AUSPEX

Auspex Architecture - FMP Past & Present

Steve Blightman

Abstract

This document describes the hardware architecture of the Auspex file server, and contrasts the FMP implementation with more classical SMP style architectures.

Auspex Systems, Inc.
5200 Great America Parkway
Santa Clara, CA 95054
(408) 986-2000

Document Serial No.:	00xxx
Version:	1
Date:	September 10, 1996
Release Status:	Preliminary

1.0 Introduction....3
2.0 NS5000 Architecture....5
  2.1 System Architecture....5
  2.2 Storage Processor....6
  2.3 Ethernet Processor....7
  2.4 File Processor....8
  2.5 Host Processor....8
  2.6 Auspex Primary Memory....8
  2.7 Hardware/Software Interface....8
    2.7.1 M16....8
    2.7.2 Hardware Fifo....9
3.0 NS6000 architecture....10
  3.1 Host Processor HP5....10
4.0 NS7000 architecture....11
  4.1 I/O Processor IOP....11
  4.2 MVIC64....12
  4.3 Hardware/Software Interface....12
5.0 Euclid Architecture....13
  5.1 File & Storage Processor....13
  5.2 Network Processor....14
  5.3 Hardware/Software Interface....15
  5.4 Future Improvements....15
    5.4.1 Memory on Blink....15
    5.4.2 Network Helper....15
6.0 Competing architectures....16
  6.1 Sun Ultra Enterprise Server....16
  6.2 Sun Ultra Enterprise Clusters....17
  6.3 DG ccNuma Servers....18

1.0 Introduction

Auspex was founded in December of 1987 with the following mission statement from the original business plan:

"Auspex will address the file server market by designing and manufacturing a file server which will conform to industry standards and increase performance, reliability, and connectivity significantly above and beyond other NFS file servers."

At the time the performance of CPUs had been roughly doubling every 18 months, but the I/O performance had not kept up. Sun, in particular, had been promoting the client/server model of computing and the idea that "the network is the computer", but they had been using their workstation architecture to build servers, not appreciating that the requirements of the server were different from the requirements of the client.

Auspex felt that there were two underlying reasons why the workstation architecture did not give the performance or the reliability required. The first was a software one, in that the servers of the day ran full Unix implementations. There was a lot of unnecessary overhead in this, and Auspex decided to write their own light weight kernel, later called FMK.

The second reason was a dataflow one. Workstations used a classic Von Neumann architecture, using one memory space for user data as well as instruction data. This one memory space inevitably lived on one memory bus, and would ultimately be the system bottleneck, limiting both the CPU and the I/O performance. Auspex's answer to this was to move to a distributed processing model, where CPUs were positioned to execute specific functions, and were kept out of the data path as much as possible. We christened this approach "Functional Multi-Processing" or FMP.

A useful analogy for FMP is a factory production line. In this environment every worker is given a dedicated function to do, and passes the results onto the next worker in line. For many years this proved to be the most efficient way of manufacturing. This is the FMP approach.

In recent times, however, in the factory these methods have been abandoned, in favor of a more general approach where each worker is capable of multiple tasks. This has the benefit of allowing one worker to more easily substitute for, or help, another. It also has some more humanistic benefits, such as being more satisfying and rewarding for the workers. Knowledge of someone else's job may also allow a worker to better perform his own functions. This is the symmetric multi-processing, SMP approach.

The SMP approach, however, demands much better communications between workers. Maintaining consistency between workers also becomes a much bigger problem. In today's computer technology the overhead and complexity of the consistency protocols does not justify the load balancing benefits gained by the SMP architecture. It seems likely to be many years before this happens.

At the risk of carrying the analogy too far, it might also be useful to point out that the factories only changed worker's assignments. They did not change the machinery used to manufacture the product. There are still machines dedicated and optimized for specific tasks. Machines, as opposed to humans, work best when they do one task repetitively. When used this way, machines can not only be more efficient and have more performance, but they also can be more reliable. These same benefits are realized by our FMP approach.

So one of the key architectural concerns was the communication between workers. As we decided on a message based protocol between these functional processors, we realized that it was critical that the message passing be done with as little overhead as possible. One of the ways we accomplished this was by putting all of shared memory in one address space. This enabled all messages, as well as data, to be passed with direct memory access, and required no protocol overhead.

