Balance 8000

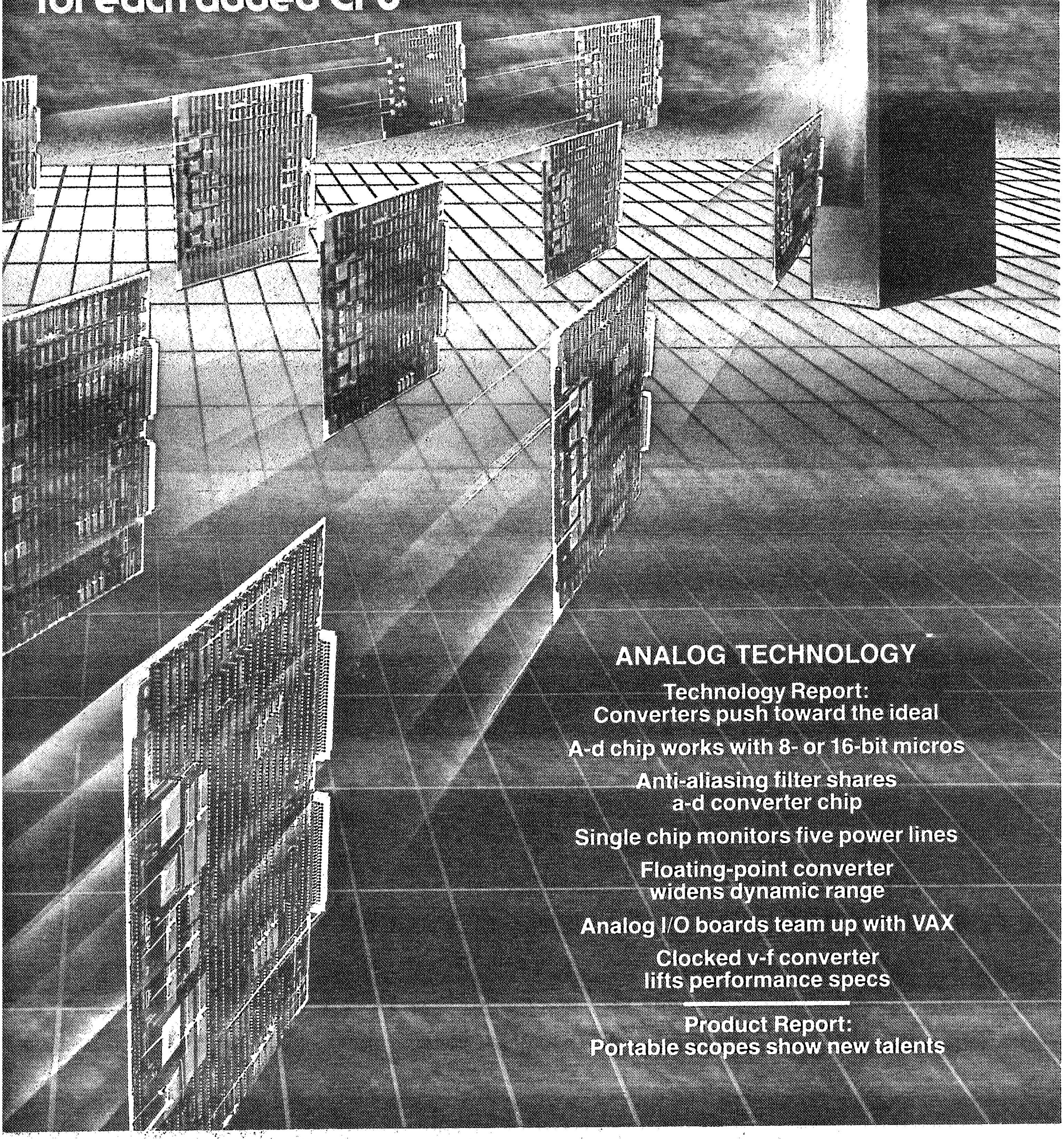
A HAYDEN PUBLICATION

ElectronicDesign®

FOR ENGINEERS AND ENGINEERING MANAGERS — WORLDWIDE

SEPTEMBER 6, 1984

Multiprocessing 32-bit computer dynamically reassigns tasks for each added CPU



ANALOG TECHNOLOGY

Technology Report: Converters push toward the ideal

A-d chip works with 8- or 16-bit micros

Anti-aliasing filter shares a-d converter chip

Single chip monitors five power lines

Floating-point converter widens dynamic range

Analog I/O boards team up with VAX

Clocked v-f converter lifts performance specs

Product Report: Portable scopes show new talents

32-bit computer system shares load equally among up to 12 processors

A multiprocessor computer automatically redistributes the work load and lets the user add more CPUs, employing one modified copy of the Unix operating system.

The performance of microprocessors has steadily increased as the demand for computer power has continued to spiral upward. What would seem to be the most powerful outcome of these trends—a computer that uses multiple processors—has run afoul of the law of diminishing returns in several ways.

Traditionally, it has been necessary to call for extensive changes when more processors are added to a system. Those additions, in turn, invite memory contention and overworked buses and require multiple copies of expensive operating systems. Furthermore, experience has shown that adding a processor to a system multiplies that system's and computing power by a factor of only 0.8, making each successive processor less effective.

