# The Sun Fireplane System Interconnect

Alan Charlesworth
Sun Microsystems, Inc.
alan.charlesworth@sun.com

## Abstract

System interconnect is a key determiner of the cost, performance, and reliability of large cache-coherent, shared-memory multiprocessors. Interconnect implementations have to accommodate ever greater numbers of ever faster processors. This paper describes the Sun™ Fireplane two-level cache-coherency protocol, and its use in the medium and large-sized UltraSPARC-III-based Sun Fire™ servers.

## 1. Introduction

Large cache-coherent, shared-memory servers have become a big business. Sales of shared-memory systems with a capacity for more than eight processors have more than doubled over the past five years to over $16 billion in 2000 [1]. This was 27% of the $60 billion-a-year total server market. In the $29 billion Unix subset of the server market, systems with room for more than eight processors account for 40% of the revenue. Figure 1 shows the sales trends of large shared-memory servers over the most recent five years.

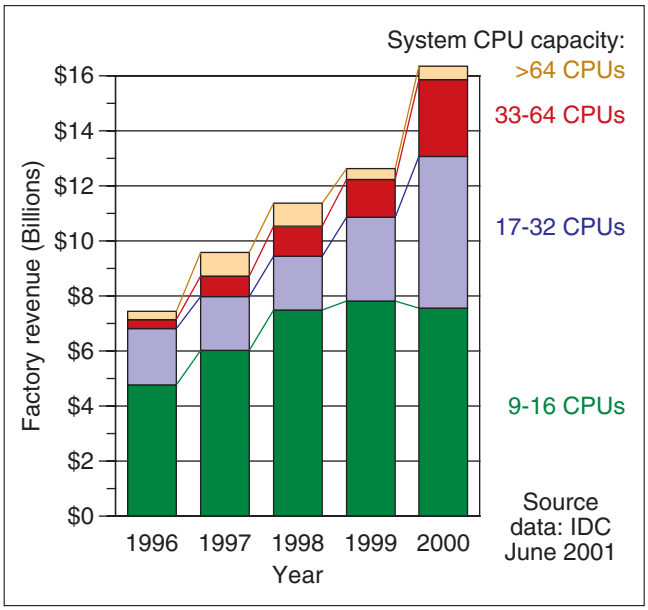


**Figure 1. Sales of large shared-memory servers**



SC2001 November 2001, Denver
(c) 2001ACM 1-58113-293-X/01/0011 $5.00

Several vendors now offer large shared-memory servers. SPARC/Solaris systems have scaled to 64 processors since 1993. The IBM pSeries 680 [2] scales to 24 processors, the Compaq AlphaServer GS-series [3] scales to 32 processors, the HP Superdome [4] scales to 64 processors, and the SGI Origin 3000 [5] scales to 512 processors. These are all Unix systems with RISC processors.

### 1.1 Cache coherency

The choice of a cache-coherency scheme is the most important design decision for a coherent shared-memory interconnect. Compared to maintaining cache coherency, moving data is easy.

The cache-coherency protocol keeps each processor's view of memory consistent. Coherency is maintained on aligned blocks of memory, called cache lines, which are typically between 32 and 128 bytes wide. Sun currently uses a 64-byte cache block.

Cache misses are satisfied from memory, unless a system device (processor or I/O controller) has modified the cache line. To do a write, a processor has to become the *owner* of the cache line. All other system devices invalidate any shared copies they have cached, and the current owner supplies the data. Henceforth, when other processors request to share a read-only copy of the data, the owning processor, not memory, will supply the data. Memory becomes the owner again when the owning processor needs to make room in its cache for new data, and it *victimizes* the cache line by writing it back to memory.

### 1.2 Cache coherency types

There are two basic types of cache coherency: *broadcast*, and *point-to-point*.

**1.2.1 Broadcast (snoopy) coherency.** All addresses are sent to all system devices. Each device examines (snoops) the state of the requested cache line in its local cache, and the system determines global snoop result a few cycles later. Broadcast coherency provides the lowest possible latency, especially for the cache-to-cache transfers that are common in transaction processing. The data bandwidth of a snoopy system is limited by the snoop bandwidth to:

Bandwidth = Cache line width × Bus clock / Clocks per snoop

**1.2.2 Point-to-point (directory) coherency.** Each address is sent only to those system devices that are known to be interested in that cache line. The hardware keeps a *directory* in memory or in special RAM to track which system devices share or own each cache line. Since every address is not sent everywhere, the total bandwidth can be much higher than with broadcast coherency. However, the latency is longer and more variable, due to the more complicated protocol. Directory coherency is usually used only in large systems, where more bandwidth is needed than snooping can provide.

See chapters 6 and 8 of [6] for more on these two cache-coherency mechanisms.

**Table 1. Sun system interconnect generations**

| System interconnect generation | 1. MBus [7] | 2. XDBus [7] | 3. Ultra Port Architecture (UPA) [8] | 4. Sun Fireplane |
|---|---|---|---|---|
| First mid-size system shipments | 1991 | 1993 | 1996 | 2001 |
| Processor | Cypress SPARC | SuperSPARC | UltraSPARC-I/II | UltraSPARC-III |
| Maximum processors in a system | 4 | 64 | 64 | >64 |
| Processor clock | 40 MHz | 40–60 MHz | 167–400 MHz | ≥750 MHz |
| System clock | 40 MHz | 50–55 MHz | 80–100 MHz | 150 MHz |
| Cache-coherency mechanism | Broadcast | | | Broadcast + point-to-point |
| Packet protocol | Circuit switched | Packet switched | | |
| Address and data | Multiplexed on same wires | | Separate wires | |
| Cache coherency line size | 32 bytes | 64 bytes | | |
| System clocks per snoop | 16 | 11 | 2 | 1 |
| Max snoop rate per address bus | 2.5 million/sec | 4.5–5 million/sec | 40–50 million/sec | 150 million/sec |
| Max data bandwidth per address bus | 0.08 GBps | 0.29–0.32 GBps | 2.5–3.2 GBps | 9.6 GBps |
| Max number of address buses | 1 | 4 | 4 | 18 |
| Max address-limited data bandwidth | 0.08 GBps | 1.28 GBps | 12.8 GBps | 172 GBps |
| Datapath width | 8 bytes | | 16 bytes | 32 bytes |
| Interconnect implementation | Bus | Buses | Mid-range: Buses<br>High-end: Switches | Switches |
| Note: 1 GBps (gigabyte per second) = $10^9$ bytes per second | | | | |

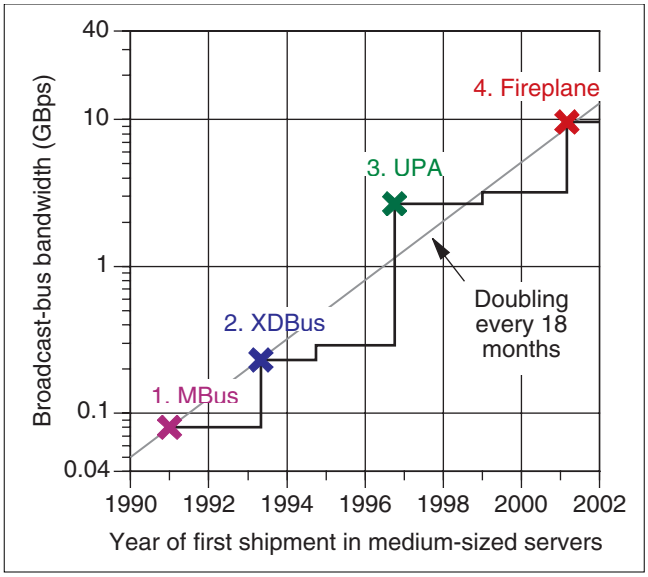


**Figure 2. Bandwidth of a broadcast bus in Sun's four interconnect generations**

## 2. Sun system interconnect generations

Sun Microsystems has been shipping shared-memory multiprocessors since the early 1990s, as summarized in Table 1. The Fireplane interconnect described in this paper is Sun's fourth generation of shared-memory system interconnect. Every three to four years Sun has introduced a new SPARC processor core together with a new system-interconnect architecture.

Sun has emphasized the bandwidth of broadcast coherency. In each of Sun's system generations, a single broadcast address bus has been the foundation for Sun's small and medium-sized servers.

Figure 2 shows the bandwidth of a single broadcast bus in Sun's four interconnect generations. Sun has increased its broadcast-coherency bandwidth from 0.08 GBps to 9.6 GBps over the last ten years — a scaling rate about the same as Moore's Law (2x every 18 months). This has been accomplished by a combination of factors: 3.7x faster system clock, 2x wider cache lines, and 11x more efficient coherency transactions.

To provide more system bandwidth than one broadcast bus can supply, Sun has put multiple snoopy buses into its largest systems since 1993: four in the XDBus and UPA generations, and 18 in the Fireplane generation.

