Hector: A Hierarchically Structured Shared-Memory Multiprocessor

Zvonko G. Vranesic, Michael Stumm, David M. Lewis, and Ron White
University of Toronto

The architecture of the Hector multiprocessor exploits current microprocessor technology to produce a machine with a good cost/performance trade-off. A key design feature of Hector is its interconnection backplane, which can accommodate future technology because it uses simple hardware with short critical paths in logic circuits and short lines in the interconnection network. The system is reliable and flexible, and can be realized at a relatively low cost.

An important aim of the Hector project is to develop an architecture suitable for a general-purpose multiprocessor whose cost is directly proportional to size. Thus, an entry-level machine would be inexpensive, but can scale to larger sizes. To accommodate configurations with varying numbers of processors, Hector has a hierarchical structure. Bit-parallel rings interconnect small bus sections. The buses and rings can transfer data independently of each other, so aggregate bandwidth increases proportionally with the number of these units.

Hector is suitable for single jobs with many parallel tasks, as well as for concurrent execution of multiple jobs consisting of predominantly serial tasks. In other words, it is effective in a Unix environment, as well as in such highly parallel commercial and scientific applications as transaction systems, finite-element analysis, and computer-aided design.

Hierarchical structure results in a fast backplane and a bandwidth that increases linearly with the number of processors. Hector scales efficiently to larger sizes and faster processors.

Hector in the multiprocessor picture

Our goal in the Hector project is to explore a design region where the most cost-effective multiprocessor solutions are likely to lie. Figure 1 shows where the Hector architecture fits with existing machines and well-known research projects. The figure indicates the relative characteristics of different machines configured to have approximately the same computational power — 0.5 to 1 Gflops.

The two axes in the figure represent two important design options. The horizontal axis represents the power of the individual processors used, with very simple processors at the origin. The less powerful the processors, the larger the number needed to achieve the desired aggregate system power.

The vertical axis represents the degree of coupling between the processing modules. In loosely coupled systems, processors can directly access only their local

0018-9162/91/0100-0072$01.00 © 1991 IEEE

memories, but use data passing to communicate with other processors over the interconnection network. In tightly coupled systems, processors communicate through shared memory at speeds normally associated with local (cached) memory access. Between these two possibilities are distributed memory systems that allow processors to access shared memory directly, but with reduced access times to some memory locations.

Loosely coupled systems are easier to build and less expensive, but shared-memory systems are considered by many to be easier to program.[1] Given two systems with processors of equivalent power, the more tightly coupled one is considered to be more general. Typically, it can execute more efficiently all applications that its more loosely coupled counterpart can. It can also execute some applications not suitable for more loosely coupled systems where sharing occurs at a finer granularity. Hence, any system represented by a point in the figure is more general than all systems represented by points directly above it.

Two extremes in the figure are immediately apparent. On the left side are machines with many simple processors, such as the Connection Machine, which has many 1-bit processors. With such machines, interconnection requires a data-passing organization. On the right side are machines that comprise only a few exceptionally powerful processors, such as the Cray Y-MP/4. These machines can make full use of the shared-memory concept.

Computational power is not simply a function of the architecture: An entry using the Connection Machine (running an application executing at 6 Gflops) won the Gordon Bell prize for supercomputing for 1989, while an entry using the Cray Y-MP/8 (running at 1 Gflops) won it for 1988.[2]

Multiprocessors that rely heavily on advanced microprocessors lie between the extremes. These machines fall into three categories, according to the coupling scheme used. A data-passing organization has been the natural choice in multiprocessors based on hypercube interconnection backplanes. Well-known examples are the Intel iPSC, the Floating Point Systems T Series (FPS-T), and the NCube machines.

The shared-memory model with full cache consistency is the goal of the recently proposed MIT Alewife,[3] Wisconsin Multicube,[4] and Stanford Dash.[5] Several machines have been built with shared-memory capability but without hardware-supported cache consistency. The BBN TC2000, the IBM RP3,[6] and our Hector multiprocessor are of this type. These systems cache read-only code and data, as well as local data, but not all shared modifiable data. The Cm* multiprocessor, designed and built at Carnegie Mellon University in the late 1970s, would probably also belong to this class if it were implemented with today's technology. When implemented, the BBN Monarch[7] will incorporate in a shared-memory organization many somewhat simpler processors with no caches.