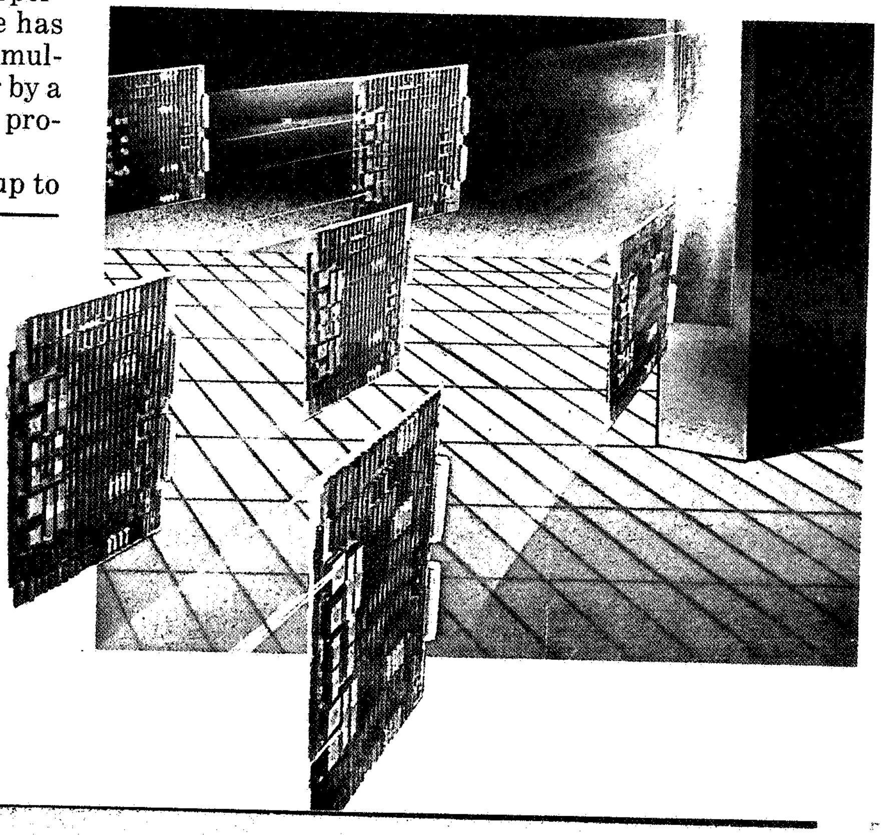
By sharing its processing load among up to 12 architecturally identical microprocessors and employing a single copy of a Unix-based operating system, a new computer eliminates the barriers associated with multiprocessor systems. Called the Balance 8000, it delivers up to 5 million instructions/s (MIPS). Its power grows almost linearly when more processors—32-bit NS32032s plus floating-point and memory management units—are added (see "More is Better, Finally," p. 154).

To make the most efficient use of its multi-

Gary Fielland and **Dave Rodgers**
Sequent Computer Systems Inc.

Gary Fielland, director of advanced development at Sequent, worked at Intel for nine years, mostly in its OEM Systems operation, before joining the Portland, Ore., company. He holds a BSME and a master's degree in electrical engineering and computer science from the University of Florida.

Dave Rodgers is vice president of engineering at Sequent. He worked at Digital Equipment Corp. for 10 years, where he was instrumental in the design of the VAX 11/780. He holds a BSEE from Carnegie-Mellon University.



processing power, the system dynamically balances its load; in other words, it automatically and continuously assigns tasks to run on any processor that is currently idle or busy with a lower-priority task. This process is carried out transparently; neither the user nor the programmer need be aware that the system sports multiple processors.

Easy to extend

At the same time, the computer is easily extensible. The user can add CPUs, memory, and I/O subsystems within a node, or more nodes within a distributed network, or more distributed and local-area networks—all with no changes in software (Fig. 1). Moreover, if the machine's assigned tasks do not run quickly enough, the user can add two more processors by simply plugging in a single circuit board.

The new computer is the first of a family to implement a processor pool architecture (see "Getting Everyone Into the Pool," p. 156). The system consists of a pool of 2 to 12 processors, a high-bandwidth bus, up to 28 Mbytes of primary storage, a diagnostic processor, up to four high-performance I/O channels, and up to four IEEE-796 (Multibus) bus couplers. It is managed by a version of the Unix 4.2 BSD operating system, enhanced to make the multiprocessor base appear invisible to any application program.

Each processor in the pool is a subsystem

More is better, finally

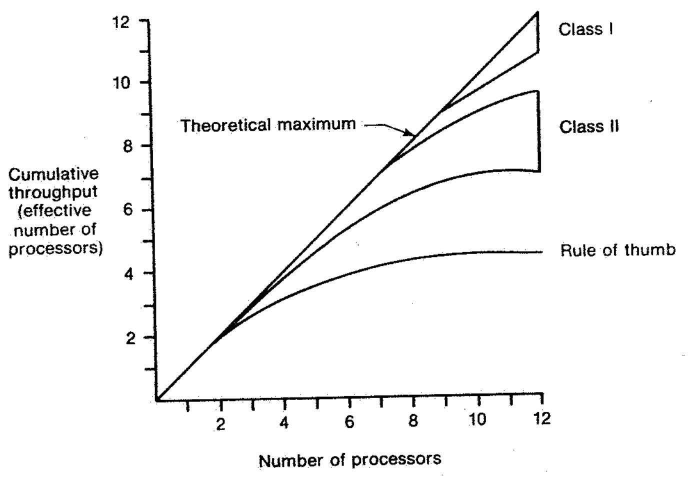
The Balance 8000, with its scalable processor pool architecture, lets OEMs easily set the number of processors to match an application. It does so in spite of a well-known saturation curve reflecting the performance of tightly coupled multiprocessors—a rule of thumb that says that each additional processor adds only 80% of the power of the one previously added. Under this rule, no matter how many CPUs are added, the maximum effective computing power does not surpass the equivalent of only five processors.

Worse, the rule only predicts the effective computing power and not an additional processor's effect on a particular application, which in many ways could actually degrade that program's performance. Therefore, this rule is best taken strictly as a measure of hardware performance.

Two classes of simple benchmarks were used to test the rule against Balance 8000: One class (Class I, in the figure) consists of traditional benchmarks like binary sort and Whetstone programs. The second (Class II) consisted of so-called pathological benchmark programs like cache busters and memory saturation generators to push the machine to its limits. All the tests are concentrated on the computer itself and produce negligible I/O activity.

Overall system performance is N times the ratio t_1/t_2, where t_1 is the run time for executing a single copy of a benchmark and t_2 is the run time for executing N copies simultaneously. The results show how the new computer's performance stacks up against the rule-of-thumb prediction and the theoretical maximum.

Clearly, the cached machine is well suited for running the Class I tests. It achieves a nearly linear improvement in performance (up to eleven effective processors when twelve are applied). With the same number of processors, the Class II tests show a much wider spread, but still yield an effective range of more than six to nine processors. And in all cases, the benchmarks show a much better performance than the rule of thumb predicts.

