Critical Issues for Operating Systems in Supercomputing Environments

1. Preface

This document resulted from a workshop held in July, 1990, jointly organized by the Supercomputing Research Center and Lawrence Livermore National Laboratory.

The workshop brought together users and designers of supercomputers from the commercial, academic, and research laboratory communities†. This report is intended to influence research, development, and standardization issues related to operating systems in supercomputing environments.

Opinions on complex issues are seldom unanimous. This report represents discussions in the framework of six critical areas. Subgroups were formed to draft the individual sections. Taxonomies are never complete and almost always overlap. Accordingly, some topics recur and different emphases are found.

Comments on this report are sought. Address correspondence to:

John Riganati riganati@super.org
Daniel Kopetzky djk@super.org
Ron Minnich rminnich@super.org

Supercomputing Research Center
17100 Science Drive
Bowie, MD 20715–4300

Dieter Fuss fuss@lll–crg.llnl.gov

Lawrence Livermore National Laboratory
P. O. Box 808, L669
Livermore, CA 94550

The workshop which lead to this document was influenced by a birds-of-a-feather session held at Supercomputing '89 and summarized in Appendix B.

2. Executive Summary

Six issues were selected as the most important to the supercomputing community. While there are certainly more than six issues some issues were dropped because they are not limited to the supercomputing community; others were dropped because they were judged to be less critical.

The 6 issues are:

I. *Distributed Supercomputing*: Distributed supercomputing involves the construction of applications which span machine boundaries. We differentiate it from parallel supercomputing in that the set of machines used is interconnected by a network, not a memory bus or backplane. The machines need not all be supercomputers. Important topics include distributed process invocation and migration; distributed file systems that support migration of files on demand; recovery strategies in the event of failure; and programming methodologies and operating systems support.

II. *Archival Storage*: There are terabytes of data on-line in formats that were created by and for one type of machine and which need to be processed on other machines. The data formats are not always compatible between the machine which stored the data and machines which need to process the data. The data must be moved at Gigabit/second rates, and conversion must be addressed efficiently.

† Institutions represented include: CCC, CRI, SSI, Unix International, OSF, Sun Microsystems, Digital Equipment Corp, Convex, Chorus Systemes, JPL, LLNL, LANL, SRC, AFWL, SDSC, NCAR, Sandia National Labs, U. Pennsynvania, U. Illinois, CMU, UC Berkeley.

III. *High Speed Networks*: Gigabit per second networks are emerging which have a higher bandwidth than the main memory bus in some of the smaller computers used in a supercomputing environment. Existing network models will probably not be usable.

IV. *Architectural Issues*: Supercomputers differ from other computers in architecturally significant ways, some of which impact operating systems. Recent operating systems research has not taken these differences into account, and thus much of this research is not applicable to supercomputers. Are changes to supercomputer architectures necessary? Can these changes be made without sacrifice of high performance?

V. *Debugging for Correctness and Performance*: Parallel and distributed applications are difficult to debug. Determining what the application is doing, whether it is returning a correct answer, and whether it will continue to return correct answers are problems that have not been solved. Determining how a highly optimized version of the application is performing and how to improve performance further is an even more difficult problem.

VI. *Models of Parallelism*: There are many models of parallelism extant. Progress in the field would be greatly enhanced if the best few were to emerge in the near term.

For each area, the discussions address five factors:

A. *The current environment*. The environment available to a supercomputer user as of July, 1990. This generally consists of software purchased from a vendor (e.g., Unicos, Cray Fortran) plus other software (e.g., NQS).

B. *The desired environment*. The environment we would like to have running on supercomputers in the next five years.

C. *The current status.* What efforts are underway to allow us to reach B. This includes work underway on supercomputers per se, and also work that is being done on non-supercomputers that may be applicable.

D. *What problems must be solved to get from* C *to* B. Depending on the individual problem, we may find that addressing that problem may require that new research efforts need to be undertaken; or that resources be furnished to appropriate organizations; or that some organizations use their existing resources to better effect.

E. *A proposed (perhaps partial) solution.* A discussion of what is needed to converge on a solution.

Attendees, listed in Appendix A, span the vendor, user, and research communities.

There was general agreement that supercomputer systems need to change to fit the distributed computing environments now being built. There are differing ideas on how to get there, but there was general agreement on the desirability of seeing some results by 1995. Also, Posix compatibility, i.e., compatibility at the operating systems interface level, is acceptable to all concerned, even those who no longer run Unix (which included *all* of the operating systems research community). In fact, while the operating systems research community does little work with the Unix operating system any more, all of the research operating systems in wide use provide a Unix-compatible operating systems interface.

Thus, we all agree that evolutionary change is required. Evolutionary change ensures that no group of users will be left out in the cold by changes in the software environment, which in turn makes it acceptable to contemplate more fundamental changes such as replacement of Unix with some other system. While the vendor representatives made it clear that they could not lead such an effort, they also made it clear that they would not hinder it, and would in fact welcome it.

It was also agreed that the gigabit/second networks had to be taken into account in any research that occurs. There is no question that such networks will require changes in the way operating systems networking support is structured, but there is no certainty about what will need to be done.

The data migration problem actually concerns a number of problem areas. One is making effective use of the high-bandwidth links from mass storage systems to computers. Another is handling data heterogeneity. Both of these problems also appear in the gigabit networking environment, so there is an area of overlap here.

The data heterogeneity problem has been solved in a low-performance regime, e.g., such systems as Sun's External Data Representation (XDR). No acceptable solution yet exists for the 100 Mbyte/second and higher regime.

Participants also expressed their need for improved language and runtime library support for building robust, secure distributed computations. No such systems currently exist.

Distributed file systems are a major concern. For example, all agreed that NFS is not a foundation of a distributed file system. File systems must support better accounting, migration to long-term storage, and migration on demand.

3. Outstanding Problems

Two problems stand out as potential "show stoppers" for the supercomputing arena:

1. Unix is now the standard supercomputer operating system. As is often the case in fast moving high technology areas, standardization is achieved well into the maturity cycle of an idea. Thus, Unix is also widely perceived by researchers to be an impediment to progress. For example, operating systems researchers generally do not use Unix. The supercomputer vendors cannot undertake research on operating systems.

   The academic Operating Systems research communities do not have access to supercomputers in a way that allows operating systems research to be effectively conducted, and so have no possibility of considering supercomputers as part of their set of target architectures. The national labs such as Livermore and Los Alamos are using Unix on their supercomputers; but their proprietary OS research activities are not anticipated to result in operational software. There is a need for operating systems research for supercomputing but each environment (industry, university, and national laboratory) is lacking in some significant way.

2. The most popular supercomputers extant do not support virtual memory. Much of the research in the computing community that is directed toward the six issues is conducted on machines which support virtual memory. Practical application of this research is difficult. We are not sure if there is a way to resolve this problem.

I. Distributed Supercomputing

I.A The Distributed Supercomputer Environment

The supercomputer environment is a complex heterogeneous one. It provides the framework for sharing a variety of capital intensive resources, including the supercomputers themselves. Within the environment can be found all or some of the following types of resources:

- supercomputers (vector, and massively parallel)
- superscalar micros/multiprocessors
- special functional units (network disk, graphics processors, frame buffers, processors dedicated to a single algorithm, etc.)
- large servers (mass storage, I/O)
- workstations
- X–terminals
- terminal servers
- authentication servers
- hierarchies of networks (local, wide-area, gigabit, FDDI, Ethernet, etc.).

There are three strong forces driving the evolution of this distributed, resource sharing supercomputing environment:

- Technology
  - Distribution is technically and economically compelling due to low cost and high performance microelectronics and computer networks. The requirement is to provide the system software framework to realize the available potential.
- User requirements
  - There is a growing need to improve the productivity of scientific engineering, and administrative personnel. In general, this requires highly interactive human engineered tools. These tools are being built around man/machine interfaces involving character-at-a-time and graphical interaction (pointing input, menus, pictures, icons, etc.). Supercomputers cannot be efficiently used in this highly interactive mode and therefore the user interface mechanisms will migrate to workstations. Many utilities and interactive modules of application systems now running on the supercomputers will migrate to machines local to the users for similar reasons. Once commands or desired operations are completely specified, a remote invocation (remote procedure call) will be made to the program on the supercomputer for action.
  - Another factor leading toward improved user productivity is relieving the users of various burdens in managing their resources, locating them, moving them to appropriate locations, managing systems, setting up consistent environments across machines. Similarly, programmer productivity should be improved if

most of these resource management chores and remote access functions can be made invisible to them.

– There seems to be a natural tendency for individuals and organizations to want as much local control of resources as possible. Thus, given the decreasing cost and increasing capability of micro and mini computers, we can expect more application functionality to migrate to locally controlled systems.

– Applications are becoming more modular, carefully splitting out problem generation, output and graphics generation, database management, computational physics algorithms, etc. Many of these modules will run more cost effectively on systems other than supercomputers both from the computer and the human productivity cost points of view, or may be most cost effectively run on different supercomputer architectures. In fact, where a given module will run most economically will change over time. Therefore, configuration flexibility is needed.

- System requirements

  – Supercomputers do not operate stand alone, but are part of a large heterogeneous resource sharing network that is in constant evolution over decades. In fact the highest performance network may be a supercomputer backplane to which are attached vector, massively parallel, and I/O processors. Therefore, an important requirement of economical supercomputer environment operation is to minimize the costs in time and people necessary to develop, integrate, and maintain this environment over decades.

  – There will be large, high performance, or specialized storage, I/O, and processing resources that are most economically shared not only by the supercomputers but by remote systems. Sharing of resources can improve load balancing and their economic utilization. Earlier, we listed the many types of resources likely to be components in a supercomputer environment. Particular systems need only implement those functions (clients or servers) that they are best suited to provide. For example, there will be dedicated systems for file, authentication, name, or process service, etc. In fact, one can envision the day when the only O.S. facilities on a supercomputer might be interprocess communication (including network access), a process server, a time service, and probably a high performance file server. A framework for their sharing and access is required.