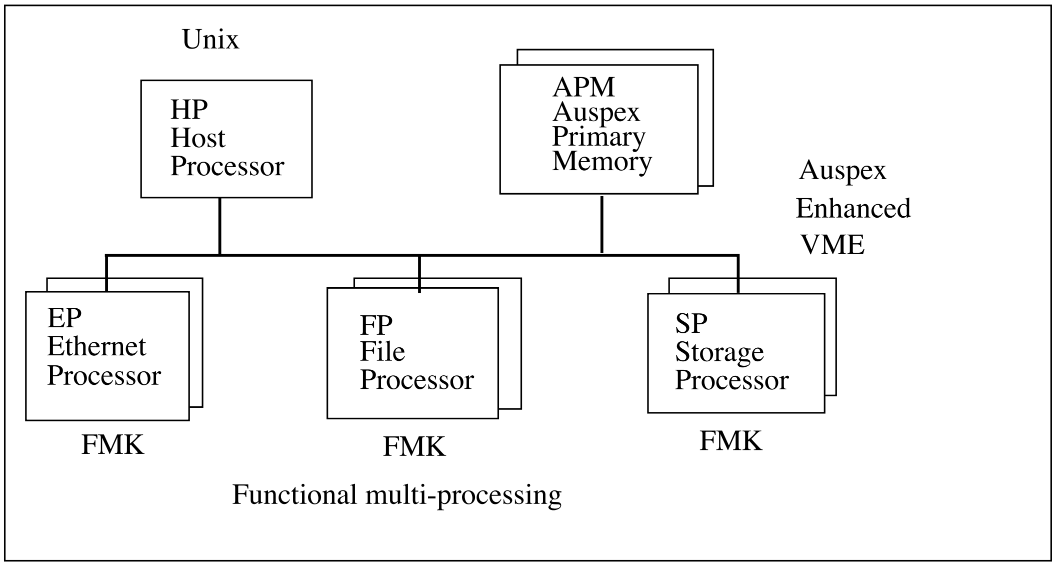
This distributed memory model is today being called NUMA (for non-uniform memory access) or ccNUMA for it's cache coherent version. ccNuma is being promoted as the model for scaling symmetric multiprocessors beyond three or four. In order for this to work, however, they must demonstrate that the cache coherency schemes scale, since they depend on this. Auspex, in contrast, does not require cache coherency, since the processors are performing different functions, and are not sharing control structures.

In summary then, we may say that there are two fundamental tenets underlying the Auspex technology. They are a light weight kernel called FMK, and a distributed processing model in one shared memory address space called FMP. From a hardware point of view, in order to maximize the performance of both the CPUs and the dataflow, it is vitally important to keep the CPU and the datapath as separate as possible. Much of the following discussion will address how the architecture of various Auspex product generations has tried to achieve this.

2.0 NS5000 Architecture

2.1 System Architecture

The NS5000 was the first product that Auspex produced. A basic system had 5 separate boards as follows:



The VME bus is used in this architecture for dataflow and for message traffic between the processors. It is important to note that it is not used for fetching instructions for any of the processors. Network client user data is transferred between the Ethernet Processor, Primary Memory, and the Storage Processor. This data path is designed to be as short as possible. The primary memory is used as a read cache for user data. The File Processor handles the file systems on the disk drives and metadata may be exchanged directly between it and the Storage Processor controlling those drives.

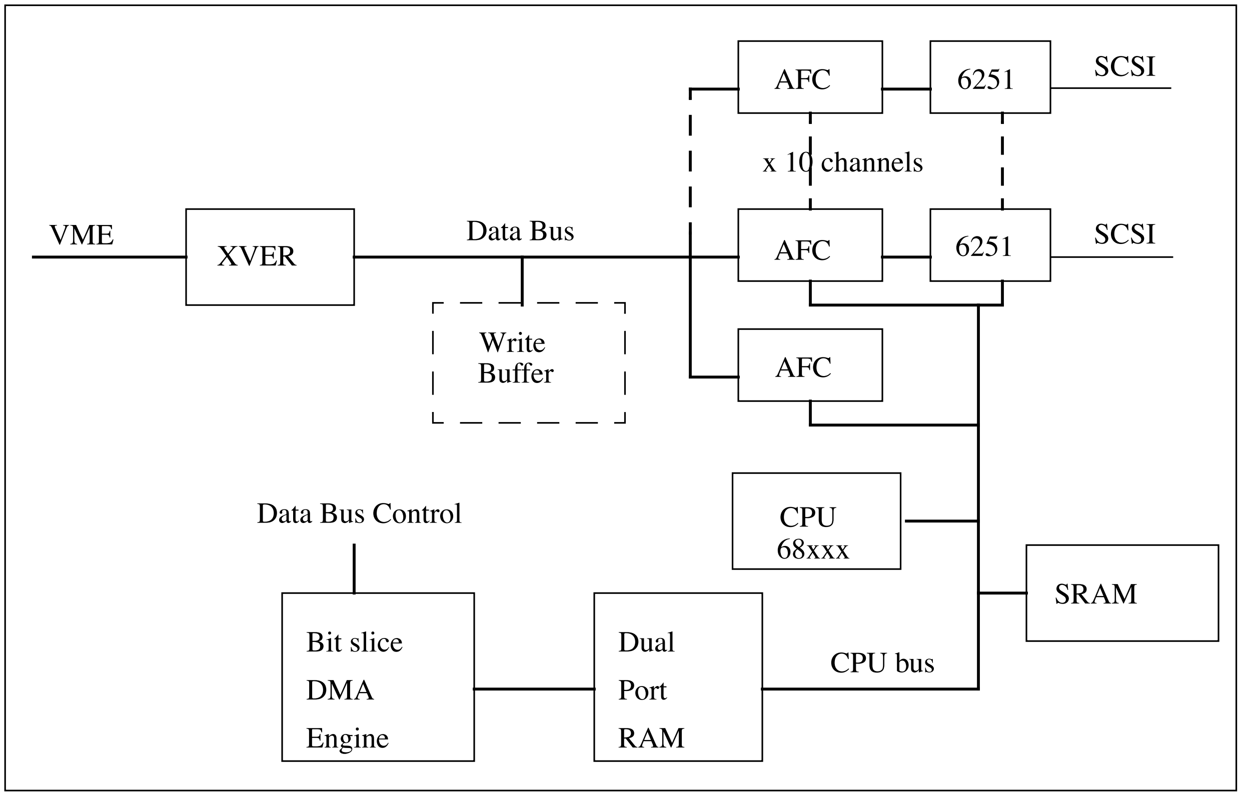
All of the processors in this system were 20Mhz 68020s. These were certainly not state-of-the-art at the time, but they were only there for control. There were high performance, special purpose, bit slice engines on both the Ethernet Processor and the Storage Processor to ensure the main data path was as fast as possible. It was felt that the VME bus would eventually limit the performance of this system, so a special Auspex mode was designed to maximize the VME throughput.