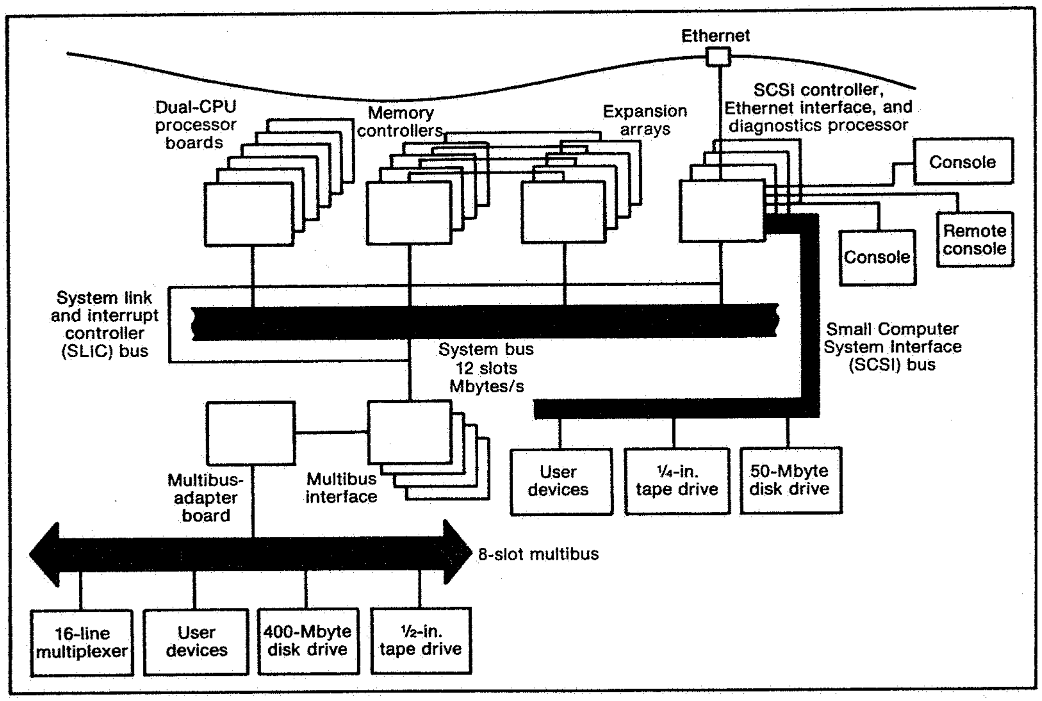


containing three VLSI parts: a 32-bit CPU, a hardware floating-point accelerator, and a paged virtual memory management unit. Two such subsystems are on one circuit card (Fig. 2). The fully 32-bit CPU and the floating-point accelerator suit technical applications, which tend to stress 32-bit operations and address calculations. The paged memory management, which handles a fully associative translation look-aside buffer and a two-level page table, accesses up to 16 Mbytes of virtual memory space for each process.

Each processor also contains a cache memory for near-zero wait states and minimized bus traffic. The two-way set-associative cache consists of 8 kbytes of very high-speed memory and stores recently accessed instructions and data. In this way, subsequent requests for the same data are satisfied from the cache, rather than main memory, to conserve bus and memory bandwidth. The cache takes in 8 bytes, a trade between hit rate and bus traffic that affords an effective hit rate of 95%.

Cache takes brains

Designing a cache for a processor pool architecture is difficult for several reasons. Since data in each cache represents a copy of some data in the primary memory, it is important that all copies and the original remain the same, even when a cache is updated. To ensure that, the new computer employs a write-



1. An extensible multiprocessor computer, the Unix-based Balance 8000 centers on a high-speed 26.7-Mbyte/s bus. Plugged into the bus are one to six dual-NS32032 processor boards, at least one memory controller, and at least one I/O controller.

Cover story: Multiprocessor computer

Getting everyone into the pool

The concept of multiprocessing embraces a full spectrum of structures that range from loosely to tightly coupled (see the figure).

Multicomputers, or specialized-function multiprocessors, encompass I/O controllers or data-acquisition subsystems whose processor boards often include special-purpose hardware. But the term "multiprocessor" may more classically be applied to general-purpose systems consisting of two or more processors of equal capability. This definition, however, covers at least three different structures: coordinated job scheduling, master and slave scheduling, and homogeneous scheduling.

In coordinated job scheduling, a loosely coupled approach, each processor is relatively autonomous. Each has its own interrupt system and storage. Each processor also has its own copy of the operating system and receives its job load from a centralized scheduling-policy manager. Once assigned, a job stays with the same processor until it ends.

One advantage of this approach is that there is relatively little coupling among the processors, simplifying the design. But there are significant disadvantages to this approach, too. First, the processing load is balanced only at the beginning of a job, when a user logs on, making the balance relatively inflexible. For this reason, it is not unusual for several users to experience poor response even when a processor within the group sits idle.

Other short-term resource imbalances are also possible, such as a memory shortage occurring in one processor while a neighboring processor has memory to spare. Also, it is not possible to employ processors concurrently to speed up a single application, as is often needed for technical tasks.

In a master and slave scheduling, a tightly coupled approach, all memory is accessible to all processors, but one processor is distinguished (usually by the software) as a master while all the others are slaves. The master maintains all of the system structures and schedules the work of all the slaves. Slaves, however, are limited to executing only user code while the master handles both user and supervisory code.

In a dual-processor, master-slave, Unix-based implementation at Purdue University, for example, the slave inspects a run queue to get the next process but can take only those marked as being user-mode code. Even then, if a process running on a slave makes a system call, the slave stops the process, marks it for supervisor mode service, and reinserts it on the run queue.

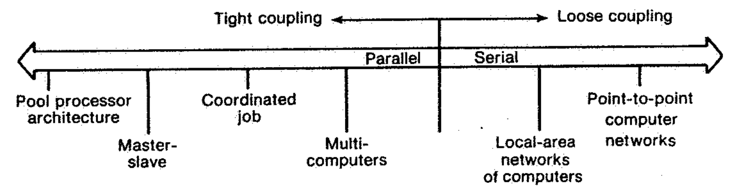
Master and slave systems, like the previous approach, are relatively easy to implement, and they do have the potential for handling parallel-programmed applications. But under heavy system loads, the master becomes a major bottleneck, and this limits the number of processors the system can have.

In the third case, a general multiprocessor system with a homogeneous architecture, all resources (memory, I/O devices, the interrupt system, and so on) are accessible to all processors. Resources are dynamically assigned to processes and not hard-wired rigidly to a processor. In this form of tightly coupled multiprocessing, a process scheduler assigns processors from a pool, earning it the name of processor pool architecture. Also, if the number of processors may be changed at need, it then is known as a scalable processor pool architecture.

The principal drawback of this approach is the difficulty of implementing it. The operating system must be carefully designed to ensure that mutual access is properly synchronized and, where necessary, excluded. In addition, the hardware must be carefully balanced to minimize performance-degrading contention.

The advantages, however, are notable. Since the system is fully symmetric, no single processor can limit the performance of the overall system. Instead, the pool of processors works as a team under all conditions to maximize the system's performance.
Also, the multiprocessing nature of the system is invisible to the user, while its shared resources and tightly coupled nature promote parallel programmed applications, which can be accelerated by adding more processors.

Of course, there are other, more exotic multiprocessor architectures, like data-flow machines, transputers, and inference machines, which promise massive parallel processing power in direct proportion to the number of processors used. While of great academic interest, though, these machines suffer from revolutionary architectures that are incompatible with existing von Neumann-based software.