## 3. Fireplane coherency protocol overview

The Sun Fireplane protocol is used by all systems built from the UltraSPARC-III processor. The Fireplane interconnect is an enhancement of the previous-generation Ultra Port Architecture (UPA) [8]. Like the UPA, the Fireplane architecture has a separate address and data interconnect to keep address and data transfers from

interfering with each other. The address interconnect disseminates read and write requests, and maintains cache coherency. The data interconnect moves 64-byte blocks from source to destination.

The major new feature of the Fireplane interconnect protocol is two levels of cache coherency: broadcast and point-to-point (see Figure 3). Logic is built into the UltraSPARC-III processor chip and I/O controller ASICs to handle both levels of the Fireplane coherency protocol. Using both types of coherency, Sun can build a wide range of system sizes from the same CPU/Memory board and same I/O controller ASIC:

- Mid-sized Fireplane systems use a single *snooping coherence domain*, which spans all the devices connected to one Fireplane address bus. Snoopy coherency gives the 24-processor Sun Fire 6800 quite low memory latency for a mid-sized system. Other recent server families (Compaq GS-series [3], HP Superdome [4], and SGI Origin 3000 [5]) all use directory coherency for their mid-sized servers.
- Large Fireplane systems (with >24 processors) use *multiple* snooping coherence domains. This protocol is called *Scalable Shared Memory* (SSM), since it is how large shared-memory systems are implemented in the Fireplane generation. Large systems retain the low latency of snoopy coherency for local memory accesses, and still have the bandwidth scalability provided by directory coherency.

This paper describes SSM as it is used in a large single-cabinet system, where the SSM transactions are implemented by wide paths across an active centerplane. This system is scheduled to be announced in the fall of 2001, after this paper has gone to press. The SSM protocol could also be used with other interconnect implementations.

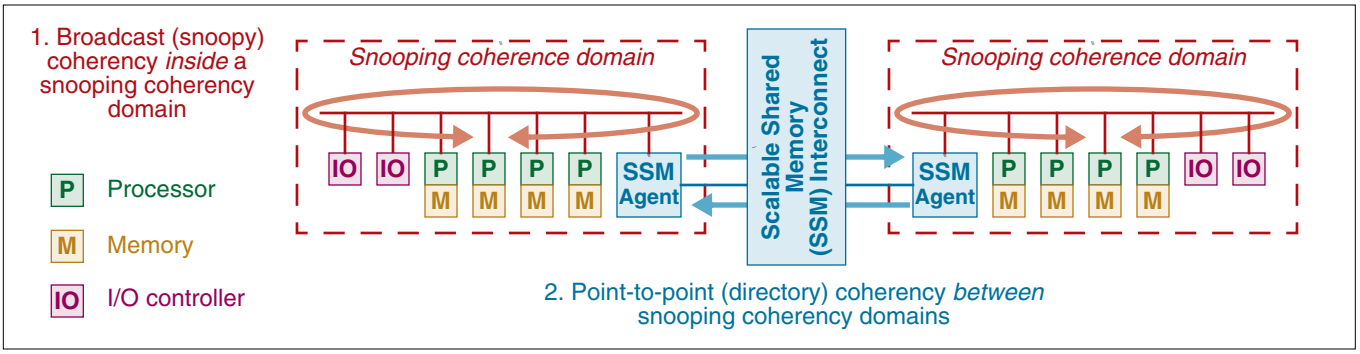


**Figure 3. Two-level coherency protocol**

## 4. Fireplane broadcast coherency

All Fireplane systems have at least one snooping coherence domain. Within a snooping coherence domain, transactions are snooped by all devices. Snoop results are output by the devices five cycles after they receive an address. The overall snoop result is combined by ASICs on each board, and is sent back to the devices four cycles later.

### 4.1 Address bus implementation

The address bus is implemented using a two-level bidirectional tree-structure of Address Repeater ASICs, as shown in Figure 4. CPU 0 issues an outgoing request (indicated by the solid red arrows) in cycle 0 to its board Address Repeater (AR0). Assuming that the bus to the top-level Address Repeater is available, the outgoing request is sent up in cycle 2. In cycle 3, the top-level Address Repeater arbitrates between requests, and selects this request to send back down to the board-level Address Repeaters.

In this example, the request is physically transmitted (indicated by a dashed green arrow) only to AR1, since AR0 sent the original request and kept a copy of it, so it only needs to be told when to use the copy. Thus, in cycle 4 the AR0 to AR2 bus is not physically used (indicated by the dotted green arrow). In cycle 6 all the system devices see the same transaction. CPU 0 sent the original request and therefore does not need a physical re-transmission of the address (indicated by the dotted green arrow). The incoming path length (green dashed or dotted) is a fixed length for all devices on the bus. The outgoing path (red) may hold queued transactions.

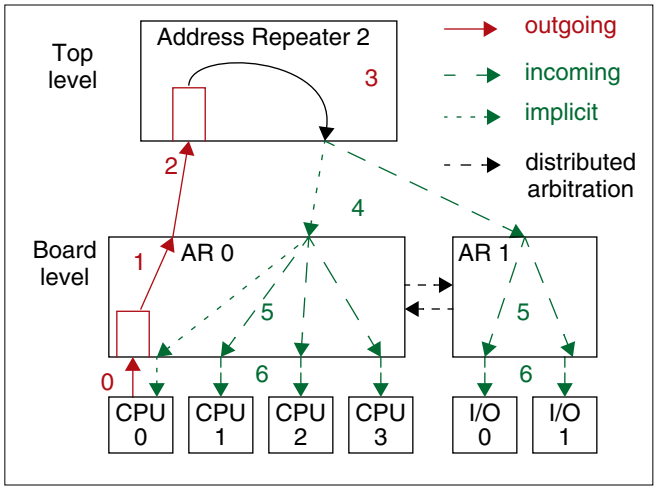


**Figure 4.** Address bus implementation

The board-level Address Repeaters use a distributed arbitration scheme among themselves to "mirror" the arbitration algorithm used by the top-level Address Repeater.

### 4.2 Snoop signals

There are three snoop-result signals: *shared*, *owned* and *mapped*:

- **Shared** is asserted by a system device that has a copy of the data. If the line is not shared, then the requesting device may place the line in a state that allows the device to write to it.
- **Owned** is asserted by a system device that "owns" the data, and may have modified it. It indicates that the data obtained from memory is not valid.
- **Mapped** is asserted by a memory or I/O controller to indicate that the request is directed to a valid memory address or I/O device.

These *snoop-out* signals are combined for the entire snooping coherence domain by the Data Switch ASICs on each board, and returned back to the system devices four cycles later as *snoop-in* signals.

### 4.3 Snoop tags

Each device has two sets of tags: Dual Tags (Dtags) and Cache Tags (Ctags). The Cache Tags represent the actual state of the data in the cache and consequently they transition with the data transfer or data modification. The Dual Tags are used for snooping and transition with transaction requests independently of any data transfer associated with the request.

**4.3.1 Cache Tags.** The Cache Tags have five states:

- **cM: Cache Modified**. Line is valid, exclusive, and (potentially) dirty in cache.
- **cO: Cache Owned**. Line is valid, (potentially) dirty and (potentially) shared.
- **cE: Cache Exclusive**. Line is valid, exclusive, and clean in cache.
- **cS: Cache Shared**. Line is valid, (potentially) shared, and clean in cache.
- **cI: Cache Invalid**. Line is invalid in cache.

*Valid* means that the line contains useful data. *Exclusive* means that no other cache has a copy. *Dirty* means that the data has been modified.

**4.3.2 Dual tags.** The Dual Tags have four states:

- **dS: Dual Shared**. Line is valid and clean in cache.
- **dO: Dual Owned**. Line is valid and (potentially) dirty in cache. Represents the cM, cO and cE states of the CTags.
- **dT: Dual Temporary**. Line is valid, clean and exclusive. This is a temporary state for read to share (RTS) while waiting for snoop input. Any intervening transactions to the line between snoop out and snoop in, or a snoop input of *shared*, will result in a final DTag state of dS, otherwise the final DTag state is dM.
- **dI: Dual Invalid**. Line is invalid in cache.

The Dtags do not exist as a separate physical entity, but instead are combined with the Ctags into a single set of physical tags.

### 4.4 Cache Tag state transitions

Figure 5 shows the transitions between the five MOESI Cache Tag states.