In fact the first performance limit found was the performance of the EP and FP. Despite hardware checksumming assistance on the EP, there was enough protocol processing that a higher performance CPU was required. This was also true of the File Processor. So the first hardware design improvements to be done were to change the processors on these boards to 40Mhz 68EC030s, and to increase their memory spaces. This was the NS5500 model.

Now let's look at the design of the individual boards.

2.2 Storage Processor

This is the block diagram of the Storage Processor.



The key thing to notice in this architecture is that the CPU is not attached to the data bus. The AFC chips are simple buffer chips, designed to buffer 2 blocks of 128 bytes, and do word to byte length conversion. Data can be transmitted with very little latency from the SCSI bus, through the 6251 and AFCs, to the VME bus. The CPU is used to control the transfer, and instruct the DMA bit slice. The DMA bit slice is responsible for the arbitration and control of the data bus. The CPU operates entirely out of the SRAM, and is unaffected by transfers on the data bus.

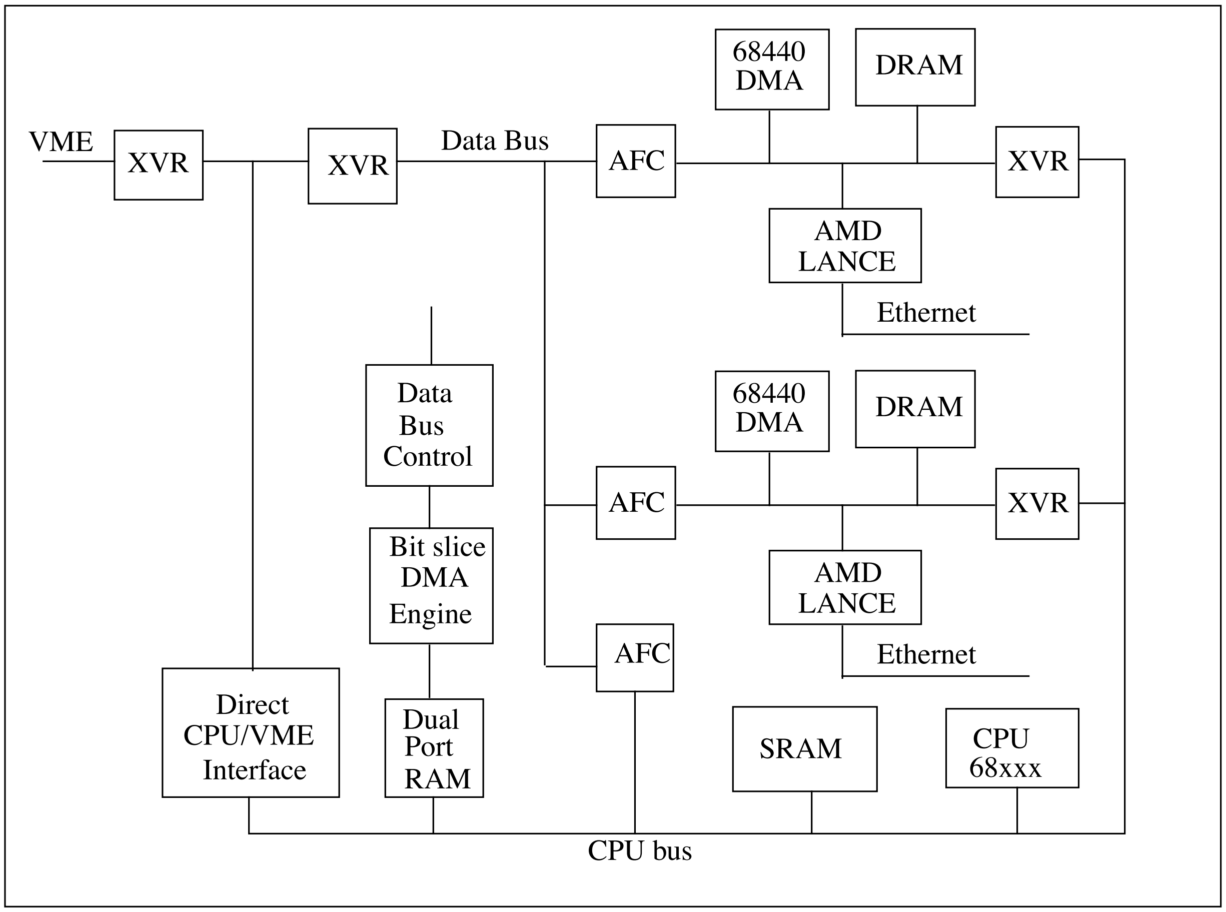
The Write Buffer was added as an option to SP3 to improve disk write performance. Write data is stored in this non volatile memory, so that disk writes may reliably be acknowledged early, before the data is physically committed to the disk drive. Again, transfers to and from this buffer do not interfere with the CPU, other than the requirement for the code to start and stop the transfer.

