The HP/Convex SPP2000 Scalable Parallel Processor

Greg Astfalk

Hewlett-Packard Company
Convex Technology Center
PO Box 833851
Richardson, TX 75083–3851
astfalk@rsn.hp.com

Abstract

The SPP2000 is the second generation Scalable Parallel Processor (SPP) from Hewlett-Packard. The SPP2000 is capable of scaling from 4 to 512 processors, and from 256 Mbytes to 128 Gbytes of physical memory. The system can also directly support up to 69 Tbytes of disk.

The SPP2000 is a hierarchical SPP, comprised of multiple nodes, with each node having up to 16 processors. The nodes themselves are tightly coupled with a low latency, high bandwidth interconnect. We utilize the latest and most powerful RISC processor from Hewlett-Packard, the PA-8000. All memory is globally shared and fully cache coherent, allowing a familiar, shared-memory programming model. Both global addressing and cache coherency are completely hardware-based. I/O capabilities, both bandwidth and capacity, scale in proportion to the number of processors. The SPP2000 is similar in many respects to its predecessors, the SPP1000 series [2, 3].

While this paper is focused on the hardware of the SPP2000, we offer some discussion of its software. For lack of space this discussion is brief, especially considering the overall importance of software to the end-user.

Architecture

The overall architecture of the SPP2000 is that of a hierarchical SPP. In Flynn's taxonomy [4] it is a MIMD machine. At the lowest level of the processor hierarchy are individual processors. Groups of up to 16 processors constitute a node. A minimal functional system can be as small as a partially populated (<16 processors) node. Within each node the processors are fully connected via a crossbar.

In Figure 1 we show the major components of a single node of SPP2000. The ERAC is the intra-node interconnect: an 8×8 nonblocking crossbar. The EMAC is the memory controller which supports 1 of the 8 memory boards within the node. The ETAC is the interface for the interconnect that ties multiple nodes together. The EPIC is the I/O chip that hosts the peripheral bus to the system. Finally the EPAC is the processor agent which interfaces the PA-8000 processors to the crossbar.

The SPP2000 can offer configurations of up to 32 nodes. Each node has an interconnect path in both the "x" direction and the "y" direction. This permits the nodes to be organized in a ring or torus topology. The aggregate interconnect bandwidth between a pair of nodes is 3.84 Gbyte/second.

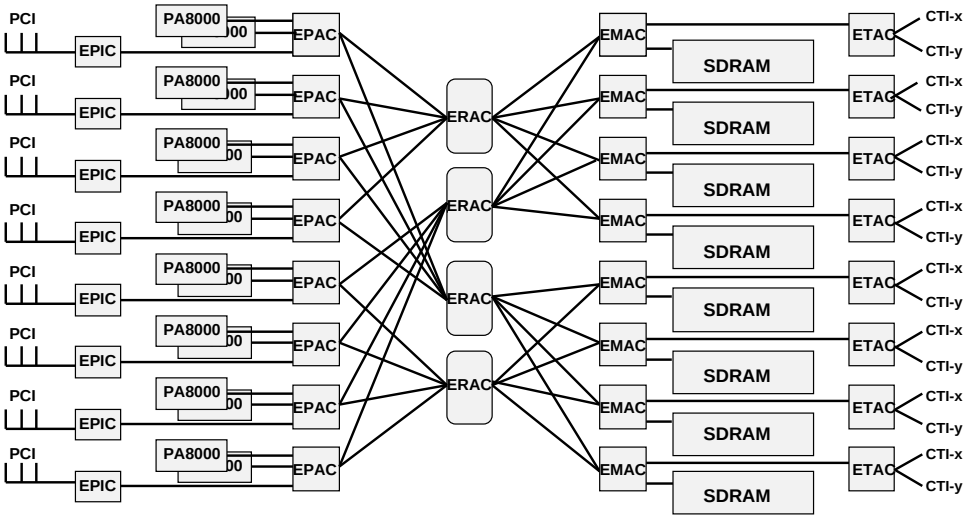

Figure 1: A block diagram of the major components of a SPP2000 node. Excalibur is the internal project name for the SPP2000. EPAC is the Excalibur Processor Agent Chip, EMAC is the Excalibur Memory Agent Chip, ETAC is the Excalibur Toroid Agent Chip, ERAC is the Excalibur Crossbar chip and EPIC is the Excalibur I/O Chip.