Tight coupling
Loose coupling
Parallel
Serial
Pool processor architecture
Master-slave
Coordinated job
Multi-computers
Local-area networks of computers
Point-to-point computer networks

through mechanism, in which each write cycle goes through to the bus and memory, in addition to updating the appropriate cache.

In this way, the write-through mechanism keeps the primary memory up to date with the caches on each processor. But what keeps the individual caches consistent among themselves? If two processors have both recently read the same data into their respective caches, and one of them updates its cache, what will keep the second processor from using its now stale data?

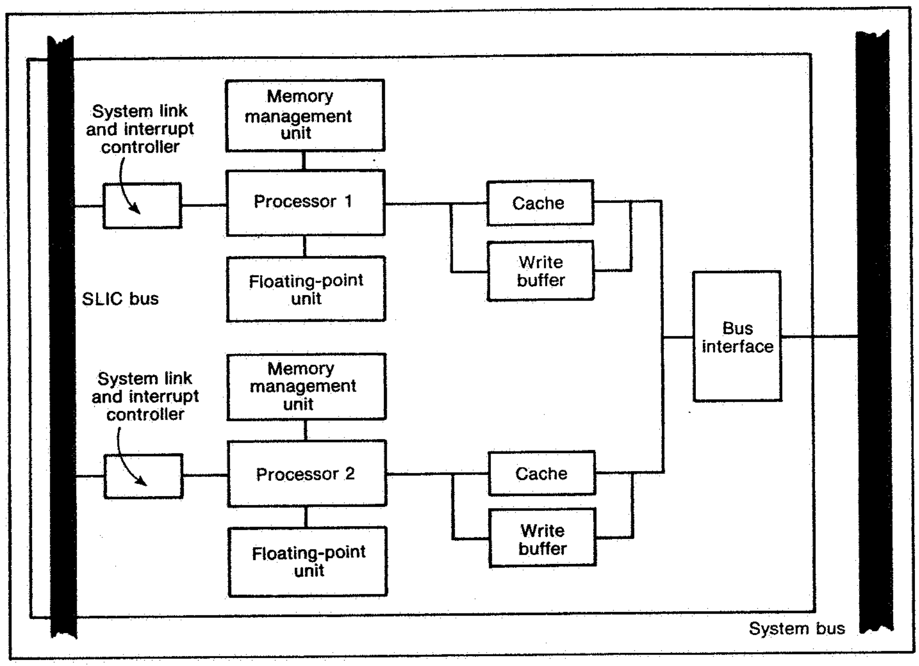
The answer is found in each cache's bus-watching logic (Fig. 3). This logic continuously monitors all write cycles on the bus and compares addresses with those in its own cache to see if any writes affect its own contents. When such an address appears, the cache invalidates the entry in question.

Also, although write cycles typically make up only 10% to 15% of the processor cycles, a processor could still waste precious time waiting for the completion of a write cycle. To avoid needless waiting, a write buffer relieves the processor of the write operation's address and data, letting it proceed while the buffer waits for the memory cycle to complete.

A SLIC chip

The last component of the processor subsystem is a custom IC, the System Link and Interrupt Controller, or SLIC. A SLIC appears with every processor in the system, as well as on every memory controller, I/O channel, and bus controller board. Communication between SLICs is accomplished with a simple command-response packet carried over a dedicated bus.

The controller serves several functions. First, it is the key element of the system's global interrupt system. Device interrupts are broadcast over the controller's bus as a packet. When that happens, the chips arbitrate among



2. Each processor in the pool is a subsystem containing three VLSI components: a 32-bit CPU, a hardware floating-point accelerator, and a paged virtual memory management unit.

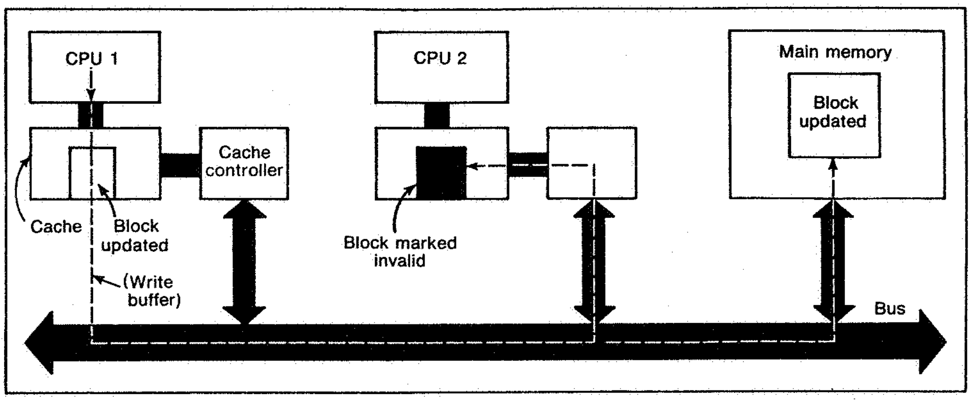
themselves on the basis of the priority of the processes running on their companion processors. The SLIC whose processor is executing the lowest-priority job at the time transforms the packet into an interrupt for its processor. With this mechanism, neither device interrupt signals nor device drivers are bound to particular processors, and only the lowest-priority processes are ever interrupted.

Cache of semaphores

A second function of the controller is to manage a cache of single-bit unit-semaphores. Such semaphores exclude access to a processor; in fact, all high-level exclusion and synchronization facilities involving the operating system are based on this function. Moreover, since each processor's SLIC is on the local bus, access to the chip does not incur any system bus cycles. As a result, spin-locks, which repeatedly request access from a SLIC semaphore, never waste bus or memory cycles.

Finally, the controller serves as a convenient communication path among modules. For example, system diagnostics and debugging routines take modules on and off line using the SLIC bus, which carries error management information. Also, the controllers note the power-up codes of modules and record their presence in an auto-configuration table accessed by the operating system. As a result, when the user plugs in a new circuit board, the system automatically reconfigures itself to include the new board without switches, wire-wrap stakes, or jumpers.

The circuit combines the functions of a serial receiver, a contention resolver, a transmitter, an interrupt controller, a semaphore cache, parallel I/O ports, and a processor interface (Fig. 4). It is implemented using a 3-μm CMOS process with gate-array technology and comes in a pin-grid array. The SLIC bus is a two-wire, bit-serial wired-OR bus that facilitates the distributed self-arbitrating access mechanism. When the bus is idle and a controller has a message to send, the chip simply sends the message



3. Bus-watching logic makes sure that a processor does not read stale data from its own cache. The logic continuously monitors the system bus, checking the addresses that appear during write cycles to see if any data associated with its own cache has been updated elsewhere. If it has, the write-through operation marks the stale data as invalid while also updating main memory.

over the bus. If it collides with another message, the lowest-priority one backs off to try again later.

