4

PARAS Microkernel at a Glance

Abstract

High performance computing requires the use of massively parallel processing systems containing thousands of powerful CPUs. Every node in these parallel systems requires an operating system (radically different from current ones) to manage its resources. To meet this requirement, we have developed an OS kernel called *PARAS Microkernel* for the PARAM family of parallel systems (being developed by C-DAC) based on state-of-the-art microkernel technology. It supports most essential services such as process, memory, and interprocess communication management. In the design of the PARAS microkernel, we have adopted the philosophy of migrating traditional operating system services into user-level processes. It removes the concept of using application ignorant systems and allows the user to build *subsystems* suitable to the application. Various subsystems such as CORE, PVM, and MPI layered on top of the PARAS microkernel have demonstrated the *simplicity* and *extensibility* features of this design approach.

Keywords: Microkernel, MPP, PARAM, PARAS, IPC, subsystems.

4.1 Introduction

It is now clear that silicon based processor chips are reaching their physical limits in processing speed, as they are constrained by the speed of electricity, light, and certain thermodynamic laws. A viable solution to overcome this limitation is to connect multiple processors working in coordination with each other to solve grand challenge problems. Hence, high performance computing requires the use of **M**assively **P**arallel **P**rocessing (MPP) systems containing thousands of powerful CPUs. To perform well, these parallel systems require an operating system radically different from current ones. Most researchers in the field of operating systems have found that these new operating systems will have to be much smaller than traditional ones to achieve the efficiency and flexibility needed. The solution appears to be to have a new kind of OS that is effectively a compromise between having no OS at all and having a large monolithic OS that does many things that are not needed. At the heart of this approach is a tiny operating system core called a *microkernel.*

A *microkernel* provides the foundation for modular and portable extensions. Every next-generation operating system will have one [1]. In theory, a microkernel with a small privileged core surrounded by user-mode services, would deliver unprecedented modularity, flexibility, and portability. The appendage *micro* suggests that the kernel provides only minimal functionality that allows user-level processes called *subsystems* to perform operating system services efficiently. Thus, it aims at migrating traditional operating system services out of the monolithic operating system kernel into user-level processes. This division of labor allows the microkernel to be small and fast and does not burden each CPU with facilities, such as a complete file system, that it does not need [2].

The *subsystems* running on top of the microkernel implement their own policies which can manage the resources better than the application ignorant general-purpose conventional OS mechanism [3]. This in turn leads to a simplified OS base, improved reliability, portability, and extensibility. Some of the dominant representatives of this new generation OS technology are QNX [1], Amoeba [2], Mach [4], and Chorus [5].

The remainder of this paper presents the PARAM 9000 architecture, PARAS microkernel and its architecture, and PARAS operating environment. It also presents communication performance of PARAS microkernel on PARAM 9000 and brings out comparative analysis with traditional network protocols under Solaris 2.x on PARAM 9000.

4.2 PARAM 9000 Machine Architecture

PARAM (**Para**llel **M**achine) is a distributed memory, message passing parallel computer. The PARAM family of supercomputers include PARAM 8000, PARAM 8600, and PARAM 9000. PARAM 8000 is based on the INMOS's transputers and PARAM 8600 is based on the Intel's i860 RISC processors. Current implementation of PARAM 9000 is based on the SUN's SPARC processors. The OpenFrame architecture of C-DAC (adopted in building PARAM 9000), designed for teraflop performance, provides seamless scalable computing power from a single interactive workstation to a cluster of workstations to the grand challenge systems [6].