In order to simplify the hardware, there is no shared memory space that may be accessed by another VME master. In retrospect, it may have been possible to improve the performance of this board by making the write buffer directly addressable on VME. Instead of having the SP's DMA read the data from another board and put it in the write buffer, it would have been possible for the other board to directly write it. Since in general VME writes are more efficient than VME reads, this may have improved VME bus performance.

Otherwise it is a testament to the architecture of this board that it has lasted as long as it has. It has gone through a number of changes, increasing the speed of the CPU, and adding VME64 support, but the architecture has remained the same.

2.3 Ethernet Processor

This is the block diagram of the Ethernet Processor.



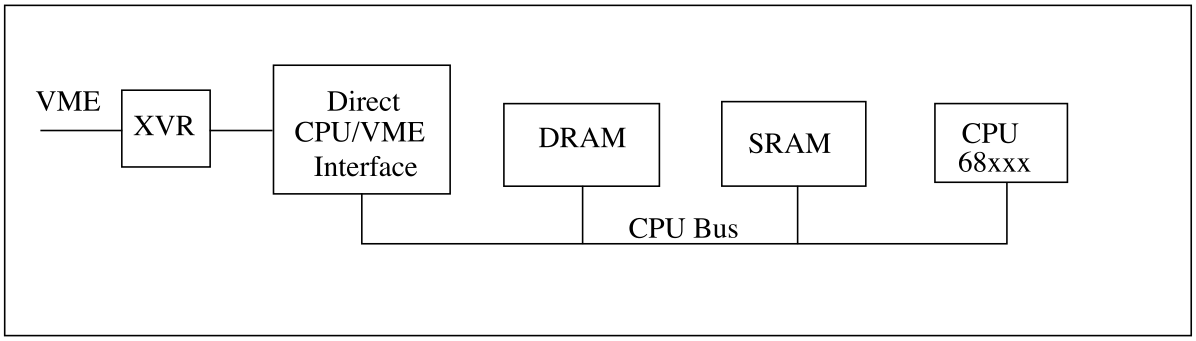
Again notice that the main data bus is separate from the CPU bus. Ethernet packets are stored in DRAM on the Ethernet bus. The headers in DRAM memory may be read by the processor, but all the actual data can be transferred via the VME bus without ever appearing on the CPU bus. The DRAM memories and the associated logic are duplicated for more performance.

The 3 AFC chips and the Bit slice DMA engines are leveraged from the Storage Processor design. An added feature of the AFC chips is that they can perform checksumming of data on the fly as it is being transmitted through them. This again is important because, without this, the Network Processor would consume a great many CPU cycles performing network checksums, and all the packet data would have to appear on the main CPU bus just for this function.