  – There is an increasing rate of change and increase in capabilities provided by the computer industry. New capabilities need to be easily and transparently integrated into the supercomputer environment and easily accessible and usable by applications.

  – Mechanisms are required to assure high availability and reliability and to simplify their maintainability. This implies increased modularity and redundancy.

  – The supercomputers are accessed from both ASCII and X–window terminals, and need to be accessed from and work cooperatively with workstations.

I.B The Desired Behavior

The goal is to make the network of heterogeneous systems that constitutes the supercomputer environment appear to terminal users, programmers (application and system) and programs as a single system with seamless transparent component system boundaries, except when explicit control of resource location is desired. That is, the user cannot tell if a resource is on the local or a remote system. This requires a location independent high–level interface that provides an application with uniform resource naming, abstract object representation and access, resource management, error handling, synchronization, debugging, accounting, and protection. All the major design areas just listed must be handled in centralized or stand-alone systems, but unfortunately the mechanisms used may not be naturally distributed. Since most existing widely used operating systems were designed in the late 60's or early 70's, before networking and distributed systems were issues, there are problems in their architecture and implementation. The types of new issues introduced by distribution that the system mechanisms must deal with include:

- heterogeneity of data encodings and internal object representation,
- arbitrary communication delays,
- new types of errors and failures resulting from network and single node problems,
- different administrative domains and corresponding heterogeneous resource allocation, accounting, security and other policies.

The general requirements above can be rephrased as the need to create the illusion of a single logical multi-processing system supporting several dimensions of transparency:

- Abstraction – the same abstract objects and interfaces must exist wherever supported. Hiding of heterogeneity of data and object encodings and representations is required.
- Syntactic – access to local and remote resources use the same operations and parameters.
- Semantic – behavior of an operation is independent of operation execution and operand location and type of failures/errors possible.
- Name – every object has globally unique machine oriented and possibly human-oriented identifiers or names which are the same independent of resource and accessor locations.
- Location – location cannot be inferred by the client from the high level name.
- Performance – performance should be as independent of location as possible. It is also desirable that response and other performance factors be predictable from run to run or transaction to transaction.
- Access right – mechanism used to provide mandatory and discretionary access rights to objects independent of accessor or object location.
- Accounting and resource allocation – mechanisms should be independent of location, although policies may be location dependent.
- Implementation and underlying operating system dependencies – access to O.S. supplied objects should only be through well-defined interfaces hiding the underlying implementation.

- Error recovery – there should be a uniform mechanism for error reporting and handling must exist for local and remote resource access.

Of the areas of desired transparency listed above, performance transparency is particularly difficult in the supercomputer environment. Performance is highly dependent on where the processing is done (e.g., the raw processing performance, load, and scheduling policies) and the effective latency and bandwidth between the distributed cooperating elements. While one could meet performance transparency by slowing jobs artificially, to some worst case common denominator, this is clearly undesirable.

Given the premium placed on performance in a supercomputer environment, the workshop consensus was that while transparency was a major goal, control of resource location and other performance parameters should be visible at appropriate interfaces.

I.C.1 Some Basic Assumptions

Some of the basic assumptions agreed on in the workshop were:

- We must evolve from where we are. This implies that we must carry a "reasonable" amount of the past along into the future. Specifically we must start from existing dominant O.S. interfaces and standards such as "Unix," and existing network protocol standards such as the Internet and OSI protocol suites.
- No one vendor or organization will implement or supply the entire environment implying significant underlying heterogeneity.
- While maintaining important upward compatibility in interfaces, the underlying implementations can and will change.
- No single organization is likely to administer or control the entire supercomputer environment, implying a confederation of administrative domains, components, and policies.
- To meet the desired behavioral goals above it will be necessary to create the model of a distributed, multi-processing computer. Such a system would contain both shared and non-shared memory. This implies the need for a distributed operating system and programming environment built on widely supported industry standards.

I.C.2 Some Applications of Distribution

The application of transparent distributed computing in a supercomputer environment will exist at several levels such as system, program development, and end application. At the system level, for example, one desires transparent access to resources such as files, processing, and hardcopy output. Files should be able to migrate, for example, between local storage and the hierarchy of storage supported by a central mass storage service. One would like to login once to the entire environment. Process structures should be able to span systems. One should be able to direct files for output to any of several hardcopy output devices whether to capital intensive high speed, high resolution central devices or to remote printers.

One would like to do application program development on workstations and have the option of controlling a distributed programming environment from the workstation with distributed *make*, source code control, and debugging facilities.

There are a variety of productivity and economic reasons to distribute actual applications. For example, in a large physics modeling system, one may want the problem setup and results analysis parts of the system on a local workstation while running the physics (often interactively) on a supercomputer. Even the physics may be best distributed between vector or massively parallel multiprocessor depending on the nature of the algorithm. These application modules will access other distributed resources such as files.

I.C.3 Status

Distributed computing will depend heavily on adherence to standards and the availability of uniform interfaces and services within the network. The widespread availability of Unix and de facto networking standards in the industry has finally reached supercomputers. Thus while distributed computing is not yet widespread, at least networking computing via standards is now common in supercomputer environments. Distributed computing in supercomputer environments is being held back by many of the issues common in other environments as well: unreliable and difficult to manage networks; poor network performance; difficulties in managing remote systems in uniform ways; problems with software distribution and configuration control; lack of adequate security mechanisms; heterogeneity problems in data formats, services, naming; performance problems with NFS; and the lack of a standard distributed programming environment and associated tools. There are also serious operating system specific problems as outlined in the next sections.

I.C.4 Supercomputing Additions to the Distributed Operating System Problem

The general resource sharing network environment and system behavioral goals outlined above are not different from those of the emerging non-supercomputer environment. Therefore this section addresses the question what new issues does supercomputing add to the problem? The following section outlines some more general O.S. requirements for distributed supercomputer and non-supercomputer environments. The following are the main factors identified:

- There is more heterogeneity and disparity in performance and resource capacity. For example floating point performance on various systems may range from the sub megaflop to hundreds of gigaflops. Floating point numbers may vary from 32 to 128 bits in size, presenting serious problems for the mechanisms used to hide heterogeneity. Files may range from kilobytes to gigabytes in size. Network speeds vary from kilobytes/sec to gigabytes/sec. Problems may need to be written to run on a single processor or multi-processors. The requirements implied are for supercomputer vendors to adhere to existing standards. It will also be desirable to make location and location control visible at the O.S. interface, if desired.
- Maximization of performance often dominates over other considerations, including transparency, ease of programming or use.
- The separation of data and processing that is often acceptable for workstation or other non-supercomputer systems is not going to provide acceptable performance for supercomputers. For example, the remote file access model may not provide acceptable performance and therefore, what is required is either process migration to the data or a caching file migration model that moves whole files close to the processing.
- Supercomputers from many current vendors do not support paged virtual memory. Many mechanisms and implementations being used in distributed computing using

non-supercomputers are built around the assumption of the universal availability of paged virtual memory. The implied requirement is that either supercomputer vendors need to accept the performance penalty associated with use of paged virtual memory or that evolving standards need to work in non-paged, non-virtual memory environments.

- Many supercomputer environments require both multilevel and need-to-know mandatory access control, as well as, secure discretionary access control. This results because there will be different domains of trust due to the confederated nature of the networks and other systems in the supercomputer environment.
- The nature of many of the problems run on supercomputers is that they run for long periods of time, generate very large files and in general consume very expensive resources. Therefore, failures in network servers, or nodes could be very costly if jobs need to be restarted from scratch. Therefore, improved restart services are required over those common in the current Unix and networking implementations, interfaces and practices.
- There is a great deal of ferment and experimentation with a wide range of supercomputer architectures from low levels of parallelism and vectorization to massively parallel systems, from SIMD to MIMD, from shared to non-shared memory or hierarchies of memory. It is therefore going to be a challenge to create a standard hardware architecture independent set of interfaces.

I.D Operating System Issues for General Distributed Environments

It was agreed that to meet the desired behavioral goals outlined above the industry needed to evolve the existing POSIX O.S. interface and command standards, network standards, and their associated implementations toward that of a distributed multi-processing operating system. For this to work well in a supercomputer environment these standards must take into account the unique characteristics outlined above of the supercomputer environment. This is a serious problem because supercomputers represent a relatively small market share and therefore their unique requirements are often ignored in official and de facto standards bodies and industry consortia. In addition, since most R & D work in distributed computing takes place on workstations, a testbed or research center is needed that allows the needed mechanisms to be efficiently demonstrated and available to the industry as a reference (and preferably portable) implementation on several supercomputer architectures. Further, being able to integrate supercomputers transparently into a reliable, easy to administer distributed environment is likely to be increasingly important to their future effective use.

Below we outline some of the major O.S. problem areas in support of distributed computing, along with some strawman requirements. These generally apply to all classes of systems.

It was a consensus of the group that the existing “Unix” interface and implementations did not support the desired transparency or the desired extensibility and modularity. It was generally agreed that the client/server model was likely to be the dominant architecture of distributed computing. However, current implementation techniques require low network latency simple services are performed efficiently. This model can be explicitly visible at the interface or be implicit in the implementation. The former approach is represented by the micro kernel O.S. architectures that basically support at least a message passing kernel and implement other O.S. services as explicit servers. MACH and CHORUS are examples of this approach implemented on a variety of systems. NLTSS is an example implemented at Lawrence Livermore National Laboratory on Cray XMP and YMP systems. The latter approach is the monolithic kernel that provides all O.S. services within a protection boundary. The pros and cons between the approaches revolved around the tradeoff in modularity and extensibility of the micro kernel approach and the possible performance advantages of the monolithic kernel approach. Those with experience with the micro kernel approach claimed they had demonstrated competitive performance. There was strong support for the micro kernel approach, but there was not consensus.

It was agreed that the three basic services needed in support of distributed computing are distributed file access, remote execution and flexible transparent interprocess communication.

