A Final Report
Grant No. 5555-25
July 5, 1993 - September 30, 1996

# HIGH PERFORMANCE DATABASES FOR SCIENTIFIC APPLICATIONS

Submitted to:

CESDIS
Code 930.5
Goddard Space Flight Center
Greenbelt, MD 20771

Attention: Ms. Nancy Campbell

Submitted by:

James C. French
Research Assistant Professor

Andrew S. Grimshaw
Associate Professor

SEAS Report No. UVA/528481/CS97/101
March 1997

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE

University of Virginia
Thornton Hall
Charlottesville, VA 22903

## UNIVERSITY OF VIRGINIA
### School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 600. There are 160 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Applied Mechanics, Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 17,000), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Phyics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

# High Performance Databases for Scientific Applications

## CESDIS Final Report
## July 1993 — September 1996

James C. French
Andrew S. Grimshaw
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

## 1. Highlights

During the first two years of funding we completed our work on high performance parallel I/O support for multidimensional range searches. This work is described more fully in [KARP94a, KARP94b, KARP94c, KARP94d, KARP94e]. In addition we began shifting our work in a new direction, file system support in high performance metasystems [GRIM94a, GRIM94b, GRIM95]. These two topics are discussed separately in greater detail below. To summarize, our main achievements in multidimensional range searching are as follows:

We developed a general approach for attacking the high-performance I/O problem.

We developed a parallel file object based on PLOP files designed to provide high-performance range queries in multiple dimensions.

We tested the performance of range retrievals on representative queries provided to us by the National Radio Astronomy Observatory.

During the final year of funding we directed our work on high performance parallel I/O to file system support in high performance metasystems [GRIM94a, GRIM94b, GRIM95]. This topic is discussed in greater detail below. Our main achievement over the period was the further development of the Campus Wide Virtual Computer (CWVC) deployed here at the University of Virginia. The campus-wide virtual computer is a prototype for the nationwide Legion system in that the computational resources at the University are operated by many different departments; sharing of resources is currently rare; resources are owned by the departments, and this equipment is used for "production" applications during the day.

Even though the CWVC is much smaller, and the components much closer together, than in the envisioned nationwide Legion, it still presents many of the same challenges. The processors are heterogeneous, the interconnection network is irregular with orders of magnitude differences in bandwidth and latency, and the machines are currently in use for on-site applications that must not be negatively impacted. Each department operates

essentially as an island of service, with its own NFS mount structure, and trusting only machines in the island.

The next section describes our work with multidimensional range searching. It is followed by an overview of *Legion*, our ongoing metasystems project and a discussion of our I/O work.

## 2. High Performance Parallel I/O Support for Multidimensional Range Searches

We have developed a general approach for attacking the high performance I/O problem, namely the Extensible File Systems (ELFS) approach based on work in [GRIM91]. This report describes an implementation following the ELFS approach for a specific class of retrieval patterns, multidimensional range searches. Multidimensional range searches appear in a wide range of applications, including many scientific applications. Such applications view a data set as an $n$-dimensional data space, where each dimension represents the values along a key field present in the data. The coordinates of each data record are its values for each of the $n$ dimensions. Using this view, subvolumes of the data space can be defined by specifying a range of values for each dimension. For example, a data set containing a set of time indexed two dimensional images can be viewed as a three-dimensional data space (*time*,$x$,$y$). Possible range searches for such a data set include retrieving a specified region of each image (a rectangle in ($x$,$y$)) for all *time* values, retrieving full images for a certain range of times, etc.

In the following sections we first present the current methods used for providing range search capabilities and then briefly describe the general ELFS approach and its benefits. This is followed by a discussion of an instance of this approach designed for high performance multidimensional range searches including details of the parallel structure of the implementation. We also describe the tests we executed using interferometry data sets from the National Radio Astronomy Observatory (NRAO). More details on these tests can be found in [KARP94a, KARP94c, KARP94d, KARP94e].