In contrast to the SP, there is a direct CPU to VME interface. This is primarily used by other VME masters to read messages from the SRAM. Although the local CPU here can read and write VME memory directly, we have found this to have a significant impact on the performance of the VME, and so this is avoided. Also this direct VME interface only supports normal VME mode as a master, and normal or block as a VME slave, and does not support enhanced VME in either direction. Enhanced VME transfers may only be done by the DMA, so VME memory is normally accessed by this CPU via DMA through the third AFC chip.

2.4 File Processor

This is the block diagram of the File Processor



This design is obviously much simpler, since it has no I/O interfaces. Indeed originally we used a Motorola VME single board computer as the file processor, but this operated only out of DRAM, which was slower, and did not support some of the Auspex specific features such as physical slot identification and hardware FIFO descriptors. So we decided to leverage the Ethernet Processor design, stripping off all the network and DMA logic that was not required. We added DRAM support to the CPU design for metadata cache, which could be large, but all the code essentially ran out of SRAM.

The main limitation in this design is that the VME interface did not support enhanced mode transfers, and that there is no DMA. The lack of DMA could be somewhat overcome by making other processors push data to, or pull data from this processor, but the lack of enhanced mode consumed a substantial amount of VME bandwidth.

2.5 Host Processor

The Host Processor designs for the NS5000 and NS5500 were basically Sun VME designs. HP1, HP2, and HP3 were all variations of a Sun 68020 design and HP4 was a Sun Sparc design. None of these designs had very good VME implementations, and none of them were capable of VME master DMA transfers. On the other hand they were naturally Sun compatible, and for most NFS transfers the Host was not involved anyway.

2.6 Auspex Primary Memory

Although the first shipments were made with Clearpoint VME memory, it was necessary to design our own memory board so that we could use Enhanced VME mode. This was the key to achieving the anticipated performance of the VME bus and the system as a whole. At the same time, we were able to incorporate the Auspex standard board registers, such as status and board type, and support physical slot identification.

2.7 Hardware/Software Interface

2.7.1 M16

From a software view, one of the key decisions was the system software that we would use. For best performance we did not want our functional processors to be burdened with a full operating system. We therefore decided to write our own lightweight kernel on which we could base the EP, FP, and SP code. In the original specification there were 16 basic messages that drove this kernel, and hence the name M16.

The software written for the EP and FP was basically interrupt driven, but the interrupt processing required was kept to a minimum. The software for the SP however, was more extensively interrupt driven, due to the requirement that for that generation of SCSI controllers, each SCSI phase change had to be handled by the CPU.

2.7.2 Hardware Fifo

Much thought was given to how best to pass messages between processors. We decided that we would not have a special interconnect dedicated to messages, but would use the VME bus for both data and messages. We therefore needed something with little overhead, that was also scalable. We did not want to classic lock mechanisms that are usually required for SMP systems, since manipulating these locks over the VME would prove costly to our performance.

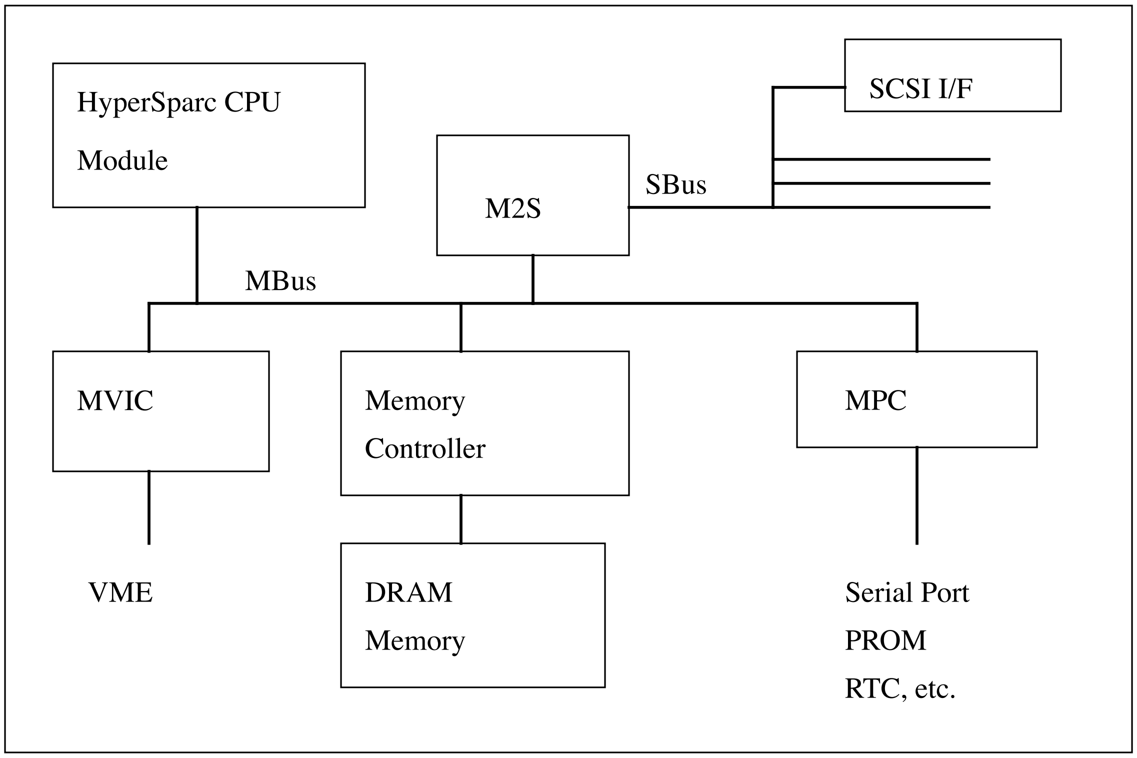
So we settled on a hardware FIFO on each processor board. In this FIFO we would store 32 bit pointers to messages in the single VME (NUMA) address space. Now in order to send an unsolicited message from one board to another, we could issue a VME 32 bit write to the receiving board’s FIFO. Each board had a single well known address for it’s hardware FIFO. Now since the VME 32 bit write was indivisible, multiple boards could send messages to one FIFO, and the VME bus arbitration would determine which order the message pointers were stored in this FIFO. The receiving board reads the pointers out of it’s hardware FIFO, and then reads the message itself from VME address space.