A bus's burden

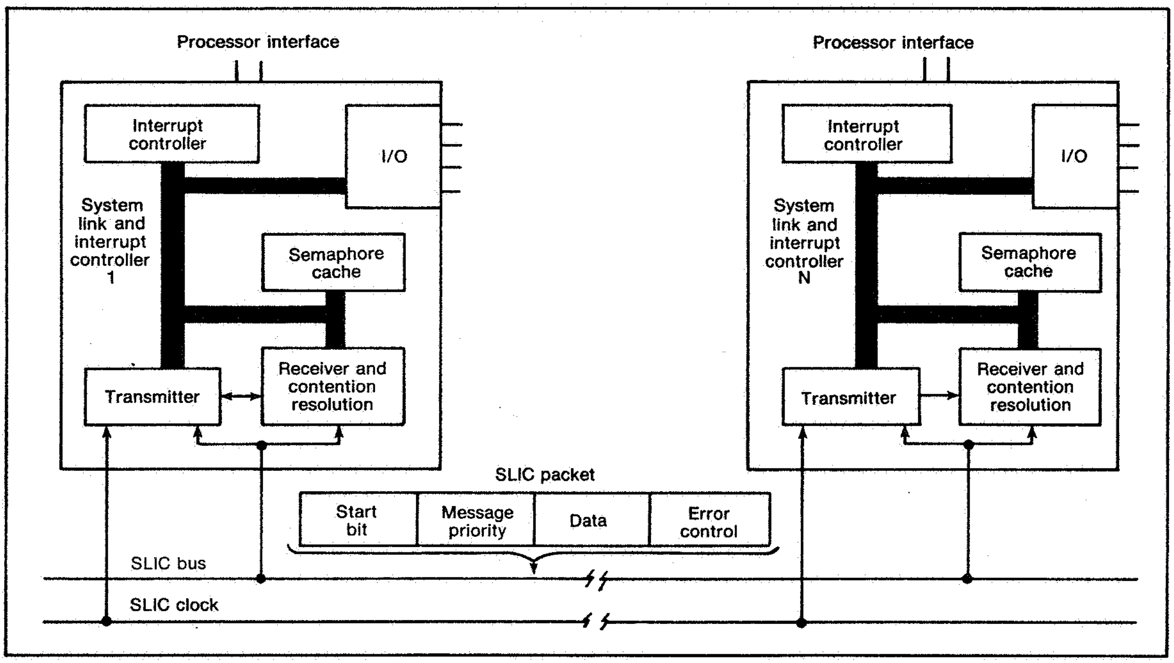
Obviously, the system bus for a scalable processor pool architecture machine is a critical element. It must provide software-transparent, symmetrical access between all processors and the system resources, including I/O subsystems of widely varying access times. And it must do so with careful regard to the high bandwidth required by the 32-bit CPUs. Yet, for the sake of economy, it must also have minimal interface complexity.

The Balance 8000's system bus achieves these goals through a combination of techniques. For example, a single, global 10-MHz synchronous bus that interconnects all processors to all other resources supplies the required symmetry. The bus yields the necessary performance with a 32-bit parallel time-multiplexed address and data path, as well as a set of control paths. The combination of time-multiplexing and Fast TTL circuits helps keep the bus interface circuitry, including transceivers, to fewer than 20 ICs.

A multiple pipeline protocol with multiple 4- or 8-bit fixed-length packets up to 8 bytes long helps to maintain the bus's bandwidth. To spare the primary storage pipelines from the possible long latency of serving relatively slow I/O buses such as Multibus, separate pipelines have been assigned to serve reading, writing, and I/O requests.

As a result, the protocol splits responses from requests so that the bus is only tied up for those cycles needed to transmit the request and response information. The storage access itself causes no bus delays and instead occurs in parallel with the traffic from other requests and responses. Moreover, the write response has its own dedicated set of wires on the bus and so happens "out of band." This frees the main 32-bit data path for more traffic.

As an example of the bus's parallelism, sup-



4. A SLIC (System Link and Interrupt Controller) chip accompanies each processor, memory controller, I/O channel, and bus coupler in the system. It provides a global interruption system, manages the cache semaphores, and establishes a packet-based communication link among the system's modules.

pose that processor P_1 sends a 1-byte request destined for the Multibus (Fig. 5). It takes only one 100-ns cycle to transmit and enqueue the request on the Multibus coupler. Next, processors P_2 and P_3 both transmit 8-byte read requests to the primary storage in the following bus cycles, after which the bus is again free for traffic.

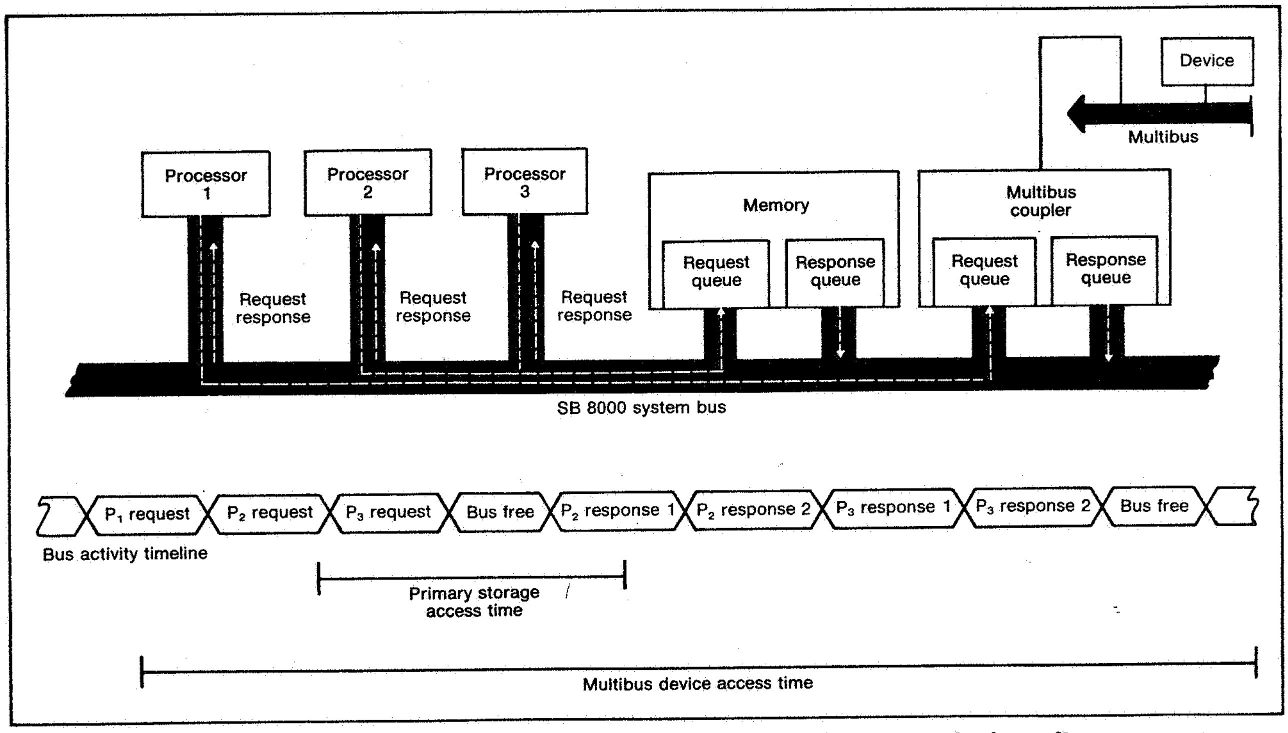
Then, when the 8 bytes of data requested by P_2 are available, the storage controller takes two bus cycles to return those 8 bytes of data to P_2, followed immediately by the two bus cycles needed to return 8 bytes to P_3. The bus is then free again until some time later when, depending on the speed of the addressed Multibus device, the Multibus coupler obtains its data from the device and uses the bus to respond to processor P_1.

