45

Computer Structures are Changing: Will UNIX Change with Them?

C. Gordon Bell
Steve Emmerich
Ivor Durham
Daniel P. Siewiorek
Andrew Wilson

Encore Computer Corporation
15 Walnut St.
Wellesley, MA 02181
uucp address: encore!emmerich

ABSTRACT

The UNIX operating system does not adequately support a significant, emerging computer structure: the Multi (symmetric shared memory, multiple microprocessor) computer. This paper offers reasons why the Multi, in particular, is a significant computer structure, and describes some of the reasons that developing and executing applications built to use multiple cooperating processors will require facilities beyond those found in UNIX. We argue (a) that multiprocessor and distributed computers for which UNIX is not currently well suited will dominate in the next decade; (b) that in order to make UNIX and the notion of a popular, non-proprietary OS survive, the UNIX technical community should focus on providing appropriate support for these new computing structures; (c) that progress towards such support should be shared in a forum sponsored by USENIX (called the "Multi- and Distributed UNIX (MAD- UNIX) forum"), so that progress is in the hands of a coalition of sponsors, not in the hands of a single company, in order to avoid the tendency for the standard to become proprietary or to ossify.

1. The Multi -- An Important New Architecture

One important, emerging computer structure is the Multi, or multiple microprocessor computer.

Multiprocessors consist of two or more processors capable of independent instruction execution and able to access programs and memory held in a common, shared memory. Mainframes have been built in multiprocessor configurations with two to four processors, such as the Burroughs B5000 (a dual symmetric multiprocessor), the IBM System 370 and 308X series computers (ranging up to a quad processor). Larger multiprocessors were not successful (a) because switching and cabling cost overhead grew as the product of the number of processors and memories, and (b) because due to program sharing, a multiprocessor with N processors requires faster, more expensive memory than a uniprocessor requires, whose cost exceeds that of N separate, slower memories.

The Multi composed of microprocessors alleviates the cabling/switching cost issue by using a fast, inexpensive and short central switch or "bus" for all communication between processors, memories, and input/output devices (impossible for physically larger mainframes). Advances in memory cacheing technology addresses the memory cost issue by providing a fast memory subsystem using high speed, expensive cache memory and lower speed, cheap main memory, while in many implementations allowing for full data concurrency between all caches in a system.

As a result, the small size, low cost and rapidly increasing performance of microprocessors enable the design and construction of computer structures that offer significant advantages in the design, manufacturing price-performance ratio and reliability over traditional computer families implemented from TTL and ECL semiconductor technologies. Current, commercial Multis typically consist of 2-28 microprocessors, common memories and input/output devices which communicate across a fully shared bus structure. [Multiprocessors with different interconnection schemes between components also exist, but they tend to be specialized for high performance in limited application areas.]

There are several commercial Multi's, built by companies such as Sequent, Arete, Synapse, Intel, and Encore, each with different processor-memory-in-out interconnect, configuration capacity, and cost. As an example, Encore's Multimax is a Multi designed for high- performance processing and high input-output data rates. Up to 20 processors with caches can be plugged into the bus (on ten modules), along with up to 32 megabytes of completely shared main memory. The bus and system architecture permits 1024 processors to address 4 gigabytes of physical and virtual memory, permitting expansion as higher degrees of component integration are achieved. The system's bus, the Nanobus, transmits 100 megabytes per second, sufficient for anticipated increases in speed of CPU chips (and resulting increases in bus loading from memory accesses) for many years. Terminal and workstation access is via one or more local-area networks.

2. Operating Systems on Multi's

The Multi structure is well-suited to high-speed of data connectivity between multiple CPU's (via shared memory), and fast synchronization between processors (via test-and-set type operations, and interrupts). An operating system can achieve high throughput in real-time, timesharing and transaction processing applications, by dynamically distributing jobs among CPU's. In a multiprogramming (timesharing) environment, throughput (the number of jobs completed per unit time) is incrased by dividing processes among the many processors in a way that is transparent to users. In real-time applications, one or more CPU's can be dedicated to device polling and/or interrupt handling as well as data transfer to mass storage, while other processors control the general programming and administrative environment. Since each process will typically spend 25 to 50 percent of its time in the UNIX kernel, UNIX must be adapted to allow multiple processors to be operating in the kernel simultaneously (as AT&T, Sequent, and Masscomp have described at these conferences). For many scientific and engineering applications in which data sets can be partitioned and operated upon in parallel, multiple CPU's can be applied to a single problem simultaneously to achieve speedup, enabling the Multi to get results at the speed of much more expensive computers, but at a fraction of the cost. Likewise, in commercial transaction-processing and database operations, processors can be working in parallel through partitioning of datasets or transaction steps to increase transaction rates and improving response time.

3. Applying Multiple Processors to Speed Up a Single Job

Consider the following idealized scenario on the development of a parallel program. There is a critical application program on a uniprocessor which has become a productivity bottleneck (e.g. VLSI design rule checker, partial differential equation solver, etc.). Dynamic parameters from the uniprocessor version are measured, such as instructions between accesses to key global data. The parameters are utilized in existing performance prediction models to determine the number of processors that can be effectively utilized. With the optimum number of processors as a guideline, the user creates a suitable parallel decomposition for the algorithm, and sets up a simulation with automatic generation of a synthetic workload. The instrumentation, data collection, and data analysis primitives and tools are used to locate bottlenecks and contentions. Once the synthetic workload has been balanced, the programmer replaces each synthetic process by actual code.