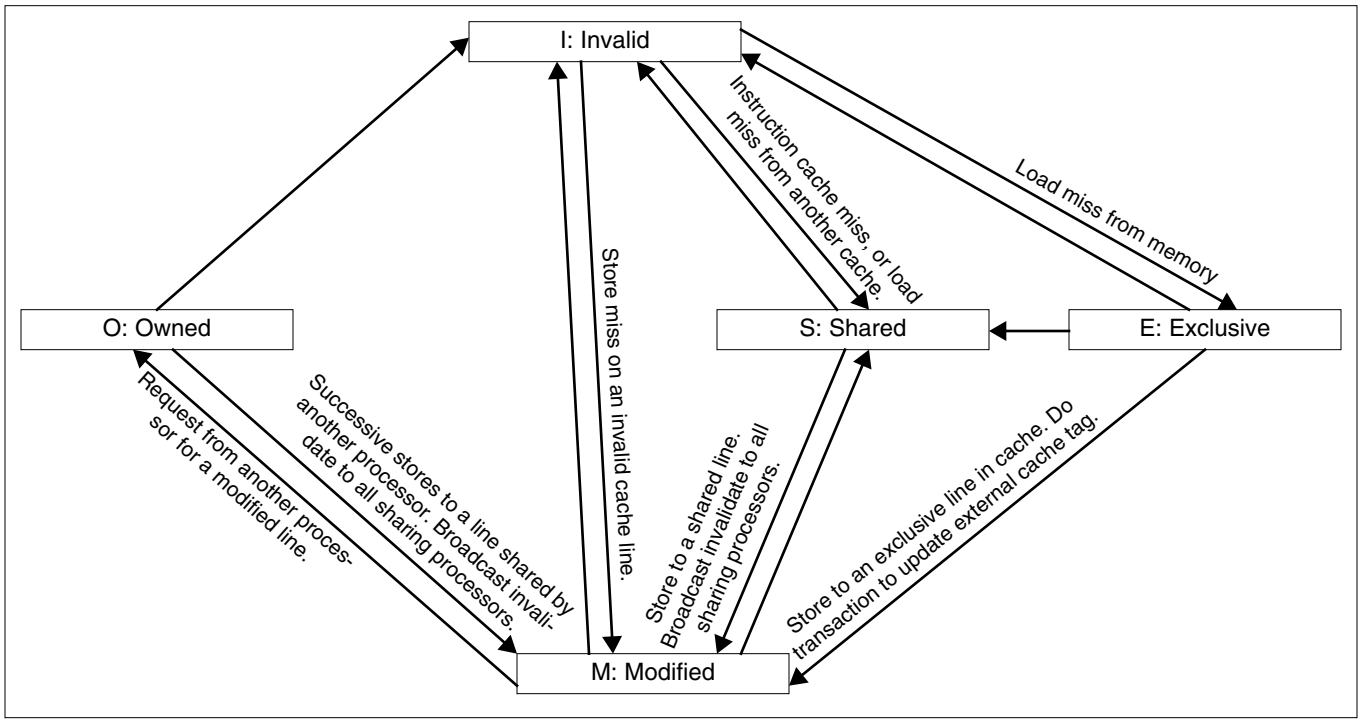


**Figure 5. Cache tag state transitions**

## 5. Fireplane point-to-point coherency

Fireplane systems with more than 24 processors are organized into multiple snooping coherence domains. An SSM agent in each snooping coherence domain forwards requests for non-local data to the *home* SSM agent. Home is where a page of memory is physically located. The home SSM agent keeps track of which snooping coherence domain any sharers are in, or where the current owner is.

The SSM protocol uses tags stored in memory (Mtags*)* to specify the global coherency state of a memory block. The Mtags are stored in the 8-bytes of error-correcting code (ECC) information that goes with each 64-byte memory block. The global coherency information is put in memory because the number of snoop tags that would have been required to represent all the data cached in other snooping coherence domains would have been too large to fit in high speed SRAM. Instead, each SSM agent has a Coherency Directory Cache (CDC) to give it quicker access to the most recent coherency information.

The Mtags have three states:

- **gM: Global Modified**. Line is valid, exclusive, and (potentially) dirty in this snooping coherence domain. Read and write operations are allowed.
- **gS: Global Shared**. Line is valid and clean in this snooping coherence domain and (potentially) shared by other snooping coherence domains. Read operations are allowed.
- **gI: Global Invalid**. Line is invalid in this snooping coherent domain. No operations are allowed.

The Mtag state is in addition to the snoopy MOESI state of a line. For example, in order to do a write, a processor must have a copy of the line that is both *modified* and *globalModified*. If the line is *globalModified,* but not *modified*, it could be promoted to *modified* with a transaction within its snooping coherence domain. If it is not *globalModified*, an SSM transaction will have to be done. Operations that are not permitted by the Mtags are not assigned a place in the global memory order until the SSM agent re-issues them.

Small and medium sized Fireplane systems that don't use SSM have all their Mtags initialized to *globalModified*, and these never change. Since *globalModified* allows both reads and writes, the protocol reduces to the simple snooping protocol.

## 6. Fireplane address transactions

These transactions are sent on the Fireplane address bus to initiate read and write operations. All addresses refer to 64-byte aligned blocks, except for the 16-byte I/O transactions. Each transaction contains an *address transfer identifier* (ATransID) which identifies the requester and this particular address transaction.

### 6.1 Requests for local data

These transactions get data from within a snooping coherence domain.

- **ReadToShare** (RTS). Obtain a cache line for read access. The data is returned by the current owner, if one exists, or the home memory location. Generated by an instruction cache miss, or a data cache load miss.
- **ReadToOwn** (RTO). Obtain an exclusive copy of a line in order to write to the line. It invalidates all other copies of the line. Generated by a data cache store miss.
- **WriteBack** (WB). Update memory with a modified cache line. Generated by a data cache miss when a dirty cache line needs to be victimized (evicted) to make room.
- **ReadStream** (RS). Read a coherent copy of the line, and remove it from the local coherency domain. Generated by a block load instruction.
- **WriteStream** (WS). Stores a block to memory. Generated by a block store instruction.

### 6.2 Requests for remote data

These transactions request the local SSM agent to get data from another snooping coherence domain. They are issued for non-local physical addresses, and for local addresses whose Mtags do not have the necessary permissions. These transactions do not cause any changes in the local snoop state.

- **Remote_ReadToShare** (R_RTS). Request the local SSM agent to get a *ReadToShare* done in another snooping coherence domain.
- **Remote_ReadToOwn** (R_RTO). Request the local SSM agent to get a *ReadToOwn* done in another snooping coherence domain.
- **Remote_WriteBack** (R_WB). Request the local SSM agent to get a *WriteBack* done in another snooping coherence domain.
- **Remote_ReadStream** (R_RS). Request the local SSM agent to get a *ReadStream* done in another snooping coherence domain.
- **Remote_WriteStream** (R_WS). Request the local SSM agent to get a *WriteStream* done in another snooping coherence domain.

### 6.3 Get data for a remote requester

These transactions are used by an SSM agent to get local data on behalf of a remote requester.

- **ReadToShareMtag** (RTSM). This transaction is issued by an SSM agent to obtain a read copy of the line for another snooping coherence domain. RTSM is similar to *ReadToOwn* except that instead of invalidating the line in all other devices, the line is marked shared with the Mtags changed to *globalShared*.
- **ReadStreamRemote** (RSR). This transaction is used by the SSM agent to provide data in response to a *Remote_ReadStream.*

### 6.4 Re-issue remote reads

These transactions are used by an SSM agent to supply the original requester with data obtained from another snooping coherence domain.

- **ReadToShareRemote** (RTSR). This transaction is used by an SSM agent to re-issue a *Remote_ReadToShare* transaction. This supplies the data to the original requester, which becomes the

new owner, and transitions to *modified* or *owned*. All other local devices transition to *shared*.

- **ReadToOwnRemote** (RTOR). This transaction is used by an SSM agent to re-issue a *Remote_ReadToOwn* transaction. This supplies the data to the original requester, which becomes the new owner, and transitions to *modified* or *owned*. All other local devices transition to *invalid*.

### 6.5 I/O transactions

These transactions do operations to I/O devices.

- **ReadIO** (RIO). Read 16-byte block of I/O space.
- **WriteIO** (WIO). Write 16-byte block of I/O space.
- **ReadBlockIO** (RBIO). Read 64-byte block of I/O space.
- **WriteBlockIO** (WBIO). Write a 64-byte block of I/O space.
- **Interrupt** (INT). Send an interrupt.

## 7. Fireplane data transfers

Data packets are transferred in response to an address transaction. The ATransID is used to match the response to the request. For a read transaction, the response is one or two data packets that have a matching *data transfer identifier* (DTransId). For a write transaction, the response is a matching *target transaction identifier* (TTransId).

### 7.1 Read data transfers

For read data transfers, the source of the data first sends the DTransId, followed two cycles later by the data and ECC. The requester must be able to accept this data when it arrives.

### 7.2 Write data transfers

For write data transfers, the destination for the data sends a TTransId and a *target identifier* (TargId) to the sender when it is ready to accept the data. The sender then uses the TargId to send the DTransId, followed two cycles later by the data.

### 7.3 Data packets

Data is organized into 128-bit sub-blocks. Associated with each sub-block is a 9-bit ECC, a 3-bit MTag and a 4-bit Mtag ECC. There are two datapath widths, with a total width (Data + Mtag + ECC + Mtag ECC) of 288 bits (wide path) and 144 bits (narrow path).

A 64-byte data transfer consists of two 32-byte packets for wide devices, and four 16-bytes packets for narrow devices. *ReadIO* and *WriteIO* commands transfer 16 bytes, which takes one transfer packets for both datapath widths.