Distributed File Access

With respect to file access it was agreed that Sun's Network File System (NFS) has been a very valuable tool, but that in its current form it has some serious limitations particularly for a supercomputer environment. In the NFS model the data remains remote and can only be accessed in relatively small pieces per synchronous NFS RPC request. A model in which files are migrated at high speed and cached local to the processing seems more appropriate to the supercomputer environment. This area is discussed further in next section on data storage.

Remote Execution

The area of remote execution is an area needing extensive work. Ideally one would like to be able to create remote processes on different systems in a machine independent way (e.g., without having to login or use a password, and without having to know architectural details of the target system), have access control, establish the appropriate environment, and incorporate the process in a distributed process tree or other structure. The current process interface does not easily support such a goal and such mechanisms as *telnet login, rexec*, or RPC to pre-existing services are inadequate. The problem becomes more complicated if requirements are added for load balancing, process migration, and establishing processing location based on functional characterization. In the remote service invocation area there is no widely agreed on and implemented standard. Such a standard is required to provide implementation inter-operability.

Interprocess Communication

It was generally agreed that the current Unix IPC services do not meet all the desired requirements. For example:

- Pipes and signals are only allowed between limited topologies of processes.
- System V messages allow unprotected message queue ids to be passed. This is too high a level of abstraction and inefficient a mechanism on which to build distributed computing.
- Signals are easily lost (no flow control).
- The blocking I/O data stream model of IPC creates problems for multi-tasking.
- The byte stream IPC model requires message or other separators and synchronization signals imbedded in data. (There is no clean separation of data and control.)

It was agreed that interprocess communication is the core of operating system design for distributed systems and network resource sharing. Systems need to be built around IPC as a central service. Some of the central IPC requirements are to support:

- A uniform transparent local and network IPC model so that programmers do not need to program differently to send local or remote messages and data.
- Arbitrary process communication structures. That is, any process can communicate with any other subject to security policy and access right constraints to support the

client/server model and in general provide increased application structuring flexibility.

- Asynchronous (non-blocking) and synchronous (blocking) communication between processes.
- Arbitrary length messages removing any constraints on application message size.
- Standard request/reply syntax and encodings to deal with heterogeneity.
- Separation of data and control to provide for third party data transfers and pipelining.
- Allowing requests and replies to be sent on separate streams so that replies can be returned from a different location than that to which the request was sent.
- Flow and error control of the communication to eliminate the need for such mechanism in applications.
- Protection of the communication streams.
- Multiple concurrent message streams between a given process and one or more others to support flexible application structures and the client/server model.
- Communication protocols that can support both low delay (transaction style applications) and high throughput (bulk data transfer applications).
- Support for multiple protocol suites.
- Hardware support that makes local IPC competitive with procedure call and supports high performance network message passing.

The section on High Speed Networking in the supercomputer environment discusses additional requirements. The “best” IPC model, if one exists, for distributed computing is still unknown. There was argument for a virtual shared memory paradigm, varieties of remote invocation, and streams.

Multi-tasking and Multi-processing Requirements

Concurrency along with communication are at the heart of distributed processing and multi-processing. Not only is there a requirement for application codes and servers to be multi-tasked, but the kernels of individual nodes will need to be multi-tasked to run efficiently on small and large multiprocessors.

- Because of blocking I/O and weak kernel synchronization mechanisms, the Unix kernel cannot be multi-tasked without significant modifications.
- Unix processes are expensive to create, communicate between, and context switch between. Even in System V where processes can share memory one still often needs to sub-multiplex Unix processes into lightweight tasks for efficient server and multi-tasking application design.
- Because of blocking I/O and synchronized kernel calls, it is difficult to multi-task within a Unix process. This requires new asynchronous system calls and other process related system changes.

The variations of multi-tasking/multiprocessing that need support are the following:

- Single processes multiplexed into many lightweight tasks sharing memory so that multiprocessing codes can be initially debugged in a simpler single processor environment and that single processor applications such as distributed resource servers that need logical concurrency can be supported.

- Many processes (sharing memory) working together on many tasks; the dominant shared memory multiprocessor supercomputer architecture.
- Many processes with their own memories working concurrently and cooperatively via message passing; This is the distributed processing model and an increasingly common multiprocessing architecture for very large numbers of processors.
- Multi, micro, and auto tasking based multiprocessing, the common approaches to supporting different levels of granularity of application concurrency.
- Symmetrical multiprocessing, that is, the operating system and applications can be multiprocessed on all processors to maximize both system and application concurrency and CPU utilization.

Successful multiprocessing requires inexpensive mechanisms to spawn and switch context among concurrent entities. These costs should be on the order of those of a procedure call. It also requires mechanisms for concurrent entities to share memory, communication ports, and other resources, and to be able to synchronize access to these shared resources.

Extensibility Requirements

One of the main purposes of an operating system is to provide commonly used abstract objects such as files, processes, directories, message streams, etc. for use as application building blocks. One thing O.S. designers have learned over the past three decades is that there is a constant need for new types of abstractions. Distributed computing we believe will accelerate this trend. Extensibility deals with the ease of adding new types of objects and who can add them, their naming, protection, etc. There is no mechanism at the Unix interface to add new types of abstract objects and support their communication, naming, protection and error control in a way uniform across resources. Unix has no unified extensible model of abstract objects and the functions operating on them; processes constitute one world, files another, newer objects such as message queues, semaphores another. The main Unix abstraction is the byte stream. Many of Unix's current problems stem from the limitation of this abstraction. Requirements in this important area are the following:

- Support an object-oriented client/server architecture that allows users as well as system's staff to add new types of objects and services (preferably without kernel modifications).
- Support uniform naming, protection, communication, etc. mechanisms for all objects added in the future, whether by users or systems staff.
- Allow object managers to be created, replicated or run anywhere in the network as most appropriate for economics, performance, reliability or other reason.

Compatibility With Existing Operating System Interfaces

In order to protect the large investment of the user community in existing programs, training, and procedures the following requirements must be met:

- Support existing user-at-a-terminal interfaces.
- Support application source code compatibility at the system interface library (e.g., POSIX) level.
- Support existing utility and public file sets.
- Support existing shared network services (e.g., storage, I/O, terminal, authentication).

- Support incremental evolution (i.e., allow mixing of existing and new native functionality).

The micro kernel advocates argued that the micro kernel approach was particularly well suited to providing compatibility with the past while allowing new interfaces to coexist in an upwardly compatible fashion.

Naming and Identification Requirements

The design of identifier and naming systems is another area at the heart of system architecture. One of the important characteristics distinguishing desired operating system designs is their explicit recognition of the importance of naming and the need to provide uniform and distributable mechanisms for this function. It is currently common for separate naming mechanisms to be used for each type of object. Often some objects can be given human-oriented names or be shared while others cannot. Some identified naming problems in the existing Unix interface that affect distributed computing are:

- Human-oriented pathnames for files are the only global, sharable name space. Therefore, new types of objects must be cast as files which may not be meaningful. Host names are often visible in many distributed versions.
- There is no clean separation between the human oriented name space for objects and their lower level machine-oriented names (which are usually not globally unique).
- Can't share processes, pipes, message queue, semaphore, other objects in any general persistent way (i.e., can't name all resources in sharable directories).
- Interprocess communication mechanisms are not integrated into the naming system.
- Directory structure is a tree that limits sharing.

Recognizing the need to remove these limitations the following goals are identified. Support:

- A distributable system viewed as global space of named objects rather than named hosts containing objects in order to make host boundaries invisible and to allow users to organize their name space as appropriate to the problem domain not network configuration.
- Clear separation of machine and human oriented names in order to allow multiple human oriented naming mechanisms, if appropriate; to allow new types of objects to include names of existing objects without name conflicts; and to support uniform naming of all types of objects.
- Allow machine-oriented object identifiers to be passed between systems or be stored in other objects in order to simplify protection and IPC mechanisms.
- Distributable mechanism for generating globally unique machine oriented object identifiers needed for transparent distributed system construction.
- Mechanism for having multiple copies of the same object for caching and redundancy without name conflicts.
- Mechanism allowing many human-oriented names for the same object so that objects can be shared and appropriately named by each user sharing the object.

Protection

Protection mechanisms in existing O.S.'s are usually weak or not securely extendable to distributed systems. Some of the problems with the existing Unix security interface include:

- lack of standard mandatory and discretionary access control mechanisms and their visibility and support at the network and utility (command) interfaces, and within the network.
- not least privilege.
- overuse of *root* and *set–user–id* mechanisms open holes in security envelope.
- does not distribute easily securely because UIDs and GIDs must be passed in the clear and be trusted.
- cannot dynamically pass access control rights.

Some needed requirements for protection are the following:

- Support a secure, consistent multilevel and need-to-know mandatory access control mechanism for all resources.
- Support least privilege discretionary access control of all resources and rights passing by application with the appropriate rights.
- Support mutual suspicion between nodes (i.e., all system component node kernels should not need to trust each other).
- Support access control lists to allow individual users to have different access rights for a given resource.

Resource Management Requirements

Some resource management goals associated with distribution are the following:

- Provide distributed accounting for all resources.
- Support distributable administrative allocation of resources if desired.
- Support resource relocation, migration, and caching.
- Support resource sharing.
- Support distributable detection and purging of lost objects if any.
- Provide a foundation that would allow improved batch facilities and more dynamic job scheduling (load sharing) across machines.

Error Control Requirements

Error control goals are the following. Support:

- Robust fail soft implementation techniques.
- Deadstart with minimal loss of state (e.g., deadstart with processes, files, etc. recovered without users having to explicitly checkpoint).
- Standard error reports from all operating system services.
- Robust communication protocols.
- Distributed exception reporting and handling mechanisms.
- Mechanism on which atomic transactions can be built when needed.

Separation of Policy and Mechanism

Because the supercomputer environment is envisioned in the general case to be a confederation of different administrative domains, issues related to separation of policy and mechanism seem particularly important.

