in what can be achieved due to a lack of storage media of one kind or another. This section demonstrates how the proposed massive information storage facility will significantly enhance our research.

Interactive Energy-Function Visualization Using a Gigabyte Memory

In two different projects, one relating to the multi-parameter optimization of analog integrated circuits[Koh87a], and the other concerning the convergence behavior of a neural network, the need arises to visualize a high-dimensional "cost-function" or "energy-surface" of about 4 to 10 parameters. In particular one would like to know the number of basins of attractions, their topological "connectedness", and the sharpness and slants of the ridges separating the valleys in order to devise a good strategy for automatic optimization tools or for learning algorithms. With the proper hardware support we can build tools and techniques to make it easier to visualize such higher-dimensional functions.

One of the tools that we plan to build is an interactive projection tool, that takes 2 to 4 slices through this higher-dimensional space and displays the function in these slices by mapping the energy value into a range of colors. In conjunction with the right computer graphics equipment (such as a PIXAR) we would like to develop an interactive manipulator that permits us to change smoothly and interactively the slice positions through this cost-function. The proposed large semiconductor primary memory will hold all the precomputed cost-information encoded in suitably chosen color maps to enable rapid loading of any particular hyperplane to the graphics display station. The keyboard will select a subset of, say, three parameters out of this higher dimensional space. The corresponding subset of data will be loaded into the PIXAR memory, and three slices can then be moved interactively with the mouse through this 3-dimensional sub-volume of data, permitting the user to follow the valleys and ridges of the cost-function minima and maxima.

A little computation shows that we need a substantial amount of memory: If we have five dimensions with 64 dots each, we need 2^{30} words, or at least one gigabyte of memory; and each additional dimension would increase the memory requirement by a factor of 64! We plan to study more effective coding schemes, so that it will be possible to move interactively through more than five dimensions.

We will also need a substantial amount of compute power to compute that information. If each point takes a hundred operations on a 100 MIPS machine, then we compute 2^{20} words in a second and need 2^{10} seconds - or 16 minutes - to fill the memory. A shared memory multiprocessor with this sort of speed residing on the network would be highly beneficial in this task.

Interactive Pictorial Manuals for Geometric Tools

The UNIGRAFIX system is a geometric modeling and rendering system that runs under the UNIX 4.2BSD operating system on VAXes and on Sun workstations. The system comprises a set of rendering programs for many different devices ranging from ASCII terminals through high-resolution color graphics terminals and hardcopy dot-raster printers. The system includes a collection of generator programs that form UNIGRAFIX descriptions of such objects as geodesic domes, mitred prismatic tubes along arbitrary paths through 3-D space, or pairs of matched gear wheels[Sequ85a]. Some of the tools in the UNIGRAFIX system have a large variety of possibly interacting option parameters, and it is often difficult to convey the effect of these parameters without the use of pictures. Sometimes even pictures are marginal, and one would like to provide a whole series of images or, better yet, a movie sequence. For instance, the influence of certain surface modeling parameters on the appearance of that surface in an environment with several objects that could be reflected and several light sources is best experienced by watching the effect as the parameters change.

Suppose we want to imitate various materials for geometric modeling and rendering with a sophisticated ray-tracer[Kaji83a, Glas84a, Mars87a]. These materials are distinguished by the way they reflect, refract and scatter light from the light sources to the human eye. Realistic rendering requires that several coefficients, for instance color, diffused reflection, specular reflection, transmitted fraction, transmission attenuation, and index of refraction, be properly set. To make matters more complicated, these coefficients may depend on the wavelength of the incident light and on the angle of incidence. The richness of these choices that need to be made is often bewildering to the user. An interactive tool is clearly required to teach the user the influence of the various parameters and efficient ways to achieve the desired look of the rendered object.

Since real-time ray-tracing is still something that not even the most powerful supercomputers today can provide, we might precompute a relevant set of examples and store them on disk for rapid perusal. The most desirable way of organizing the material for such a tutorial would be to store pictures in all relevant parameter directions. The user could then 'move' along any one of the desired axes, change the parameter value (almost) continuously, and observe the effect on the displayed scene interactively.

However, a little computation shows that even optical disks are insufficient to solve this problem in a brute force way. Assume that there are only six relevant coefficients and that we would like to show the effect for 10 discrete values for each. This would require storing one million images if we want to give the user the opportunity to move around freely in this six-dimensional space. More sophisticated approaches for selecting and organizing images will be required. It is the goal of our research to find effective solutions to this problem.

Combined Use of the Gigabyte Memory and of Optical Disks

Both applications described above stretch the limits of what one might achieve with the technology expected to be available in the next 3 to 5 years. The crude solutions outlined above will be inefficient, provide limited resolution along the various axes of the high-dimensional spaces we want to explore, or may not be truly interactive.

However it seems promising to exploit a clever combination of the use of the best features of both types of storage devices. To stay with our second example, we are currently evaluating whether it is practical to run a worst-case ray-tracing rendering of the demonstration scene, assuming all surfaces to be potentially reflective and refractive, and to trace the rays to some limiting depth, say 8. In the worst case this would result in 254 secondary rays for each ray that bounces suitably through the scene. All the geometrical tests for ray/polygon intersections are computed only once, and the resulting arithmetic expressions that show how the color of an individual pixel on the screen is composed from these 255 contributions are stored on an optical disk. When the user chooses a particular set of material coefficients, the chosen numbers are plugged in, the expression for each pixel is evaluated on the fly, and the result is sent to the screen.