The DTransId for all data packets associated with the same transaction is the same, so the data packet order determines the position of the data in the block.

## 8. Example Fireplane interconnect operation

This section describes the steps used to implement a coherent read-to-share from memory. This is done to satisfy an instruction-cache miss or a data-cache load miss. These transaction sequences assume that the data is currently unmodified in memory, rather then having been modified (owned) in a system device's cache. The diagrams portray the actual interconnect topology, which can be also be seen in more detail in Figure 9.

### 8.1 Transfer inside a snooping coherence domain

This is the case where the target memory location is in the same snooping coherence domain as the requester. The address is broadcast on the local address bus, and the system devices snoop their caches. The 64-byte data block is transferred over the data switch from memory to the requester.

See Figure 6 for the block diagram showing these operation steps:

1. **Address request** (cycle 0–2). The requesting CPU makes a *ReadToShare* request to its board Address Repeater, which sends the request up to the top-level Address Repeater.

2. **Broadcast address** (cycle 3–6). The top-level Address Repeater chooses this request to broadcast throughout the snooping coherence domain. It sends the request back down to the board-level Address Repeaters. Every system device gets the address on cycle 6.

3. **Snoop** (cycle 7–15). Each system device examines its coherency tags to determine the state of the cache line. There are three snoop-state signals: *shared*, *owned*, or *mapped*. Each device sends its snoop-out result to its Board Data Switch on cycle 11.

   Each Board Data Switch ORs its local snoop results, and sends them to every other Board Data Switch. Each Board Data Switch computes the global snoop result, and sends the snoop-in result to its system devices on cycle 15.

   If the snoop result had been a hit, then a cache-to-cache transfer would have been initiated on cycle 16 by the owning device. In that case, the memory cycle would have been ignored.

4. **Read from memory** (cycle 7–22). The target memory controller (which is inside a CPU) recognizes that the request is in its address range, and initiates a read cycle in its memory unit. In cycle 22, the data block is sent from the DIMMs to the local Dual CPU Data Switch.

5. **Transfer data** (cycle 23–36). The data block is moved through the Board Data Switch, the System Data Switch, the requester's Board Data Switch, and the Dual CPU Data Switch. The four 18-byte portions of the data block arrive on the wires into the requesting CPU in cycles 33-36.

The address interconnect takes 15 system cycles (150 ns) to obtain the global snoop result and send it to the system devices. The DRAM access is started partway through the snoop process, and takes an additional 7 cycles (47 ns) after the snooping is finished. These two latencies have a fixed minimum time.

The data transfer takes a variable number of clocks, depending upon how many data switch levels are between data source and destination. It takes 14 cycles (93 ns) when they are on different boards, 9 cycles (60 ns) when they are on the same board, and 5 cycles (33 ns) when they are on the same CPU.

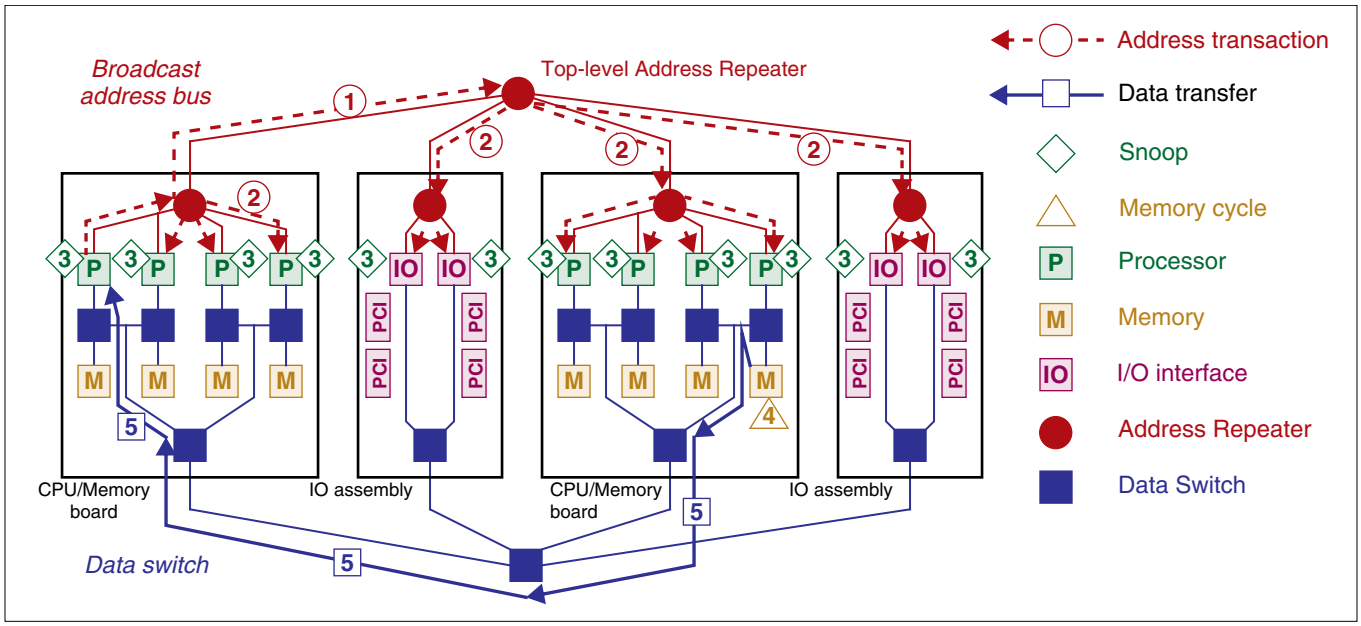


**Figure 6. Read to share operation within a snooping coherence domain**

Memory is usually interleaved across the 16 banks on each board, so the average pin-to-pin latency (from address out of the CPU to all of the data into the CPU) is 197 ns when memory is on the same board, and 240 ns when memory is on a different board.

Back-to-back memory latency, such as would be measured by a pointer-chasing latency benchmark like *lmbench* [10], is typically about 40 ns more than these pin-to-pin latencies.

### 8.2 Transfer between snooping coherence domains

This is the case where the target memory location is in a different snooping coherence domain from the requester. The SSM agent on the requester's local address bus forwards the request to the SSM agent in the home snooping coherence domain via the address crossbar. The home SSM agent broadcasts the request on the home address bus. Either memory or a local cache supplies the data. The data block is transferred through the home data switch, the data crossbar, and the requester's data switch to reach the requester.

See Figure 7 for the block diagram showing these operation steps:

1. **Address request** (cycle 0–2). The requesting CPU determines from its Local Physical Address (LPA) registers that the desired location is not in this snooping coherence domain. It makes a *Remote_ReadToShare* request to its board Address Repeater, which in turn sends the request to the top-level Address Repeater.

2. **Send address to home SSM agent** (cycle 3–8). The Address Repeater arbitrates for the local address bus. When it gets the bus, it broadcasts the *Remote_ReadToShare* transaction on the local bus. The local CPUs and I/O controllers ignore this transaction. The requester's SSM agent determines the home board from the physical address. It sends the *Remote_ReadToShare* across the address crossbar to the SSM agent for the home board.

3. **Lock line and check coherency** (cycle 9–11). The home SSM agent locks the line, so that no other transaction can be made to this cache line. It checks its coherency directory cache. In this case, we assume a hit that indicates that the requested location is not owned. If the coherency directory cache had missed, then the home SSM agent would have had to wait an extra 16 system cycles (106 ns) for the Mtags to arrive from memory.

4. **Send expected responses**. The home SSM agent sends the number of slave responses to be expected (in this case 0) across the response crossbar to the requesting SSM agent. The home SSM agent tells the home data agent to be ready to forward the data to the requester.

5. **Broadcast address on home bus** (cycle 12–14). The home SSM agent arbitrates for the home address bus. When it gets the bus, it broadcasts a *ReadToShareMtag*.

6. **Snoop on home bus** (cycle 15–23). Each system device examines its coherency tags to determine the state of the cache line. If there is a hit, the owning cache supplies the data, and the memory cycle is ignored.

7. **Read data from memory** (cycle 15–30). The memory controller for the requested location (which is inside a CPU) recognizes that the request is in its address range, and does a read cycle in its memory unit. The data is sent to its local CPU Data Switch.

8. **Transfer data to home data agent** (cycle 31–34). The data block is sent to the home data agent.

9. **Move data across centerplane** (cycle 35–41). The home data agent saves the data block for possible Mtag update. It arbitrates for the system data crossbar. It sends the data across the data crossbar to the requesting data agent, with Mtags set to *globalShared*.