Accounting, resource allocation, security were some of the areas discussed. It was felt that the current interface and implementation either does not adequately support such separation or has inadequate mechanisms to deal with heterogeneity in such policies.

System Administration

An area discussed which is holding back setting up more tightly coupled distributed computing in the supercomputer environment is system administration. System administration is currently difficult and a source of additional heterogeneity.

I.E.1 Follow-on Work

It was generally felt that the workshop just scratched the surface of the issues outlined above. It was agreed that two useful follow-on work items could be pursued:

1. Develop more detailed distributed supercomputer application scenarios to provide for the refinement of requirements.
2. Analyze the current Unix interface semantics (Unix Reference Manual Section 2) in detail for specific problems with respect to support for distributed computing in the supercomputer environment and propose modifications or additions to solve them.

I.E.2 Conclusions

Six main conclusions resulted from the working group on distributed computing in a supercomputer environment:

1. Distributed, resource sharing computing is the wave of the future for use of supercomputers and possibly vital to supercomputer economic viability.
2. A standard distributed O.S. interface is required.
3. The supercomputer environment has some unique requirements for a distributed computing environment.
4. The existing Posix/Unix interface does not naturally support distributed computing requirements in many areas.
5. There is a need to demonstrate the efficient, practical, viability of distributed O.S. features as a reference implementation on several supercomputer architectures.
6. More work is needed to refine the issues presented in these discussion notes.

II. Archival Storage for Distributed Supercomputing Environments

II.A Current Storage Capabilities†

The long term storage of data has been a continuing concern within the supercomputer community. In fact, the supercomputer community has continually pushed the forefront of mass storage systems. Mass storage currently implies the storage of data in excess of 10^{12} bits and it is not uncommon in this community to see requirements of over 10^{15} bits.

Hardware Network Connections

There are several technologies that can be used to connect distributed MSSs. These technologies include fiber optics, coaxial cable, satellite relay, microwave and radio relay. The fiber channel standards work and ongoing research indicate that a single faster striped channel, composed of 16 channels each capable of peak rates of 800 Mb/s, will provide peak rates of 12.8 Gb/s for distances of up to 10 km. Metropolitan area connections will require store and forward nodes, which will reduce the peak rate of an end to end link.

Mass Storage System to Supercomputer

There are second order effects that impose fundamental limits on the throughput of traditional protocol stacks, and that come into play at speeds of 100 Mb/s and greater. The second order effects are due to packet size and bit error rate. Reagan Moore of the San Diego Supercomputer Center reports that Cray Research Inc. has measured a rate with TCP/IP of 640 Mb/s on the 800 Mb/s (80%) Cray high speed channel between two supercomputers. This requires that 12.5% of the total processing power of the supercomputer is available to run the protocol. The protocol overhead rate is 1.56 ns per bit.

Moore has developed a model that predicts the usable transfer rate and CPU availability [Moore]. With fiber optic links the bit error rate decreases substantially and Moore's model predicts a usable rate of 95% of a 2.4 Gb/s link with a bit error rate of 10^{-8}, protocol overhead rate of 1.56 ns per bit, packet size of 480 Kb and 47% of the total CPU cycles. The dominant overhead is due to the protocol overhead with error correction as a second order effect. Assuming a significant part of the rate can be achieved on a busy supercomputer, a mass storage system to supercomputer link at a sustained rate of 0.8 Gb/s is achievable.

Strategic Standards

Standards work is very important for several reasons. With standards, development time is shortened so that state of the art hardware will in fact be delivered and for a reasonable cost; evolution of systems is made easier; and a spectrum of choices in hardware with differing performance and cost trade–offs becomes available.

The IEEE Storage System Standards Working Group (which will set several standards for aspects of the storage system), Fibre Channel, HIPPI, FDDI, IPI3, SCSI 2, ESCON, FIPS60 and high level protocol standards groups, which include users, are currently meeting regularly. Support in terms of users from the Intelligence Community is needed to evaluate the issues and attend the meetings to work on the issues. The trade–off of RAIDS (Redundant Array of Inexpensive Disks) type magnetic tape or optical disk architectures versus expensive, fast and sometimes unreliable recorders is an issue [RAID].

II.B Desired Storage Characteristics

An archival storage system permits data to be moved to slower, less expensive, larger capacity devices when immediate access is no longer needed. These systems deal with bitfiles as their smallest unit of data. Systems are expandable and bitfiles may reside in an archive for decades.

† This section is extracted from Mass Storage Requirements in the Intelligence Community, Tom Myers and Elizabeth Williams, to appear in *Proceedings of Supercomputing '91*, November 18–22, 1991, Albuquerque, NM.

The migration of files toward archival storage should normally be automatic. The site administrators determine a policy that dictates data migration and allocation inside the DCE archival storage system.

The implementation of archival storage and bitfile management for the complete environment exposes several problems. Bitfiles need globally valid names that can be used as they are being moved through the DCE to/from the archive. A file system hierarchy must be defined. Each component in the DCE needs to know the file system hierarchy for which it is responsible, and the location of the boundary through which the network and the archival system operate. The file system hierarchy is also divided across a DCE and within a supercomputer.

Several related capabilities are addressed by data storage standards. Examples of such standards are illustrated by the IEEE Mass Storage System Reference Model functionality:

1. Separation of dataflow and control messages. This allows the creation of network disks which are accessible by multiple computers, with access controlled by the distributed file system.
2. Separation of user file name and the machine bitfile identifier. This allows arbitrary mappings to be developed for defining the user interface to the distributed file system such as a UNIX file system structure or a Database Management System interface.
3. Separation of the logical file system from the physical media used to store the files. This allows transparent data migration of the least referenced files to cheaper storage media and also removes many of the UNIX disk administration limitations. Files can be moved off of failing disks and disks can be dynamically added or removed from the system without having to rebuild the entire file system.

Although the third point above advocates transparency with respect to storage media there may be cases where components of the DCE need to know the storage media differences. This requirement manifests itself both at the distributed nodes and at the supercomputer because of the differences in the physical media structure.

Data exchange must also address the issue of multilevel security. While this is normally provided by the operating system, access to a distributed file system can open new security holes. An example is migration of secure files from disk to tape. The remnant space on disk must be overwritten to guaranteed that the classified information has been completely eradicated.

II.C Current Work on Storage Problems

Two illustrative examples of an approaches to solving global naming problems are the Internet IP name service and the IEEE Mass Storage System Reference Model concept of nameservers. The first separates names by local computer, site, and region. The second allows the construction of gateways to control the data associated with a local site. As part of the IEEE_MSSRM, remote data is cached locally upon reference, allowing fast access for future references to the data. Global nameservice then resides in the gateways. The issues for implementation are:

1. administrative: Who maintains the gateways and the unique site and region names?
2. performance: Can local caching meet the requirements for fast file access and for minimizing network traffic?
3. file system hierarchy: Consistency between locally cached files and archival storage must be maintained. This either complicates nameservice by requiring DCE file

references to be directed to the cached data or archived data must be locked until cache updates are received.

Each bitfile has a set of associated attributes. The attributes relate to how the data must be maintained as it migrates through the hierarchy, as well as through the DCE. The bitfile carries attributes based upon its location in the hierarchy, or in the DCE, that may be different from those needed at the supercomputer level. Many of these attributes are related to the data content and where it resides in time within the DCE. There are evolving meta-data representation methodologies but few are standardized.

Data must be delivered in a reliable, rapid manner to the supercomputers, and perhaps at a different rate to other systems in the DCE. Throughout the DCE we have to target the data transfer protocol issues to be addressed by the operating system. Thus, we must deal with network and data storage interfaces such as those between memory, a solid-state disk, and physical media. It must be assumed that any given operating system inside the DCE must receive and deliver reliable data. Thus, data integrity must be maintained by the system for any given domain inside the DCE.

Of great importance is the issue that the file index for the data must be secure. Under no circumstances should a file index be lost, destroying the equivalence between file names and locations on a physical storage device. Current systems solve this problem by shadowing the file index directory on multiple disks and by journaling all changes to the file index.

Several working committees are attacking the various problems delineated above, and are trying to confront the difficulties in these environments. this work is progressing mostly in the United States. The IEEE Computer Society Technical Committee on Mass Storage Systems is in the process of developing a Computer Society draft standard on data storage systems. The current working draft provides a consistent terminology and an object-oriented design for defining storage subsystem components, whether they are being build around a single system or in a DCE. Other groups in the computing community are currently dealing with the problems of data migration within a distributed environment. This distributed environment may or may not include a supercomputer, but it almost always includes a high-volume storage system of some sort delineated as a “mass storage system.”

II.D Problems of Data Representation and Exchange

The problem of data exchange between different computer architectures and operating systems has been existent since the inception of the computer. This problem has been exacerbated by the acceptance of the client–server paradigm as the provider of distributed services. Distributed computer services require immediate data exchange. In the past, data was exchanged on some media, such as tape, and could be examined at leisure. Ad hoc data conversion routines were created to process the data and were often embedded in the programs using the data. Data exchange in the client-server paradigm does not permit this leisurely data examination. Both the client and the server must be able to “call” software that is guaranteed to convert the exchanged data “on the spot.” This guarantee also implies a standard format rather than the ability to convert all formats because it would be impossible to maintain multiple architecture conversion software and, of course, the size of such conversion software would be enormous.

The issue of data exchange has been addressed resulting in many data exchange software packages. A few of the currently more popular packages are XDR, HDF, NetCDF, PostScript and CCSDS. Each of these packages addresses a specific type of data. Some address bitmap data; one addresses the general encoding of “display” information. Some of these packages address various numerical representations in computers. It is unclear that any existing package could or should be extended to solve all data exchange problems. However, a more realistic approach would be a collection of data exchange packages formed as the “standard.”

Another important issue in the exchange of data is reliability through unscheduled interrupts, whether caused by hardware failures, network interruptions, or software failures. If a data transfer fails, the data transfer should automatically requeue at the discretion of the user.