Since the bus traffic is decoupled from the main-storage access, the bus can be accessed with the most efficiency. This decoupling, along with the fact that the memory controllers can be interleaved, results in a sustainable bandwidth of 26.7 Mbytes/s out of a theoretical maximum of 40 Mbytes/s.

Resolving problems

Beyond bandwidth considerations are the issues of bus arbitration, congestion, and control of and recovery from errors. To address these problems, the system employs a central multi-level arbiter. It has multiple priority levels to serve mechanical mass storage devices whose performance would suffer if their data transfers were not accepted in real time.

All processors share a given priority level as well, and within that level the arbitration circuit guarantees fairness. This is important because it ensures that, even under a very heavy load, there is no condition that would deprive a



5. One technique for achieving the bus's 26.7-Mbyte/s transfer rate is to release the bus after a memory request without waiting for a response, so that the bus is occupied only for the cycles needed to transmit requests and response information. Requests and responses are queued until the bus is free.

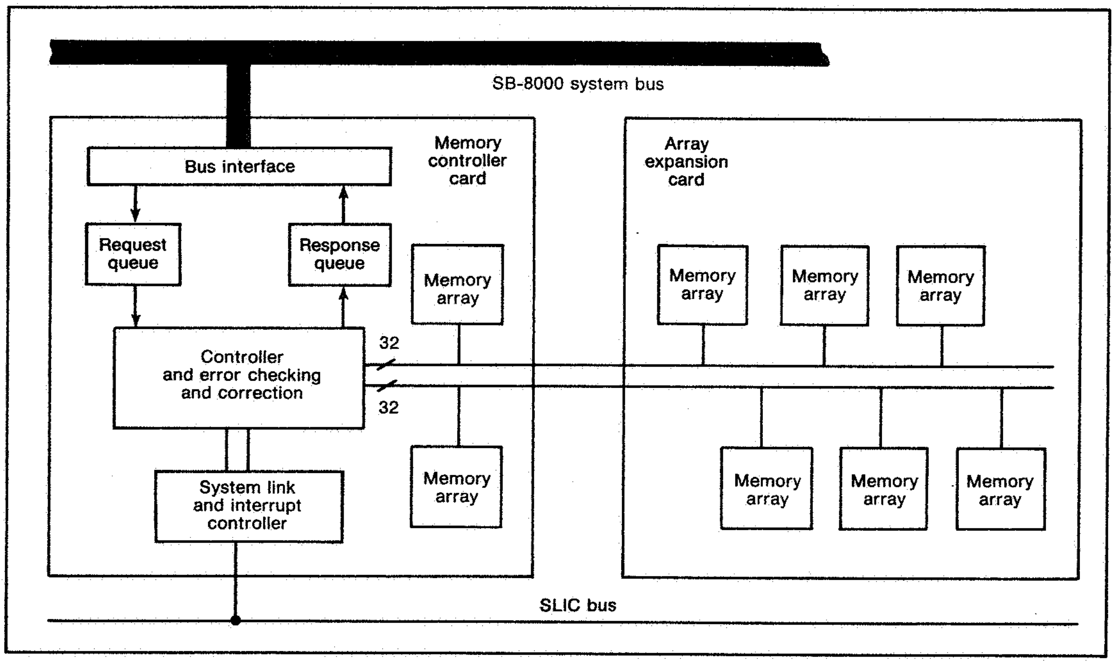
given processor of bus access. This in turn prevents deadlocks or intolerable degradation in performance.

A related issue for split-response protocol is that of managing congestion and controlling the flow of queue requests. Instantaneous heavy loads on the bus can fill up the request queues of the responders. In some systems, the flow control mechanism uses a negative acknowledge (NAK) response to keep those queues from overflowing. However, such a system can degrade from excessive request and NAK cycles. In the Balance 8000, however, each requester stores status information about each relevant queue. Requests are not propagated onto the system bus unless they are guaranteed a slot in the address queue. Therefore, even under heavy loads, each and every bus cycle is used productively.

Controlling and recovering from errors is always a difficult issue, but even more so in a multiprocessor system where there are many contenders for the bus and a high degree of concurrency. The system's bus checks for parity and uses other error-checking schemes. Whenever a serious error occurs, the hardware records the identity of each party involved and then freezes the bus so that the error does not create further trouble. Next, a diagnostic processor takes control and, through the link and interrupt controller chips, investigates the problem. Typically, recovery software will reset the computer, run selected confidence tests, disable the faulty module, and reboot the system on the remaining hardware.

Memory matters

In a scalable processor pool architecture, the single global memory is the center of the universe. It stores all resident code and data and



6. A pipelined packet bus and 8-byte transfers help maintain the high bandwidth needed by the global memory in the multiprocessor system. Requests arrive from the bus interface and are stored in the request queue, while responses queue up and await access to the bus. At the same time, the controller cycles the 64-bit-wide arrays to access data and check errors.

must satisfy all requests that are not met by the local caches. Therefore, a high bandwidth for the memory is critical.