The memory is NUMA (NonUniform Memory Access) shared-memory. Common in the literature is the term CC-NUMA, for Cache Coherent, NonUniform Memory Access. The SPP2000 also fits this designation. Each node can have up to 4 Gbytes of physical memory. Thus a fully configured system could support, in 1996, 128 Gbytes of physical memory.

Regardless of the amount of physical memory on the nodes the address space that the user operates within can span the physically distributed memories of all the

nodes. This globally shared-memory will be referred to as GSM in the remainder of this paper. With this architecture the user is given the appearance of a classic shared-memory computer supporting symmetric multi-processing. The implementation of GSM is entirely hardware-based; no software or operating system is involved. The entire memory is deterministically cache-coherent. Coherency is also entirely hardware based. Users can view every byte of the physical memory, as seen by every processor, as a `load/store` device.

The I/O of the SPP2000 is truly scalable. Within each node there can be as many as 24 PCI I/O busses. Each PCI bus is 120 Mbyte/second. Essentially any of the PCI devices can be supported. The I/O channels within the node are connected to the same logic block that supports the processors, the EPAC (see Figure 1). Thus, as processors are added, so is the potential I/O capacity.

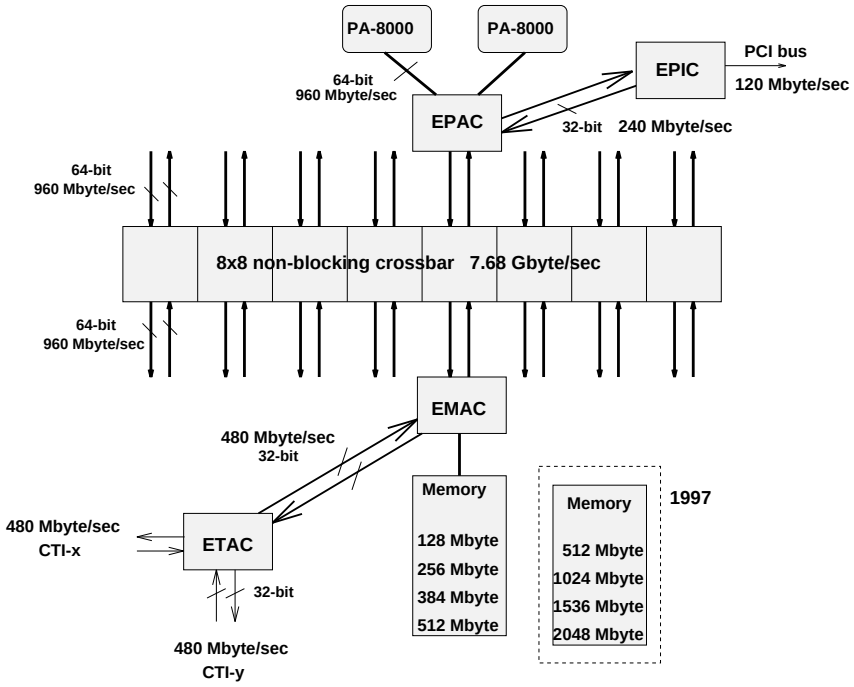

Figure 2: The hardware speeds and feeds for the SPP2000 node. Note that we are only showing the componentry on one of the 8 ports of the crossbar. The "1997" memory board is explained in a later section of this paper.

As a concise picture of the overall "speeds and feeds" of the SPP2000 node we refer to Figure 2. In the following sections we cover each of the major components of the architecture in greater detail.

Processor

The processor used in the SPP2000 is the Hewlett-Packard PA-8000 [5]. The PA-8000 is a 4-way super-scalar RISC processor that is binary compatible with the previous members of the PA-RISC family, which includes the PA-7100 and PA-7200. The PA-8000 operates at 180 MHz, which gives the processor 720 (peak) MFLOPS of performance. Current measurements show that the processor will deliver, in the HP server infrastructure, 11.8 SPECint95 and 20.2 SPECfp95. On the well-known Linpack benchmark the PA-8000 achieves 158 MFLOPS for the 100×100 case and 510 MFLOPS for the 1000×1000 case.