References

[Moore] Reagan Moore, "Distributing ASpplications Across Wide Area Networks", *General Atomics Report GA–A20074,* April 1990.

[RAID] David Patterson, Garth Gibson and Randy Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *ACM SIGMOD*, Chicago, IL, June, 1988, pp. 109–116.

III. High Speed Networking and Supercomputer Operating Systems

III.A The Current Environment

Networking structures are hierarchical, consisting of wide area linked subnets of metropolitan or local area networks (MANs or LANs). LANs are typically contained within a building; MANs cover a college campus or institution; and WANs connect the world. Examples of existing WANs are the *Internet* or *NSFNet*. For discussion, consider high speed to be 1 billion bits per second (Gbps) or greater, medium speed to be 0.1 Gbps (or 100 million bits per second (Mbps)), and low speed to be below 0.01 Gbps. LANs exist with these definitions (e.g., Ultra, FDDI, and Ethernet respectively), and currently these network types are used to connect supercomputers to other computer systems, including supercomputers. High-speed WANs are on the near horizon (e.g., there are five Gbps testbed collaborations involving Government, Industry, and Universities). While, in general, the topologies are broadcast in the LAN arena, and point-to-point in the WAN, there are a few point-to-point LANs.

The machines serving as end nodes are heterogeneous, both in the sense of architecture (e.g., word length and floating point formats) and in the sense of performance (e.g., a Cray-2 and a Sun-2). Both types of heterogeneity make it extremely difficult to construct a network architecture that is flexible without sacrificing the performance of the network.

There is a mixture of bulk and interactive traffic; it might be expected that the future WAN traffic will tend towards bulk data transfers due to propagation and switch induced latency. On the other hand one of the most common uses of a large existing WAN–the Internet–is for *telnet* and other non-bulk transfers of data.

Security requirements for components of the network also vary. Machines on a LAN may be more trusting of other machines and users on the same LAN; connection over a MAN or WAN must be viewed with greater suspicion.

III.B Desired Networking Capabilities

In the supercomputing community and the high speed networking community, the “Big Goal” is delivering a significant fraction of the raw channel bandwidth to applications. This means that applications should be able to obtain Gbps end-to-end bandwidth. It is possible to envision achieving this goal between supercomputers without a significant hardware assist because the machines themselves are so fast. On smaller systems, such as workstation and minicomputers, a hardware assist of some sort is mandatory. Existing commercial network interface architectures are not suitable for the Gbps domain.

While it may seem sufficient to multiplex the Gbps channel and thus provide many more Mbps channels than are currently possible, there are applications which demand the higher bandwidth–data migration, distributed supercomputing, and scientific visualization are current examples.

Thus, there are certain sub-goals which should be realized by an operating system intended to support these networks. In particular, we expect that the operating systems will be multiprocessing in nature, and thus there will be a need for fast interprocess communication (IPC) and low-overhead protocol suites. Given the varying mixtures of bulk and interactive traffic, it is desirable that performance be good for both cases, i.e., high bandwidth and low latency. Experience suggests that protocol implementations must be augmented by reasonable support from the operating system, and we suggest light-weight processes, fast context switches, and high-performance scheduling as desiderata for high-speed network interface software.

It is desirable to have as clean an interface between the machine architecture and the network interface as possible. This desire is motivated by the observation that much software (and consequently, performance) is spent in melding the abstractions of each. Research into new I/O architectures will be necessary to determine the architecture of such an interface. This research will necessarily involve both hardware and software issues.

In addition, we see the need for reference implementations of network abstractions, in order that new implementations can be verified and tested. Internetworking facilities and features are necessary, and serve the additional purpose of adding the flexibility to incorporate new hardware and network abstractions, as necessary. The desire for internetworking capabilities is tempered to some degree by the need for varying levels of security, which implies that an ability to construct "secure" systems is desirable.

III.C The State of the Art in Networking

In the area of network abstractions no real choice can yet be made. Possibilities include memory, streams, messages, remote procedure call (RPC), sockets, I/O devices, et cetera. In addition, the relationship between these abstractions and the IPC mechanism is not clear; some are clearly "nice fits" but given the heterogeneity of traffic and performance there is no consensus. Generally, systems have been characterized by a single network abstraction for all sorts of traffic, both bulk and interactive. TCP/IP, and network software which uses this suite, is in common use and optimized implementations have achieved bandwidths in the range of 500 Mbps on supercomputers, kernel-to-kernel.

- Gbps LANs exist, as do high-speed switch fabrics, and several more of each are in various stages of prototyping.
- Security is generally poor or achieved by separation of networks.
- Networking management is presently in a poor state, and worsens when heterogeneity is considered.

Given this state of affairs, we have identified various operating system problems which stand in the way of our goals; these are discussed in the next section.

III.D Problems That Must Be Addressed

We saw a variety of operating system problems that could be addressed by implementations or further research.

The network abstraction is inextricably tied up in the structure of the operating system, which has a strong influence on the implementation of networking abstractions and their performance, and much of the bulk and resource usage of present operating systems is due to migrating functionality into the kernel for performance. Memory architectures present a problem for many of the network and IPC abstractions which have been proposed, especially those predicated upon the existence of virtual memory support. Highly layered protocols, or more correctly, current implementations of highly layered protocol abstractions, present a severe performance problem; much of this is due to the layering mechanisms provided by the operating system.

Several points of view as to proper operating systems structure were represented. These could be described, more or less, as monolithic kernels and microkernels.

The main goals of the microkernel approach are to provide a framework which efficiently supports modular, distributed, higher–level operating systems. Ideally, microkernels provide a minimal set of fundamental operating system primitives which are common to varying operating system environments. Higher–level operating system environments, such as UNIX, can then be constructed as sets of independent, cooperating servers using the primitives supplied by the microkernel. These primitives, typically, provide a machine–independent interface to manipulate such basic operating system objects as address spaces and process contexts. The semantics of these primitives tend to be as general as possible in order to support many different high–level operating systems. In addition, microkernels often provide a location-transparent IPC mechanism based upon *ports* and *messages*. If a higher-level operating system has been constructed as a set of communicating servers, these servers can be loaded independently, depending upon specific needs or resource availability.

Monolithic kernels, on the other hand, are characterized by a single loadable image that comprises all operating system services inside a single set of protection boundaries which are crossed by system entry points for the kernel. Though a monolithic kernel may be modular and well-structured, it is inherently necessary to provide all of its services, even if they are not needed on certain sites.

There is a perceived trade-off between speed and manageability. Proponents of microkernels believe that any minimal overhead in the local case can be eliminated or made negligible through careful tuning and optimizations. Furthermore, they believe that this overhead will be compensated for by performance gains due to distribution. Proponents of monolithic kernels believe that any added overhead is inherently unacceptable.

The granularity of scheduling is an important problem, and has a strong impact on the performance of protocol implementations and the ability to multiplex traffic. Interrupt and event-driven scheduling often preclude polling where it would be a better design decision in practice.

The lack of a trusted operating system precludes the design of systems which interoperate with other systems having unlike security requirements, and this in turn precludes the sharing of expensive network resources.

Since many networking functions are essentially grafts onto existing operating systems for standalone systems, network status information and control facilities are minimally available, if at all. In particular, the lack of counters and trace facilities precludes effective network management, and where they are available, they are often implemented in such a fashion as to impact performance.

Supercomputer operating systems must support asynchronous operations, which are a pervasive solution to speed mismatches between supercomputers and external services such as I/O devices and networks. These latencies are especially severe when in a WAN setting, since the propagation delays are a physical limit rather than a technological one.

Supercomputers are not available to a large segment of the operating systems and high speed network research communities, from which many current technologies, such as UNIX and TCP/IP, have emerged. Where some supercomputers are on the network, they are available for applications research only. The result has been that much research has ignored supercomputers and supercomputing environments. An unintended side effect of this is that the Japanese manufacturers benefit most from this situation, since their supercomputers are architecturally similar to systems available to OS researchers in important respects, such as support for virtual memory. We noted that there must be availability of both supercomputers and high-speed networks to the OS research community, for experimental use, in order for solutions to be realized.

III.E Conclusions

Thus, our conclusions are:

1. Current operating systems are not designed for the sort of high-speed bulk data transfer which characterizes much of the traffic involved in supercomputer data

migration. We propose that either a single network abstraction be developed which handles these transfers well, or that multiple network abstractions be used. This latter choice may be one of multiple physical networks or of multiple abstractions for as single network underpinning.

There are number of possible network and network interfacing paradigms possible, such as virtual circuit and datagram. At a higher–level, network paradigms display such characteristics to programmers as node names, domain names, connection types, etc. It is attractive to develop a paradigm such internetworking, where a uniform set of services is provided to all participants in a network. TCP/IP is not the only possible view one can take of networking, but it shows the value of simple naming and uniform interfacing (by all TCP/IP participants). However, the TCP/IP strategy of building a service provision model where almost any underlying technology can be used (T1, 802.3, etc) may reduce any advantages native to that technology (hardware multicast, megabyte packets). This can cause a split, where one networking paradigm is chosen for one class of applications, and another paradigm is chosen for applications which can make better use of the technology. To the degree that the paradigm resolves differences between technology, provides a consistent user interface, and preserves any advantages offered by the technology, it will be effective.

2. We (and others) have achieved no consensus on the issue of network abstraction, and workshop consensus was that we should let experimental evidence accumulate before drawing conclusions and setting directions. Various experiments are underway to test the feasibility of some alternatives such as distributed memory. None of these experiments currently addresses the needs of the supercomputing community, however.

3. Heterogeneity is a problem, and is expected to remain one, for a variety of reasons. Particular drivers are (1) the cost and quantity of existing equipment; (2) the competitive advantage due to differentiation; and (3) rapid advances in technology. A solution to this problem is adaptable operating system configurations, where adaptable does not necessarily imply portability of the operating system itself, but rather portability of applications. This can be achieved by the addition of modules, where necessary, to mask architectural heterogeneity.