Clearly, the desire to exploit concurrency within a particular piece of software increases the complexity of the overall task. In the experience of researchers at CMU, there are two major differences between sequential and parallel programming. The first is the requirement that the speed of data communication and synchronizaton between processors take advantage of the speed of the shared memory between processors, to maximize aggregate performance. The information communicated between processors is comprised of raw data to be processed, plus information to synchronize tasks so that the integrity of the information is maintained. The second difference is the control of concurrent activities. Controlling concurrent activities involves not only scheduling tasks so that the "producers" deliver information in time to keep the "consumers" busy, but also the need to stop and start collections of activities together, and to avert undesirable interactions including both contention and deadlock in the sharing of resources. Both communication and control issues have been addressed in distributed systems interconnected by Local Area Networks. However, the high-overhead solutions for loosely coupoled LAN's are not appropriate for tightly coupled shared memory multiprocessors. Communication and control on multiprocessors has been the subject of study on C.mmp and Cm* at CMU, Ultracomputer at NYU, and TRAC at Texas. Experience shows us that special scheduling algorithms, IPC mechanisms, data abstractions, and tools are needed to assist with parallel program development and execution. (A bibliography on these topics are being distributed at the conference, separate from the Proceedings).

The complexity of developing parallel programs suggests that a "Parallel Programming Environment" needs to be devised, in which each step of design, specification, implementation, instrumentation and debugging are supported by tools.

The design stage should provide various models of computation including proven templates for typical techniques of parallel processing. Some of these are as follows: parallel decomposition (e.g. master-slave, pipeline, asynchronous, synchronous, multiphase, partitioning, and transactions); templates for high availability and fault tolerance (e.g. journaling, shadow processes, duplicate and compare, replicate and majority vote); and performance prediction models (e.g. speed-up prediction as a function of parameters measured on a uniprocessor version of an algorithm). Because performance will be even more sensitive to design in a parallel environment, tools for creating synthetic workloads and simulating multiprocessor contention (both hardware and OS) would be useful.

Tools for defining data abstractions and their functionality, and the synchronization mechanisms for those abstractions, would assist in the specification, implementation, and debugging phases. Libraries of such abstractions could be built up that have well understood cost, performance, and reliability characteristics. Extensions to programming languages should be provided to describe concurrency. Structure editors would be available to assist writing of syntactically correct code, and compiler and run-time checks should be included for semantic checking. The debugging and instrumentation stage should be supported by facilities for managing multiple activities, e.g. starting, stopping, and scheduling of events, as well as notification upon detection of a set of events.

4. How can we Cooperate To Support Distributed and Parallel Processing?

Several vendors have adapted UNIX to multiple processors by allowing multiple threads of execution through the kernel, but this level of support does not address how the aggregate computing cycles of Multi's can speed up the time to perform single applications that are organized as multiple tasks or processes. Research in alternative computer architectures, operating systems, languages, and algorithms is not likely to provide answers in the near future. This is up to us, the UNIX community, to learn and adapt to UNIX technology.

Until now, the motivation for changing UNIX has properly been to make it more fully realize its original intended purpose of being a better, more portable timesharing environment that exploits its underlying hardware, particularly for software development and technical documentation preperation applications. Support for evolutionary hardware developments such as virtual memory and peer-to-peer networking were successfully added, largely as a result of public domain, government-funded efforts at U.C. Berkeley. More recently UNIX has been adopted as a workstation development and execution environment, and as a multi-tasking OS for PC's.

It is ironic that after having been a source of much creative change in OS development over the past 15 years, UNIX(tm) is commercially supported and subject to pressure to be standardized when it is least suitable for supporting the fastest-growing computing styles and most cost-effective hardware structures. Networking, bitmap displays, and multi-processors are the new components of these computing structures and styles which UNIX does not succeed in managing very well. In order to manage these hardware resources properly, UNIX clearly must support integral networking (or better capabilities for integrating network- based systems capabilities into the operating environment), and flexible distributed resource sharing and control between heterogeneous machines. Progress is being made in this direction, notably by SUN and AT&T and numerous universities. More progress could be made towards supporting closer control by database and transaction processing systems over process and task scheduling, and disk access, for improved performance, as these are areas where much research and practical systems building has taken place since the UNIX filesystem was invented. In addition, UNIX must provide suitable constructs for supporting diverse bitmap displays that provide heterogeneous graphics services and windowing facilities. This is an area in which must experimentation is occurring, as it is not very well understood yet how screen objects, operating systems, and languages ought to interact for good performance and flexibility.

Interested members of the USENIX Association could make a difference in addressing distributed and parallel processing needs in UNIX, and in helping to shape a good development and execution environment for distributed and multiprocessors with UNIX, by joining Encore in the founding of the "MAD-UNIX forum." We would like to discuss with you appropriate mechanisms for participants to share ideas, over the network or in person.

It is our belief that if UNIX does not evolve fast enough to support distributed resource sharing across networks, and parallel applications on Multi's, other proprietary OS's will. Given how significant we believe the Multi structure to be, this could mean that UNIX could become AT&T's operating system for their computers, and merely a "compatibility interface" for other successful computers. Though possibly inevitable, the loss of momentum behind UNIX as an OS for the future (rather than the past) would be a loss for all of us who believe that UNIX can continue to be a source of creative, practical ideas in computing.