3.0 NS6000 architecture

Although the NS5500 was leading the way in I/O performance, Auspex was being asked for more Host Processor performance. This has never been an area in which we believe we can, or should, add much value. So the idea here was that we should leverage as much standard CPU technology as we could, and thus the HP5 was based on the MBus used in the Sparcstation 2. The one area we felt we could improve, however, was the interface between the HP and the rest of our system, and so we developed MVIC, the MBus to VME interface. This allowed data transfers between the MBus and VME to be done at a much faster speed, and also allowed the HP to become a full participant in the message passing by supporting a hardware descriptor FIFO.

3.1 Host Processor HP5

This is the block diagram of the HP5



Notice here that this is basically a workstation design with the typical Von Neumann architecture. Data and instructions are fetched from the single memory, and this of course becomes the performance bottleneck. Everything wants to access the DRAM memory via the MBus. As in all popular architectures today, the Hypersparc CPU mitigates this problem with the use of data and instruction caches, but it's performance is still heavily dependent on memory access times. In typical implementations, even with cache hit rates above 95%, we have observed that CPUs can end up stalled waiting for data more than 50% of the time. Since there is not generally extensive I/O traffic to the Host Processor, however, performance of the system is not unduly affected.

In retrospect today, the major mistake here was not to strive harder to make this design 100% Sun compatible, but at the time it wasn't clear exactly what was needed to do this, and the choice of the Fujitsu chip set restricted us.

4.0 NS7000 architecture

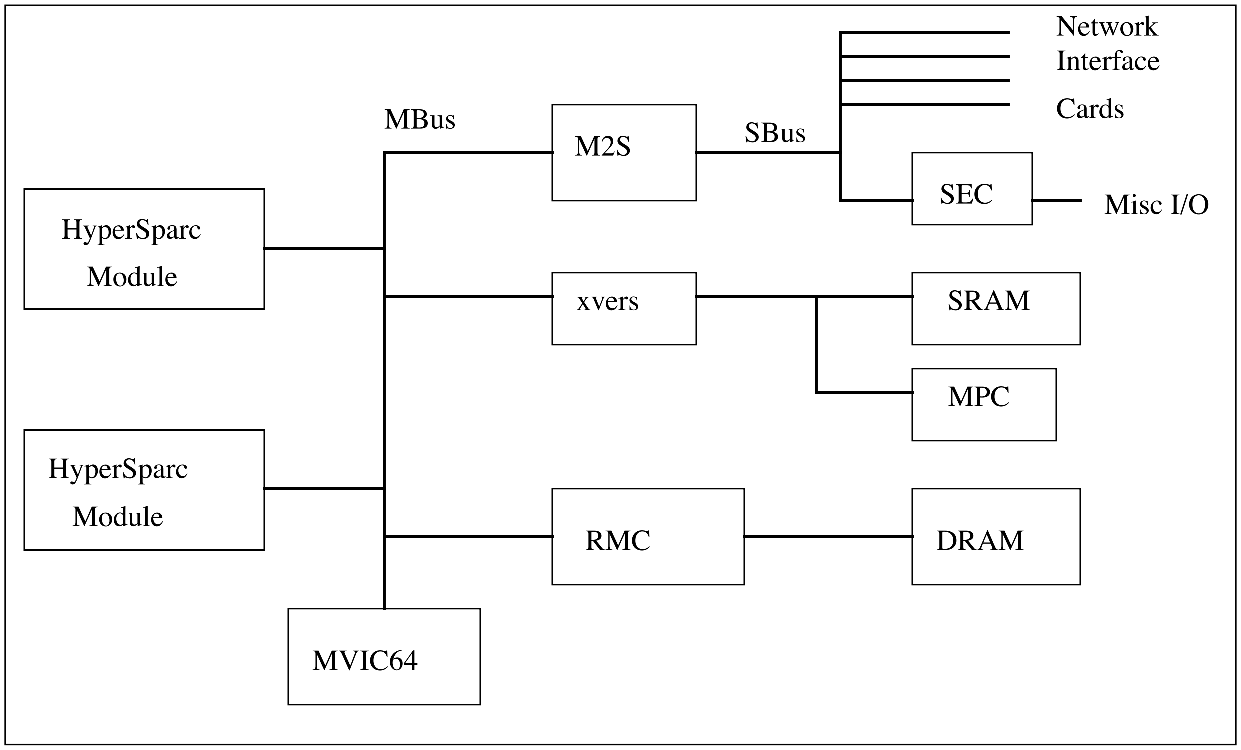
MVIC as an IC development would never had been justified if it had only been used in the Host Processor. The intent at the beginning of the IC design was that the part would be leveraged into an I/O processor design, so that the functional processors could continue to evolve. Since it was obvious that we should transition our Host Processor from 68000 to Sparc technology, it also made sense to transition our functional processors this way as well. Looking at the performance of the NS6000 system, the limiting factor was the performance of the Ethernet Processor, followed fairly closely by the File Processor.