4. Security is a concern, and it should be noted that many supercomputers are in environments where it is a rather more sensitive issue than workstations connected via the Internet. There are no hard and fast solutions, but there was some agreement that trusted operating systems with support for authentication protocols were a good basis. For example, this basis and adaptability allows the use of end-to-end or link encryption. There was some concern expressed that link encryption in the Gbps range was not readily available.

5. Operating system structure was the focus of a great deal of discussion during the workshop, and there was consensus that the "microkernel" abstraction, exemplified by the Mach, and CHORUS operating systems, was a good one. However, there was concern expressed about the performance realizable by these schemes. In the high-speed networking realm, the microkernel systems are attractive due to their

inherent flexibility and ability to “dekernelize” much of the networking functionality.

6. Scheduling in multiprocessing systems is an issue which has particular impact on the performance of high-speed networking implementations. Our conclusion was that fine-grained scheduling (where threads of control are divorced from what usually comprises the process state, e.g., an address space) was desirable, if possible, since context-switch burdens can be reduced. Such scheduling is well-matched to the implementation of layered abstractions and reduces the performance burdens. In addition, non-preemptive scheduling is desirable in multiprocessor systems where individual processors may be dedicated to specific tasks, or where sets of processors may be dedicated to long-running jobs requiring no intervention by the operating system.

IV. Architectural Issues for the Supercomputer OS Environment

IV.A Current Architectural Issues

A supercomputer operating system exploits the architectural features of the supercomputer by functionally satisfying three basic areas of utilization, each of which has an associated time constraint:

1. Real-time (minimize frame time)
2. Interactive (minimize user response time)
3. Batch (minimize application wall-clock time).

Current supercomputer performance levels are achieved only by architectures that employ multiple processors. Two styles of machines prevail. First, the power of an individual processor can be maximized; groups of processors are configured to implement a MIMD (Multiple instruction, Multiple data) environment. Efficient control implies an effective set of hardware inter-processor synchronization mechanisms as well as specific hardware features which allow efficient exploitation of the multiple processors from user applications. Second, massively parallel arrays of relatively inexpensive, highly integrated, processors can be used to construct a SIMD (Single instruction, Multiple Data) environment. Existing massively parallel machines traditionally do not have operating systems which execute across all of the processing elements. These types of machines are usually controlled by a more traditional, front end computer through well defined hardware interfaces.

Every multi-processor supercomputer vendor has a specific implementation of its operating system. Some implementations are Multi-tasked while others are Multi-threaded. It is unclear which operating system architecture currently exhibits the best performance and which will scale properly with an increased number of faster, Von Neuman style processors.

Real memory:

Since the supercomputer community is transitioning toward a standardized operating system (i.e., UNIX), it is appropriate for supercomputer vendors to require relatively easy movement of “standard” operating system reference ports to real memory, supercomputer architectures.

Almost all of the currently available supercomputers utilize a “real memory architecture” to minimize hardware translation overhead time. A virtual memory architecture adds enough hardware overhead to effectively decrease the overall peak performance of the supercomputer.

It is currently still possible to port most “standard” reference ports to real memory architectures. However, it is apparent that the directions of the operating system standards committees is away from real memory implementations. Several user level interfaces which are being considered as “standard” rely heavily on virtual memory support both in the operating system and in the hardware. As the standards evolve and become more dependent on virtual memory, it will be quite difficult to provide “standard” updates to the supercomputer user community.

Performance monitoring:

Hardware and software architectural features to allow efficient monitoring and optimization of performance is also an important concern. This capability has historically been eliminated because of the inability to perform the functions required in a non-intrusive fashion with a minimum impact on performance. Currently, it is fairly difficult to obtain detailed tuning information from existing operating systems. It is even more difficult to interpret existing available data.

Fault handling:

With the advent of general purpose supercomputers entering markets traditionally held by slower, more general purpose machines, a larger "mean time to interrupt" (MTTI) is required. Supercomputer architectures and their associated operating systems have historically gained performance by sacrificing hardware redundancy and "fail-safe" hardware and software. Current standardized operating system software relies heavily on hardware fail safe features. Many supercomputer operating systems lack the extensive error recovery that insulates the end user from these problems.

IV.B Desired Architectural Features

The desired behavior of multiple processors is to decrease the wall clock times for large applications both in a heavy mix of jobs and in a dedicated environment. An efficient supercomputer operating system allow the execution of mutually exclusive job streams on each processor or allow all of the processors to be applied to a single application in parallel or some combination of these approaches.

The challenge to the supercomputer community will be how to bring massively parallel processors into the traditional supercomputer mainframe, and tightly couple them to the other, more general purpose processors. Essentially, the massively parallel processor is viewed as a special purpose hardware entity from the operating system perspective. From the user perspective, a specialized class of applications can utilize massively parallel processors more efficiently than it can utilize the traditional supercomputer processors.

A supercomputer operating system must be designed to effectively manage very large, real memories and the additional mass storage they require. As part of the real memory requirement, very large memories (16 Gbyte) and larger should be recognized as a given in the next 2–4 years on supercomputer architectures. An additional part of the problem is the lack of capability (both bandwidth and density) of I/O devices to keep up with the increased real memory supercomputer architectures.

An end user should be able to extract extensive hardware and software statistics about a particular application to aid in the tuning of the application on a multi-processor architecture. Effective presentation of the data to the end user is as important as the raw data itself.

Standard operating systems will address recoverability. Hardware and software designers should make intelligent tradeoffs between performance and resilience of the supercomputer system. The ideal behavior is to have non-stop supercomputing which makes system reboots due to hardware and software problems transparent to the end user.

IV.C Problems and Current Status

For the most part, standards committees have focused on multi-processing on smaller, non-supercomputer types of systems which exhibit a different set of performance issues and are used for the solution of a different class of problems. In addition, since a supercomputer is such an expensive resource, there are very few sites that can afford to dedicate a fast, multi-processor supercomputer strictly for supercomputer operating systems development.

Parallel execution must be the norm in both applications and within the operating system. This can be accomplished by either designing a Multi-threaded operating system (multiple processors allowed to execute the same instruction sequence simultaneously within the operating system) or a Multi-tasked operating system (separate, mutually exclusive tasks comprise the operating system. Each task can be executed only on a single physical processor).

An additional issue is to attempt to delineate how efficient additional numbers of processors in future supercomputers will really be. Adding additional processors into future generations of supercomputers adds large amounts of additional complexity to the basic machine design thus increasing the cost and decreasing the performance because of additional hardware and software management. It is currently unclear how the existing base of applications and future applications will benefit from more physical processors. Many user applications contain a finite amount of additional parallelism because of the nature of the physics inherent in the problem.

IV.D Steps Toward Solving Architecturally Related Problems

To begin to understand the above issues, the supercomputing community must provide additional, multi-processor supercomputers (or effective models) for operating system research and development. The basic question to research is the degree of parallelism that is appropriate to provide the best price/performance ratio both in the operating system and in user applications.

In addition, hardware and software research should be performed to treat arrays of massively parallel processors as asymmetric processing entities. From an operating system standpoint, the arrays of asymmetric processors may share a common, global memory with its standard supercomputer processor counterparts. It then becomes an operating system issue of how best to allow user access to the asymmetric processor.

A real memory architecture porting base should be established as part of the standard operating system releases. Also, the standards committees should recognize the existence of real memory machines in general and the problems associated with the operating system management of user applications which are many gigabytes in size. Standards committees must also recognize the standardization of low overhead, high bandwidth user interfaces like asynchronous I/O to take advantage of larger memories and faster and more dense I/O devices.

Extensive work should be performed to incorporate non-intrusive performance data into existing hardware and software supercomputer operating systems. Extensive work should also be performed to provide a graphic visualization interface for the end user to present the data in a meaningful manner.

V. Debugging for Correctness and Performance

V.A Introduction

This section addresses the supercomputing operating system issues related to the analysis of proper performance and functionality in supercomputer applications. The underlying philosophy of our recommendations reflects the fact that time, in particular supercomputer CPU time, is an irrecoverable resource. Thus the design of a supercomputer operating system must assure that computational cycles are rarely lost. To this end, we recognize the following as system features prerequisite to proper debugging and performance analysis:

1. A robust checkpoint/recovery facility.
2. A checkpoint/recovery mechanism which is both system administrator and user controllable and configurable.
3. An exception processing facility which captures and retains sufficient process state to enable recovery, "editing" and continued execution of that process.

Few operating systems support even one of the above prerequisites (indeed, standard UNIX systems support none), and those which do have, for the most part, been written specifically either for supercomputer class machines, or for transaction or real-time processing, where loss of either CPU cycles or the work performed could have catastrophic results. Furthermore, the present lack of adequate debugging tools in a supercomputing environment simply compounds the loss of this irrecoverable resource.

There are a number of reasons for tying the notion of debugging for performance to conventional debugging (for correctness). In the area of supercomputing, achieving an expected level of performance is important due to the high cost of machine time. An application that is correct, but not performing to expectations, can be considered to have performance bugs. Both types of debugging can utilize common mechanisms and functionality for monitoring and controlling the execution of an application, and for correlating the state of the running program with source code. Furthermore, the user can benefit from a common language and interface for these activities, especially since performance debugging is typically done right after (or in conjunction with) correctness debugging.

Though the issue at hand is the problem of debugging an application (or portion thereof) which is running on a supercomputer, the solution cannot be limited only to the scope of the supercomputer system. A supercomputing environment is comprised of a diverse mixture of systems, from glass ttys to graphic workstations to supercomputers. The desired solution is one which works as well in one section of the environment as another, but in turn makes use of the superior functionality, of say a workstation, when it is available. Further, though there is a growing class of applications which utilize distributed computational resources, and the ability to debug such applications is of paramount importance.

Indeed, future supercomputing environments will require not only distributed debuggers, but the need to debug distributed applications. Rather than attempt to solve this complex and elusive problem here, we prefer to focus on laying the groundwork for addressing the immediate needs for debugging in a non-distributed supercomputing environment first, while assuring that the immediate solution does not inhibit or preclude its extension and enhancement into a distributed environment.