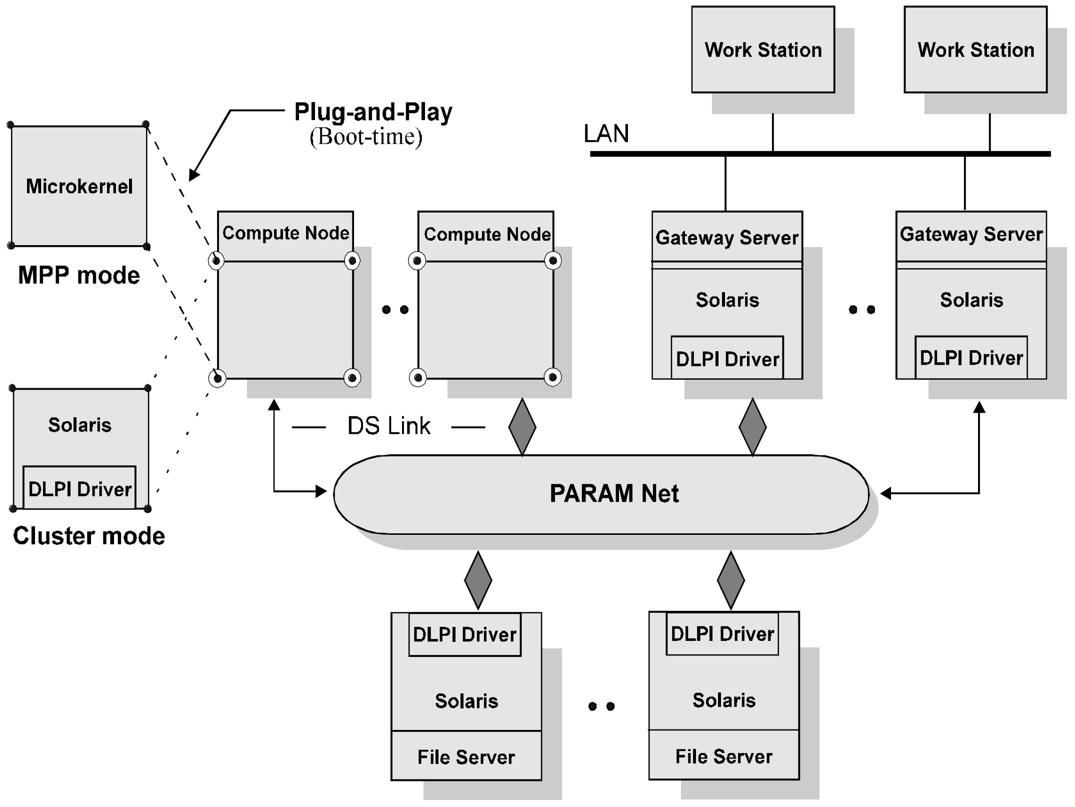

Figure 4.1: PARAM 9000 as Dual Personality System - MPP and Cluster

The battle between MPP and Cluster computing began with the emergence of powerful workstations in the 90's and the possibility of connecting them through high speed networks. This is the motivation for unification of both MPP and cluster personality within a single framework in PARAM 9000. The architecture of PARAM 9000 exhibiting dual personality is shown in Figure 4.1. The PARAM communication Network (PARAM-Net) connects all processing elements and allows smooth transition from Cluster to MPP computing. It offers a high speed, low latency, and packet switched network based on wormhole routing algorithm [7].

The PARAM 9000 hardware architecture allows the nodes to be configured as *compute* nodes or *service* nodes. The *compute* nodes can only execute user jobs, while the *service* nodes in addition provide various services required to support the execution of user jobs. The services provided include file services, gateway services for external connectivity, etc. The service nodes providing gateway services (gateway nodes) support external connectivity through Ethernet, FDDI, or token ring.

In both the personalities (MPP and Cluster), the basic hardware of PARAM 9000 remains the same. These personalities differ only in terms of operating environment and their configuration in terms of software. In the cluster personality, all nodes of the system are configured with the industry standard Solaris OS from SunSoft. In the alternate personality, few nodes (service nodes) of the system run Solaris OS while the others (compute nodes) are configured with PARAS microkernel and system servers. This allows *plug-and-play* of the needed personality; during boot-time PARAM can be configured to support either the MPP or the Cluster personality. The use of a custom microkernel provides the required flexibility to extract maximum performance from the system.

Cluster Personality

The system exports the Solaris 2.x operating system on all the nodes and the PARAM interconnection network supports a **D**ata **L**ink **P**rovider **I**nterface (DLPI) driver which implements ISO Data Link Standard Definition and Logical Link Service Definition. The DLPI support allows mapping of common network protocols such as TCP/IP to the underlying network PARAMNet. Thus the system provides an environment which is functionally equivalent to a cluster of Solaris workstations over a LAN. Much of the software available under a distributed Solaris environment becomes readily available on PARAM also.