To do its job, the central memory uses a pipelined operation and a memory controller, which together sustain a data transfer rate of 8 bytes every 300 ns (Fig. 6). Memory requests arrive from the bus interface and are stored in the request queue. At the same time, the controller can cycle the 64-bit arrays to access the data and generate and check error-correction codes. Also in parallel is the response queue which returns responses to previous requests.

In addition, the SLIC circuit on the memory controller card is used to report any error information and identify a memory module and its configuration to the auto-configuration software. It also allows that software to set attributes on a memory controller card, like a base address and the number of controllers to be interleaved.

I/O reaches out

Although technical applications often require sheer computational intensity, it is still important to have a balanced I/O capability. This is the job of the computer's I/O channel processor card (Fig. 7). Each channel card includes a 32016 whose instruction set is compatible with the 32032 CPU and which interprets channel commands and drives the on-board I/O adapters. Also, the processor on the first channel board serves as a diagnostic processor, taking advantage of the card's fully self-contained environment. The processor operates normally even in the event of hard errors in such vital portions of the system as the bus, memory, or processor pool.

Among the I/O adapters are an IEEE 802.3-compatible Ethernet interface and a Small Computer Systems Interface (SCSI) mass storage bus. This emphasis on industry standard interfaces makes it easy for the OEM to mix and match from among the growing set of compatible mass storage devices. The system's software device driver also benefits, accommodating a wide range of disks without modifications.

In order to peak the I/O's throughput, the system employs individual FIFO "surge" buffers and direct memory access controllers that are designed for 8-byte transfers. Using surge buffers instead of a conventional RAM to buffer bursts of data from an I/O device minimizes the delay between the data's arrival and its updating of the main memory.

The I/O channel processor board is designed to sustain both (Ethernet and SCSI) adapters at once. Its high transfer rate and ability to gather scattered data into consecutive memory locations make it an excellent match for the large disk transfers used in Unix 4.2 BSD's fast file system. Since the system supports up to four such I/O channel boards, the designer has the capability to balance the I/O bandwidth to match the computational bandwidth for a given application.

The computer system's IEEE-796 (Multibus) bus conveniently obliges an OEM who wants to add custom hardware or put in one of the many Multibus-compatible, special-purpose I/O adapters. Up to four such buses can be connected. Moreover, data can move directly between the system bus and the Multibus and the bus coupler maps addresses and connects protocols.

Unix for everyone

As mentioned earlier, the computer's operating system is based on Unix 4.2 BSD. This version is a standard in the area of technical processing and offers several advantages over other Unix derivatives. The fast file system, for example, vastly improves the I/O throughput with clever data placement algorithms and by transferring large blocks of contiguous data. These file-management improvements count for even more in a multiprocessor system. Also, the virtual memory and demand-paging features of Unix allow several large programs to run concurrently and reduce the overhead in starting a new process.

In addition, the 4.2 BSD version takes advantage of the proven DARPA (Defense Advanced Research Projects Agency) standard TCP/IP (Transmission Control Protocol/Internet Protocol) to facilitate integrated and heterogeneous network services. Some examples are remotely transferring files, logging on, or executing commands.

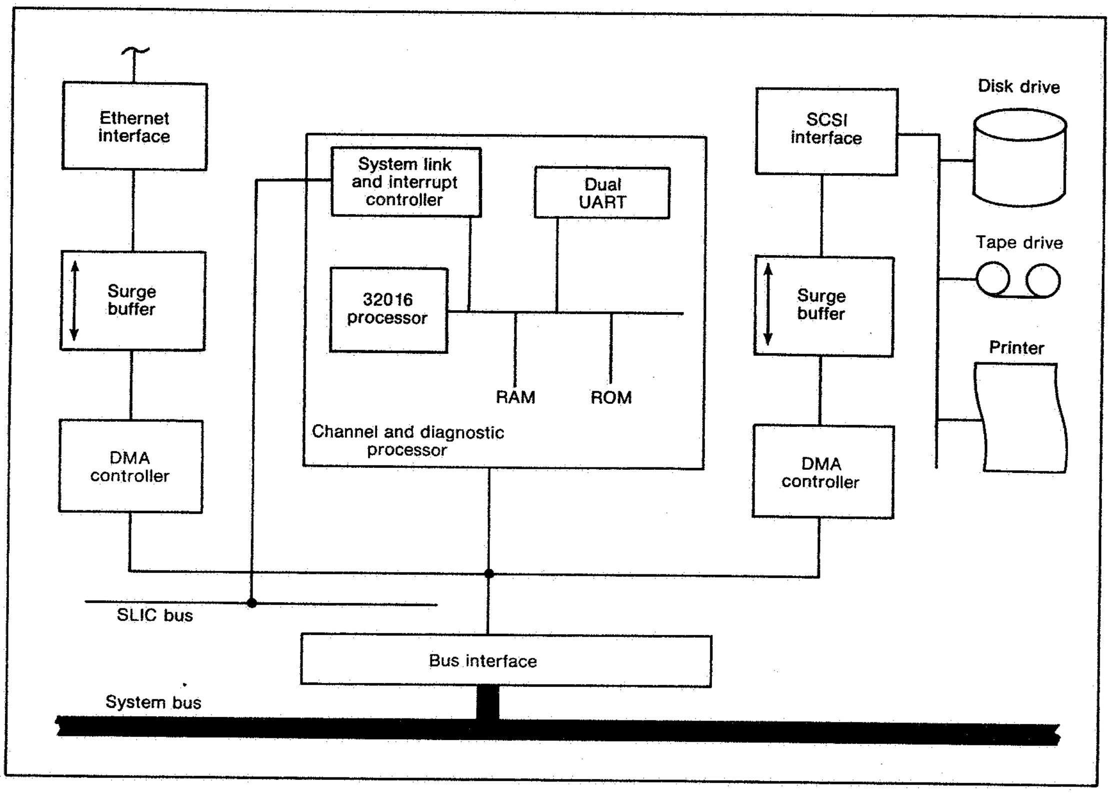
Much of the power and elegance of Unix is in its concise syntactical notation and support for multiprogramming. Unix encourages the programmer to construct applications by piecing together collections of smaller programs; these are interconnected by a simple inter-process communications (IPC) facility: the pipe. A pipe connects the output of one program to the input of a second, whose output goes to the input of a third program, and so on.

The IPC facilities of Unix 4.2 BSD extend to include a so-called socket-based message-passing mechanism. Rather than doing everything in a large monolithic program, this philosophy advocates smaller, simpler, and easily reused programs all communicating through the IPC facilities.