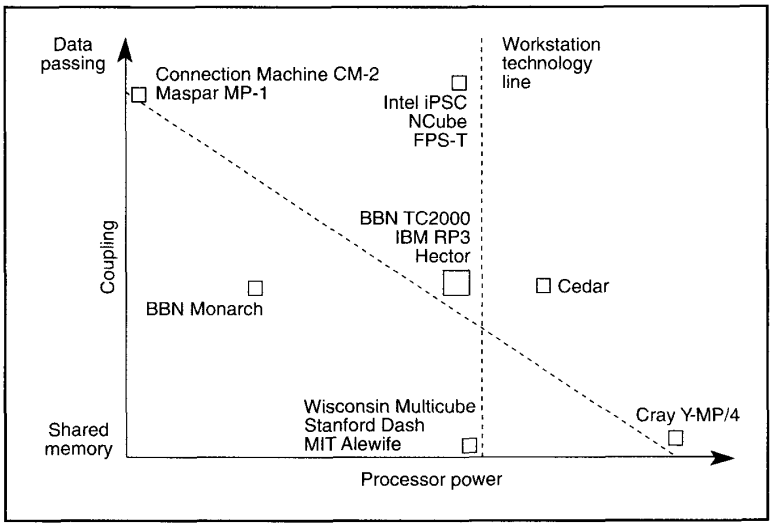

Figure 1. Coupling and processor power in multiprocessors.

In Figure 1, a line joins the Connection Machine and Cray extremes. The machines above this line are economically feasible with today's technology. Indeed, all of the machines shown in that region have been built, although not necessarily in full size. On the other hand, machines that lie below the line are more difficult to build; none of those shown in the figure have been implemented. With these systems, the challenge is to provide coherent cached shared data with good performance, without introducing excessive interprocessor traffic.

The systems at both ends of the diagonal are expensive. The most cost-effective solutions may lie along a line in the region where workstation technology is used, indicated by the vertical line. The point at the top end of the line probably represents the most inexpensive solution: processors with the best price-performance ratio in a simple interconnection structure. The workstation technology line is rapidly moving to the right; 100-MIPS microprocessors may soon be available. Increasing the speed of the interconnection network is more difficult than increasing the speed of the processor chips. Thus, it will be a challenge to build systems at the point where the workstation technology line intersects the diagonal. To address this challenge, Hector incorporates a novel hierarchical pipelined backplane that will scale to future technologies.

Organization of Hector

Although it is the simplest structure for interconnecting several processing modules, the bus does not scale well in size and saturates quickly if too many modules are connected. With high-speed processors, typically two to six processors can be connected to a bus without significant performance degradation. Connecting a larger number of processors to a single bus is feasible only if the processors' request rate for bus transfers is low.

Hector exploits the natural advantages of the bus by using relatively small bus sections tied together through hierarchical bit-parallel rings. Figure 2 shows the overall organization. A bus section, with the associated processing modules, is called a *station*. Several stations are interconnected by a bit-parallel *local ring*. Local rings are, in turn, interconnected by a *global ring*.