MPP Personality

The PARAM 9000 OS architecture allows the microkernel to be loaded on the compute nodes instead of Solaris. The communication subsystems over the microkernel have been highly optimized providing an order-of-magnitude improvement in performance. Custom designed **Co**ncurrent **R**untime **E**nvironment CORE [10] and standard message passing interfaces **P**arallel **V**irtual **M**achine (PVM) [11] and **M**essage **P**assing **I**nterface (MPI) [12] are supported for parallel programming.

4.3 PARAS Microkernel Architecture

PARAS microkernel is a message passing, multithreaded operating system kernel designed for high performance massively parallel processing systems. It supports multiple tasks with a paged virtual memory space, multiple threads of execution within each task and message based interprocess communication. Message passing is the primary means of communication among tasks, and to some extent between the tasks and the kernel itself. Location independent interprocess communication is supported through the port abstraction, with support for both synchronous and asynchronous modes of communication. The microkernel integrates the following essential services:

- An executive — derived from the Mach microkernel [9] which preemptively schedules priority-based threads.
- A location transparent interprocess communication mechanism.
- A simple virtual memory model.
- A low level hardware supervisor which dispatches interrupts and traps.

Microkernel Architecture

The architecture of the PARAS microkernel is shown in Figure 4.2. All the client requests are notified to the PARAS microkernel by using system calls. These system calls are accessible both to subsystems and applications. The PARAS service request *dispatcher* will in turn route all requests to the appropriate service providers, which are built in the form of *resource managers*. They are Process Manager, Virtual Memory Manager, and Inter-Process Communication (IPC) Manager.