### 2.1. Current Methods

There are several approaches typically employed to provide range searching capabilities on a set of data, each with varying degrees of implementation effort and performance. Many implementations store the data sets as simple sorted sequential files and scan the file, filtering out unwanted data outside of the desired subvolume. This approach is easy to implement, but performs poorly, especially when small amounts of data are desired relative to the file size. An improvement to this scheme uses an indexed, sorted file, to reduce the number of accesses needed to the file, improving performance at a modest complexity cost. This scheme works well for a one-dimensional space (for the sorted, indexed key), but does not generally perform well for multidimensional accesses. Some implementations designed for parallel applications, improve the performance of a single file by either replicating the data sets or partitioning the data set into separate disjoint sets. Each of these approaches is designed to alleviate the contention for the single file resource among multiple concurrent processes, but does not improve upon the basic access methods for each distributed file.

Another common approach is to use a commercial database management system (DBMS) which allows for the specification of range queries. Relational DBMS are particularly popular where range searches are easily defined using the Structured Query

Language (SQL). This approach is easy from the implementation standpoint, but may not achieve acceptable performance. DBMS are built to support a wide range of possible access patterns and types of data and are not tuned to range searching in multiple dimensions. In addition, DBMS incur overhead for the guarantee of consistency within the database, which may not be an issue for many applications.

A less often used approach implements file structures specifically tailored for range searching such as PLOP files[KRIE88a, KRIE88b], grid files [NIEV84], k-d & k-d-b trees [BENT79, ROBI81], or quadtrees [SAME84]. These file structures offer performance advantages by attempting to preserve physical data locality in all of the dimensions of the data space and by providing efficient methods for finding particular regions of the data space. The drawback is that these file structures can be difficult to implement properly, especially in a distributed manner. Even when these structures are implemented, the implementations are often highly application-specific and not reusable, so the common practice is to build them virtually from scratch.

## 2.2. The ELFS Approach

The ELFS approach is to create file objects that satisfy four criteria:

(1) Match the file structure to the access patterns of the application and the type of the data. As the examples in the previous section point out, the organization and structure of the underlying file can greatly influence performance by reducing the number of accesses required. For distributed file structures, effective data placement can potentially improve performance by reducing latency. Therefore it is important to match file structures with their use.

(2) Use parallel and other advanced I/O techniques. In a file type-specific manner exploit parallelism to overlap application computation with I/O requests to reduce the effective latency of a request (i.e. the wait actually experienced after issuing a request). For distributed file structures, true parallel access can be used to better utilize the file system's bandwidth. Other I/O-related functions, such as data conversion and sorting, may be performed in parallel to speed the overall performance of using stored data. Prefetching and caching are two other well-known performance-oriented techniques that can be employed when applicable.

(3) Improve the I/O interface to application programs. There are two main reasons for improving the file interface. Most importantly from a performance standpoint, is to allow the user to convey useful information that can be exploited by the file object implementation. For example, knowing the stride of accesses in a matrix file can be exploited to effectively prefetch data, or knowing that the file will be used in a read only fashion can allow the implementation to avoid potentially costly consistency protocols. The second reason for improving the interface is to make file objects easier to use by application programmers, reducing their programming burden.

(4) Encapsulate the implementation details in file objects. This goal is aimed at increasing the maintainability and reusability of the file objects. By using the object-oriented paradigm for the file objects, application programmers can derive new file objects from existing base objects and can then extend them and tailor them without reimplementing much of the file object functionality.

A suite of extensible file objects can be developed using this methodology, each performing best for a particular class of data types and access patterns. Application designers can then choose the best file object for there purposes and extend or tailor the file definition as needed, hopefully requiring only a modest amount of effort. Our early design work in this area has been reported in [KARP94b].

### 2.3. Parallel File Objects for Multidimensional Range Searches

Using the ELFS approach we have created a parallel file object designed to provide high performance for range queries in multiple dimensions. Our implementation uses the PLOP file as the basic underlying file structure. Though other file structures could be used for multidimensional range searches, it is our opinion that none of these candidates is clearly superior to PLOP files, while PLOP files have a relatively straightforward implementation. For a more in depth analysis of the choice of file structure see [KARP94a]. A PLOP file views a data set as a multidimensional data space. The data space is partitioned by splitting each dimension into a series of ranges called slices. The intersection of a slice from each dimension defines one logical data bucket. Data points are stored in the bucket that has corresponding values in each dimension. Therefore, within a bucket, the data points exhibit spatial locality in all dimensions. A tree structure for each dimension tracks the physical location of each bucket within the file, so that each bucket can be accessed very efficiently. This structure allows retrievals to eliminate parts of the file that do not correspond to values within the range search based on all dimensions, while quickly accessing those parts that may contain valid data.