This hierarchical structure is similar to

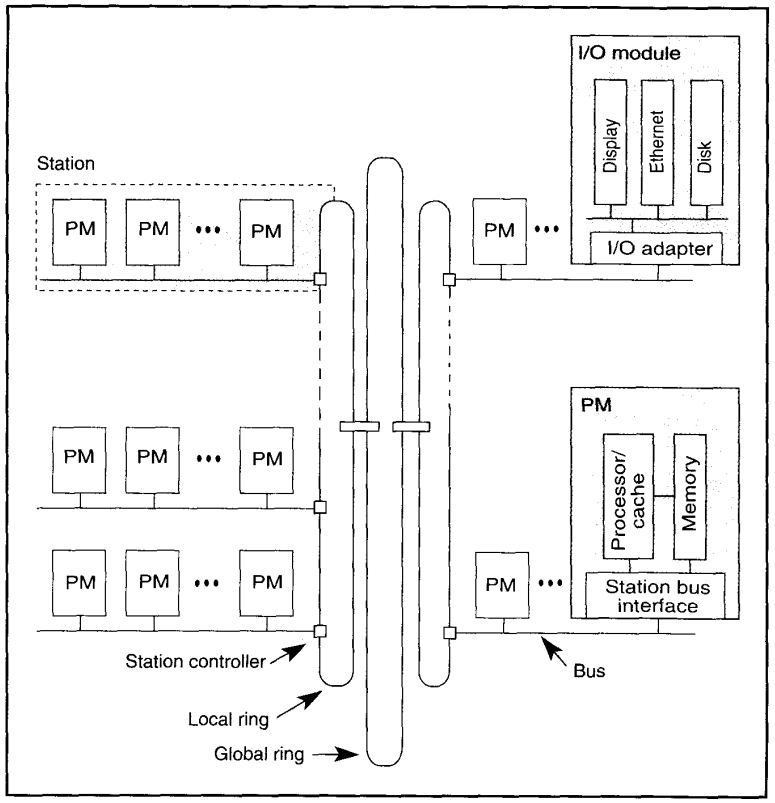

Figure 2. Structure of Hector.

those of Cm* and Cedar. In Cm*,[8] multiple processor-memory pairs are connected by a "map-bus" to form a cluster, and multiple clusters are interconnected via "intercluster" buses. Rings at higher levels of the hierarchy, as used in Hector, have the advantage that they can be constructed using only short point-to-point transmission links, allowing for faster signaling. Moreover, the overall bandwidth of the ring increases proportionally with the number of ring segments (although increasing the number of ring segments also increases latency by a corresponding amount). A second difference between Hector and Cm* is in how communication is controlled. In Cm*, a relatively powerful, horizontally microcoded processor controls communication, whereas in Hector, control is by simple discrete logic (implemented in PALs in our prototype). This simplicity allows for faster switching and higher throughput.

In Cedar,[9] the lowest level consists of several Alliant FX/8 systems, each comprising eight processing elements (with vector units) connected by a crossbar switch to a cache and higher level communication structures. These, in turn, are connected by another crossbar switch to memory. We believe that the relative simplicity of Hector's rings allows Hector to be more easily scaled to larger sizes and faster technology.

Basic Hector architecture. A station consists of a number of *processing modules* (PMs), connected by a bus. A typical PM comprises a microprocessor, cache memory, on-board memory, and various I/O interfaces. While we use the term PM to refer to a module (board) plugged into a slot in the backplane, a PM need not have any processing capability. Some may be just memory modules; other specialized PMs provide interfaces to various I/O devices.

Hector provides a flat, global (physical) address space, where each PM is assigned a unique range of addresses. All processors can transparently access all memory locations. Information transfer takes place in a packet format using a synchronized packet-transfer scheme. The traffic is controlled by three types of interface circuits. A *station bus interface* in each PM handles the communication requirements of the PM. A *station controller* controls on-station transfers as well as the local ring traffic at the station. It contains a set of latches that hold an entire packet and a set of transceivers that isolate the station bus from the local ring. A packet traverses the ring by being transferred from the latches of one station into the latches of the next station. An *inter-ring interface* connects a local ring to the global ring. Within one clock cycle, a packet can be transferred

- between two PMs within a station,
- between the latches of two adjacent station controllers or inter-ring interfaces,
- from a PM in a station into the latches of the next station controller on the ring, or
- from the latches of a station controller to a PM in the station it controls.

An on-station transfer and a transfer from the latches of that station controller to the latches of the next station can occur simultaneously. The low complexity of each operation makes the backplane scalable.

By controlling both the station bus and the corresponding ring segment, the station controller can give priority to packets being transferred on the ring. It does not allow a local PM to access the bus when there is a packet in the ring latches addressed to this station (thus allowing the packet to be latched onto the station bus). Also, it allows only on-station transfers whenever a valid packet in its latches is to be transferred to the next station. This strategy prevents packets from having to be dropped or queued at the station and local ring levels.

The inter-ring interface, which is essentially a 2×2 crossbar switch, requires FIFO buffers to store packets when collisions occur — that is, when packets coming from both the local and global rings in a given clock cycle have to be routed to the same output.

The addressing scheme is simple, so packets can be routed with minimal overhead in a fraction of a clock cycle. Each ring is identified by the most significant r bits of the (memory) address, the station is identified by the next s bits, and the slot in the station is identified by the least significant p bits. This allows simple and fast address decoding.