As described, a full image could be regenerated in a matter of seconds. If we want to perform the display-update in real time, we need more sophisticated ordering and grouping of these computations. A first idea is to run-length-encode adjacent pixels on the same scan line with the same expression for their display value; the expression evaluation will then be carried out for all of them simultaneously. With a suitable indexing scheme and some clever caching scheme involving the large semiconductor memory, we can further reduce the necessary computations and treat much bigger groups of pixels with the same computation. For instance, think of all the background pixels as referencing the result of a single expression for their color value.

Conclusion

With the emerging optical disk technology and large random access memories, pictorial presentation of ideas will become practical. The key task is to find ways to organize a multidimensional set of mutually related displays and to provide rapid random access to them. This research is part of our broader goal to make graphical entities equal citizens with text-files in the UNIX environment. The optical disk is the right medium to store the large amounts of data associated with images.

3.6. Research in Computer Vision (Jitendra Malik)

The major theme of our research in computer vision is the inference of geometric descriptions of a scene useful for object recognition. This is to be done either from a single image or from sequences of images. Storing images requires a lot of memory. A 500 X 500 pixel image with 8 bit resolution occupies a quarter megabyte if it is a grey scale image and three-quarter megabytes for a 3-color image.

The research we are conducting requires both rapid access to images and secondary storage of scenes for retrospective analysis. The proposed facility will allow us to study the following interesting problems.

Problems requiring expanded main memory.

A very large main memory on the order of 100 - 1000 megabytes would be useful in the development and testing of storage-intensive algorithms. This corresponds to being able to keep approximately 400-4000 black and white images or 130-1300 color images at a time in main memory. This will enable us to experiment with a whole new class of algorithms for the following two problems:

(a) Analyzing time-varying images:
The importance of motion has been well recognized both for biological systems and artificial vision. In lower animals like frogs and flies the best developed vision module is that of motion detection and measurement. For robots, relative motion can be used to segment objects in the scene considerably more reliably than is possible with a single view.

A major problem with analyzing time varying images is the correspondence problem – finding the points in the different images which correspond to the same point in the scene. This can be done easily if the views are closely spaced, however that makes measurements of the changes between images inaccurate. The solution then is to use a large number of closely spaced images. This approach has been successfully demonstrated at SRI by Bolles, Baker and Marimont[Boll87a] for some simple types of camera motion e.g. when the camera is aimed perpendicular to its linear trajectory.

We will consider the problem for a more general situation allowing for (a) the camera trajectory to be an arbitrary planar curve, and (b) object motion. One motivation for this is to be able to use mobile robots.

(b) Object Recognition:
The problem here is to identify the name, position and orientation of the objects in the image, given a database of geometric descriptions of the objects. It is a very difficult problem, if one considers the variety of shapes, reflectance and textures, which could be imaged under different lighting conditions and viewed from different directions.

Present day algorithms have limited applicability. One approach to this problem, introduced by Koenderink and Van Doorn[Koen79a] is that of precomputing the 'aspect graph' of topologically distinct views of an object. The gaussian sphere of viewing directions is partitioned into connected sets such that in each set the graphs corresponding to the projected line drawings are isomorphic. Koenderink and Van Doorn had specified how this could be done for an object bounded by a single smooth surface, and more recently Gigus and Malik [Gigu87a] came up with an algorithm for polyhedral objects.

Unfortunately the worst case size of the partition can be quite high ($O(n^6)$), where n is the number of vertices in the object. For practical objects, the partition is probably much smaller and can be reduced further by introducing a multi-resolution hierarchy. Nevertheless, it seems clear that a large main memory is needed in order to use this algorithm in practice.

Problems requiring massive secondary storage.

A large secondary (optical disk) storage on the order of 10 - 1000 gigabytes will permit us to store an extensive database of images taken under a variety of imaging situations. That would considerably facilitate the testing and validation of computer vision systems developed by us and other researchers. In addition to this, we will use the database for the following two tasks:

(a) Measurement of scene statistics:
Measurement of scene statistics has an important role in the design of computer vision systems. Various vision tasks, when examined in their mathematical formulation are underconstrained. Inferring 3-D shape from photographs and line drawings is an obvious example. In biological vision these tasks are made solvable by the help of additional assumptions which reflect statistical and other prior knowledge about the scene. For example, in cultural scenes junctions are often interpreted as arising from edges which are mutually orthogonal in 3-D space.

A large database would facilitate the measurement of various scene statistics. Some examples–how bright are specularities?, how sharp are shadow edges?, how good is the lambertian assumption for the diffuse component of reflected light? This information would be of considerable help in designing general purpose vision systems.

Sometimes these statistics can suggest new constraints and algorithms. An example of the nontrivial consequences of such studies is the Maloney-Wandell color constancy algorithm [Malo86a] which exploits the fact that typical illuminations, reflectances have a spectral distribution that can be adequately represented using only 3-5 basis functions. If one does not exploit this statistical fact about the world, the problem is underconstrained and unsolvable.

(b) Providing examples for learning algorithms:
A database of images could also be used as a training set for 'learning' algorithms. An obvious example is that of learning the discriminating function for texture segmentation. Both the Beck and Julesz models tell us that textural segmentation occurs as a result of differences in the first-order statistics of local features of textural elements such as length, orientation, color, density etc. The weights to be given to the various terms can be determined by giving a set of images with or without texture boundaries and adjusting the weights to yield the correct result in each instance. One of the earliest strategies for this is the one used by Samuel[Samu63a] in his checkers learning program; there have been others. We will develop these ideas further in the context of our applications and study them both theoretically and experimentally.