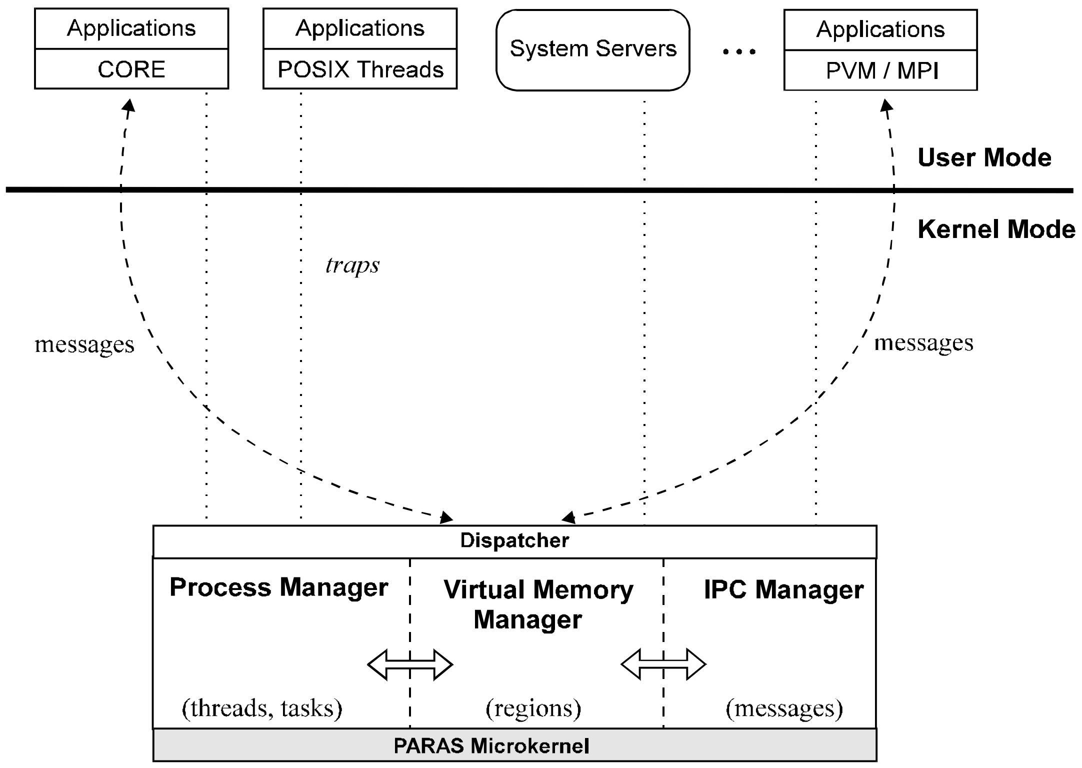

Figure 4.2: PARAS Microkernel Functional Block Diagram

The *process manager* is responsible for handling issues related to task/thread creation, scheduling, dispatching, control, resource handling, etc. The *memory manager* is responsible for serving user requests for memory. It handles memory allocation/deallocation management and maps user addresses (logical) to the machine (physical) addresses. The *IPC manager* in cooperation with system servers, provides port abstraction for communication among the tasks. These ports can be created, named, located, and deleted.

Abstractions

The microkernel supports [8] five basic programmer visible abstractions: *tasks, threads, ports, messages, and regions*. The first and second abstractions (*tasks* and *threads*) arise from separating the traditional notion of a process into two subconcepts. *Tasks* contain the resources associated with a process such as the address space, file descriptors, and port access capabilities. Tasks do not perform computations themselves, but serve as a framework in which threads can operate. A *thread* is the control unit most basic to CPU utilization, containing the minimal processing state associated with a computation: a program counter, a stack pointer, and other hardware register state information.

In terms of these concepts, a UNIX process corresponds to a PARAS task containing a single thread. A PARAS task can contain multiple threads, but each thread is associated with exactly one task. Since each thread may access all its associated task's resources, including shared memory, the PARAS design naturally supports parallel-programming techniques. Multiple threads within a task can execute in parallel, yet incur slightly more overhead than a single UNIX process. Hence, *threads* are generally termed *lightweight processes*.

The third of the five fundamental abstractions is a *port*. A PARAS port is a queue of messages protected by the kernel. All message passing operations within PARAS make references to a port as read or write destinations.

Tasks and threads can have an *exception port* associated with them. The thread's exception port is the port to which the kernel sends messages indicating an exception in the thread. Exceptions raised by the microkernel include illegal memory accesses, protection violations, arithmetic exceptions, etc. The kernel sends exception messages to the task's exception port if the thread causing the exception has no exception port registered. If neither the task nor the thread have exception ports registered, the thread encountering the exception is terminated.

A *message* is a stream of bytes used to communicate between threads. It contains the data to be communicated.

Each task has a large virtual address space within which its threads execute. The virtual address range may be sparse, with ranges of addresses which are not allocated, followed by ranges of addresses that are allocated. A *region* of an address space is that memory associated with a continuous range of addresses; that is, a start address and an end address. *Regions* consist of pages which may have different protection attributes, enabling a task to protect pages in its address space to allow/prevent access to that memory. Regions are not swapped, which means that tasks have to be resident in memory to execute. This extremely simple memory model has been chosen for reasons of performance and simplicity. Having a task resident in memory allows faster interprocess communication. Also, a design without swapping makes the kernel simpler, smaller, and more manageable.

Design Issues

The design issues such as portability, data sharing, protection mechanisms, efficiency are achivied in the following ways.

Portability: It refers to the ability of an operating system to run on several machines with different architectures. The PARAS kernel components that are heavily relies on the idiosyncrasies of the underlyimng architecture, which is an impediment to the portability of the operating system are segrigated as machine dependent. This has substantially increased portability and applicability of PARAS microkerenel.

Data Sharing: PARAS microkernel support data sharing for communication and synchronization among the multiple processes runnung on different/same processors under PARAM.

Protection: PARAS kernel support mechanism of protecting shared memory among several processes.

Efficiency: PARAS is desinged to run on multiprocessor environment to support the execution of parallel application. In fact, the PARAS operating system itself is designed to run in parallel; all algorithms are designed to run parallel and data structures are designed to allow to access then in highly parallel way.