The PA-8000 is the first chip to implement the PA-RISC 2.0 instruction set architecture [6]. The primary purpose of the 2.0 architecture is to support 64-bit integers and 64-bit addressing. Performance increases and new functionality are also an important and integral part of the new instruction set architecture. Some of the new features include variable size pages, new floating-point operations and better branching, in both displacement and prediction.

The PA-8000 is a decoupled architecture. This means that the instruction decode logic is not integrated with the functional unit's pipeline logic. This allows the chip to partially decode instructions independent of, and in advance of, the instruction's actual execution by the functional unit(s). Decoded instructions are staged in queues within the chip. The PA-8000 can have up to 56 decoded instructions in the "ready to execute" queues at any given time.

Owing to space restrictions we can only list, with a brief explanation, the major features of the PA-8000.

Data cache: 1 Mbyte, direct-mapped, 3 clock latency (1 under pipeline conditions), dual-ported, 32-byte line, and write-back.

Instruction cache: 1 Mbyte, direct-mapped, delivers 4 instructions per cycle.

56 entry IRB: The Instruction Reorder Buffer holds decoded instructions that are ready to execute when the data arrives and the resources are available, somewhat akin to a data flow engine.

Out-of-order execution: Since a pool of ready to execute instructions are sitting in the IRB, the processor can launch these whenever the requisite data and functional units are available. This includes execution which is not in strict program order. This helps with latency hiding and keeping the functional units doing useful work.

Branch prediction: A static mechanism of "biasing" the branch instructions and a dynamic mechanism involving historical branching information work to minimize mis-predicted branches.

Speculative execution: The processor will "speculate" which way branch instructions will go, and begin to execute instructions on that path in order avoid stalling the processor's pipelines.

Prefetching: Support for explicit loading of data from memory to cache in advance of its actual usage. Prefetching does not stall the processor.

Outstanding loads: The processor can continue to execute with up to 10 data caches misses outstanding. This is a significant latency hiding feature that increases the aggregate memory bandwidth.

Cache bandwidth: At two loads per cycle this is 2.88 Gbyte/sec at 180 MHz (8 bytes per load).

Floating-point units: There are 2 fully pipelined, and independent, floating-point functional units.

ALUs: There are two independent 64-bit ALUs.

Divide and square-root: Two independent, non pipelined units for square root or floating-point divide.

Shift/merge units: Two independent units.

Load/store units: Two independent units that can issue two loads. two stores, or a load and a store per cycle. Only one store can be retired per cycle.

Register renaming: Each slot of the IRB has a result register to hold its resultant. Data is moved from these rename registers to the architectural registers when the completed instruction is formally retired.

Memory bus: The Runway bus is 960 Mbyte/sec from memory to cache.

The PA-8000 processor incorporates many contemporary features in order to deliver high bandwidth, high computation rates and effective latency hiding. Readers are urged to consult [5] for further embellishment of the brief bullet list items above.

Memory

The memory of the SPP2000 is characterized as physically distributed and logically shared. Common semantics label this as GSM (Globally Shared Memory) or DSM (Distributed Shared Memory). In spite of the physical distribution of the memory among the nodes of the system, any given processor on any node can directly address any byte of memory on any node. Accessing memory is done via the `load` and `store` instructions generated by the processor itself. Thus, all of memory is globally shared. Message-passing is *not* required for inter-processor or inter-nodal communication. Also not required are explicit `gets` or `puts` of globally addressable data. Programmers are afforded the comfortable and familiar view of memory as a load/store device. The inherent hardware of the SPP2000 provides

for memory to be globally shared among nodes.

A very important point related to GSM on the SPP2000 is that it is completely cache-coherent. A user, or a running process, never needs to be explicitly concerned with insuring the consistency of the data in multiple processor caches; the hardware insures coherency.