3.7. Using Gigabyte Storage for the Visualization of Shape and Control Parameters (Brian A. Barsky)

The *Beta-spline* is a recent mathematical formulation developed expressly for representing curves and surfaces in computer graphics and computer aided geometric design[Bars81a, Bars87a]. One of the ideas underlying the Beta-spline is the addition of two *shape parameters* that control *bias* and *tension*. Bias is a weighting of tangent vectors between two adjoining segments (patches) of the curve (surface). As the tension parameter is increased, the curve (surface) tightens and converges to the control polygon (graph). Experience has shown that these shape parameters provide a natural way to control shape, even for a mathematically-naive user. Mathematically, one of the novel aspects of the Beta-spline is the substitution of a notion of *geometric continuity* (unit tangent vector and unit curvature vector) for the *parametric continuity* (in terms of the parametrization) required by traditional formulations.

The Beta-spline has been generalized to allow the shape parameters to vary along the curve or surface, thereby enabling even more local control[Bars83a, Bars83b]. In this case, the shape parameters can either be set *globally* with one value for the entire curve or surface, or *locally* with a different value for each joint. Furthermore, from a design system standpoint, a user can set an initial global value, and then refine the design by altering a few local values.

Recent work has concentrated on generalizing geometric continuity to n^{th} *order geometric continuity*, or G^n continuity and on developing the mathematical expressions for these conditions. It is this form of continuity that yields shape parameters; general interpretations and mathematical expressions are being derived for higher-order Beta's[DeRo85a]. We will generalize Beta-splines to higher degree to reflect this higher order geometric continuity. One of the goals of generalizing geometric continuity to higher order is that the resulting continuity conditions can be used as a foundation upon which to construct higher-order Beta-splines. With this higher order will come more kinds of shape parameters. We intend to study the behavior of such parameters. Such a study can be facilitated by the use of a gigabyte storage.

Gigabyte storage will enable us to compute and store many images depicting different values of each of the various shape parameters, and to do so for several different order Beta-splines. The result will allow interactivity in a setting of pre-computed images, each of which might have been the result of extensive calculation.

The Beta-spline introduced two shape parameters using a local approximating spline curve or surface formulation. Later, a local interpolatory spline curve using the same parameters was explored by DeRose and Barsky[DeRo84a, DeRo87a]. Another approach for a local interpolating spline curve with parameters to control the shape was introduced by Kochanek and Bartels[Koch84a, Bart87a] in the context of computer animation. Their parameters, which they call "control parameters", are different from our shape parameters but have similar effects. In addition, there is a third control parameter, called "continuity."

We have generalized the use of these three control parameters for curves to modelling free-form surfaces[Du87a, Du87b]. The introduction of control parameters provides us with useful methods for representing surfaces. Using these methods, we are able not only to create smooth surfaces, but also to easily introduce and control discontinuities on the surfaces.

All these parameters need to be studied experimentally to gain an understanding of their behavior. In the future, we will also develop new parameters to respond to certain ideas of intuitive specification descriptors. To accomplish such studies requires the generation of images for different parameters. This leads to the idea of a space/time tradeoff, where such images for a wide variety of parameter values are pre-computed and stored in a gigabyte memory. Then, it will be possible to scan through the memory, examining these images. This will result in an animation simulation that will appear as if the images were being computed in real-time in response to the specification of parameter values. This will enable the interactive browsing through storage allowing the user to effectively alter the shape of the surface by adjusting parameters and seeing the results in real-time.

The capability just described will require the study of encoding schemes for images (Run Length Encoding is an example). Moreover, storing animation engenders the idea of developing encoding schemes for image sequences, perhaps considering the *change* between images. Finding efficient schemes for storing image sequences taking into account changes as well as a basic static encoding amenable to such approaches will be a crucial area of our work.

3.8. Symbolic Scientific Computing (Richard J. Fateman, Wm. Kahan)

Symbolic mathematics systems have historically represented some of the largest and most memory-intensive programs in the research community. As one of the most successful areas of application of Lisp, such systems have had a substantial effect on support environments.

The Macsyma system at MIT prompted numerous efforts to acquire large memory systems, and in the period from 1968-78 inspired innovations in operating systems (ITS for the DEC-6/10) and architecture (the MIT Lisp Machine). At UC Berkeley, Lisp and the Macsyma system were once again a major driving force for large-memory systems, this time on the VAX. In addition to being a factor in originating the Berkeley UNIX project, certain features (e.g. "vadvise") were specifically added to UNIX for Lisp/Macsyma paging performance. A similar situation prevailed for early versions of the Sun Microsystems 4BSD. operating system. Benchmarks for the shared-memory multi-processor SPUR (Symbolic Processing Using RISCS) project[Hill86a] are also drawn from symbolics mathematics.

Although the demands of systems of the past for physical memory and address space are quite modest compared to available commercial mainframe and engineering workstation systems with 8-32 megabytes of physical memory, important problems of the past which were simply not worth doing in a heavily disk-bound environment can be considered once again. Issues which once seemed important (such as software paging of un-used programs) can be ignored; issues of memory management such as Lisp garbage collection or alternatives using other technology, can now be explored more easily. In the past, we assumed there would always be a generally severe penalty for systems in which address space exceeds physical memory. Now it appears that physical memory will be quite large, and penalty-free address space will be even larger. Lisp implementations have traditionally benefited from large "flat" address spaces. Lisp machine architects of the 1970's and 1980's advocated short addressing modes (CDR-coding). When fully-qualified addresses require (with tags) more than 32 bits, how should they be encoded? Should Lisp systems promote the use of segmented address spaces ("areas")? Gigabyte address spaces will require resolution of these problems.