4.4 Operating Environment

The operating environment includes the PARAS microkernel running on compute nodes and Solaris 2.x running on service nodes. The power of PARAM 9000 in MPP mode is provided to the user with the help of two sets of servers: one running on the **S**ervice **P**artition (SP) and the other running on the **C**ompute **P**artition (CP). The microkernel as a whole interacts with various other components which are migrated into user level processes. They include server loader, process server, file server, name server, etc., to provide a complete operating environment as shown in Figure 4.3. The process server, partition manager, file server, and name server run on the compute partition. The server loader and a front end server (Param Resource Manager) run on the service partition.

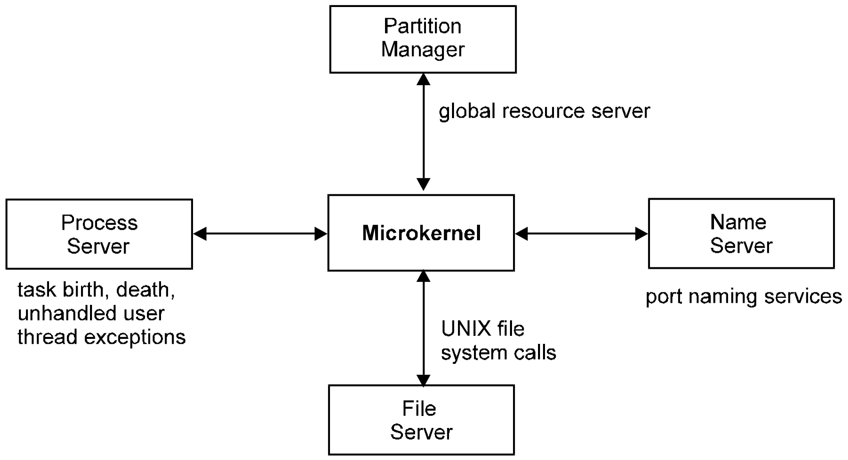

Figure 4.3: Microkernel and System Servers

Since the microkernel runs on the *compute partition*, a front-end server running on the *service partition* is required to *download* user tasks on to the *compute partition* and service its input/output requests.

Partition Manager - PM

Partition Manager serves as a global resource manager for a partition and is responsible for management of that partition's resources. It maintains information about the user tasks, system servers, and memory usage on each of the nodes in the partition. In addition to this it also maintains the system configuration information. Partition manager runs on any one of the compute nodes of a partition.

Process Server - PS

The microkernel interacts with the process server on the birth and death of any task and on any unhandled user thread exception. The PS is responsible for *spawning* tasks on a node and providing a *remote system call* interface using which the user can access the microkernel facilities, from the service partition itself. Process server runs on each and every compute node of a partition.

MicroKernel Filesystem Server - MKFS

The microkernel communicates with the file server (MKFS) in order to serve UNIX file system calls made by user applications. The MKFS services the Solaris file system requests on behalf of the user tasks on the compute nodes. File server runs on each and every compute node of a partition.

Name Server - NS

The microkernel also interacts with the name server for providing port naming services (like naming, locating, etc.) to user applications. It handles all the issues related to naming services and maintaining legacy of communication. Name server runs on any one of the compute nodes of a partition.

4.5 External Interface - Microkernel System Calls

The following are the system calls exported by the PARAS microkernel which can be accessed by subsystem designers and application developers. These system calls provide a consistent interface on all PARAM range of machines. Hence subsystems or applications developed on earlier PARAM machines are portable to newer versions also. For detailed discussion on system calls supported by the PARAS microkernel can be found in later chapters and also in the manual "*PARAS Microkernel interface manual for PARAS 9000*."

Task Related Calls

□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□

Thread Related Calls

□□□□□□□□□□□□
□□□□□□□□□□□□
□□□□□□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□

Memory Management Calls

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□

□□□□□□□□□□□□□

IPC Management Calls

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□

4.6 Communication Performance

Evaluating the performance of a parallel computer working in a scientific and engineering environment is a complex process since, it is a function of many interrelated considerations. The performance depends upon varied number of factors such as the particular application, underlying algorithm(s), the ability of the compiler to optimize the code, the operating system abilities, the system architecture, node processor technology, network speeds, and most predominantly, *communication pattern*. However, on distributed MIMD machines performance measurement mainly concentrates on the communication overhead, which helps in evaluating the overall performance of the systems. The communication performance of the PARAS microkernel and the UDP under Solaris observed on the PARAM 9000 (in MPP and cluster modes respectively) is discussed in the following subsections.