Memory access and communication protocol. Accesses to remote memory modules are transparent to the processors. A nonlocal memory request is recognized by the control circuitry at the station bus interface, which forms a request packet. The request packet includes a 32-bit destination memory address and a source PM address. In the case of a write, the packet also contains 32 bits of data. When the packet arrives at the destination, the destination station bus interface performs the action requested (read or write) and returns a *response* packet to the source. For a read, the response packet contains up to 64 bits of data. For a write, the response packet is just the acknowledgment that the write operation has been completed. The response packet is sent to the PM identified by the source address in the request packet.

If the source PM receives no response packet within a time-out period, it retransmits the request. If it receives no response after multiple retries, the operation fails. The requesting PM also retransmits the request if it receives a negative acknowledgment response, which happens, for example, when a remote PM recognizes a request but cannot service it. To allow a PM to have multiple outstanding requests, a tag is included in each request packet and returned with the response packet so that the response can be matched to the correct request. (Our prototype implementation allows only one outstanding request.)

Considerable flexibility is permitted in the memory-access requests, which can involve 8-, 16-, 32-, or 64-bit data. A burst read of 128 bits can be used to load cache lines. For burst reads, the responding PM automatically generates multiple response packets, each containing 64 bits of data. The entire operation is retried if any response packet does not arrive within the time-out period.

Atomic operations. Packet transfer is usually reliable. Therefore, to reduce hardware complexity and cycle time and hence increase performance for the common case, we believe that individual packet transfers can be aborted without specifically signalling an error condition. For example, a packet may be dropped when a transmission error is detected (by parity bits) or when a buffer overflows. Requiring the hardware to generate negative acknowledgment packets in these cases would add to its complexity. Hector's communication protocol allows source PMs to detect aborted transfers by timing out, at which time they can retransmit the request packets.

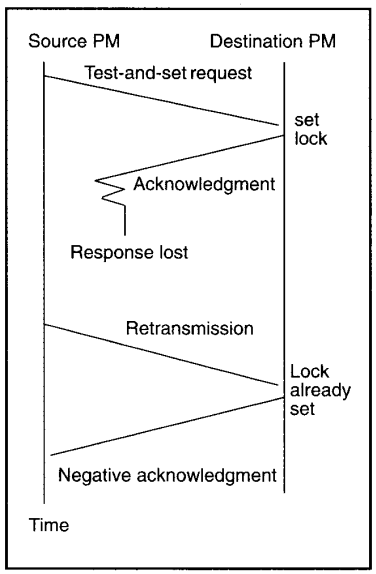

Figure 3. The problem with nonidempotent requests.

For read and write operations, the request-response protocol with time-out works correctly because of the idempotent nature of the operations. (Strictly speaking, the write operation is not idempotent: A retransmission of a write request may result in the write operation occurring twice. This is acceptable for most applications; for the others, the write must be made part of a critical region.)

An important operation for which the request-response protocol will not work directly is read-modify-write, which is needed to implement a processor's test-and-set or swap instructions. Here, a difficulty arises if a PM does not receive a response to a read-modify-write request within a time-out period. The source station bus interface does not know whether its request packet was dropped or whether the corresponding response packet was dropped (in which case all further retries may fail). This situation is shown in Figure 3, where the response to a test-and-set request is dropped. When the destination PM receives the retransmission of the test-and-set request, it returns a negative acknowledgment because the lock is already set. The source PM cannot determine whether the lock was set by its initial request or whether it was set by another processor in the meantime.

To handle this situation, the read-modify-write operation is performed in two separate stages. In the first stage, the PM sends a read-and-lock request, which causes the responding PM to read the addressed memory location and return its contents in a response packet. When the requester receives the data requested in its read-and-lock packet, it starts the second part of the read-modify-write by sending the data to be written to the destination memory in a write-and-unlock packet. To guarantee atomicity, the responding PM must prevent other processors from performing a read-and-lock at the same memory location between these two operations. Each station bus interface maintains a set of <proc, addr> pairs for this purpose, so an entry is recorded during the read-and-lock operation and cleared during the write-and-unlock operation. The station bus interface also uses these <proc, addr> pairs to detect and appropriately handle duplicate requests resulting from retransmissions.