Within a node there are 8 memory boards. Each memory board is supported by a memory controller which is connected to a single, dedicated port of the intra-node interconnect, the crossbar. The data path from the crossbar to the memory controller is 64-bits wide and operates at 120 MHz (8.33 nanoseconds). Thus the port bandwidth is 960 Mbyte/second. For each node there is 7.68 Gbyte/second of crossbar bandwidth connecting processors and memory.

The memory system and memory boards are designed to permit the use of 16 Mbit SDRAMs or 64 Mbit SDRAMs. In 1996 while 64 Mbit SDRAMs are not yet cost effective, we will use 16 Mbit SDRAMs. This allows a maximum of 4 Gbytes of physical memory per node. Sometime in 1997 we expect the higher density SDRAMs to be cost effective enough to use in the SPP2000. With the use of 64 Mbit SDRAMs the maximum physical memory per node is increased to 16 Gbytes.

The memory within each node is interleaved. Interleaving is a technique commonly used in the classic supercomputer to increase the sustainable memory bandwidth. The principle is quite simple: divide the memory into what are called "banks" and have consecutive physical addresses run across these banks. Since DRAMs take a certain amount of time to refresh, the expectation of the design is that by the time the consecutive memory requests come back to the original bank it will have had sufficient time to refresh and it will be ready to deliver another datum. Each memory board contains four separate banks. Thus, for the 8 boards, there is a 32-way interleaved memory. Since the PA-8000 always loads a cache line of 32-bytes, the "width" of the bank is sized at 32-bytes. This means that each memory access to the bank retrieves a processor cache line's worth of data from that bank alone. This degree of interleaving is unheard of in classic workstation or server design. It is a distinct differentiator of the SPP2000 over these other types of systems.

The memory latency for a read of local memory is 0.54 μsecond. This time gets a 32-byte cache line from memory into processor cache and the requested operand into a register. This time is the most conservative time (i.e., it is an honest number) from the user's perspective. This latency is from the time of the dispatch of the `load` instruction until the receipt of the data in the specified register. Often vendors cite numbers which do not include the delays that occur

within the chip. For modern RISC architectures there can be ($\mathcal{O}(10)$) processor cycles to get the data from the processor pins to the register. We are accounting for this in the latency number that we report.

The latency to remote memory is 1.6 μseconds. As we did for the local memory latency we use the most conservative number. The 1.6 μseconds returns the data, a 32 byte cache line, from a remote node's memory, encaches it in the CTI cache (this is discussed in a later section of this paper) portion of the local memory, encaches it in the requesting processor's cache and places the requested operand into a register.

For *both* local and remote memory references the entire memory subsystem is pipelined. There is no place, other than busy bank waits, in the memory system that is "single-threaded." To appreciate the effect of this we can consider a series of `load` instructions. The first load instruction requires the latency times we detailed earlier for local and remote memory: 0.54 and 1.6 μseconds respectively. The second load instruction, owing to the pipelined memory system, and the ability of the PA-8000 to support multiple outstanding requests, completes in an additional 50 nanoseconds after the first. This means that from the dispatch of the first load instruction until receipt, in the register, of the second load instruction's data, 0.59 μseconds transpires. Likewise there is an additional 50 nanoseconds between the completion of the second and third load instructions. Note that this applies to both local and remote memory requests. We can fairly view the memory system as supporting a series of data items flowing with a 50 nanosecond separation in time.

If we take a processor-centric perspective, the combination of the PA-8000's abilities (in particular the 10 outstanding data cache misses), and the memory subsystems capabilities allows the processor to *sustain* a memory to cache bandwidth of approximately 510 Mbyte/second from local memory. If all the data are remote, the SPP2000 can sustain a remote memory to cache bandwidth of 200+ Mbyte/second.

Internode Interconnect

The SPP2000 can be comprised of multiple nodes. Topologically the nodes can form a ring or a torus. This requires interconnection paths in both the x-direction and the y-direction, for the torus topology (see Figure 3). Specifically there is an x-direction and a y-direction interconnect path supported on each of the 8 interconnect controllers (the ETAC) in each node. Thus every node has 8 independent data paths to each of its neighbors in both the "horizontal" and "vertical"

directions.