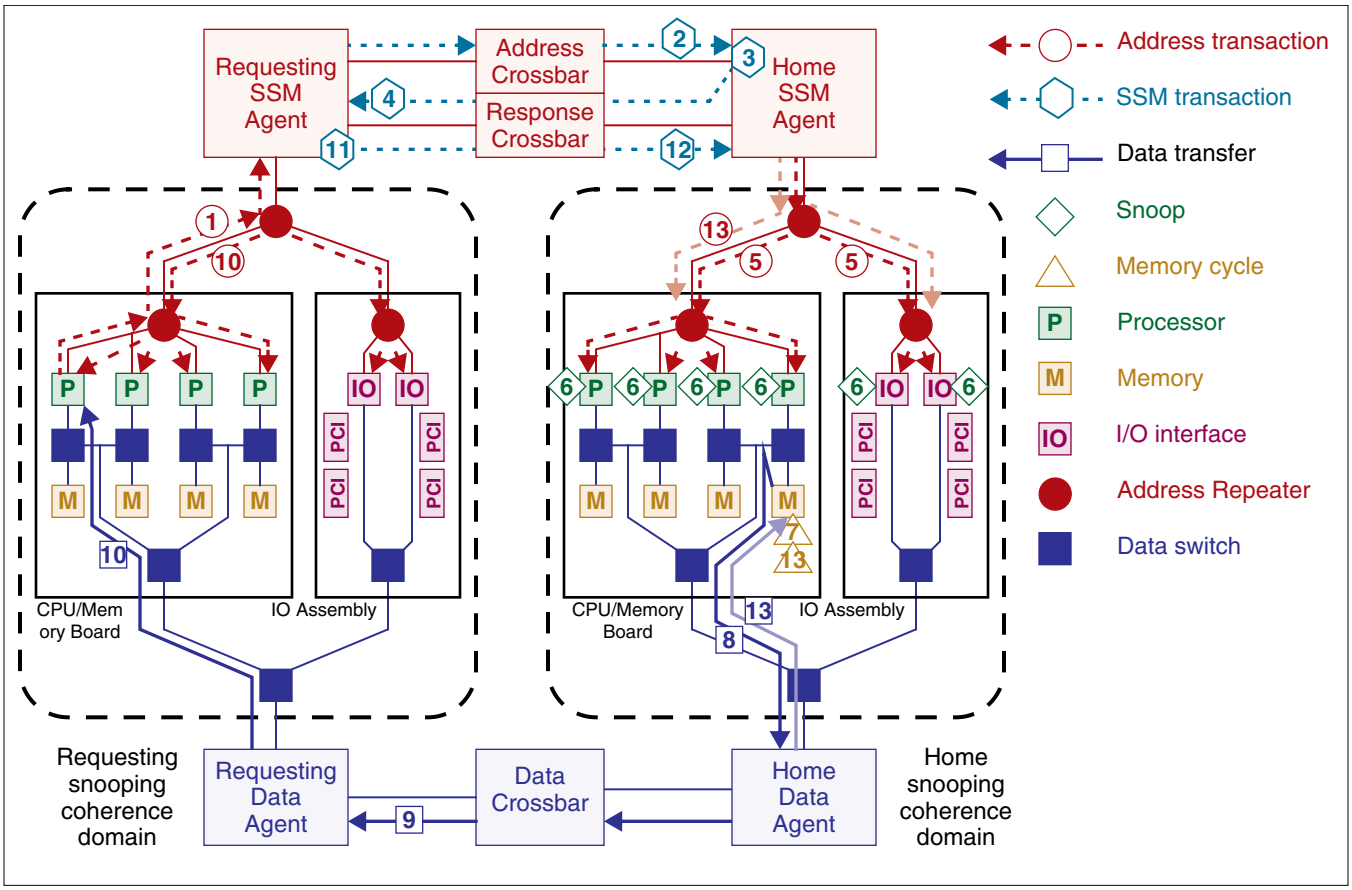


**Figure 7. Read to share operation between snooping coherence domains**

10 10 **Transfer data to requester** (cycle 42-50). The requesting data agent buffers the data until the local datapath is available. It notifies the requester's SSM agent that data has arrived. It arbitrates for the local datapath, and sends the data to the requesting CPU. The requesting SSM agent arbitrates for the local address bus, and reruns the request as a *ReadToShare-Remote.*

11 **Count responses**. The requesting SSM agent counts expected responses (in this case, one from the home SSM agent). It looks for data arrival notification from the requesting data agent.

12 **Unlock line**. The requesting SSM agent tells the home SSM agent via the response crossbar to unlock line. The home agent unlocks line.

13 13 13 **Update the Mtags**. If the previous home Mtag state was not *globalShared*, the Mtags need to be rewritten. The home agent arbitrates for the home address bus, and broadcasts a *WriteStream*. The Home data agent writes the data block to memory.

A load from memory between snooping coherence domains does four local address bus transactions, one SSM request, and two SSM replies. It takes 14 more system cycles (93 ns) than a transfer between boards inside a snooping coherence domain. This 38% increase in pin-to-pin memory latency for SSM versus snooping allows the peak bandwidths of SSM systems to scale up at a rate of 9.6 GBps per snooping coherence domains.

## 8.3 Cache-to-cache transfers

This is the case when data is owned (modified) in a cache.

**8.3.1 Inside a snooping coherence domain.** The owning system device asserts a snoop result of *owned*, and sends the data directly to the requester. The memory cycle is ignored. The pin-to-pin latency is increased by 11 system clocks (73 ns) over the time required to do a load from memory.

**8.3.2 Between snooping coherence domains.** A three-way transfer is done. The coherency directory cache entry of the home SSM agent tells it which snooping coherency domain currently owns the data. The home SSM agent sends the owning SSM agent a *ReadToShareMtag*. The owning SSM agent runs the transaction on its local address bus, and supplies the data directly to the requesting data agent. The latency is 21 system clocks (140 ns) more than for an unowned SSM transfer, and 24 system clocks (160 ns) more than for a cache-to-cache transfer in a non-SSM system.

## 9. Fireplane interconnect implementation

### 9.1 UltraSPARC-III processor

The UltraSPARC-III processor [9] shown in Figure 8 is used by all Fireplane-generation systems. It does in-order execution of up to four instructions per clock, with out-of-order completion. To reduce chip count and latency, the external cache tags, the Fireplane coherency controller, and the DRAM controller are all integrated onto the processor chip.

**9.1.1 Fireplane system interface.** The Fireplane coherency controller can do a snoop of the processor's five caches every system clock. The processor can have up to 15 outstanding Fireplane transactions, including up to: 1 instruction-cache miss, 1 data-cache miss, 8 prefetches, and 4 writebacks or block stores. The outstanding transactions can complete out of order with respect to the original request order.

**9.1.2 Memory controller.** The memory unit consists of two groups of four dual-banked 144-bit-wide SDRAM DIMMs. An individual memory bank has a peak bandwidth of approximately 0.8 GBps. The peak bandwidth of all four memory banks is 2.4 GBps. The maximum DIMM size is currently 1 GB.

**9.1.3 Processor error protection.** The major on-chip SRAM structures (instruction cache, data cache, and external cache tags) all have parity or error correcting code (ECC) error protection, as shown by the symbols P E in Figure 8. The parity protection on the two internal caches causes the data to be automatically refetched from the external cache, and the instruction to be re-executed. The external cache is protected by ECC.

The system interface generates ECC bits when a data block is sent out of the processor. The ECC bits are stored in memory, and checked and corrected when the data block is read back into a processor.