We first implemented a sequential version of the PLOP file based file object. Though unable to take advantage of parallel techniques, this version exploits the structure of PLOP files to achieve efficient accesses. In addition to the obvious benefits of using the tailor-made structure of the PLOP file, a subtle performance improving enhancement was implemented for sorting by a key that is one of the dimensions. Because the data points contained in each slice along a dimension are disjoint, the data in each slice along the sort key can be sorted separately, and with each slice returned in order. By sorting in smaller batches, the complexity of the sort is reduced from $O(n \log n)$ to $O(n/p \log n/p)$ in the ideal case ($p$ = number of slices spanned by the request).

The parallel version is being implemented using Mentat [GRIM93c], an object-oriented parallel processing system. The design of the parallel implementation includes several significant changes from the sequential version. First, PLOP files have been modified to accommodate distributed pieces. These pieces can be created and distributed in three patterns: segmented (or partitioned) along a dimension, striped along a dimension, or blocked by some set of dimensions (e.g. using two dimensions each piece would be a rectangular region). The distributed PLOP file allows not only parallel access to the data, but also allows an application program to map processes to nodes near the data they will require.

Second, parallel I/O workers have been added to access each distributed file piece and a manager has been added to coordinate their activities. The workers asynchronously handle all requests for data at their piece, including I/O device access, data conversion and, if possible, data sorting. Our initial design has only a single manager process for all worker processes and clearly does not scale well for increased numbers of application processes requiring data. We already plan to replace this design with a

scheme that will scale for increases in the number of application processes by enabling the manager to replicate itself and assign different managers to different application processes.

Third, the interface has been improved. A major improvement to the interface to decouple the definition of a query request from the retrieval of the data. The idea is to allow the user to specify a query ahead of the time the data will be used whenever possible, and to submit the query to be performed. The file object can asynchronously begin buffering the request while the application continues to do useful work. When the application actually wants the retrieved values, a call is made to ask for the data.

### 2.4. Performance Tests

To test our implementation, we have converted two interferometry data sets from NRAO's Very Large Array (VLA) radio antenna installation. The first file, a line spectrum file, is ~50 megabytes and 126,092 records, while the second file, a continuum spectrum file, is ~270 megabytes and contains 8,440,092 records. Initial results for the sequential version have been very encouraging. The converted PLOP files utilize space fairly efficiently, 79% and 66% for the line and continuum files respectively (efficiency is calculated by comparing actual storage used for records versus the total storage allocated, including overhead and fragmentation).

To test the performance of range retrievals, a set of twelve representative queries for NRAO's applications has been developed. Both files have been tested using these queries for the sequential version, with impressive results. The parallel version will be tested with the same suite of queries for various file distribution patterns and numbers of file pieces. These results and a comparison of results across the various parameters can be found in [KARP94a, KARP94c, KARP94d, KARP94e].

## 3. Legion: File and Data Access

Our work in file systems for high-performance heterogeneous metasystems is being done within the *Legion* project. We describe that next.

### 3.1. Legion

Legion will consist of workstations, vector supercomputers, and parallel supercomputers connected by local area networks, enterprise-wide networks, and the NII. The total computation power of such an assembly of machines is enormous, approaching a petaflop; this massive potential is, as yet, unrealized. These machines are currently tied together in a loose confederation of shared communication resources used primarily to support electronic mail, file transfer, and remote login. However, these resources could be used to provide far more than just communication services; they have the potential to provide a single, seamless, computational environment in which processor cycles, communication, and data are all shared, and in which the workstation across the continent is no less a resource than the one down the hall.

A Legion user has the illusion of a single, very powerful computer on her desk, which is used to invoke an application on a data set. It is Legion's responsibility to transparently schedule application components on processors, manage data transfer and coercion, and provide communication and synchronization, while trying to minimize execution time via parallel execution of the application components. System boundaries will

be invisible, as will the location of data and the existence of faults.

The potential benefits of Legion are enormous: (1) more effective collaboration by putting coworkers in the same virtual workplace; (2) higher application performance due to parallel execution and exploitation of off-site resources; (3) improved access to data and computational resources; (4) improved researcher and user productivity resulting from more effective collaboration and better application performance; (5) increased resource utilization; and (6) a considerably simpler programming environment for the applications programmers. Indeed, it seems probable to us that the NII can reach its full potential only with a Legion-like infrastructure.