The acronym for the interconnect is CTI (Coherent Toroidal Interconnect). The CTI is based on the Scalable Coherent Interface (SCI) [1] which is an IEEE standard. Each of the CTI links is 32-bits wide and their frequency is 120 MHz (8.33 nanoseconds). Each link thus supports 480 Mbyte/second. Given that there are 8 such paths, the interconnect bandwidth, between any pair of nodes is 3.84 Gbyte/second.

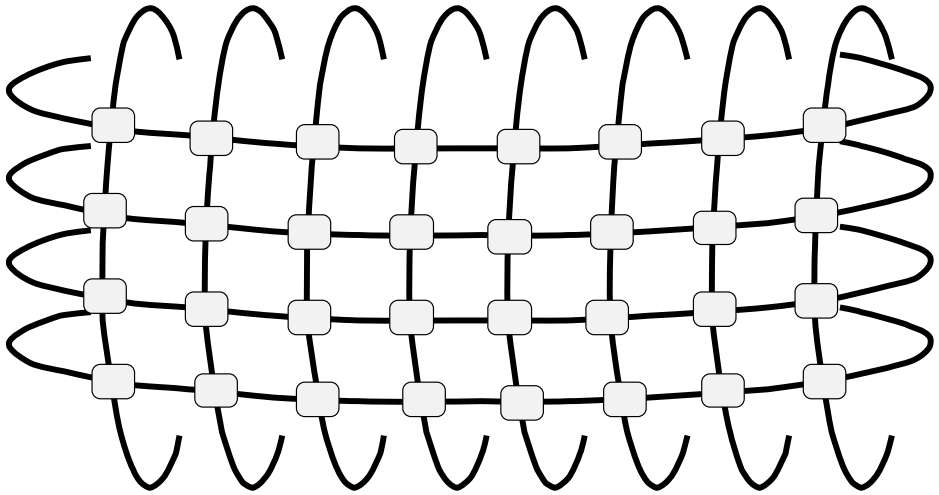

Figure 3: The torus topology of the SPP2000 "wall." Note that each block in this figure represents a node containing up to 16 processors. Each path that is connecting nodes in this picture is actually 8 separate CTI links.

If we consider a single interconnect, say the x-direction, then it can be considered as a unidirectional ring. The ring is actually comprised of the 8 independent data paths from each of the eight interface controllers (ETAC) in each node. Each link between a pair of nodes for each of the 8 paths is completely independent from all others. Thus we call this a series of point-to-point interconnects. All 8 data paths are moving data in the same direction; the interconnect is *not* bidirectional. For data to move from a given node to its downstream neighbor it is necessary to travel around the ring. We will explain more about this shortly.

Consistent with SCI the CTI uses a split transaction protocol. An example serves best to illustrate this. Consider that a specific node requests a cache line from a remote node. The first use of the interconnect is to send a request packet out over the interconnect. (Without loss of generality we assume both nodes are on the same ring). The request packet travels over the ring, passing through intervening CTI controllers on the non-target nodes in the path, until the target node sees

and removes the packet from the interconnect. The memory subsystem on the target node then undertakes the actions needed to retrieve the cache line from its local memory. When the target node has retrieved the cache line, it constructs a response packet that contains the data. This response packet, with 32 bytes of data, is placed on the interconnect and is sent to the requesting node. The "split" is that the request packet is separate and distinct from the response packet.

The split transaction protocol also removes the notion of "nearest-neighbor" from the topology. The length of the request path plus the length of the response path is constant, regardless of the spatial relationship between the two nodes in the topology. The latency for interconnect related memory references was discussed earlier. To reiterate, it is 1.6 μseconds for a remote memory reference.

Each CTI controller can support up to 32 outstanding requests. This is necessary to support the PA-8000's ability to have multiple outstanding load requests. This is substantially different from the SPP1000/1200/1600 in which the CTI controller only supported a single outstanding request to remote memory. For a given node, the set of 8 CTI controllers can support 256 outstanding remote memory load requests. The selection of which ring is used to carry interconnect traffic is based on the address being requested.

The selection of which of the 8 links to use for resolving a remote address request is based upon the address itself; this is very similar to the interleave, or bank, selection for local addresses.