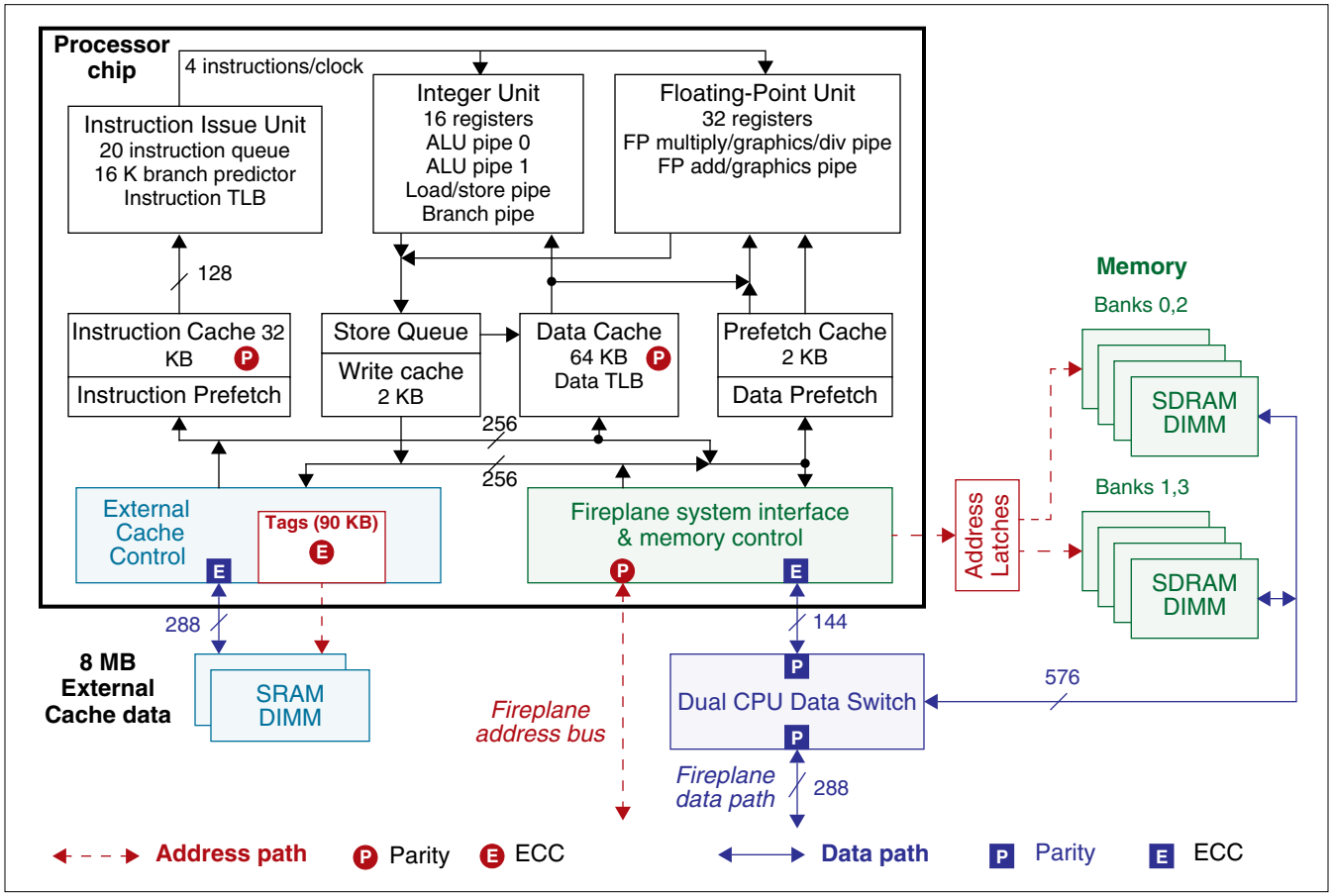


**Figure 8. UltraSPARC-III processor**

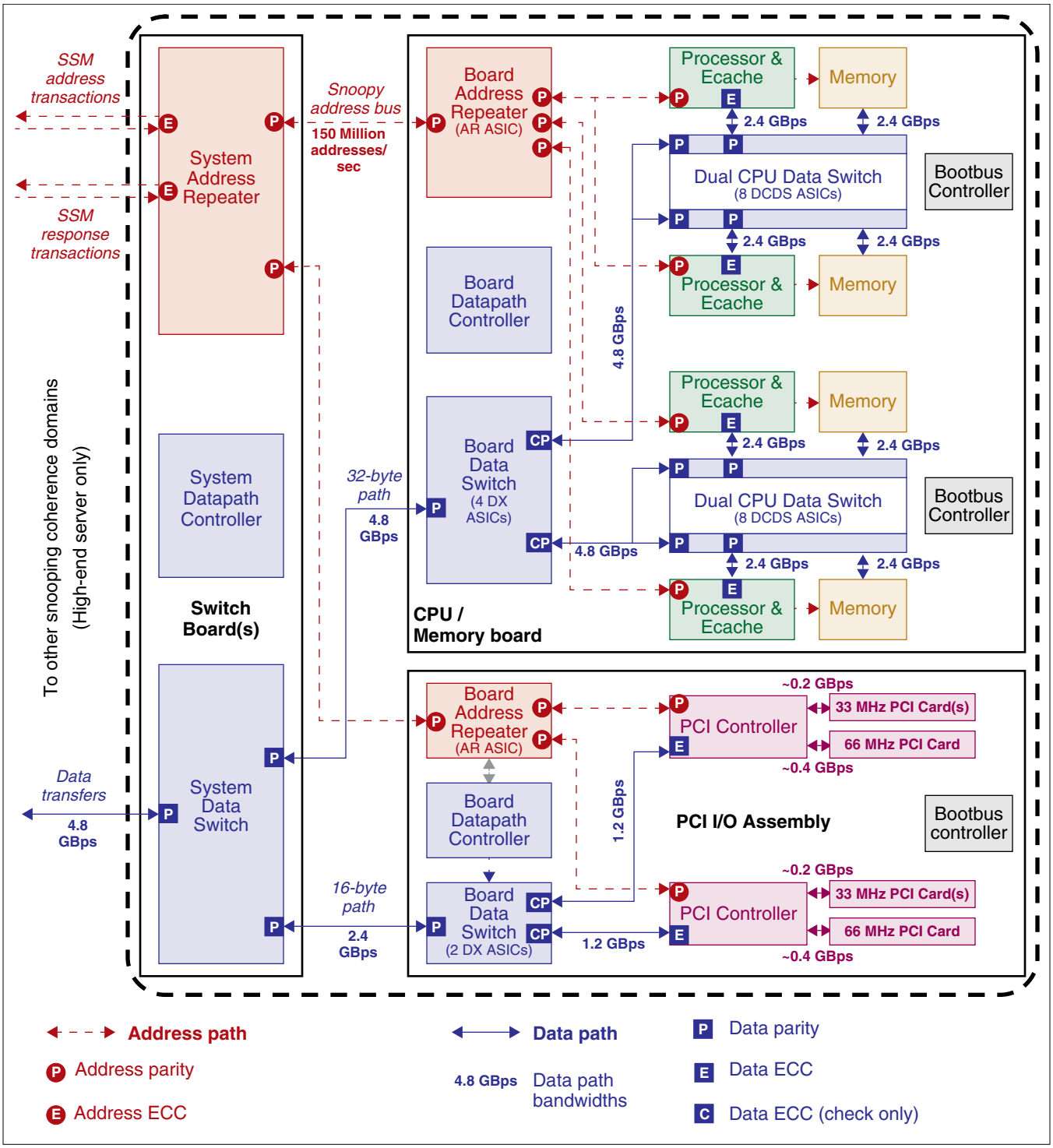


**Figure 9. Snooping coherence domain block diagram**

## 9.2 CPU/Memory board

The CPU/Memory board block diagram is shown in the upper right of Figure 9, and the physical board layout is shown in Figure 10. This board is used by all mid and high-end Sun Fire servers.

The board-level Address Repeater collects address requests from the four processors on the board, and forwards them to a top-level Address Repeater on the switch boards. It also disseminates addresses back to the processors so they can snoop them against their cache contents.

A CPU and its associated memory unit each have a 2.4 GBps path to a 3x3 crossbar in their half of a Dual CPU Data Switch. Both CPU/memory pairs share a 4.8 GBps path to the Board Data Switch. The Board Data Switch is a 3x3 crossbar that connects the two halves of the board to the off-board 4.8 GBps Fireplane switch port. If all memory accesses from each of the four CPUs were to each CPU's own local memory, then the peak memory bandwidth of a board would be 9.6 GBps. However, memory is usually 16-way interleaved across the memory banks of all four processors. In this case, the path between the two Dual CPU Data Switches and the Board Data Switch limits the peak memory bandwidth of a board to 6.4 GBps.