Platform Specifications

PARAM 9000-SS with SuperSPARC processors running at 75 MHz and **C**-DAC **C**ommunication **P**rocessor (CCP) connected to the PARAMNet operating at 50 Mbits/s. The three communication parameters measured are latency, bandwidth, and exchange bandwidth on PARAS microkernel and UDP (**U**ser **D**atagram **P**rotocol) under Solaris 2.4 on PARAM 9000.

Latency

One half of the time taken for the round-trip of a zero byte message is considered as the *latency*. The measured latencies on microkernel and UDP are given in Table 4.1. TCP latencies are typically higher than UDP latencies, due to the nature of the protocol. The latencies on the Microkernel are much lower than that on UDP/Solaris, as expected. The round-trip time increases almost linearly in both cases.

Bandwidth

Bandwidth is measured in the same manner as latency. The message size divided by one half the round trip time, is a measure of the bandwidth. Table 4.2 shows the performance obtained by using Microkernel and UDP. It is found that the Microkernel has a better performance than UDP. It should be noted that the maximum communication bandwidth attainable on the Microkernel is usable by the application, due to the reliability provided by the communication protocol. In contrast, the maximum bandwidth obtainable on Solaris over UDP is not usable by the application, since UDP does not provide reliable communication.

Message Size (Bytes)	Latency in (μs)	
	Microkernel	UDP/Solaris
1	62	325
101	86	365
201	109	390
301	133	410
401	158	455
501	182	495
601	207	500
701	230	545
801	253	555
901	278	625
1001	301	621
2001	542	960

Table 4.1 : Communication Latencies

Message (KB)	Bandwith (KB/s)	
	Microkernel	UDP/Solaris
2	2560.0	2122.3
4	3413.3	2874.4
6	3780.9	3142.7
8	3996.1	3399.2
10	4137.3	3200.0
20	4501.1	3413.3
30	4619.5	3572.1
40	4681.1	3810.2
50	4718.8	4015.7
60	4735.2	4068.9
70	4754.9	4179.5
80	4769.5	4289.6
90	4768.9	4296.6
100	4790.6	4376.1
200	4844.4	4602.2
300	4856.9	4726.1
400	4861.7	4708.0
500	4864.6	4818.8
1000	4880.7	4876.2

Table 4.2: Communication Bandwidth

Message Size (K Bytes)	Exchange Bandwidth (KB/s)	
	Microkernel	UDP/Solaris
1024	9279.4	7137.5

Table 4.3: Exchange Bandwidth

Exchange Bandwidth

Exchange bandwidth is the sum of the throughput that may be obtained when data transfer proceeds in both directions simultaneously, utilizing the bidirectional capability of the communication link. On the Microkernel, send and receive are handled simultaneously, by dedicating one thread for each operation. On Solaris, nonblocking variants of send and receive are used. The *asymptotic* exchange bandwidth for a message size of 1 MB is given in Table 4.3.

Remarks

Communication latency on the Microkernel is less than one-fifth of the latency on Solaris. The maximum throughput obtainable asymptotically, is the same on UDP/Solaris and the Microkernel, and is limited only by the speed of the communication link. It is expected that the asymptotic throughput obtained on the microkernel could be much more than that on Solaris with faster communication links as the copy overhead under the UDP protocols will become more predominant. The maximum *asymptotic* throughput is achieved with a smaller message size on the microkernel when compared to the UDP.

4.7 Conclusions and Future Work

PARAM is being increasingly adopted at numerous institutions worldwide for high performance computing. Many scientific, industrial, business, and medical applications are being deployed over PARAM. Abstractions provided by the PARAS microkernel do not expose the machine's architecture in great detail to the user. Hence, the acceptance of the machine has (will be) significantly increased with the availability of standard parallel programming interfaces layered on top of the microkernel.