This protocol allows atomic operations on any memory location and can survive losses of packets, regardless of the packet lost. If either the read-and-lock or its response is lost, the requester will time out and retransmit the request. If the destination PM received the original request, it will recognize the retransmission (since it stored the requester's identifier) and respond with an acknowledgment. If either the write-and-unlock or its response is lost, the requester will also time out and retransmit the request. If the original write-and-unlock packet was lost, this will result in a write-and-unlock action. Otherwise, the destination PM will send an acknowledgment.

Hector's backplane compared with other interconnection structures. As a hierarchical system, Hector provides for fast local operations over a bus. Most systems with nonuniform memory-access times support memory accesses at only two time-cost levels — namely, local on-board accesses and remote accesses. For example, in a 16-processor BBN TC2000, the access-time differential between local memory and remote memory is 1:4. In contrast, Hector has multiple levels in its hierarchy, so the cost of accessing data increases incrementally with the distance. In our prototype implementation, the access time differentials between local, on-station, on-ring, and off-ring memory are 1:1.2:2:4. The advantage of a hierarchical structure is that much of the communication remains within the lower levels of the hierarchy because of the locality in data ac-

Table 1. The cost and complexity of interconnection structures ($n \leq 256$ is the number of processors).

Parameter	Hector	Banyan	Crossbar	Hypercube
Longest wire length	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Gates in switch path	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Total switch cost	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Switch data bits	$O(1)$	$O(1)$	$O(n)$	$O(\log n)$
Switch control bits	$O(1)$	$O(1)$	$O(n \log n)$	$O(\log n)$
Switch complexity	$O(1)$	$O(1)$	$O(n^2)$	$O(\log n)$
Fan-out	$O(1)$	$O(1)$	$O(n)$	$O(\log n)$

cesses, resulting in relatively low average memory-access times. Moreover, since a bus is used at the hierarchy's lowest level, a simple snoopy protocol can provide cache consistency among the PMs attached to the bus.

Other important and distinctive advantages of the Hector backplane are its simplicity, low cost, and scalability to higher clock rates. Direct comparisons with other systems are difficult. Nevertheless, Table 1 summarizes our attempt to compare metrics that affect the cost and speed of several interconnection structures. Besides Hector, we considered Banyan networks, crossbars, and hypercubes.

The ability to scale an interconnection network to higher speeds depends largely on the longest wire length needed to connect system parts. Signal quality degrades over long wires at higher clock rates, and skew between signals in a cable makes it difficult to reduce the clock cycle time. The first line in Table 1 gives the length of the longest wire required by each type of network. We assume that these systems can be implemented to fit in a single rack, since a system within a single rack has a maximum wire length that is a linear function of processor distance, while in a larger system the wire length might be the square root of the distance. Hector has a constant-length wire, since each ring connects only to its immediate neighbor. In our prototype, with two levels of rings, the longest wire is only seven inches. All the other systems require a wire length proportional to the size of the system, because they require connections to processors at least halfway across the entire set of processors. This makes it difficult to scale to high clock speeds.

The number of gates in the path can also cause significant delays. All of the systems have identical orders of delay, although they still may differ by a constant factor that can be quite large (for example, 50 nanoseconds, the time it takes to transfer a packet in Hector, as opposed to 7 ns, the delay through a crossbar multiplexer). The third line in the table measures the total cost of the switch for an entire system. Hector maintains $O(n)$ cost, so the switch cost is always a small constant fraction of the total system cost. At the other extreme, the cost of a crossbar is $O(n^2)$, which can rapidly dominate the total system cost as n becomes larger.

The remaining lines of Table 1 measure the complexity of implementing the interconnection network using application-specific integrated circuits (ASICs). The fourth line measures the amount of data processed by a switch, and the fifth line measures the number of bits required to control data routing at a switch. These metrics are important because the number of pins that can be connected to a chip is limited. The sixth line gives the logic complexity in a single switching unit. All networks, with the possible exception of large crossbars, are simple enough to be implemented with ASICs. The last line gives the fan-out per wire.

Table 1 supports our claim that Hector offers a scalable architecture through its use of short electrical connections, and does so at a lower cost than other interconnection networks.

Implementation

Figure 4 shows a block diagram of the PM board. The processor is a Motorola MC88100 microprocessor running at 20 MHz. The cache consists of up to four Motorola MC88200 16-Kbyte cache chips. The on-board memory comprises up to 16 Mbytes of RAM, implemented as part of the processor boards rather than as separate modules to reduce bus loading and the average number of bus accesses. The PM contains two on-board buses called the processor bus and the memory bus. They are separated by buffers to isolate the processor from the memory bus, allowing other PMs to access this memory while the processor is accessing off-board memory.