We project that symbolic mathematics systems and environments for scientific program development and execution can rise to the "challenge" of using new facilities and can provide the benchmarks for new developments in fields of operating systems, data representation, and architecture. The underlying technology of Lisp and related programming language issues may also present challenges.

The following paragraphs describe our current research directions.

More Flexible Binding Time

One of the major tensions in computer programming language design and implementation is the choice which must be made between compile-time and run-time analysis and execution of a higher-level-language algorithm.

In conventional languages, compile-time is used for syntax analysis, certain types of error detection or correction, semantic analysis, optimization, and a variety of other tasks. Yet in today's object-oriented languages, once a program is apparently correct (so far as a compiler can tell), many decisions (bindings) still remain: the typical generic or polymorphic function must still figure out its arguments and their types at run-time. Various architectural features (tags, invisible pointers) have been built to minimize the costs. Yet these are never entirely adequate: a clever compiler given the precise types (or even constant values), can perform a vastly superior optimization of an algorithm compared to that of a run-time system.

We propose to explore systems in which optimization can continue even at run-time.

(1) We are pursuing the use of large scale "tabulation" (sometimes called "memo-ization") in which the input-output relations of side-effect-free functions are stored in a hash-table and later retrieved when the same input is used. Examples from large symbolic mathematics problems suggest this is a viable technique for saving time and reducing program complexity. (This technique is used in the Univ. of Waterloo "Maple" system[Char83a].)

(2) We are also exploring the use of program transformations, providing a higher level optimization either by specific request or automatic means. Macsyma has a number of programs that will pick out common subexpressions, transform polynomials to Horner's rule form, compute power-series approximations, unroll loops, etc. These can be used explicitly, although whether this is done at run-time or "pre-compilation" time is open to debate.

The automatic optimization can be explained by an example: when it appears that an interpreted function is going to be evaluated numerous times (as when a mathematical function is plotted), we propose to transform it to an efficient compiled function. In particular, if $f(x,y,z)$ is plotted for z ranging over the interval [-1, 1], then x and y are presumed to be constants and eligible for various optimizations; the program can be specialized for this computation. In fact, it may be appropriate (especially in expensive cases such as solids modelling) where it is known that f will be applied to many argument sets, to approximate f itself (e. g. by Chebyshev approximations). Existing programs provide the capability to write out symbolic expressions in a form suitable for parallelizing Fortran compilers. A more restricted version of this approach has been developed by the TAMPR project at Argonne Labs[Boyl82a]. The researchers there have demonstrated that from one model routine (say, for Gaussian elimination), they can produce dozens of specialized codes, for various precisions (single, double), various domains (complex, real), various representations (tri-diagonal, symmetric, sparse, etc.). Sometimes such specialized programs have better asymptotic running times than the model routine would suggest.

We are eager to pursue more general results in the context of symbolic mathematics systems: whenever a new type-vector of arguments is provided to a function, an incremental analysis and compilation could be performed that would result in a newly optimized code. This idea was proposed in the Newspeak system[Fode84a]. This system, in fact, became too unwieldy for mere 12 megabyte VAX systems available at the time.

This technique is important for symbolic mathematics in that there is a fairly well understood, huge, and badly-in-need-of-optimization, collection of programs running on diverse types. For example, we have hierarchical types such as "truncated power-series in one variable (z) with coefficients in the quotient-field of polynomials in the two variables (x,y) over the ring of integers".

In general, we believe the availability of large spaces in which to cache precomputed *values* and precomputed specialized *procedures* can revolutionize the role of the system and application programmer, who must otherwise spend significant resources deciding what to throw out and what to keep. Now the prospect of a system which learns facts and procedures can emerge from truly large memory systems. Symbolic mathematics is a "hard" area yet with some important technical consequences for A.I. reasoning discussed in an earlier section of this proposal. Although there may still be a need for occasional "garbage collection" it may be removed to a time when it is convenient, or perhaps done unobtrusively as has been proposed for various incremental schemes.

Searching/mathematics knowledge

Another new direction we intend to pursue with huge memories is the storage of otherwise poorly-indexed information. Consider the world's currently largest table of indefinite and definite integrals of elementary and special functions[Prud86a]. A preliminary study suggests this table could be represented in perhaps 10 megabytes, but its use could be vastly enhanced by a multi-dimensional search simultaneously on various keys (e.g. there is a square-root, there is a cosine, there is a logarithm). Our preliminary estimates suggest that it would be sufficient to have (say) 1 megabyte of keys available in memory, and the full table stored on (say) magnetic or optical disk[Brav86a].

This is a specialized instance of a very general problem: How do you rapidly find a close match in a large table. Traditional methods of search require either an exact match (for hash-coding), or ordered search. Ordered search where the cost of testing the order is as expensive as (say) a pattern match or a unification algorithm, can be prohibitively expensive in a large table.