The communication performance study has revealed that the PARAS microkernel has less protocol overhead when compared to the UDP/Solaris. The optimization of the PARAS microkernel and other subsystems is under progress, which is expected to further improve overall system performance. The PARAS operating environment needs to be enhanced to support multiuser and timesharing concepts. The future enhancement (see Figure 4.4) of the PARAS microkernel would be to support a fully preemptible kernel which will facilitate real-time extensions.

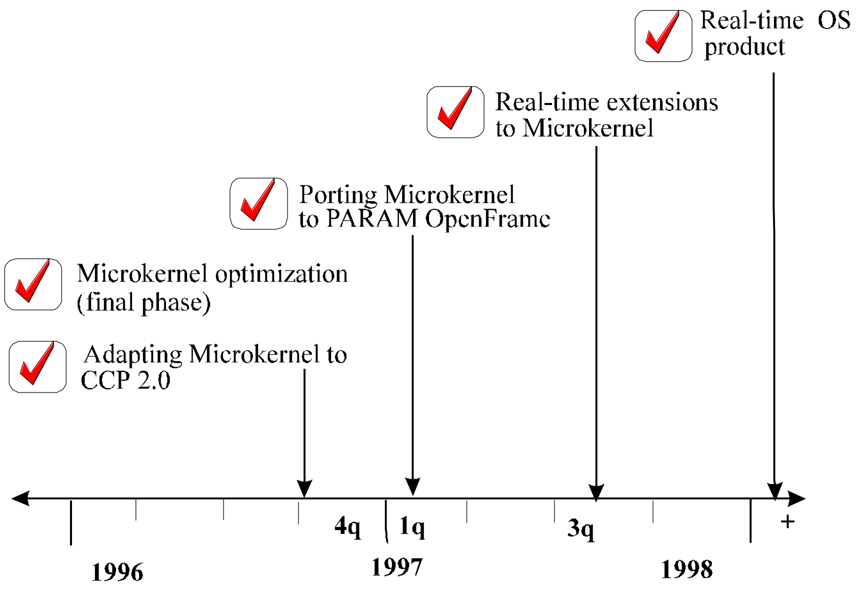
Real-time OS product
Real-time extensions to Microkernel
Porting Microkernel to PARAM OpenFrame
Microkernel optimization (final phase)
Adapting Microkernel to CCP 2.0
4q
1q
3q
+
1996
1997
1998

Figure 4.4: Microkernel Roadmap

References

1. Peter D Varhole, *Small Kernels Hit it Big*, Special Edition, Byte Magazine, January, 1994.
2. Andrew S. Tanenbaum, *A Comparison of Three Microkernels*, The Journal of Supercomputing, Volume 9, Number 1/2, 1995.
3. Rajkumar et al, *PEACE Threads Interface on Microkernel*, The Third Conference on Advanced Computing, Tata McGraw Hill, India, 1995.

4. Michael et al, *Microkernel Modularity with Integrated Kernel Performance*, Operating Systems, Collected Papers, Vol.3, OSF Research Institute, April 1994.
5. Dick Pountain, *The Chorus Microkernel*, Special Edition, Byte Magazine, January, 1994.
6. V P Bhatkar et al, *Philosophy of PARAM 9000*, PARAM 9000 - Towards a National High-Performance Information Infrastructure, (c) C-DAC, 1995.
7. P R Eknath et al, *PARAM 9000 Systems Architecture Overview*, PARAM 9000, (c) C-DAC, 1995.
8. C-DAC, *PARAS Microkernel interface manual for PARAS 9000*, 1996.
9. Mohan Ram N et al, *PARAM 9000 Operating Environment*, PARAM 9000, (c) C-DAC, 1995.
10. C-DAC, *COncurrent Runtime Environment (CORE) manua*l, 1996.
11. V S Sunderam, *PVM: A Framework for Parallel Distributed Computing*, Journal of Concurrency: Practice and Experience, December 1990.
12. David W Walker, *The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers*, Oak Ridge National Laboratory, Oak Ridge, 1993.