Before the Legion vision can be realized, several technical challenges must be overcome. These are software problems; the hardware challenges are being addressed and are the enabling technologies that provide the opportunity. The software challenges revolve around eight central themes: achieving high performance via parallelism, managing and exploiting component heterogeneity, resource management, file and data access, fault-tolerance, ease-of-use and user interfaces, protection and authentication, and exploitation of high-performance communications protocols.

### 3.2. Objectives

From our Legion vision we have distilled six primary design objectives:

*Easy-to-use, seamless computational environment.*

Legion must mask the complexity of the hardware environment and the complexity of communication and synchronization of parallel processing. Machine boundaries should be invisible to users. Legion will provide both user and programmer with a uniform interface to service. As much as possible, compilers, acting in concert with run-time facilities, must manage the environment for the user.

*High performance via parallelism.*

Legion must support easy-to-use parallel processing with large degrees of parallelism. This includes task and data parallelism and their combinations. Because of the nature of the interconnection network, Legion must be latency tolerant. Further, Legion must be capable of managing hundreds or thousands of processors. This implies that the underlying computation model and programming paradigms must be scalable.

*Single, persistent namespace.*

One of the most significant obstacles to wide-area parallel processing is the lack of a single name space for file and data access. The existing multitude of disjoint name spaces makes writing applications that span sites extremely difficult. Therefore, Legion must provide a single name space for persistent objects (files).

*Security for users and resource owners.*

Because we cannot replace existing host operating systems, we cannot significantly strengthen existing operating system protection and security mechanisms. However, we must ensure that existing mechanisms are not weakened by Legion.

*Manage and exploit resource heterogeneity.*

Clearly Legion must accommodate heterogeneity, i.e., it must support interoperability between heterogeneous components. In addition, Legion will be able to exploit diverse hardware and data resources, executing subtasks of large applications on

different heterogeneous processors, and using heterogeneous data sources. Some architectures are better than others at executing particular kinds of code, e.g., vectorizable codes. These affinities, and the costs of exploiting them, must be factored into scheduling decisions and policies.

*Minimal impact on resource owner's local computation.*

The noticeable impact of Legion on local resources must be small, particularly with regard to interactive sessions. If users notice a significant performance penalty when their site is attached to Legion, they will withdraw; an observed penalty must be more than offset by the benefits of Legionnaire status.

We have the additional objective of demonstrating the effectiveness of wide-area heterogeneous computing on serious applications drawn from a variety of application domains, including "grand challenges," such as global climate modeling and the human genome project, and from economically significant problem areas such as electrical engineering and medicine.

### 3.3. Approach

The principles of the object-oriented paradigm are the foundation for the construction of Legion; our goal will be exploitation of the paradigm's encapsulation and inheritance properties. Use of an object-oriented foundation will render a variety of benefits, including software reuse, fault containment, and reduction in complexity. The need for the paradigm is particularly acute in a system as large and complex as Legion. Other investigators have proposed constructing application libraries and applications for wide-area parallel processing using only low-level message passing services such as those provided by PVM [SUND90] and P4 [BOYL87]. Use of such tools requires the programmer to address the full complexity of the environment; the difficult problems of managing faults, scheduling, load balancing, etc., are likely to overwhelm all but the best programmers.

Objects, written in either an object-oriented language or other languages such as HPF Fortran, will encapsulate their implementation, data structures, and parallelism, and will interact with other objects via well-defined interfaces. In addition, they may also have inherited timing, fault, persistence, priority, and protection characteristics. Naturally these may be overloaded to provide different functionality on a class-by-class basis.

Our approach to constructing Legion is evolutionary rather than revolutionary. We have begun by first constructing a Legion testbed by extending Mentat, an existing object-oriented parallel processing system [GRIM93d]. Mentat attacks the problem of providing easy-to-use high performance parallelism to users. Mentat has been used to implement several real-world applications on hardware platforms spanning the bandwidth/latency space in a heterogeneous environment [GRIM93a, GRIM93b, GRIM93e]. Mentat's object-oriented structure, and its ability to achieve high-performance on platforms with very different communications characteristics are the key factors in our choice of Mentat as our implementation vehicle. The testbed provides us with an ideal platform to rapidly prototype ideas, forcing the details and hidden assumptions to be carefully examined, and exposing flaws in the ideas or in the system components.