Various direct indexing techniques based on discrimination vectors similar to key-word searches are a start to finding information in our example of the integration table; we suspect that ultimately, a pattern-match plus more specific information about derivatives, domains, and other constraints must be used. The use of such a general framework involving pattern matching, procedure invocations, and perhaps other object-oriented programming ideas may be a way of approaching such partly-ordered search in other domains. We have been discussing these prospects with Professor Karp and Seidel (see their section of this proposal).

3.9. Run-Time Support for the Integrity of Huge Software Collections (Wm. Kahan)

The paradigms by which we manage collections of information that matter to just ourselves and a few colleagues do not serve adequately to manage huge collections of information intended to serve a wider public. Because each user is far less confident of his ability to find his way through a large corpus than a small one, the large corpus can be poisoned more readily than the small by a relatively tiny dose of misinformation. That is why the integrity of a large collection of information poses serious technical problems out of all proportion to our experience with our own collections of information in personal libraries and computer files. The problems are more difficult with software than with other forms of data because software is intended to be executed rather than merely read, to act rather than merely be acted upon. The acquisition of a facility with very large storage capacity will encourage the creation of vast archives of software, and thus will oblige us to confront the problem of software integrity.

Our primary concern is with the diagnosis at run time of anomalies that show up in software that was presumed to have been debugged. These anomalies may be the first evidence that something is still wrong with the software; pursuing that evidence to a correct conclusion is the first step toward the restoration and maintenance of integrity in software. Many of these anomalies stem from the bugs that arise when programs transgress the implicit limits of computation on finite machines. These limits manifest themselves as signals caused by overflow or (in floating point) by underflow, as inaccuracies due to finite precision, and as invalidities caused by attempts to access nonexistent parts of complex data structures. We have developed a set of policies for handling such exceptions in a generally satisfactory way at tolerable cost. These policies are designed so that a programmer need not be preoccupied with exceptions but may deal with them as afterthoughts if they arise; so that irrelevant exceptions that do not affect the final results of a computation are hidden from view; and so that what few exceptions come to the programmer's attention will be localized well enough that he or she can decide easily what to do about them. The policies are based on the appropriate use of default values such as the Nan (not a number) data type of the IEEE floating-point standards 754 and 854[Cody84a], on the use of flags to signal to the user that his program has had to do something disputable, and on the use of modes to enable or disable the detection of various exceptions. They also involve a technique called "Presubstitution", which is a generalization of the "Default" values required by the IEEE Standard. The aim of this technique is to handle exceptions automatically by software designed to hide the irrelevant exceptions from users, without recourse to the spaghetti-like control paths required by most contemporary programming languages. There are real-life examples [Kaha87a, McLe87a] in which appropriate policies used to treat the sign of zero, and to handle exceptions associated with attempting to divide by zero or to compute the square root or the logarithm of a negative number, can spell the difference between success and failure.

Our schemes have been tested separately on various systems over the past two decades, but have not yet appeared together in any one system. They are compatible with imprecise interrupts, a disagreeable characteristic of some of the fastest computer architectures now and very likely of most of them in the future. The schemes work best if integrated into the operating system and the language processors, but that is not essential. What is essential is that they work satisfactorily even if the applications programmer is unaware of their existence.

No matter what policy might be adopted in advance to cope with a class of exceptions, someone will have good reason to take exception to that policy. Consequently, a computer system must be equipped with something we call Retrospective Diagnostics that memorializes every instantiation of these policies, so that a computer user who doubts their wisdom can investigate whether they hurt him. Our experience with primitive implementations of Retrospective Diagnostics[Kaha66a] gives convincing evidence of the essential role they play when numerous users of innumerable layers of software they cannot (and do not wish to) read find themselves wondering what went wrong. David Barnett, a former student, is currently engaged in a more ambitious implementation of Retrospective Diagnostics on a VAX running 4.3 BSD UNIX.

Continuation of this work will provide a significant technique for reducing the overburden of conceptual difficulties faced by a naive user of an elaborate library. Growth of such libraries, as we incorporate a more complete environment in integrated application systems will inevitably provide a challenge to programmers and end-users. We intend to reduce the behavior of complex systems to comparatively few carefully stated mathematical and pragmatic useful assertions. By providing the tools, both to the programmer/tester and the end user to investigate problems stated at an appropriate linguistic level (presumably closer to high-level language or application level than machine language), we hope to enhance productivity of the scientific application developer. This work is the next logical step in a progression begun with the earlier, necessary foundation of IEEE arithmetic standards.

4. References

Adob85a.
Adobe Systems, Inc., *PostScript Language Manual,* Addison-Wesley Pub. Co. (1985).

Aigr84a.
Philippe Aigrain, Susan L. Graham, Robert R. Henry, Marshall Kirk McKusick, and Eduardo Pelegri-Llopart, "Experience with a Graham-Glanville Style Code Generator," *Proc. SIGPLAN'84 Symposium on Compiler Construction,* (June 20-22, 1984). Also appears as SIGPLAN Notices **19**(6)

Ande87a.
D.P. Anderson, D. Ferrari, P.V. Rangan, and S.-Y. Tzou, "The DASH Project: Issues in the Design of Very Large Distributed Systems," Report No. UCB/CSD 87/338, University of California, CS Division (EECS Dept.), Berkeley, CA (January 1987).

Ball86a.
Robert A. Ballance, "Design of the PAN Language-Based Editor," unpublished working paper, Computer Science Division, EECS Department, University of California, Berkeley, CA (February 1986).

Ball87a.
Robert A. Ballance, "Pan: An Introduction for Users," Version 1.2, Computer Science Division, EECS Department, University of California, Berkeley, CA (July 1987).