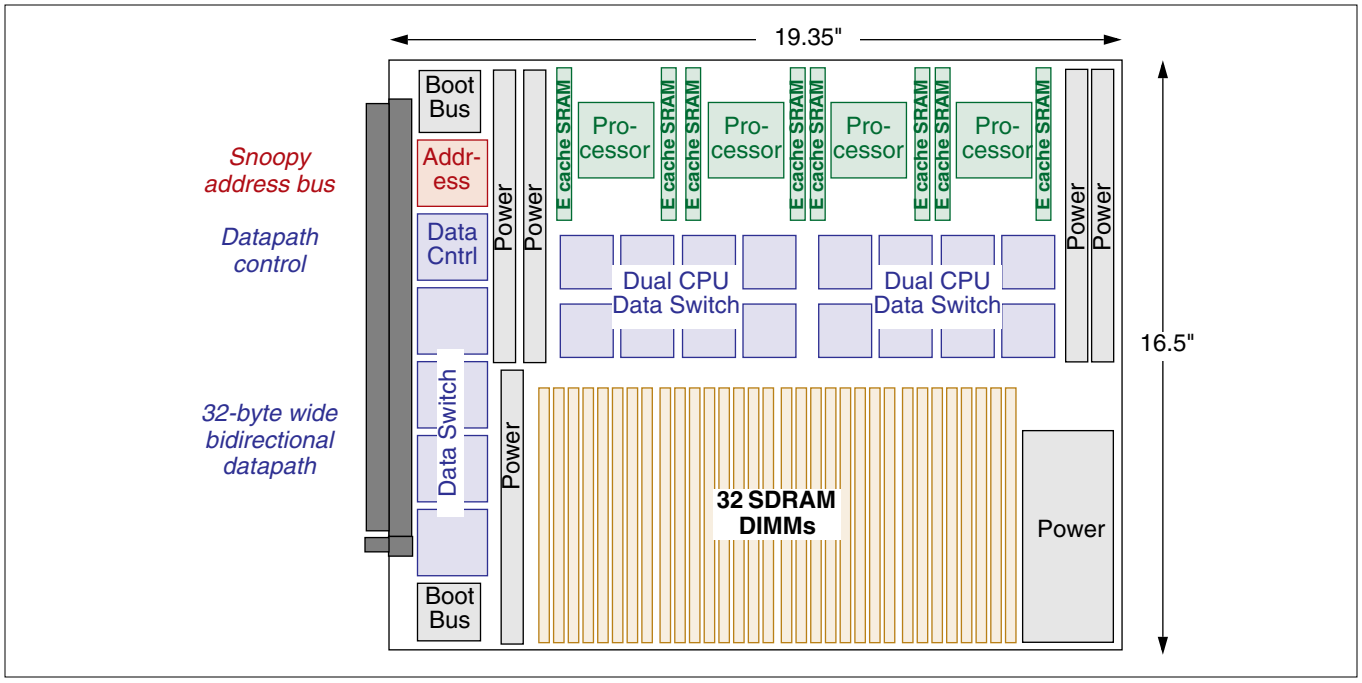


**Figure 10. CPU / Memory board physical layout**

## 9.3 PCI I/O assemblies

The PCI assemblies for the mid and large-sized Sun Fire systems all share the same architecture, which is shown in the lower right of Figure 9. The board-level Address Repeater collects address requests from the two PCI Controllers on the PCI assembly, and forwards them to a top-level Address Repeater on the switch boards. Each PCI Controller provides one 33 MHz PCI bus and one 66 MHz PCI bus. The two PCI Controllers connect to the Board Data Switch, which in turn connects to a 2.4 GBps Fireplane switch port.

There are several physically different implementations of the PCI assembly to accommodate small versus large cabinets, and PCI versus CompactPCI form factors. All PCI assemblies have two 66 MHz PCI slots, and from two to six 33 MHz PCI slots.

## 9.4 Inter-board interconnect

The logic of the interconnect between the boards of a snooping coherence domain is shown on the left-hand side of Figure 9. There is a top-level Address Repeater, and a top-level data crossbar. A CPU/Memory board has a 4.8 GBps (32-byte-wide bidirectional) data connection, and an I/O assembly has a 2.4 GBps (16-byte-wide bidirectional) connection.

**9.4.1 One snooping coherence domain.** Figure 11 shows the mid-size server cabinets, which all have one snooping coherence domain. The inter-board interconnect of the Sun Fire 4800-6800 is packaged on Fireplane switch boards. Two switch boards are used in 12-processor systems, and four switch boards are used in 24-processor systems. Each switch board has one Address Repeater ASIC and two Data Switch ASICs.

The switch boards can be hot-swapped out to replace failed components. In the eight-processor Sun Fire 3800, the Fireplane address and data ASICs are mounted on an active backplane to save the space that would have been required for separate switch boards.

In the 12 processor systems, one system-level Address Repeater ASIC accepts address requests from three CPU/Memory boards and two I/O assemblies, and broadcasts the address transactions back to the boards. In 24-processor systems, two system-level Address Repeaters work in tandem to handle six CPU/Memory boards and four I/O assemblies. The peak system bandwidth is determined by the snoop rate of 150 MHz x 64-byte cache line width = 9.6 GBps. The system-level data interconnect is a 4x4, 5x5 or 10x10 32-byte-wide crossbar, which is bit-sliced across the switch boards.

**9.4.2 Multiple snooping coherence domains.** Each snooping coherence domain consists of one CPU/memory board and one I/O assembly, which are connected together by a switch board called an *expander board*. This board provides the system-level Address Repeater and Data Switch logic required of any snooping coherence

domain. In addition, there is SSM address logic and data logic to implement transfers between the snooping coherence domains.

When all transfers are to locations inside the same snooping coherence domain, the peak data bandwidth is 9.6 GBps per snooping coherence domain. When all accesses are to non-local snooping coherence domains, then the peak data bandwidth is limited by the Fireplane datapath to 2.4 GBps per snooping coherence domain

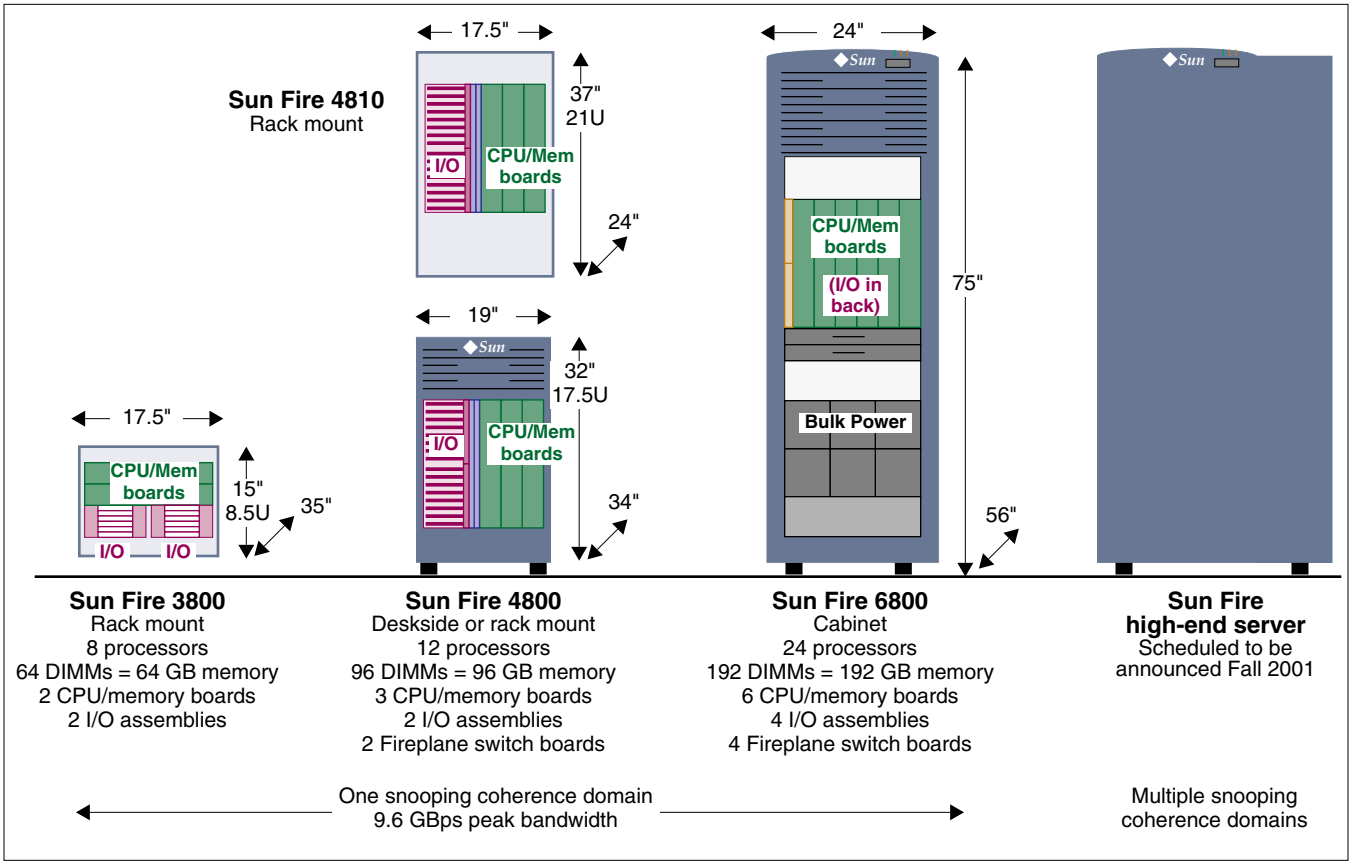


**Figure 11. Mid and high-end Sun Fireplane servers**

### 9.5 Interconnect error protection

**9.5.1 Addresses.** The Fireplane address bus has three parity error bits, which separately cover: 60 bits of command, address, byte mask, and address transaction ID; 12 bits of state information; and 38 bits of transaction ID. This is denoted by the symbol P in Figure 9.

The SSM address and response transactions are protected by ECC, which is denoted by the symbol E in Figure 9.

**9.5.2 Data.** The end-to-end data path is protected by ECC, which is denoted by the symbol E in Figure 9.

Two additional checks are made to help isolate the cause of datapath errors:

- Individual point-to-point connections are covered by parity. This is denoted by the symbol P in Figure 9.
- ECC is checked as it enters or leaves a CPU/memory pair or an I/O controller by the Board Data Switch. This is denoted by the by the symbol C in Figure 9.

The ECC checks that are done by the Board Data Switches can identify the source of ECC errors in most cases.

A particularly hard case for detecting the cause of ECC errors is when a device writes bad ECC into memory. These get detected much later by other devices reading these locations. Since the bad writer may have written bad ECC to many locations, and these may be read by many devices, the errors appear to be in many memory locations, while the real culprit may have been a single bad writer. Since the Board Data Switch checks ECC for all data entering or leaving each device from other devices, the original source of errors can be isolated.