### 3.4. File/data Access

File and data access is one of the most crucial issues for Legion, particularly with respect to providing a seamless environment. Today, distributed file systems such as NFS, Andrew, and Locus are commonplace in local area networks. The unified level of service and the naming scheme that they present to their users make them one of the most successful components of contemporary distributed systems. In Legion we intend to provide the same level of naming and access transparency provided in local area networks. This cannot be accomplished either by directly extending current systems onto a national scale, or by imposing a single file system for both local and Legion access. Instead we propose to adopt a federated file system approach. The Legion file system will provide naming, access, location, fault, and replication transparency. It will permit users (or library writers) to extend the basic services provided by the file system in a clean and consistent fashion via class derivation and file-object instantiation and manipulation. The extensions that we intend to design and implement ourselves include application-specific file objects designed to improve application performance by reducing observed I/O latency.

Issues such as naming, location transparency, fault transparency, replication transparency, and migration have been addressed both in the literature [LEVY90] and in several existing operational systems. Rather than duplicate those efforts we will build on them and extend them into a larger context. The difficulty that arises when borrowing from an existing system is that most of the systems do not have a scalable system architecture or flexible semantics, or they require the imposition of a unified file system model, contrary to our federated file system goal. Therefore we will borrow ideas but not implementations, looking more to combine the work of others with our own.

Although we will continue to refer to the Legion "file system," we intend to create a persistent object space as has been proposed for distributed object management systems [NICO93]. There are several other efforts in the distributed object literature with which we share many goals, e.g., SHORE [CARE94] and CORBA [MANO92, NICO93]. Legion is distinguished from these efforts by the emphasis we place on performance - Legion expects to provide a high performance computing environment and this goal is paramount. To this end we will focus more on file system support than database support.

The model that we will employ is simple and driven by the observation that the traditional distinction between files and other objects is somewhat of an anachronism. Files really are objects - they happen to live on disk, as a consequence they are slower to access, and they persist when the computer is turned off. We define a file-object as a typed object with an interface. The interface can also define object properties such as its persistence, fault, synchronization, and performance characteristics. Thus, not all files need be the same, eliminating, for example, the need to provide Unix synchronization semantics for all files even when many applications simply do not require those semantics. Instead, the right semantics along many dimensions can be selected on a file-by-file basis, and potentially changed at run-time.

### 3.5. Agenda

Our agenda consists of three stages: (1) the construction of a campus-wide virtual computer at the University of Virginia, (2) packaging the campus-wide system for preliminary experimentation and use by Legionnaires, and (3) expansion to a nationwide

demonstration system. Each of these three stages will build upon the previous.

Before any major project is undertaken, one must ask how to measure success. In parallel processing, success is measured by application performance (speedup, elapsed time) and the flexibility and ease of use of the tool. Other important metrics include acceptance by the user community, fault-tolerance, cost per used MIP/FLOP, and whether tasks can be performed that were not possible before (e.g., run an application in Virginia on data that resides at NASA-JPL, or collect and use data in real time from sensors in orbit, but have that data look like any other "file").

Application performance will be measured for a variety of real-world applications, as well as selected kernel codes and parallel processing benchmarks. The applications will be drawn from a diverse set of disciplines: biology, physics, electrical engineering, chemistry, economics, radio astronomy, and command and control. The applications will possess different granularity characteristics, as well as different latency tolerances. It is not our intent, however, to show that all applications will be capable of exploiting the nationwide resources of Legion. Some applications, those with inherently small granularity or that are latency intolerant, will remain best suited to local operation, e.g., on a single processor or on a single tightly-coupled parallel processor.

### 3.5.1. Construction of a Campus-Wide Virtual Computer (CWVC)

The campus-wide virtual computer is a direct extension of Mentat to a larger scale, and is a prototype for the nationwide system in that the computational resources at the University are operated by many different departments; sharing of resources is currently rare; resources are owned by the departments, and this equipment is used for "production" applications during the day.