Bars81a.
Brian A. Barsky, "The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures," Ph.D. Dissertation, University of Utah, Salt Lake City, Utah (December, 1981).

Bars83a.
Brian A. Barsky and John C. Beatty, "Local control of Bias and Tension in Beta-splines," *Transactions on Graphics* **2**(2) pp. 109-134 (April, 1983). Also in the SIGGRAPH'83 Conference Proceedings 25-29 July, 1983 pp. 193-218

Bars83b.
Brian A. Barsky and John C. Beatty, "Controlling the Shape of Parametric B-spline and Beta-spline Curves," *Proc. Graphics Interface '83,* pp. 223-232 (May, 1983).

Bars87a.
Brian A. Barsky, *Computer Graphics and Geometric Modelling Using Beta-splines,* Springer-Verlag, Tokyo (To appear, 1987).

Bart87a.
Richard H. Bartels, John C. Beatty, and Brian A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling,* Morgan Kaufmann Publishers, Inc., Los Altos, CA (To appear, 1987).

Blac87a.
Christina Black and Charles Farnum, "A Static Analysis of an NFS File Server," Final report, CS 262 term project, University of California, Berkeley, CA (May 1987).

Boll87a.
R. C. Bolles, H. H. Baker, and D. H. Marimont, "Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion," *International Journal of Computer Vision* **1**(1) pp. 7-55 (1987).

Boyl82a.
James M. Boyle, "Program Transformation and Language Design," pp. 285-295 in *The Relationship Between Numerical Computation and Programming Languages,* ed. J. K. Reid,North Holland (1982).

Brav86a.
M. Braverman, "ATHEIST: A Table-Driven Heuristic Integration System," CS282 class project, Computer Science Division, EECS Department, University of California, Berkeley, CA (1986).

Char83a.
B.W. Char et. al., "The design of MAPLE: A compact, portable, and powerful computer algebra system.," *Proc. EUROCAL '83* **162** pp. 102-115 Springer-Verlag, (1983).

Chen86b.
Pehong Chen, Michael A. Harrison, John L. Coker, Jeffrey W. McCarrell, and Steven J. Procter, "An improved user environment for TeX," pp. 45-54 in *Proc. Second European Conf. on TeX for Scientific Documentation, Strasbourg, France,* Lecture Notes in Computer Science **236**, Springer-Verlag, New York (June 19-21, 1986).

Chen86c.
Pehong Chen, John L. Coker, Michael A. Harrison, Jeffrey W. McCarrell, and Steven J. Proctor, "The VorTex document preparation environment," pp. 23-24 in *Proc. Second European Conf. on TeX for Scientific Documentation, Strasbourg, France,* Lecture Notes in Computer Science **236**, Springer-Verlag, New York (June 19-21, 1986).

Chen87a.
Pehong Chen and Michael A. Harrison, "Multiple Representation Document Development," UCB Tech Report, UCB/CSD 87/367 (July 30, 1987).

Chen83a.
P.S. Chen, *The Entity-Relationship Approach to Software Engineering,* Elsevier Science (1983).

Chen86a.
Yih-Farn Chen, Atul Prakash, and C. V. Ramamoorthy, "The Network Event Manager," *Computer Networking Symposium,* (November 1986).

Cody84a.
Cody et. al., "A Proposed Radix- and Word-length-independent Standard for Floating-point Arithmetic," *IEEE Micro,* pp. 86-100 (August 1984).

Davi79a.
Randall Davis, "Interactive transfer of expertise: Acquisition of new inference rules," *Artificial Intelligence* **12**(2)(1979).

DeRo84a.
Tony D. DeRose and Brian A. Barsky, "Geometric Continuity and Shape Parameters for Catmull-Rom Splines (Extended Abstract)," *Proc. Graphics Interface '84,* pp. 57-64 (27 May - 1 June 1984).

DeRo85a.
Tony D. DeRose and Brian A. Barsky, "An Intuitive Approach to Geometric Continuity for Parametric Curves and Surfaces," pp. 159-175 in *Computer-Generated Images -- The State of the Art,* ed. Daniel Thalmann,Springer-Verlag (1985). Earlier version in Proc. Graphics Interface '85 Montreal 27-31 May 1985 pp. 343-351

DeRo87a.
Tony D. DeRose and Brian A. Barsky, "Geometric Continuity, Shape Parameters, and Geometric Constructions for Catmull-Rom Splines," submitted to ACM Trans. on Graphics (1987).

Doyl79a.
J. Doyle, "A Truth Maintenance System," *Artificial Intelligence* **12**(3)(1979).

Dris86a.
James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan, "Making Data Structures Persistent," *Proc. 18th ACM Symposium on Theory of Computing,* pp. 109-121 (1986).

Du87a.
Wen-Hui Du, Brian A. Barsky, and Francis J. M. Schmitt, "New Formulations Using Brown's Interpolant with Control Parameters," *Proc. SIAM Conference on Applied Geometry,* (20-24 July, 1987).

Du87b.
Wen-Hui Du, Francis J. M. Schmitt, and Brian A. Barsky, "Modelling Free-form Surfaces Using Brown's Interpolant with Control Parameters," *Proc. International Conference on Computer-Aided Drafting, Design, and Manufacturing Technology,* pp. 240-247 (21-25 April, 1987).