I/O

The i/o of the SPP2000 is scalable in proportion to the number of processors, in both aggregate bandwidth and capacity. All I/O is based on the PCI (Peripheral Channel Interconnect) standard. In the remainder of this section, we detail the I/O system of the SPP2000.

As previously discussed there are, within each node, 8 agent chips. Each agent chip supports an I/O channel which is 32-bits wide and is, at its clock rate of 120 MHz, rated at 240 Mbyte/second. Thus each node offers 1.92 Gbyte/second of I/O channel capacity.

On the "external" side of the I/O channel is the EPIC chip. The EPIC chip takes the 240 Mbyte/second I/O channel feed and converts it to 120 Mbyte/second and to the 32-bit PCI protocol. The PCI bus from the EPIC chip can support 3 PCI controllers. Thus an individual node can support up to 24 PCI controllers.

The disk controllers that we will use are Ultra-SCSI-II which has a peak bandwidth of 40 Mbyte/second. Theoretically, Ultra-SCSI-II can support up to 15 disks per controller. For performance reasons we will limit this to something closer to 10 disks per controller.

For disks we will be using SCSI-Fast&Wide in the 3.5 inch format, with 9 Gbyte capacity. We also support the SCSI-Fast&Wide 5.25 inch format disks, with a capacity of 9 Gbytes.

Since there are 3 PCI controllers per PCI bus and 8 PCI busses per node, each node can support 2.16 Tbytes of spinning media, using ten 9 gigabyte disks per controller. This is in absence of any other type of peripheral devices since the 2.16 Tbytes would consume all of the available slots on the PCI busses.

Striping of file systems (RAID-0) can be implemented across the PCI controllers and across the I/O channels on the agent chips. This will permit highly striped file systems in order to offer high aggregate I/O bandwidth disk access.

Coherency

Earlier in this paper we stated that the memory of SPP2000 was fully coherent. In this section we will offer enough details to convey an idea of how coherency is accomplished in the presence of physically distributed, but globally shared, memory.

A word about the importance of coherency is in order. Purely distributed-memory machines dictate a programming method called message-passing. All data transfers and synchronizations between multiple processors are done via messages sent between the processors over their interconnection. All messages are explicitly accomplished by user-written code. This is a well accepted and powerful parallel programming technique. However, it is tedious and more time-consuming for the programmer than the simpler case of a shared-memory programming style. In the case of shared-memory, the compiler can accomplish much of what the programmer needed to do in the case of message-passing. Shared-memory also permits existing codes to be more easily ported. If you have a shared-memory machine that is not coherent, then the onus is on the user to insure the consistency of the data. The resulting code looks very much like message-passing code with different semantics. A true coherent shared-memory machine is simply easier to program.

To achieve cache coherency, the SPP2000 employs a distributed cache directory. This directory is comprised of a set of "tag-bits" that can be thought of as being

alongside the memory. It is called "distributed" since the tag-bits are present for all of the physical memory. The tag-bits do not subtract from the user-addressable memory. A system with, for example, 1 Gbyte of physical memory on a node will actually have more than this amount. The additional memory, above and beyond, the 1 Gbyte will be tag-bits.

The tag-bits contain information about the state of the cache line they are associated with. The tag-bits are always examined, and possibly updated, when memory references are made by any of the processors in the system. Some of the tag-bits are used to indicate the intra-node coherency of the cache line, others are used to specify the inter-node coherency. If the processors involved in sharing memory are within a single node, then the interconnect is never involved in maintaining cache coherency. This is a result of the SPP2000 having a "two-level" cache coherency mechanism. One level is the sharing vector within the tag-bits for all intra-node coherency. The second level is the sharing list formed in the tag-bits for inter-node coherency.

Inter-node coherency is specified in the tag-bits by constructing a linked list, also called the sharing list (not to be confused with the sharing vector), of the node numbers that contain a copy of the cache line in question. In the above example, when node Y sent the cache line over the interconnect to node X, it updated some of the tag-bits associated with that line to point to node X. When node X received the line into its CTI cache it updated some of the tag-bits associated with that cache line. Again by examination of the tag-bits devoted to the inter-node coherency, the memory subsystem can insure coherency.