Unix also encourages multiprogramming at the human level. With the simple shell command-language syntax, a user can invoke one or more processes and have them run in the background simply by terminating the command with an ampersand (&). Alternatively, the user can invoke several communicating process simultaneously by using the pipe syntax " / ."

Done with mirrors

On a single-processor computer, apparent concurrency is not real, since the processes are multiplexed (time-sliced) through a single processor. However, on a multiprocessor system with a processor pool architecture, the pool of symmetrical processors will execute the pro-



7. Each I/O channel card contains a 32016 processor that interprets channel commands and drives the on-board I/O adapters. The processor on the first channel board also serves as a diagnostician; it can operate normally even if hard failures befall the system bus memory or processor pool.

cesses in true concurrency, resulting in a multiplicative improvement in throughput.

This architecture, then, combines the benefits of Unix's existing application set with the scalable power of multiple processors. Simply adding or deleting processors changes the level of performance without a single software change.

Some changes needed

However, although the Unix phiiosophy is based on multiprogramming, the standard implementation is still based on a single-processor architecture. By implication, that would limit the operating system functions to one process at a time. In contrast, a processor pool architecture often accommodates several simultaneous processes executing within the kernel on several processors. For this the kernel needs to be fully sharable by multiple, concurrently executing processors.

The version of Unix ported to the Balance 8000, called Dynix, required modifications in four major areas: mutual exclusion, interrupt distribution, virtual memory management, and process scheduling.

On a single-processor system, mutual exclusion of processes from the operating system is achieved simply by disabling processor interrupts, therefore guaranteeing that no other process will get control until the interrupts are enabled. However, this technique is not enough to guarantee mutual exclusion for a processor pool multiprocessor, since other processors may be simultaneously executing processes via the kernel. The Dynix kernel, therefore, incorporates a more robust model of exclusion based on three mechanisms: gates, locks, and counting semaphores.

Gates and locks

A gate, the lowest-level mechanism, is implemented in the SLIC circuit and used only for routines where time is critical, since there are only a limited number of them available. A lock is a software version of a simple, non-queueing unit semaphore built over the gate mechanism; it is used when the requesting process cannot afford to "sleep" (lapse until notified) while waiting.

The highest-level exclusion mechanism is the counting semaphore, consisting of a counter and a waiting queue. Semaphores completely replace the conventional sleep-and-wakeup mechanism, providing more structure and eliminating unnecessary context switching. Semaphores are used by processes to avoid waiting for events or to hold the sought resource for a long time.

In a conventional Unix system, a single processor receives all device-interrupt signals. However, in a processor pool architecture, any processor can accept any interrupt signals. The resulting decrease in interruption latency is another improvement over single-processor systems.

If necessary, however, device drivers can avail themselves of the kernel lock and semaphore mutual-exclusion mechanisms to protect critical data. There is also a mode that emulates a single-processor system to simplify the porting of existing monoprocessor drivers to the new multiprocessor machine. Moreover, Dynix supports configuration at the object-code level, so users can add drivers and rebuild the kernel without the source code.

Spreading the work around

Process scheduling in Dynix is conceptually quite similar to that of conventional Unix. There is a single, priority-oriented run queue; since the system is fully symmetrical, any process can execute on any processor, whether in supervisory or user mode. As for priorities, the basic scheduling philosophy is to always run the highest-priority, ready-to-run processes first.

Dynix's global visibility minimizes the number of context switches that will result from system events. It works this way: a periodic interrupt causes the processor accepting it to determine if there is a process running that should be preempted by a higher-priority one in the run queue. Similarly, when one currently executing program initiates a second process, the processor checks to see if it should preempt any other program running on any of the other processors. If so, the processor may use its SLIC's software to nudge whichever other processor is running the lowest-priority task into rescheduling it and taking on the new job.

Obviously, Dynix contributes to the processor pool architecture's ability to balance loads dynamically. As already mentioned, this is a function of the fact that all processors are identical and all code and data are equally accessible to all processors. There is no rigid binding of processes to processors, and there are no restrictions limiting the ability of processes to execute on any of the processors. In short, any process in an executable state can run on any processor.

To get down to specifics, the Dynix scheduler continuously monitors the set of ready-to-run processes and dynamically schedules the highest-priority ones so that each processor is always working on the most important job as long as there is work to be done.

Many applications

A computer of this kind opens up new application possibilities, the breadth of which is exemplified by two extremes: a multi-user, multipurpose network server, and a single-user, application-specific workstation.

At the first extreme, technical OEM applications require a broad range of distributed and centralized computing power, but the overriding requirement is that each user have full access to all the system's resources. In part, this means that a computer's files be fully available to dumb terminals, personal computers, and workstations alike.

A network server fits well into this scheme because it can manipulate the contents of a data base for the workstation, offer files and computing power to the personal computer, and provide all but display services to the dumb terminal. Moreover, the computational power required of the server depends on the size of the network. Here the scalable pool idea is a natural fit.

The multiple users will undoubtedly generate a large number of independent processes that will be transparently and dynamically scheduled by Dynix to run in order of priority on the available set of processors. Therefore, an increased throughput attributed to the two or more processors is available without the application designer expending any extra effort.

With this approach, the OEM can create a "product line in a box," offering a basic configuration that processors can be added to or subtracted from to meet the particular needs of the network. Similarly, the installed base of systems can be upgraded in the field, without software changes, by the simple addition of a processor card.

Workstation potential

At the other extreme, some individual applications are so computationally intensive that they justify the expense of their own dedicated, single-user workstations. Here the OEM needs to offer a low-cost starter system and the opportunity to add processing power later. In addition, the workstation must fit into a network with other, and probably different, computer systems, in order to conveniently share data bases.

One example has an image server as one node, a high-resolution film recorder as another, and a workstation demanding high-quality graphics. Here again, the pooled processor system is a powerful solution. The OEM can configure a low-cost starter system with only 2 processors, while offering to upgrade it to 12 as needed. And the high-performance graphics subsystem is easily connected to one of the industry standard I/O ports, like the SCSI bus; for the highest possible performance, it could go directly on the system bus.

In addition, most computationally intensive technical applications have a great deal of inherent parallelism that can exploit the power of the multiprocessor computer by making simple changes to a program originally intended for a one-processor computer. The ray-tracing algorithm used to realistically model and shade three-dimensional solids is a good example. The parallel computational paths are easily separable; a typical algorithm can be so modified in less than a man month. In many cases, the algorithm can even be designed to dynamically adapt to the number of available processors, thereby providing the convenience of self-configuring software, even at the application level.□