Three main memory activities on a processor board are important: processor on-board requests, processor off-board requests, and requests from the station bus to the memory. The processor accesses on-board memory by first obtaining control of the memory bus and then accessing the memory. Off-board references from the processor use the station bus interface circuitry, as explained earlier. The station bus interface places arriving memory requests in a two-deep FIFO before requesting control of the memory bus. Once control is granted, it performs the memory operation and returns the acknowledgment together with any data. It signals a negative acknowledgment on a distinct bus line if the FIFO is full when a packet arrives.

The station bus and local ring interface. Station bus operations are pipelined as follows. A source PM sends a bus request to the station controller during one cycle. If there is no contention, the station controller returns the bus grant at the beginning of the next cycle and the PM places the packet on the bus during the same cycle in which it received the grant. In the case of an on-station transfer, the destination PM acknowledges reception of the packet in the next cycle by asserting the Received line of the bus. A separate acknowledgment packet is therefore not necessary for on-station write requests. If the Received line is not asserted, the source PM will immediately attempt to retransmit the request. The entire process — from the time the source module asserts its Request line to the time it recognizes its Received line — takes three cycles, but it ties up the bus for only one. Independent transfers from different source modules can therefore be placed on the bus in each cycle, allowing full use of the bus bandwidth.

The station controller is responsible for arbitration of the station bus and data transfer between the station bus and the local ring. It gives the highest priority to packets on the ring addressed to its station. Lower

priority is given to requests for off-station transfers by PMs on the station. An off-station transfer is accommodated whenever the ring segment has an empty slot. The lowest priority is given to on-station transfers. Within this priority strategy, the bus is granted in a round-robin fashion when multiple off-station or multiple on-station bus requests are outstanding.

The inter-ring interface. The inter-ring controller has a two-deep FIFO buffer. In general, it gives priority to packets coming from the global ring; hence, packets that successfully reach the global ring will reach their destination (if no transmission errors occur). Extensive simulations indicate that this strategy provides the best performance for virtually all data-access patterns. Simulation results also indicate that for the type of data-access patterns we expect to be typical, a global ring connecting n rings with interfaces containing two-deep buffers has fewer collisions than an $n \times n$ crossbar with no buffers.

Instrumentation support. We have implemented an instrumentation facility that allows nonintrusive hardware monitoring. It can operate in a number of modes. For example, the address histogram mode allows generation of histograms based on address information, conventional memory-use profiles, or interreference-jump-distance distributions. The state histogram mode allows analysis of bus-cycle and machine-state distributions. Two tracing modes are also supported. The full trace mode traces address information and the timestamp mode is used when references to a small number of objects may be distributed over long periods of time. This experimental facility allows us to monitor and measure low-level activity on real workloads.

Scaling for speed. Hector's interconnection network is designed to be scalable to higher speeds. The minimum cycle time is 46 ns in our prototype, as shown in Table 2. The relatively long time to drive the bus is due to the large capacitive load offered by the multiple bus drivers and receivers on each station.

Processor clock speeds will continue to increase. The clock cycle of the Hector backplane can be reduced by adding pipeline stages in the bus controller. This will lead to better throughput but increased latency. For example, the bus operations in Table 2 can be split into two pipeline stages. In the first stage, a packet is enabled onto the bus from one bus driver and clocked into all receivers without being decoded, reducing the electrical loads on the bus from the current three to one per PM. The time needed to accomplish this essentially dictates the cycle time. The second stage is decoding: The controller determines in which buffer the packet should be placed. If the packet is addressed to a given module, the controller selects a multiplexer to pass the packet into an appropriate set of latches.

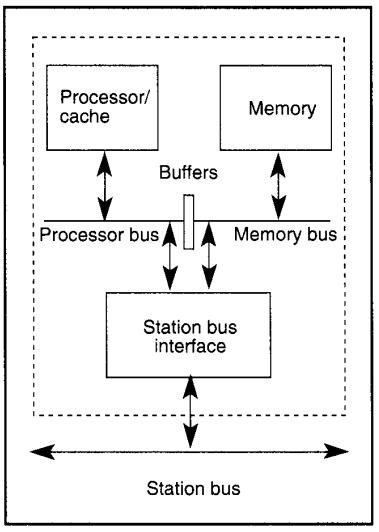

Figure 4. Processor board block diagram.