On the SPP2000, a configurable portion of the physical memory is allocated, by the operating system, to CTI cache. The simplest view is that the interconnect cache performs the same function as a processor cache only it does so on a nodal basis. This is best illustrated by an example.

Assume that some processor on node X references, via a `load` instruction, an address that is mapped to the physical memory on node Y. The agent chip, in conjunction with the memory controller and interconnect chip, will perform the requisite actions to send a request out over the interconnect from node X to node Y requesting the target cache line. The memory subsystem of node Y will send a copy of the cache line over the interconnect to node X. When the cache line arrives at node X, it is copied into the CTI cache portion of the physical memory and it is propagated to the requesting processor's cache. At this time, two copies of the cache line exist within the memory of the SPP2000. This can be generalized to as many nodes as desired. If any processor on node X references an address that is within the cache line that is now in the CTI cache, then the data is taken from the CTI cache copy rather than going over

the interconnect again. Thus the interconnect cache acts as a dynamic repository for data reuse. If a remotely referenced cache line has been copied to the CTI cache and is subsequently addressed, it is found locally with the latency of a local memory access.

Packaging

The SPP2000 node is packaged into a single chassis. Within the chassis is included everything associated with the node; the processors, memory, PCI cards, power supplies, and some disks. The chassis size is approximately 29×36×35 inches (74×91×89 centimeters). Recall that a single node is up to 11.5 GFLOPS of peak performance with up to 4 Gbytes of physical memory. Within the node there can also be 162 Gbytes of disk. Naturally more disk can be configured in separate cabinetry.

The nodes can be stacked two high, which is the maximum stack height allowed. A stack of two nodes is a cabinet, see Figure 4. At the bottom of the cabinet is some room for casters to allow for its easy movement. The casters and their associated cover adds a few inches to twice the height of the individual chassis. The overall height of a 2-node cabinet is 71.5 inches (182 centimeters).

When fully populated, the node chassis weighs approximately 450 pounds (205 kilograms). The fully configured power requirement for a node is 6900 watts, using 30 amp service. The node is air-cooled by fans within the chassis.

Operating System

The SPP2000 is a fundamentally different architecture from the HP workstations and servers. In spite of this, the operating system offers the user the "look and feel" of HP-UX. The operating system on the SPP2000 is called SPP-UX. It is important to note that the SPP2000 does *not* run HP-UX. The architecture of SPP-UX is micro-kernel based. For those programmers that write code utilizing system calls, SPP-UX is compatible with those of HP-UX.

There are actually a number of levels or types of compatibility the SPP2000 offers with respect to HP-UX. As mentioned the system calls are compatible. Users can actually take executables off HP workstations and servers running HP-UX and execute them under SPP-UX. There are some very minor exceptions to this. Seldom does an end-user encounter these corner-conditions. The compilers under

Figure 4: A two-high stack of SPP2000 nodes, which would be a 32-processor system. Note that the "skins" are molded plastic.

SPP-UX can utilize object files that were created with the HP-UX compilers. Finally, the Unix level commands are the same as those under HP-UX.

Generally a user will perceive no difference when working on an SPP2000 or a HP server.

Parallel Programming

Since the SPP2000 is a parallel machine we offer a number of means to use and exploit parallelism. The means to achieve parallelism centers around the compilers, Fortran and C, and several libraries. The user can approach parallel programming on the SPP2000 in a number of ways. In the general order of increasing user involvement and effort we have:

- Vanilla source code compilation (sequential) — does not achieve parallelism but many scalar optimizations are applied to improve single processor performance.
- Vanilla source code compilation (automatic parallelism) — the only effort on the part of the user is to set a compiler flag and the compiler will, when possible, generate parallel code for loops and loop nests.
- Explicit use of directives/pragmas — the user inserts directives (in Fortran) or pragmas (in C) to direct the compiler to generate parallel code. These directives are quite simple but powerful, covering things such as parallel loops, parallel tasks, parallel regions, critical sections, and much more.
- Message-passing — the SPP2000 offers highly optimized implementations of both PVM and MPI. The notion here is that there are multiple processes communicating through explicitly sent, and received, messages.
- Explicit use of threads — this is explicit parallel programming where the user codes calls to invoke parallelism, synchronize threads, etc.
- Hierarchical parallelism — the compilers and parallel libraries on the SPP2000 offer the ability to have several concurrent levels of parallelism in operation at the same time. For example, node-way and thread-way, message-passing processes that are using explicit threading, and message-passing processes that are node-way and thread-way parallel.