Even though the CWVC is much smaller, and the components much closer together, than in the envisioned nationwide Legion, it still presents many of the same challenges. The processors are heterogeneous, the interconnection network is irregular with orders of magnitude differences in bandwidth and latency, and the machines are currently in use for on-site applications that must not be negatively impacted. Each department operates essentially as an island of service, with its own NFS mount structure, and trusting only machines in the island.

The CWVC is both a prototype and a demonstration project. The objectives are to: demonstrate the usefulness of network-based, heterogeneous, parallel processing to university computational science problems; provide a shared high-performance resource for university researchers; provide a given level of service (as measured by turn-around time) at reduced cost; and act as a testbed for the nationwide Legion.

The prototype consists of over sixty workstations and is now operational. In [GRIM95] we present the performance of two production applications that we have used to test the efficacy of our approach: complib, a biochemistry application that compares DNA and protein sequences, and ATPG, an electrical engineering application that generates test patterns for VLSI circuits. The performance results are encouraging.

## 4. I/O Status

We conclude this report with a summary of the I/O status in the Legion project. This section presents an overview of the current research initiatives related to file systems and I/O in the Legion Metasystem [GRIM94b] project. Given the intended nation-

wide to world-wide scope of Legion, the system poses many new challenges in the area of scalable I/O applications, but at the same time holds the promise of exciting new tools for wide-area collaboration and large-scale information management and retrieval.

Legion is a distributed, object-oriented, virtual-machine based metasystem intended to present a single, seamless computational environment to users and application developers. A central part of the Legion environment is its single, persistent namespace. Current research in the area of persistent objects in Legion is focused on three of different levels. These are:

(1) Design and implementation of the basic system functionality to support persistent objects.

(2) Design and implementation of basic, useful, user-level persistent object classes.

(3) Development of applications to avail of the unique facilities supplied by Legion persistent object classes.

### 4.1. System Support for Persistent Objects

Current research at the low-level Legion system implementation level is addressing some of the basic problems in supporting distributed persistent objects. These include:

*Naming and Binding*

Central to the ability to support persistent objects is the need for a well defined concept of the persistent object name-space. Legion objects have associated with them system-wide unique identifiers, LUIDs. A distributed scheme utilizing Binding Agent objects is employed to bind LUIDs to object addresses.

*Persistent Object Creation and Scheduling*

Another active research area is related to the placement and instantiation of persistent objects in order to best utilize available resources.

*Object States*

Legion objects can be in one of two basic states \m active or inactive. Active objects have an associated thread of control and address space, while inactive objects are dormant and have all needed state saved to stable store. Currently, the basic mechanisms by which objects are moved between active and inactive status are being developed. All object classes will support "Save" and "Restore" asynchronous member functions which will be invoked by system scheduling and instantiation objects. Class authors will be responsible for utilizing Legion supplied mechanisms to save and restore user level object state, while system internal object state will be saved and restored automatically.

*Advanced Features*

Features such as persistent object migration and object replication are also important goals of the Legion persistent object model. While such features will be under the control of persistent object class authors, the needed system support to elegantly utilize these mechanisms are currently being investigated and developed.

### 4.2. Basic User Level Persistent Object Classes

While the central core of the Legion system will provide the ability to develop persistent object classes, the system will be of little use unless an existing base of useful,

user-level persistent object classes is also provided. Among the most important of these are:

*File Objects* - The basic staple of information-based applications is the file. While the Legion system will not prescribe any limited set of file objects, it will provide a useful set of basic file classes. The most basic among these is the currently implemented "Byte Vector" object class - an unstructured vector of bytes supporting a set of member functions similar in functionality to Unix standard library file manipulation system calls. Along with the byte vector object class, utility programs such as "legion cat" have been developed to demonstrate the basic functionality of location independent, distributed files. Other more complex application utilizing basic Legion "Byte Vector" objects are described below.

*Context Objects* - While the Legion LUID naming scheme addresses the basic issues of object naming and binding, it does not directly address the basic needs of name-space "navigation" mechanisms - the ability to explore the name space, create logical links between objects, and map human-user comprehensible string names to system readable LUIDS. This role will be played by "Context" object classes. These objects, at the highest level, will provide a mapping between user-level string names and system-level LUIDs. They will also provide mechanisms to create links between contexts, building a graph of object directories. Well constructed context objects will provide the basis for structuring the Legion namespace at the level of user comprehensible meaning.