Fode84a.
J. K. Foderaro, "The design of a language for algebraic computation systems," Ph. D. Dissertation, EECS Department, University of California, Berkeley, CA (1984).

Fox84a.
M. Fox et. al., "Experiences with SRL: An Analysis of Frame-based Knowledge Representation," *Proc. 1st Intl. Wkshp. on Expert Data Base Systems,* (October 1984).

Gane84a.
Shivaji Ganesh, "Availability and Consistency of Global Information in Computer Networks," Ph. D. Dissertation, University of California, Berkeley (Aug. 1984).

Gigu87a.
Z. Gigus and J. Malik, "Computing aspect graphs for line drawings of polyhedral objects," *submitted to IEEE Workshop on Computer Vision,* (1987).

Glas84a.
A.S. Glassner, "Space Subdivision for Fast Ray Tracing," *Computer Graphics and Applications* **4**(10) pp. 15-22 (1984).

Good83a.
Nelson Goodman, *Fact, Fiction and Forecast,* Harvard University Press, Cambridge, MA (1983).

Guse83a.
Riccardo Gusella and Stefano Zatti, "TEMPO: Time Services for the Berkeley Local Network," UCB/CSD 83/163, University of California, Berkeley, CA (December 1983).

Guse85a.
Riccardo Gusella and Stefano Zatti, "An Election Algorithm for a Distributed Clock Synchronization Program," UCB/CSD 86/275, University of California, Berkeley, CA (December 1985).

Gutm84a.
A. Gutman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. 1984 ACM-SIGMOD Conf. on Management of Data,* (June 1984).

Hill86a.
M. D. Hill et. al., "Design decisions in SPUR," *IEEE Computer* **18**(11) pp. 8-22 (November 1986).

Hodg87a.
D.A. Hodges and L.A. Rowe, "Information Management for CIM," *Proc. Adv. Res. in VLSI,* (Mar 1987).

Kaha66a.
W. Kahan, "7094 II System Support for Numerical Analysis," *SHARE SSD 159 item C4537,* pp. 1-54 (December 1966).

Kaha87a.
W. Kahan, "Branch Cuts for Complex Elementary Functions," in *The State of the Art of Numerical Analysis,* ed. M.J.D. Powell,Oxford University Press (1987).

Kaji83a.
J.T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," *Computer Graphics* **17**(3) pp. 91-102 (1983).

Koch84a.
Doris H. U. Kochaneck and Richard H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *SIGGRAPH'84 Conference Proceedings* **18**(3) pp. 33-41 (July, 1984).

Koen79a.
J.J. Koenderink and A.J. van Doorns, "The internal representation of solid shape with respect to vision," *Biological Cybernetics* **32** pp. 211-216 (1979).

Koh87a.
H.Y. Koh, C.H. Sequin, and P.R. Gray, "Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models," *Proc. ICCAD*, (1987).

Kure87a.
Oivind Kure, "File Placement and Migration in Distributed Systems.," Ph.D. Dissertation (expected December 1987).

Leff83a.
Samuel J. Leffler, William N. Joy, and Robert S. Fabry, *4.2 BSD Networking Implementation Notes*, CSRG, Computer Science Division (EECS), University of California, Berkeley, CA (July 1983).

Lena86a.
D. Lenat, M. Prakash, and M. Shepherd, "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks," *AI Magazine* **6**(4)(1986).

Lena87a.
D. Lenat and E. Feigenbaum, "On the Thresholds of Knowledge," *Workshop on the Foundations of Artificial Intelligence*, (1987).

Malo86a.
L.T. Maloney and B.A. Wandell, "Color constancy: a method for recovering surface spectral reflectance," *Journal of Optical Society of America A* **3**(1) pp. 29-33 (1986).

Mars87a.
D.M. Marsh, "UgRay - An Efficient Ray-Tracing Renderer for UniGrafix," CS Division Report No. UCB/CSD 87/360, University of California, Berkeley, CA (May 1987).

McLe87a.
P.J. McLellan, "An Equation Solver for a Handheld Calculator," *Hewlett-Packard Journal*, pp. 30-34 (August 1987).

Nels87a.
Michael Nelson, Brent Welch, and John Ousterhout, "Caching in the Sprite Network File System," *Proc. Eleventh Symposium on Operating Systems Principles*, (To appear, Dec 1987). Also appears as technical report UCB/CSD 87/345; March, 1987

Norv87a.
Peter Norvig, "Inference in Text Understanding," Ph.D. Dissertation, University of California, Berkeley, CA (1987).

Oust87a.
John Ousterhout, Andrew Cherenson, Fred. Douglis, Michael Nelson, and Brent Welch, "The Sprite Network Operating System," Technical report UCB/CSD 87/359 (June 1987).

Oust85a.
John Ousterhout et. al., "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proc. Tenth Symposium on Operating Systems Principles*, pp. 15-24 (December 1985).

Perk84a.
Charles L. Perkins, "The Multiple Representation Problem," Master's Thesis, Computer Science Division, University of California, Berkeley, CA (December 1984).

Porc82a.
Juan Porcar, "File Migration in Distributed Systems," Ph.D. Dissertation (June, 1982).

Prud86a.
A. P. Prudnikov et. al., *Integrals and Series (2 volumes)*, Gordon and Breach Science Publishers, (1986). Russian Publication 1983, 1984

Rama82a.
C. V. Ramamoorthy and Y. W. Ma, *Algorithms for Reconfiguration Control in Dynamic Computer Networks*, Computer Science Division (EECS), University of California, Berkeley, CA (1982).