## 10. System performance

The mid-range Sun Fire servers have set six world records between their introduction in March and July 2001. These records were set

using 750 MHz processors with 8 MB caches running Solaris 8 update 4. In the continual process of benchmark leapfrogging, the two SPEC records have been exceeded by later results.

### 10.1 SPECweb99 benchmark

SPECweb99 [11] measures web serving speed. It reflects real-world usage where the server supports home pages for a number of different organizations. Certain files within the home page are more popular than others. The workload simulates dynamic operations such as "rotating" advertisements on a web page, customized web page creation, and user registration.

In April 2001, a 12-processor Sun Fire 4800 set a record of 8,739 simultaneous connections. This was 5% more than the previous record, which was set by a 12-processor IBM p680. The Sun Fire benchmark system had 44 GB of memory, and 320 GB of storage in two Sun StorEdge™ T3 disk array trays (9 drives × 18 GB each). Networking was done on 12 gigabit Ethernets. The server software was iPlanet Web Server 6.0, which is an enterprise-class web serving solution.

### 10.2 SPECjbb2000 benchmark

SPECjbb2000 [12] is a Java server-side benchmark that emulates a 3-tier system, with emphasis on the middle tier. Random input selection represents the first tier user interface. SPECjbb2000 fully implements the middle-tier business logic. The third tier database is replaced by binary trees in memory. SPECjbb2000 is inspired by the TPC-C benchmark and loosely follows the TPC-C specification for its schema, input generation, and transaction profile. The benchmark runs from memory, and so does no disk or network I/O.

In March 2001, a 24-processor Sun Fire 6800 set a record of 109,146 SPEC JBB operations/sec. This was 36% faster than the previous record set by a 24-processor IBM AS-400.

The Sun Fire benchmark system had 48 GB of memory, and 18 GB of disk. It used Java 2 Standard Edition (J2SE) 1.3.1, which is a 32-bit java Virtual Machine (JVM).

Runs were also made using 1, 8, and 12 processors, to show multiprocessor scaling relative to one processor running one warehouse.

**Table 2. Sun Fire 6800 SPECjbb scaling**

| CPUs | Warehouses | SPECjbb ops/sec | MP efficiency |
|---|---|---|---|
| 8 | 8 | 43,353 | 91% |
| 12 | 12 | 62,463 | 91% |
| 24 | 24 | 109,146 | 80% |

### 10.3 Oracle Applications Standard Benchmark

The Oracle Applications Standard Benchmark [13] measures Enterprise Resource Planning (ERP). It simulates a realistic customer scenario using a selection of the most commonly used Oracle Applications modules. The transaction mix includes:

- 4 modules from Oracle Financial Applications: accounts payable, accounts receivable, fixed assets, and general ledger.
- 3 modules from Oracle Supply Chain Management Applications: inventory, order entry, and purchase orders.
- 18 OLTP transactions
- 4 batch jobs representing a substantial part of the overall benchmark load

The database is sized to represent a mid-market businesses whose annual revenues range from $100 to $500 million. There are 9,588 tables.

In March 2001, a 24-processor Sun Fire 6800 set a record of 16,016 benchmark users, with an average response time of 1.01 seconds. This was 14% more users than the previous record set by a 24-way IBM S80.

The Sun Fire benchmark system had 72 GB of memory, and 1.2 TB of storage in eight Sun StorEdge T3 disk array trays (9 drives × 18 GB each). It ran Oracle RDBMS 8.1.7.

### 10.4 PeopleSoft 8 Financials Online benchmark

The PeopleSoft 8 Financials Online benchmark [14] measures OLTP performance. It does 17 different transactions on a schema that contains 12,362 tables, and includes a batch update running in background.

In March 2001, a 24-processor Sun Fire 6800 set a record of 12,000 concurrent users, with an average response time of 3.99 seconds. This was 2.4x more users than the previous best, which was set by an 8-processor Compaq Pentium-III system.

The Sun Fire benchmark system had 24 GB of memory, and 1.2 TB of storage in eight Sun StorEdge T3 disk array trays (9 drives × 18 GB each). It ran Oracle 8.1.6.

### 10.5 PeopleSoft8 General Ledger benchmark

The PeopleSoft8 General Ledger (with combination editing) benchmark measures large batch runs typical of OLTP workloads during a mass update.

In May 2001, a 6-processor Sun Fire 6800 set a record of 7,255,139 journal lines per hour. This was 36% more than the previous best, which was set by a 6-processor IBM p620.

The Sun Fire benchmark system had 16 GB of memory, and 1.2 TB of storage in eight Sun StorEdge T3 Arrays (9 drives × 18 GB each). It ran Oracle 8.1.6.

### 10.6 TPC-H ad-hoc decision support benchmark

TPC-H [15] is a decision-support benchmark with an ad-hoc querying workload. It and TPC-R replaced TPC-D in 1999. TPC-H consists of a suite of business oriented ad-hoc queries and concurrent data modifications. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. TPC-H represents a situation where users don't know which queries will be executed against a database system; hence, the "ad-hoc" label. Pre-knowledge of the queries may not be used to optimize the DBMS system.

In May 2001, a 24-processor Sun Fire 6800 set a record price-performance of 581 $/QphH @1000GB. This was 19% better than the previous best price-performance, which was set by a 128-processor NCR Worldmark 5250. The Sun Fire result is currently has the best per-CPU performance.

The Sun Fire benchmark system had 80 GB of memory, and 6 TB of storage in 16 A5200 disk trays (22 drives × 18 GB each). It ran the DB2 UDB EEE 7.2 database manager. Table 3 shows the cost distribution of TPC-H configuration. For this medium-sized database problem, the server hardware represented 36% of the cost, 42% of the power dissipation, and 27% of the floor space.

**Table 3. Sun Fire 6800 TPC-H 1000 GB solution costs**

| Component | TPC-H cost ($millions) | Measured AC power consumption | Floor space |
|---|---|---|---|
| 6800 server | $1.10 | 5.4 KW | 8.8 sq ft. |
| Disk storage | $0.64 | 7.5 KW | 24.4 sq ft. |
| Software | $0.45 | | |
| 5 year maintenance | $0.88 | | |
| Total | $3.07 | 12.9 KW | 33.2 sq ft. |

## 11. Conclusion

The success of a multiprocessor system interconnect lies in efficiently delivering application performance and throughput from the processors, disk drives, and DIMMs. The medium-sized Fireplane-based servers have posted a series of performance records on standard benchmarks. Results for the high-end system will come after this paper has gone to press.

Like the preceding interconnect generations, the Sun Fireplane interconnect is another step along the path of supporting more and faster processors in a shared-memory system. Using two types of cache coherency looks like a promising method of keeping low latency for small and medium sized systems, and for local accesses in large systems, while scaling up the bandwidth in large systems.

## 12. References


[1] International Data Corp, Latest *Quarterly Server Tracker spreadsheet*, June 2001.

[2] IBM, *The IBM pSeries 680 Technology and Architecture White Paper*, Dec. 2000, http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/p680_technology.pdf.

[3] Kourosh Gharachorloo, Madhu Sharma, Simon Steely, and Stephen Van Doren, "Architecture and Design of AlphaServer GS320," *ASPLOS-IX*, November 2000, http://www.crl.research.digital.com/projects/scalable/formalmethods/wildfire/alphaserver.pdf.

[4] Hewlett Packard, *HP Superdome White Paper*, Sept., 2000, http://www.hp.com/products1/unixservers/highend/superdome/pressroom/whitepapers/wp_superdome-technical.pdf.

[5] SGI, *SGI 3000 Family Reference Guide*, Aug. 2000, http://www.sgi.com/origin/3000/3000_ref.pdf.

[6] David Culler and Jaswinder Singh, *Parallel Computer Architecture*, Morgan Kaufmann, San Francisco, 1999

[7] Ben Catanzaro, *Multiprocessor System Architectures*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[8] Kevin Normoyle, Zahir Ebrahim, Bill VanLoo, Satya Nishtala, "The UltraSPARC Port Architecture," *Hot Interconnects III*, August 1995.

[9] Tim Horel and Gary Lauterbach, "UltraSPARC-III: Designing Third-Generation 64-Bit Performance," IEEE Micro, May–June 1999, pp 73-85, http://dlib.computer.org/mi/books/mi1999/pdf/m3073.pdf.

[10] Larry McVoy and Carl Staelin, "Lmbench: Portable Tools for Performance Analysis," *Proceedings: USENIX 1996 Annual Technical Conference*, Jan 22-26, 1996, pp. 279-294, http://www.usenix.org/publications/library proceedings/sd96/full_papers/mcvoy.pdf.

[11] SPECweb99 benchmark description and results, http://www.spec.org/osg/web99/.

[12] SPECjbb2000 benchmark description and results, http://www.spec.org/osg/jbb2000/.

[13] Oracle Applications Standard Benchmark description and results, http://www.oracle.com/apps_benchmark/.

[14] PeopleSoft-8 description, http://www.peoplesoft.com/en/us/products/timely_topics/current_release/ps8_index.html.

[15] TPC-H benchmark description and results, http://www.tpc.org/tpch/default.asp.