V.B Desirable Features for Debugging Supercomputer Applications

With respect to supercomputer operating systems, a number of desirable features have been identified. One of the primary features is that the debugging system must be applicable to jobs running in a batch mode or an interactive mode. Supercomputer operating systems must be designed so that debuggers or performance monitors can be as effectively utilized for batch processes as for interactive processes. It is not uncommon to run a long job on a supercomputer and have it accumulate hours or days of CPU time before it encounters a bug. Presently, the operating system responds to such an exception by creating a dump of the process (in UNIX, a core file). Supercomputer operating systems should assure that sufficient process context is saved so that it is possible to recover the process and, if desired, fix the exception and continue the execution of the process as if the exception did not occur. The implication is that the operating system supports a robust checkpoint/recovery mechanism, and that the debugging capabilities be tied to that mechanism.

With respect to performance issues, there is a need for standardization of the various metrics for measuring performance. Some architectures have hardware support for, say, measuring the number of instructions issued per unit time, some have software support. Supercomputing operating systems and support libraries should support the collection and user interrogation of information relating to measurement of performance. These capabilities could then be utilized by debuggers or user-level performance monitoring tools to better characterize not only the process itself, but the environment in which the process is running and the effects of each on the other.

When debugging applications for a multi-processor supercomputer, the ability to stop all running tasks simultaneously is critically important. When an application is actually using multiple processors in parallel and any one task encounters either a break point or exception, the latency in getting all other associated processors stopped is of critical importance.

When stopped more quickly, the state of the job reflected in the debugger more closely resembles the state of the job at the time of the exception, and the more likely it is that the user can find timing bugs. From this point, it should be possible to restart only one task while keeping the others stopped. When the user is ready to let the job resume, it is equally important that all tasks be started as close to the same time as possible, for reasons similar to those for getting them all to stop at the same time, e.g., for consistent timing.

The operating system must support the most expeditious method of stopping and starting all processors associated with a multi-tasked application. In fact, we do not believe that a software solution alone provides adequate capabilities for nearly simultaneous execution control and encourage hardware vendors to provide hardware support for operating system clustering of physical CPUs and control over instruction issue.

Regarding performance debugging, typical facilities such as total CPU time, I/O and page fault counters, and so forth, are often inadequate for understanding why a particular code performs the way it does, especially when parallel processing and/or multi-programming come into play. It would be desirable to have the ability to collect traces of system events, time-stamped with a high-resolution clock (again, hardware comes into play). From the user's point of view, event and statistical data collection must be performed on a per task basis. Furthermore, traces of system-wide events and activity must be user accessible in order to correlate the varying effects of system load levels on individual application performance.

Not only should the operating system allow a user to monitor performance (e.g., examine system accounting, system process table, or actual process memory) but it should not preclude multiple users from simultaneously obtaining such information for or from a single process. Such capability is crucial to real-time system administration, system performance monitoring, but also to "guided debugging" (where more than one user, e.g., a scientist and a consultant, may attach to, monitor, debug and control an individual process). More sophisticated tools, including establishment of standards between machine types, are essential. In addition, the operating system should, ideally, support its own debugging and monitoring by reducing the number of time-critical paths to a minimum.

V.C The Current Situation: Problems and Solutions

In a UNIX environment, debugging has traditionally been available in two forms: interactively or post-mortem. The interactive form usually requires starting a new process from a debugger and running that process until it reaches the point of interest. The post-mortem form only allows a “snapshot” view of the process, from a core file, at the time at which a hardware or system exception occurred. Both these forms impose quite restricted capabilities in a supercomputing environment. It is essential to provide the ability to connect to an already existing process, as well as to disconnect from a process and allow it to execute normally. Further, there is a critical need to be able to reinstantiate a process after it has encountered an exception (i.e., to take a “core” file, recreate its process context, debug the process and then, if desired by the user, allow the process to continue as if no exception had occurred).

The `/proc` paradigm in AT&T System V provides much of the functionality being sought for connecting to, initiating, controlling, examining and detaching from process and its memory without inordinate intrusion. Thus the basic capabilities for interactive debugging are available in System V. Extension of these capabilities to include process lineage (parent, sibling and child processes) of the process being debugged, process context (port control, open files, signal handling, etc.) control and some process monitoring (e.g., trapping of signals, system calls, etc.) must be provided. The failing of most systems, though, is in the area of post-mortem process analysis and debugging. As mentioned previously, a robust checkpoint/recovery mechanism is an absolute requirement for a supercomputing operating system, and we view the integration of a debugging capability into that requirement as being as critical. It is our opinion that the core file paradigm be replaced with a checkpoint paradigm (much like that afforded by CTSS and DDT) to allow the examination and debugging (and reinstantiation) of failed processes in the same manner as interactive processes.

To access system state relevant to a process (e.g., file description information), the basic Unix approach would be to poke around in `/dev/kmem`. This is unacceptable due to issues of security, efficiency, and portability. As such, no current debuggers use this approach. Again, `/proc` is seen as a good model for solving these problems. It may need to be extended to get access to more system information, some of which may be machine or version dependent, but at least a well defined “user” interface can be constructed around this model.

One problem that still exists with the /proc model is how it can handle multi-tasking applications on a multi-processor. The problem here really stems from the lack of definitions and standards for models of parallelism. Multi-tasking support is at best weakly defined in Unix systems, and is still evolving. A micro-kernel approach, such as Mach, solves the “definitions” problem for some architectures, but then may provide some solution other than the `/proc` model.

The capability to stop and start all running tasks simultaneously in a parallel application is something that no operating system does well enough on any general purpose multi-processor system. In fact, the operating system may never support this well enough to find some bugs. This is an area where new hardware/architectures may be needed to allow the operating system to adequately solve this problem. This in turn may have ramifications on the model(s) of parallelism that become popular.

For the purposes of analyzing program performance, few tools are available beyond the usual accounting types of information, such as CPU time and I/O request counts. Furthermore, this information is usually obtained in the context of the program through direct system calls, or by using a standard utility to cull through system files, but is not often readily available from debuggers. In parallel environments, one often wants to know specific times at which events occur so that these can be collated with events in other parallel tasks. The `/proc` paradigm, along with library instrumentation, provides the potential for relatively non-intrusively monitoring performance characteristics of a process; where the library tallies the data and a tool (e.g., debugger or process monitor), via /proc, reads the information from process memory and displays the information appropriately.

This kind of detail is best achieved using a tracing approach. Some of the events of interest occur within the operating system, so some kind of interface is required to enable tracing, control the level of detail, and make the data accessible. Some experiments have been done to selectively trace context-switch events in two different systems [Lehr 89][Shar 90]. These results have been very encouraging in terms of efficiency and usefulness. But further development is required to integrate system event collection with user event collection, variable monitoring, and debugging facilities. This integration may be difficult because performance measurement techniques usually run in the context of the target program, in order to efficiently trace user events without requiring system services. On the other hand, most feel that debugging is best done using a separate context to monitor and control the target program, with minimal interference.

For purposes of measuring and monitoring the operating system itself, it appears practicable to trace a number of internal events [Tram 90]. This information can be collected in system buffers and periodically read out by a system daemon or written to a communication link to a dedicated monitoring system, such as a workstation. If standard data formats are developed, common tools could be built for analyzing and displaying the data across a variety of hardware systems. Of course, which information is collected can be site specific and dynamically enabled.

References

[Lehr 89] Ted Lehr, David Black, Zary Segall and Dalibor Vrsalovic, "MKM: Mach Kernel Monitor Description, Examples and Measurements," *Technical Report CMU–CS–89–131,* School of Computer Science and Department of Electrical and Computer Engineering, Carnegie Mellon University, March 1989.

[Shar 90] Sanjay Sharma and Allen Malony, "The Tracing Facility in the Cedar System," *CSRD Report No. 1015*, University of Illinois at Urbana–Champaign, Center for Supercomputing Research and Development, June 1990.

[Tram 90] R. D. Trammel, "The Big Picture: Visualizing System Behavior in Real Time," *Usenix Summer Conference Proceedings*, Anaheim, CA, June 1990, pp 257–266.

VI. Models of Parallelism

VI.A Current Models of Parallelism

The supercomputer environment of today is dominated by two models of parallelism, namely,

1. Systems composed of small numbers of extremely powerful processors;
2. Systems composed of large numbers of cost-effective processors.

We examine these two types of systems. Systems composed of small numbers of extremely powerful processors are typified by the multiprocessor Cray Research machines, with less than 20 Central Processing Units. Each CPU is a previous generation supercomputer in its own right, and the composition of these processing units into a multiprocessor system with shared memory results in an expensive configuration with impressive performance. The processing units can be employed on a single parallel job or multiple job streams, and considerable operating system mechanism is employed to keep each CPU as active as possible. In either case, OS support must be present for multithreaded use of the processors. Advanced compiler technology, typically FORTRAN, is employed to exploit detectable parallelism in the program source modules. The compiler run-time support then employs the multithreaded operating system capabilities to map the program's parallelism on to the available hardware. The ability of this model to exploit parallelism is then tied to two factors: first, what parallelism the compiler technology can discover in the programs, and second, what level of parallelism can be obtained from multiple job streams. The evidence seems to point to an evolutionary pace (4–>8–>16–>64) as the number of processors slowly increases, rather than a revolutionary pace, where huge numbers of high-performance processors are used.