Rama82b.
C.V. Ramamoorthy and S.L. Ganesh, "Global Information Management," *Proc. UCLA Packet*

Radio Analytical Workshop, (Aug. 1982).

Robe77a.
J. Roberts and I. Goldstein, "The ERL Manual," Memo 409, MIT, AI Laboratory, Cambridge, MA (September 1977).

Robi81a.
J. Robinson, "The K-D-B Tree: A Search Structure for Large Multidimensional Indexes," *Proc. 1981 ACM-SIGMOD Conf. on Management of Data,* (May 1981).

Rowe86a.
L.A. Rowe, "A Shared Object Hierarchy," *Proc. Int. Wkshp. on Object-Oriented Database Systems,* (Sept 1986).

Rowe87a.
L.A. Rowe and C.B. Williams, "An Object-Oriented Design for Integrated Circuit Fabrication," *Proc. Conf. on Data and Knowledge Sys. for Eng. and Manuf.,* (To appear, Oct 1987).

Russ86a.
Stuart Russell, "Preliminary Steps Toward the Automation of Induction," *Proc. AAAI-86,* (1986).

Russ86b.
Stuart Russell, "Analogical and Inductive Reasoning," Ph.D. Dissertation, Stanford University, Stanford, CA (1986).

Samu63a.
A.L. Samuel, "Some studies in machine learning using the game of checkers," in *Computers and Thought,* ed. J. Feldman,McGraw-Hill, New York (1963).

Sand78a.
Erik Sandewall, "Programming in an interactive environment: the LISP experience," *ACM Computing Surveys* **10**(1) pp. 35-71 (March, 1978).

Sequ85a.
C.H. Sequin, "Berkeley UNIGRAFIX, A Modular Rendering and Modeling System," *Proc. of the 2nd USENIX Computer Graphics Workshop,* pp. 38-53 (Dec. 1985).

Smit81a.
Alan Jay Smith, "Long Term File Migration: Development and Evaluation of Algorithms," *Communications of the ACM* **24**(8) pp. 521-532 (August 1981).

Smit81b.
Alan Jay Smith, "Input/Output Optimization and Disk Architecture: A Survey," *Performance Evaluation* **1**(2) pp. 104-117 (1981).

Smit85a.
Alan Jay Smith, "Disk Cache - Miss Ratio Analysis and Design Considerations," *ACM Transactions on Computer Systems* **3**(3) pp. 161-203 (August 1985).

Ston86a.
M. Stonebraker and L. Rowe, "The Design of POSTGRES," *Proc. 1986 ACM-SIGMOD Conference on Management of Data,* (May 1986).

Ston87a.
M. Stonebraker et. al., "The POSTGRES Rules System," *Proc. 1987 IEEE Data Engineering Conference,* (February 1987).

Swea86a.
Paul Sweazey and Alan Jay Smith, "A Class of Compatible Cache Coherency Protocols and Their Support by the IEEE Futurebus," *Proc. 13th Ann. Int. Symp. on Computer Architecture,* pp. 414-423. (June, 1986).

Tand85a.
Tandem Computer, "A Measure of Transaction Processing Power," Technical Report 85.1, Cupertino, CA (April, 1985).

Thom87a.
James Thompson, "Algorithms and Results for Caching in Processors and File Systems," Ph.D. Dissertation (expected September 1987).

Wile84a.
R. Wilensky, Y. Arens, and D. Chin, "Talking to UNIX in English: An Overview of U.C.," *Communications of the ACM* **27**(6)(1984).

Wino86a.
T. Winograd and F. Flores, *Understanding Computers and Cognition,* Ablex, Norwood, NJ (1986).

G. Results for Prior CER or II Award(s)

Berkeley has never had a CER or II Award from NSF. We applied once before, in 1981. As part of an inter-agency agreement, we were instead funded by DARPA, with the intention of developing infrastructure, and also increasing our involvement with the DARPA programs. That DARPA contract has led to subsequent DARPA sponsorship of the research projects of some of our faculty, as indicated elsewhere in this proposal. It has significantly increased the amount of experimental work we have been able to do, and has provided a research opportunity for many of our recent graduates and current students.

The DARPA program is to some extent mission-oriented; it does not encompass all of the research topics that we collectively wish to pursue, nor all of the faculty members participating in this proposal. Our previous CER-style award was a one-time occurrence and unconventional for DARPA. None of our current DARPA funding has that character.

A major share of the present DARPA funding is supporting the Aquarius project of Professor Despain, who is not a faculty investigator on this proposal. Of the present equipment we have listed, the Apollos, the NCR Tower, a VAX 785, and some Sun workstations and servers are used exclusively for his project. A major share of the VAX 8600 is also devoted to that project. DARPA is also providing partial support for the POSTGRES, SPUR, VorTeX, and Pan efforts described earlier. Much of that current support ends next year. Support for DASH and for continuation of our database and software environment research is pending.

The infrastructure support from DARPA demonstrated the enormous value to our program of a shared focus of the kind we are proposing here. Since 1981 we have had considerably more experimental research and significantly more collaboration than we had had before then. The technical interactions we have had as part of the preparation of this proposal suggest that the infrastructure support we are requesting from NSF would once again be of enormous benefit to the quality of our program.

H. Research Environment Study

H.1 Faculty List

omitted

H.2 Ph.D. Graduates

omitted

H.3 Metrics for the Research Environment

omitted