### 4.3. Applications Utilizing Persistent Objects

In order to demonstrate the utility of Legion persistent objects, as well as to drive the further development and refinement of the Legion object model, a number of applications are currently being updated to utilize Legion file facilities. Some of these include:

*Text Editing / Word Processing.* The ability to collaborate on documents conveniently in a wide-area distributed environment will be one of the basic benefits of Legion. Of the basic applications to support in this domain, text editing and word processing are among the most obvious candidates. Versions of the standard unix vi and emacs word processors have been updated to utilize Legion files. These have been demonstrated in truly wide-area (cross country) environments. Currently, an add-on module for the FrameMaker word processing system is being developed to allow Frame users to transparently manipulate files in Legion space.

*Distributed Software Development.* Another potential area being investigated is distributed software project control. The ability to effectively collaborate on software development projects is a natural application of the Legion persistent object facility. The problem of developing a source code control / configuration management system utilizing Legion file objects is currently being investigated.

*Distributed Simulation.* The use of distributed simulations in military and commercial applications is wide spread and growing consistently. The application of the Legion persistent object space to distributed simulation is an area of active research in the group. A recent paper examining this potential application of the system is [FERR95].

**References:**


[BENT79] J. L. Bentley and J. H. Friedman, "Data Structures for Range Searching", *ACM Computing Surveys 11*, 4 (Dec. 1979), 397-409.

[BOYL87] J. Boyle et al., *Portable Programs for Parallel Processors*, Holt, Rinehart and Winston, New York, 1987.

[CARE94] M. J. Carey et al., "Shoring Up Persistent Applications", *SIGMOD 1994*, 1994, 383-394.

[FERR95] A. J. Ferrari, "Distributed Interactive Simulation in the Legion System", *ELECSIM 1995, Electronic Conference, 1995. <ftp://ftp.cs.virginia.edu/pub/ajf2j/legion_dis.ps>.*

[GRIM91] A. S. Grimshaw and E. C. Loyot, Jr., "ELFS: Object-Oriented Extensible File Systems", Tech. Rep. CS-91-14, Dept. of Computer Science, University of Virginia, July 1991.

[GRIM93a] A. S. Grimshaw, W. T. Strayer and P. Narayan, "Dynamic Object-Oriented Parallel Processing", *IEEE Parallel & Distributed Technology: Systems & Applications*, May 1993, 33-47.

[GRIM93b] A. S. Grimshaw, E. A. West and W. R. Pearson, "No Pain and Gain! - Experiences with Mentat on Biological Application", *Concurrency: Practice & Experience 5*, 4 (June 1993), 309-328.

[GRIM93c] A. S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat", *IEEE Computer*, May 1993, 39-51.

[GRIM93d] A. S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat", *IEEE Computer*, May 1993, 39-51 .

[GRIM93e] A. S. Grimshaw, J. B. Weissman and W. T. Strayer, "Portable Run-Time Support for Dynamic Object-Oriented Parallel Processing", *submitted to ACM Transactions on Computer Systems*, July 1993.

[GRIM94a] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver and P. F. R. Jr., "A Synopsis of the Legion Project", Tech. Rep. CS-94-20, Dept. of Computer Science, University of Virginia, Charlottesville, VA, June 1994.

[GRIM94b] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver and P. F. R. Jr., "Legion: The Next Logical Step Toward a Nationwide Virtual Computer", Tech. Rep. CS-94-21, Dept. of Computer Science, University of Virginia, Charlottesville, VA, June 1994.

[GRIM95] A. S. Grimshaw, A. Nguyen-Tuong and W. A. Wulf, "Campus-Wide Computing: Early Results Using Legion at The University of Virginia", Tech. Rep. CS-95-19, Dept. of Computer Science, University of Virginia, Charlottesville, VA, March 1995.

[KARP94a] J. F. Karpovich, A. S. Grimshaw and J. C. French, "Breaking the I/O Bottleneck at the National Radio Astronomy Observatory (NRAO)", Tech. Rep. CS-94-37 , Dept. of Computer Science, University of Virginia, Charlottesville, VA, August 1994.

[KARP94b] J. F. Karpovich, A. S. Grimshaw and J. C. French, "Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O", *Proc. OOPSLA '94: Object-Oriented Programming Systems and Languages*, 1994, 191-204.