The second way to reduce cycle time is to use ASIC technology. The top half of Table 3 shows our timing estimates for the pipelined implementations using standard complementary metal-oxide semiconductor (CMOS) cells with conventional medium-scale integration (MSI) bus drivers and receivers (because of the low drive capability of CMOS ASICs). The reduced time to drive the bus is due to decreased loading on the bus. The bottom half of Table 3 shows timing estimates for an implementation using an emitter-coupled logic (ECL) gate array that includes the bus drivers and receivers on chip.

Table 2. Current 20-MHz bus timing.

Operations	Time(nanoseconds)
Clock → bus grant	7
Bus grant → drive bus	18
Drive bus → decode address	7
Decode address → controller output	10
Setup time	2
Skew	2
Total	46

An implementation using ASICs would eliminate many constraints that limit performance in our prototype. In particular, because of the large number of chips required, the MSI technology we use makes it difficult to implement buffering or pipelining for data paths. With ASIC technology we could implement many such buffers on a small part of a chip. This would also allow us to reduce memory-access times significantly by interleaving the memory system, pipelining the memory error-correction circuitry to deliver corrected data at full memory bus speed, and increasing the size of the data path to 64 or 128 bits. In addition, an eight-packet-deep FIFO in the station bus interface would dramatically reduce the number of retries required. Ultimately, VLSI technology will advance to the point where it may be feasible to implement an entire Hector station on a single chip.

Hector has three important advantages. First, the hierarchical structure allows short transmission lines and construction of a simple and fast backplane. This makes Hector scalable to match the needs of future high-speed microprocessors and leads to increased performance, reliability, and flexibility, as well as to lower cost. Second, the cost and the overall bandwidth of the structure grow linearly with the number of processing modules. This makes Hector expandable to large sizes, yet allows small configurations at a low cost. Finally, the cost of a memory access grows incrementally with the distance between the processor and memory location. This allows the low cost of localized memory accesses to be exploited by single threaded applications, applications with a small degree of parallelism, and applications with a high degree of locality in their memory accesses.

A possible shortcoming of our current design is the lack of cache consistency across all processing modules. While hardware-based cache consistency mechanisms for larger systems is an active area of research,[3-5] it appears that their complexity

Table 3. Timing estimates for CMOS and ECL ASIC technology.

Technology	Stage (time in nanoseconds)		Stage (time in nanoseconds)	
CMOS	Stage 1		Stage 2	
	Clock → grant	7	Clock → clocked bus data	11
	Grant → drive bus	14	Clocked bus data → ASIC bus	2
	Setup time	3	ASIC bus → decoded address	4
	Skew	2	Decoded address → controller output	5
			Controller output → multiplexer output	3
			Setup time	1
	Total	26	Total	26
ECL	Stage 1		Stage 2	
	Clock → grant	1	Clock → clocked bus data	1
	Grant → drive bus	5	Clocked bus data → decoded address	2
	Bus → ASIC internal bus	1	Decoded address → controller output	2
	Setup time	1	Controller output → multiplexer output	1
	Skew	2	Setup time	1
			Skew	1
	Total	10	Total	8

and cost will be excessive if consistency is maintained at the granularity of relatively small cache lines. It is not easy to provide consistency at this granularity because of the excessive intermodule traffic and bottlenecks caused by synchronization and serialization requirements. We expect consistency will be even more difficult to maintain as faster processors with multilevel caches are introduced. Software techniques that keep caches consistent at a coarser granularity[10] may be a workable solution, although it is not yet clear how effective they can be for different parallel applications.

Nonuniform memory-access times impose a challenge in the design of operating systems and parallel applications. To exploit the performance potential of a system like Hector, memory, I/O, and processors must be managed to minimize the average memory-access costs by reducing the number of remote memory accesses. The memory-management subsystem can replicate or move pages to bring them closer to the processes accessing them, but must prevent excessive paging overhead. The scheduler can place processes close to the data they are accessing, simultaneously attempting to balance the load on the processors. Hector's raw I/O capacity is large because each PM has I/O capabilities, and I/O devices can be attached directly to the station bus. But this capacity cannot be exploited if all I/O traffic must traverse the system. It is, therefore, important to localize I/O accesses.