The first choice, then, was to transition the Ethernet Processor. This had the added benefit of allowing us to leverage other SBus network designs, so that we would not have to develop FDDI or ATM interfaces ourselves. It was also apparent that we could take advantage of the fact that MBus was designed to support more than one processor, and so we could transition the File Processor function to this board as well. Although the pairing of the File and Ethernet processors may not have been as natural as the File and Storage processors, the Storage Processor was not limiting the performance. It was therefore opportunistic to pair the functions this way.

At this time it also became apparent that the VME bus was going to limit the system performance. So we also decided to integrate the primary memory function into this design. This enabled us to move the user data cache closer to the network, as well as reducing the basic board set from 5 to 3 boards. On the down side, it did mean that, in a multi-board configuration, the user data cache was distributed across multiple IOPs. So in order to get maximum system performance, it is necessary to configure the system so that the file systems accessed by clients are managed by the IOP to which their networks are attached. In this way we can minimize the IOP to IOP communication.

4.1 I/O Processor IOP

This is the block diagram of the IOP3.



We also designed the Network Accelerator card to offload the CPU from having to perform checksumming. This card was an SBus card that could be installed on the same SBus as the Network Interface cards.

We also designed the Network Accelerator card to offload the CPU from having to perform checksumming. This card was an SBus card that could be installed on the same SBus as the Network Interface cards. It worked in conjunction with the buffered Ethernet adaptor that we designed. The way this worked was that the network packets buffered on the Ethernet card could be transferred through the Network Accelerator to accumulate the checksum. This was done without putting the data onto the MBus. Unfortunately with FDDI cards, the network packets go directly to memory on MBus, so moving them back out to the SBus to accumulate checksums was not deemed worthwhile. In this case, the network CPU does the checksum.

Overall, however, as in the HP5, the major bottleneck in this board design is the MBus. Both processors and VME transfers are competing for this bus. The SRAM was added to the IOP3 design in order to alleviate some of this problem. This enabled instructions to be fetched from SRAM memory more quickly, but the fundamental problem remained.

In summary then, as often happens, the strength of this design was also its weakness. The fact that we leveraged standard workstation design allowed us to use standard CPU modules and support chips. On the other hand, by doing this we mixed user data and instruction paths, limiting the performance gain that we could realize.

4.2 MVIC64

Despite the performance issues discussed above, when we transitioned the IOP to HyperSparc, the functional multiprocessors were finally fast enough that we had reached VME saturation. We then decided to redesign the MVIC chip to support 64 bit, rather than 32 bit, transfers. At the time IEEE was enhancing the VME specification to support 64 bit transfers, by multiplexing the address and data lines, so allowing the same physical backplanes to achieve the higher transfer rates. This was a natural thing to do, and we decided to do it too.

We had many discussions about which VME transfer modes we would support. In the end our decision was to keep MVIC64 as simple as possible. We would do VME64 transfers in much the same way as we did enhanced VME32 transfers. This also meant that our VME64 transfers would then also be enhanced and run potentially faster than the IEEE standard. Unfortunately we would not be able to interface with other manufacturer's VME boards. We also defined the 64 bit VME address to closely match the 64 bit MBus address internal on the IOP, and so were also able to further simplify the design.