[KARP94c] J. F. Karpovich, A. S. Grimshaw and J. C. French, "Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O", Tech. Rep. CS-94-28 , Dept. of Computer Science, University of Virginia, Charlottesville, VA, July 1994.

[KARP94d] J. F. Karpovich, J. C. French and A. S. Grimshaw, "High Performance Access to Radio Astronomy Data: A Case Study", Tech. Rep. CS-94-25 , Dept. of Computer Science, University of Virginia, Charlottesville, VA, July 1994.

[KARP94e] J. F. Karpovich, J. C. French and A. S. Grimshaw, "High Performance Access to Radio Astronomy Data: A Case Study", *Proc. 7th Inter. Conference on Scientific and Statistical Database Management*, Oct. 1994, 240-249.

[KRIE88a] H. Kriegel and B. Seeger, "Techniques for Design and Implementation of Efficient Spatial Access Methods", *Proc. of the 14th VLDB*, 1988, 360-370.

[KRIE88b] H. Kriegel and B. Seeger, "PLOP-Hashing: A Grid File without a Directory", *Proc. of the Fourth Inter. Conf. on Data Engineering*, Feb. 1988, 369-376.

[LEVY90] E. Levy and A. Silberschatz, "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys 22*, 4 (December 1990), , 321-374.

[MANO92] F. Manola, S. Heiler, D. Georgakopoulos, M. Hornick and M. Brodie, "Distributed Object Management", *International Journal of Intelligent and Cooperative Information Systems 1*, 1 (June 1992).

[NICO93] J. R. Nicol, C. T. Wilkes and F. A. Manola, "Object-Orientation in Heterogeneous Distributed Systems", *IEEE Computer 26*, 6 (June 1993), 57-67.

[NIEV84] J. Nievergelt and H. Hinterberger, "The Grid File: An Adaptable, Symmetric Multikey File Structure", *ACM Transactions on Database Systems 9*, 1 (Mar. 1984), 38-71.

[ROBI81] J. T. Robinson, "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes", *Proc. of the Annual Meeting of the ACM Special Interest Group on Management of Data*, 1981, 10-18.

[SAME84] H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys 16*, 2 (June 1984), 187-260.

[SUND90] V. S. Sunderam, "PVM: A framework for parallel distributed computing", *Concurrency: Practice and Experience 2*, 4 (December 1990), 315-339.

DISTRIBUTION LIST

1 - 3 CESDIS
Code 930.5
Goddard Space Flight Center
Greenbelt, MD 20771

Attention: Ms. Nancy Campbell

4 - 5 J. C. French

6 A. S. Grimshaw

7 - 8 M. Rodeffer, Clark Hall

* SEAS Postaward Research Administration

9 SEAS Preaward Research Administration

*Cover Letter

JO#7525:ph

NASA

# Report Documentation Page

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| | | |

**4. Title and Subtitle**
High Performance Databases For
Scientific Applications

**5. Report Date**

**6. Performing Organization Code**

**7. Author(s)**
James French

**8. Performing Organization Report No.**

**10. Work Unit No.**

**9. Performing Organization Name and Address**
University of Virginia
P.O. Box 9003
Charlottesville, VA 22906

**11. Contract or Grant No.** NAS5-32337
**USRA subcontract No.** 5555-25

**12. Sponsoring Agency Name and Address**
National Aeronautics and Space Administration
Washington, DC 20546-0001
NASA Goddard Space Flight Center
Greenbelt, MD 20771

**13. Type of Report and Period Covered** Final
July 1993 - September 1996

**14. Sponsoring Agency Code**

**15. Supplementary Notes**
This work was performed under a subcontract issued by
Universities Space Research Association
10227 Wincopin Circle, Suite 212
Columbia, MD 21044 Task 18

**16. Abstract**

The goal for this task is to develop an Extensible File System (ELFS). ELFS attacks the problem of the following:

1. Providing high bandwidth performance architectures
2. Reducing the cognitive burden faced by applications programmers when they attempt to optimize their I/O operations to fit existing file system models; and
3. Seamlessly managing the proliferation of data formats and architectural differences

The approach for ELFS solution consists of language and run-time system support that permits the specification o a hierarchy of file classes.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| extensible file system | Unclassified--Unlimited |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 1 | |

NASA Form 1626 Oct 86