We are expanding our Hector prototype, which now consists of several stations connected by a local ring. Two research groups at the University of Toronto have developed operating systems to support Hector and allow it to run application software. Hector provides an excellent experimental testbed for studying such software issues as processor scheduling, parallel I/O, memory management, and software cache consistency. ■

Acknowledgments

The Hector project has been funded by the Natural Sciences and Engineering Research Council of Canada strategic grant STR0032675. We thank Kennedy Attong, Yonatan Hanna, Jan Medved, Peter Pereira, and Ron Unrau for their substantial work in helping us implement the prototype. We also acknowledge the help of Ben Bacque, Keith Farkas, Carl Hamacher, Ric Holt, Roman Kiss, Orran Krieger, Maitreya Makhopadhyay, Chuck Pilkington, Ken Sevcik, Martin Snelgrove, and Safwat Zaky, who have participated in the Hector project.

References

1. A.H. Karp, "Programming for Parallelism," *Computer*, Vol. 20, No. 5, May 1987, pp. 43-57.
2. J. Dongarra et al., "Special Report: 1989 Gordon Bell Prize," *IEEE Software*, Vol. 7, No. 3, May 1990, pp. 100-104, 110.
3. A. Agrawal et al., "APRIL: A Processor Architecture for Multiprocessing," *17th Int'l Symp. Computer Architectures*, 1990, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 2047, pp. 104-114.
4. J.R. Goodman and P.J. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor," *15th Int'l Symp. Computer Architectures*, 1988, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 861, pp. 422-431.
5. D. Lenoski et al., "The Directory-Based Cache Coherence Protocol for the Dash Multiprocessor," *17th Int'l Symp. Computer Architectures*, 1990, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 2047, pp. 148-159.
6. G.F. Pfister et al., "The IBM Research Parallel Processor Prototype," *Proc. Int'l Conf. Parallel Processing*, 1985, IEEE Computer Soc. Press, Los Alamitos, Calif., pp. 764-771.
7. R.D. Rettberg et al., "The Monarch Parallel Processor Hardware Design," *Computer*, Vol. 23, No. 4, Apr. 1990, pp. 18-30.
8. E.F. Gehringer, D.P. Siewiorek, and Z. Segall, *Parallel Processing: The Cm* Experience*, Digital Press, Billerica, Mass., 1987.
9. D.J. Kuck et al., "Parallel Supercomputing Today and the Cedar Approach, *Science*, Vol. 23, Feb. 1986, pp. 967-974.
10. S. Owicki and A. Agrawal, "Evaluating the Performance of Software Cache Coherence," *Third Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 1989, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 1936, pp. 230-242.

Zvonko G. Vranesic is a professor of electrical engineering and computer science at the University of Toronto. His research interests include computer architecture, VLSI systems, fault-tolerant computing, local area networks, and many-valued switching systems. He has been a senior visitor at the Computer Laboratory at Cambridge University, England, and at the Institute of Programming at the University of Paris.

Vranesic received his BS, MS, and PhD from the University of Toronto in 1963, 1966, and 1968, respectively. He is a member of the Association of Professional Engineers of Ontario and a senior member of the IEEE.

Ron White is a research associate in the Department of Electrical Engineering at the University of Toronto and is a member of the Hector hardware development team.

White received his BA in electrical engineering from the University of New Brunswick in 1984 and his MS in electrical engineering from the University of Toronto in 1988.

Readers may write to the authors at the University of Toronto, Toronto, Canada M5S 1A4.

Michael Stumm is an assistant professor of electrical engineering and computer science at the University of Toronto. His research interests are in computer systems. He manages the Hector project.

Stumm received his diploma in mathematics and PhD in computer science from the University of Zurich in 1980 and 1984, respectively. He is a member of the IEEE Computer Society and the ACM.

David M. Lewis is an assistant professor of electrical engineering at the University of Toronto. His research interests include logic and circuit simulation, computer architecture for high-level languages, logarithmic arithmetic, and VLSI architecture.

Lewis received his BS and PhD in electrical engineering from the University of Toronto in 1977 and 1985, respectively. He is a member of the IEEE and the ACM.