The second model of parallelism is the "massively parallel" school, where very large (e.g., 65536) numbers of processors comprise a supercomputer system. Rather than combine a small number of very high performance (and very high-priced) processors, a large number of lower-performance processors are combined. The idea is that processor technology can be chosen from a market segment where the economics of ''commodity'' processors apply. Large numbers of commodity-priced processors are combined into a computing system, where it is hoped that both price/performance and absolute performance advantages accrue. A prominent example of this style of supercomputing are hypercube connected machines which use commodity-class microprocessors in large configurations. These machines have very little in the way of operating system support, and instead rely more on powerful machine-dependent run-time systems associated with the higher level language compilers for the machines. In practice, these machines seem most suited to running single job-streams with highly parallel component programs, such as fluid-flow analyses and other problems where the solution methods do not disguise the physical parallelism inherent in the problem. Multiple job streams are very difficult, as (1) the dynamic processor allocation and deallocation, and associated policy decisions, requires more performance than the component processors can provide, and (2) the memory architecture (non-shared) makes system management difficult, as communication can be expensive and thus treating a block of processors as an aggregate is difficult to do dynamically.

While these two types of parallel processors hardly provide a complete taxonomy, they do allow us to get at the issues of parallelism, and its management by supercomputer operating systems. In addition, they provide a practical characterization of the current environment.

VI.B The Desired Environment

The desired environment is a ''computing system'' which provides extremely high performance on a large variety of problems. Since most problem solutions are not simply characterized, and may comprise several phases, the components of the computing system should be managed to give maximum end-to-end performance. For many problem programs, this means that different supercomputers should be employed as the need for parallelism expands and contracts. Operating systems and language run-time systems should cooperate to identify phase changes in program execution, and to identify resource requirements associated with these phases, so that problem programs are appropriately placed in the supercomputing system. High-speed networks of one type or another (as were discussed in the workshop and are reported in section III) will interconnect the components of the supercomputing systems, and connect them to various display technologies.

Supercomputing operating systems will then:

1. Make policy decisions for allocating components of parallel supercomputers,
2. Dynamically map parallel workloads to heterogeneous parallel hardware,
3. Manage disparate I/O devices and high-speed network interconnects,
4. Require more intensive compiler analysis and run-time system support.

VI.C Problems

The common denominator of mechanism we agreed on at the workshop was a thread. While a precise definition of a thread was not available, the term was understood to represent a schedulable flow of control, e.g., an instruction stream. Typical sequential programs are characterized by a single flow of control, while parallel computations are characterized by multiple flows of control. Traditionally, Operating Systems have assigned each flow of control to a distinct address space, and made this the schedulable unit of computation, the process. Associated with each process is a great deal of state, such as file control blocks, buffers, and custom exception–handling facilities. On machines which support virtual memory, the state includes support for private address spaces. This state is used to provide the illusion that the process has a private ‘‘virtual machine’’ with which it can process its instruction stream.

However, it is clear that this process notion leads to a great deal of overhead for supporting each process. the overhead is most burdensome when:

1. there are many flows of control, i.e., a high degree of parallelism,
2. the flows communicate frequently,
3. the ‘‘virtual machine’’ state, such as i/o buffers, is logically shared by the flows of control.

Unfortunately, many of the problems which supercomputers are best suited for are high–overhead cases, and thus the traditional process model can present a problem. The thread model separates flows of control from virtual–machine support, typically by supporting multiple threads which are associated with a single address space. Certain support features must continue to exist, such as independent stacks, but in general the state required for a thread is little more than a register set and a private stack. Thread models are available in a number of operating systems, including Mach and Chorus. The operating system architectures are known as micro–kernel architectures, due to their structure. The structure builds operating system facilities used by programmers, such as file servers, using low–level primitives for scheduling and communication. Scheduling and resource management are provided at the level of threads. Workshop participants provided some new insight into scheduling strategies which might be applicable in these systems. One difficulty which became apparent as we tried to understand the applicability of these research systems was the lack of support for either variety of parallel supercomputer. Two reasons were posited by the group:

1. To the best of our knowledge, Cray-style machines were not available to the University OS researchers.
2. Cray-style machines did not provide virtual memory, which was widely used to support interprocess communication and other facilities in at least one of the micro kernel architectures (Mach).

This suggests that among the problems to be solved are:

1. Precise communication from the compiler or run-time system to the operating system of the current requirements of a computation
2. Operating system mechanisms which can map most of these requirements onto appropriate components, in an efficient way.
3. Policy modules which are adaptive and efficient enough so that they use the multi-supercomputer system effectively rather than consuming it computing policy. Policy modules may want to become more autonomous with increasing parallelism, decreasing communications capability, or decreased single-processor power.

Research towards a solution would identify:

- A reasonable primitive for parallel scheduling, e.g., a thread; such a primitive should make minimal assumptions about other available mechanism, so that it could work in a wide variety of environments, such as real-memory environments.
- Methods for making this primitive visible to programmers, and available to run-time systems for parallel compilers.
- Language constructs which would permit exploitation of the thread model, e.g., schedulable chunks of code, with marked dependencies, so that finer granularity could be exploited when more parallelism is available.
- Solutions to heterogeneous supercomputing, especially where data representation issues (such as floating point formats) create problems.
- Flexible policy mechanisms that could exploit varying availability of processor types and components. This would have the additional benefit of providing fault tolerance, as a failed processor would merely reduce availability of a certain component type in an advanced supercomputer system.

Research in other areas which might be exploited includes work in distributed operating systems such as process migration and network shared memory. Work in networking, such as protocol conversion, external data representations, and inter-networking might also be exploited.

Appendix A. Attendance List

Bob Aiken aiken@es.net
SCS, U.S. DOE, ER–7, GTN
Washington, D.C. 20585
(301) 353–5800

Tom Bishop bishop@ui.org
UNIX International
20 Waterview Blvd
Parsippany, NJ 07054
(201) 263–8400 x235

David Black dlb@osf.org
Open Software Foundation Research Institute
11 Cambridge Center
Cambridge, MA 02142
(617) 621–8745

Allan Bricker allan@chorus.fr
Chorus Systemes
6, Avenue Gustave–Eiffel
78182Saint–Quentin–En–Yvelines CEDEX
France

Edward Chow chow@logos.jpl.nasa.gov
Jet Propulsion Laboratory
MS 198–231, 4800 Oak Grove Drive
Pasadena, CA 91109

Perry Emrath emrath@uiuc.edu
University of Illinois
104 S Wright – 305 Talbot
Urbana, IL 61801
(219) 333–4354

Tom Engel engel@craycos.com
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80935

Tom Faulkner
Cray Computer Corporation
Lawrence Livermore National Laboratory
Livermore, CA 94550

Dieter Fuss fuss@lll–crg.llnl.gov
Lawrence Livermore National Laboratory
P. O. Box 808, L–669
Livermore, CA 94550
(415) 422–4027

Peter S. Ford peter@lanl.gov
Los Alamos National Laboratory
MS–B258
Los Alamos, NM 87545

Michael K. Gleicher gleicher@ddnvxl.aswl.af.mil
Air Force Weapons Laboratory
WL/SCS
Kirtland AFB
Albuquerque, NM 87117–6008

Bruce Griffing griffing@ccc.nersc.gov
Lawrence Livermore National Laboratory
P. O. Box 808, L–561
Livermore, CA 94550
(415) 422–4498

Dave Grubb dgrubb@lance.tis.llnl.gov
Lawrence Livermore National Laboratory
P. O. Box 808, L–631
Livermore, CA 94550
(415) 423–4745

Malcolm Hammerton mah@craycos.com
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80906

John Hartman jhh@sprite.berkeley.edu
U.C. Berkeley
571 Evans Hall, CS Division
Berkeley, CA 94720

Roger Hayes r_hayes@eng.sun.com
MTV 29–112
Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043

Morris Jette jette%NERSC@ccc.nersc.gov
Lawrence Livermore National Laboratory
P. O. Box 808, L–561
Livermore, CA 94550
(415) 423–4856

Dan Kopetzky djk@super.org
Supercomputing Research Center
17100 Science Drive
Bowie, MD 20715–4300
(301) 805–7474

Ron Larson lars@craycos.com
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80935

Paul Leskar pal@craycos.com
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80935

Jon Masamitsu lll–lcc!ssiwest!masa
Supercomputer Systems
2021 Las Positas Ct., #101
Livermore, CA 94550

Karl J. Melendez kjm@lanl.gov
Los Alamos National Laboratory
MS B294
Los Alamos, NM 87545

Ron Minnich rminnich@super.org
Supercomputing Research Center
17100 Science Drive
Bowie, MD 20715–4300

Dennis Moen
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80935

Reagan Moore moore@sds.sdsc.edu
San Diego Supercomputer Center
P. O. Box 85608
San Diego, CA 92186–9784

Dan Nessett nessett@llnl.gov
Lawrence Livermore National
P. O. Box 808, L–60
Livermore, CA 94550
(415) 422–4033

Bernie O'Lear olear@ncar.ucar.edu
National Center for Atmospheric Research
P. O. Box 3000
Boulder, CO 80307
(203) 497–1268

Dan Poznanovic dsp@cray.com
Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55121

John P. Riganati riganati@super.org
Supercomputing Research Center
17100 Science Drive
Bowie, MD 20715–4300
(301) 805–7450

Gene Schumucher gene@ncar.ucar.edu
National Center for Atmospheric Research
P. O. Box 3000
Boulder, CO 80307

Karen Sheaffer karen@snll–arpagw.llnl.gov
Sandia National Laboratories
Div 2913
P. O. Box 969
Livermore, CA 94550
(415) 294–3431

Jonathan M. Smith jms@cis.upenn.edu
University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104–6389
(215) 898–9509

John Sopka sopka@syzygy.dec.com
Digital Equipment Corporation
TAY 1–2 / H03
151 Taylor Street
Littleton, MA 01460
(508) 952–4347

Eric Varsanyi ewv@craycos.com
Cray Computer Corporation
P. O. Box 17500
Colorado Springs, CO 80935

Richard Watson watson@ocfmail.ocf.llnl.gov
Lawrence Livermore National Laboratory
P. O. Box 808, L–631
Livermore, CA 94550
(415) 422–9216

Tom Watson watson@convex.com
Convex Computer Corporation
154 Classen
Dallas, TX 75218
(214) 497–4338

Appendix B. Supercomputing '89 BOF Session