It is generally easy to invoke parallelism in an application via the mechanisms described above. Parallel applications, using PVM or MPI, that exist on other parallel platforms can be trivially ported to the SPP2000. Those applications involving other vendor directives or explicit threading are reasonably easy to port to the SPP2000.

Support Tools

To complement the compilers and the parallel programming task, the SPP2000 offers three support tools which are all "parallel aware." These are a debugger, a profiler and a tracer.

CXdb is the debugger. CXdb allows the user to independently debug multiple threads of a multi-threaded application. This is accomplished from within a GUI that can control any number of threads from within a single window. Additionally, CXdb offers the user the ability to perform accurate debugging of optimized code. That is, CXdb understands a bijective map between source statements and the optimized location(s) of the corresponding machine instructions. Thus the user can exert complete control over any part, or parts, of an optimized, parallel application.

CXpa is the profiling tool for the SPP2000. CXpa allows the user to profile not just routines, but also loops, loop nests, and even basic blocks. CXpa is not a statistical-based profiler but rather it instruments the code with calls to counting and timing routines with the result of very precise characterization.

CXpa permits gathering more than just processor time. Other important metrics such as wall-clock time, cache misses, average memory access latency and others can be obtained. CXpa offers the ability to gather and display this information on an individual thread basis. Integrated two- and three-dimensional graphics offer an easy to comprehend visual presentation of the performance metrics, with source code click-back.

CXtrace is a tracing tool that enables the user to see the temporal behavior of the parallel application. CXtrace is primarily intended for message-passing applications. The user can pan and zoom on the time-line (x-axis) and see the various states the message-passing processes are in, as well as the messages that are passed between them. CXtrace also offers the ability to click on the message lines to see the detailed information about the individual messages. Like CXdb and CXpa, CXtrace is largely point-and-click driven.

Summary

The HP/Convex SPP2000 is a scalable parallel system that incorporates the latest and most powerful RISC processors. The SPP2000 offers up to 370 GFLOPS of peak performance as well as 256 Gbytes of physical memory and an aggregate of up to 31 Gbyte/second of i/o bandwidth to support as much as 70 Tbytes of spinning media. The infrastructure of the system is designed with parallelism and high-performance in mind. While achieving this goal it has also maintained compatibility with the desktop.

Parallelism is offered to the user in the hardware, operating system, compilers and support tools. There are automatic parallelizing compilers and parallel-aware support tools, such as debuggers, profilers and tracers. A large number of the most important third-party application codes are available and tuned to the architecture.

In the space we have for this paper we tried to cover the most important aspects of the SPP2000. Readers that are interested in more details of the hardware or the software, of which there are many, are urged to contact either this author, or their local Hewlett-Packard representative.

References

[1] ANSI/IEEE Std 1596–1992, IEEE Standard for Scalable Coherent Interface (SCI), available from the Institute of Electrical and Electronics Engineers, Inc., Service Center, 445 Hoes Lane, P.O.Box 1331, Piscataway, NJ 08855–1331, 800–678–4333.

[2] T. Brewer, *A highly scalable system utilizing up to 128 PA-RISC processors*, Proceedings of COMPCON'95, IEEE Computer Society Press, Los Alamitos, CA, pp. 133–140, March, 1995.

[3] *Convex Exemplar Architecture*, Second Edition, Convex Computer Corp., Richardson, TX, Order No. DHW–014, November, 1994.

[4] M.J. Flynn, *Some computer organizations and their effectiveness*, IEEE Transactions on Computers, **C-21**(1972), pp. 948–960.

[5] D. Hunt, *Advanced performance features of the 64-bit PA-8000*, Proceedings of COMPCON'95, IEEE Computer Society Press, Los Alamitos, CA, pp. 123–128, March, 1995.

[6] G. Kane, *PA-RISC 2.0 Architecture*, Prentice Hall, Upper Saddle River, 1996.