4.3 Hardware/Software Interface

In an on-going quest for performance, some changes were made in this timeframe to some of the low level software routines.

As stated above, since we had transitioned to this Sparc based IOP, the performance on the MBus had become a major concern. Having processors using the MBus and VME bus to directly access message data became a significant performance penalty, so we altered the software for the IOP so that all messages were fetched with DMA.

The other significant change made was to go to a polling method, rather than being interrupt driven. This was demonstrated to enable appreciably better throughput, without impacting latencies.

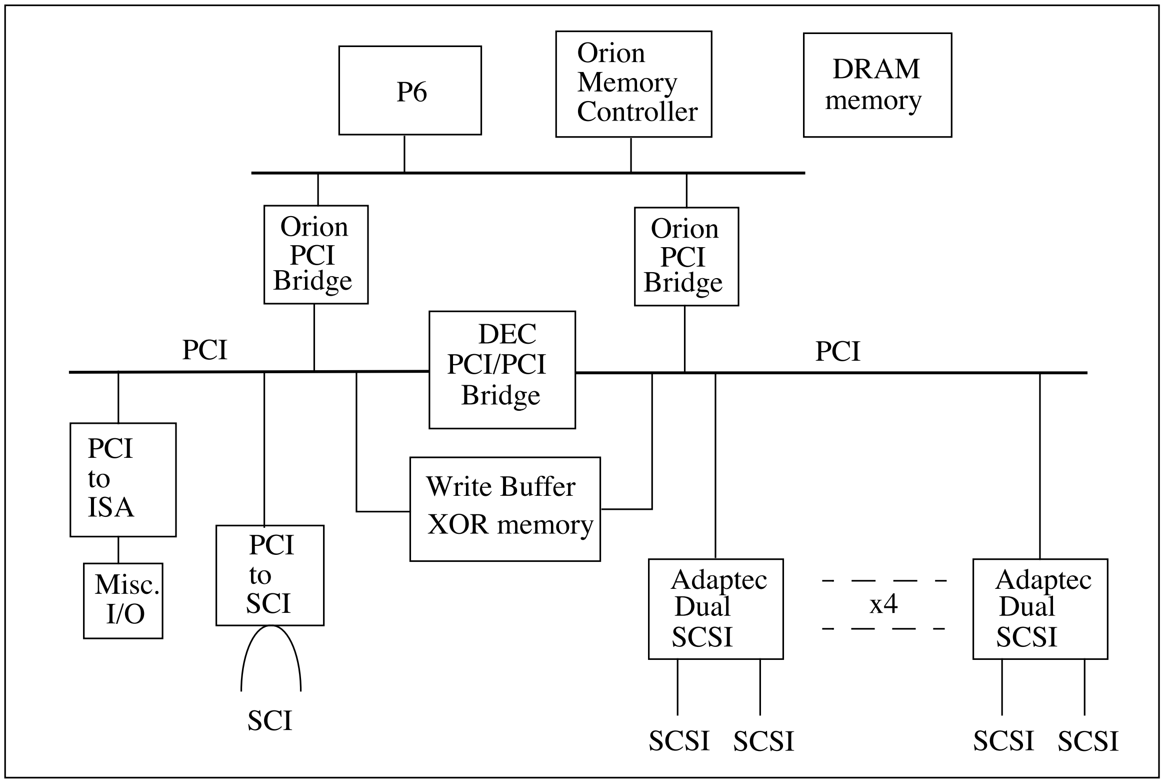
5.0 Euclid Architecture

The Euclid project was started to transition the Auspex Netserver products to different technologies. Instead of VME we will use SCI for the interconnect; instead of SBus we will use PCI for the local bus; and instead of Sparc CPUs we will use Intel Pentium Pros. We will still use standard workstation technology, but instead of attaching the interconnect to the Processor's memory bus, we will attach it directly to the local PCI bus. This gives us the advantage of being able to transfer data directly from a peripheral on the PCI bus to the interconnect without affecting the CPU's access to it's local data.

5.1 File & Storage Processor

In contrast to the NS7000, we also decided to move the File Processor function to where it perhaps more naturally belonged; that is, with the Storage Processor. In fact, because we are using more intelligent SCSI host adaptors than in the NS7000, a large part of the Storage Processor function disappears, and we will not dedicate a processor to this function. So we believe that one processor can be shared to perform both the File and Storage processor functions. One of the anticipated benefits of combining the File and Storage processor functions is that it is no longer necessary to transfer raw metadata over the interconnect. In the NS7000 system this has been seen to account for as much as 35% of the VME bandwidth.

This is the block diagram of the FSP in Euclid 1

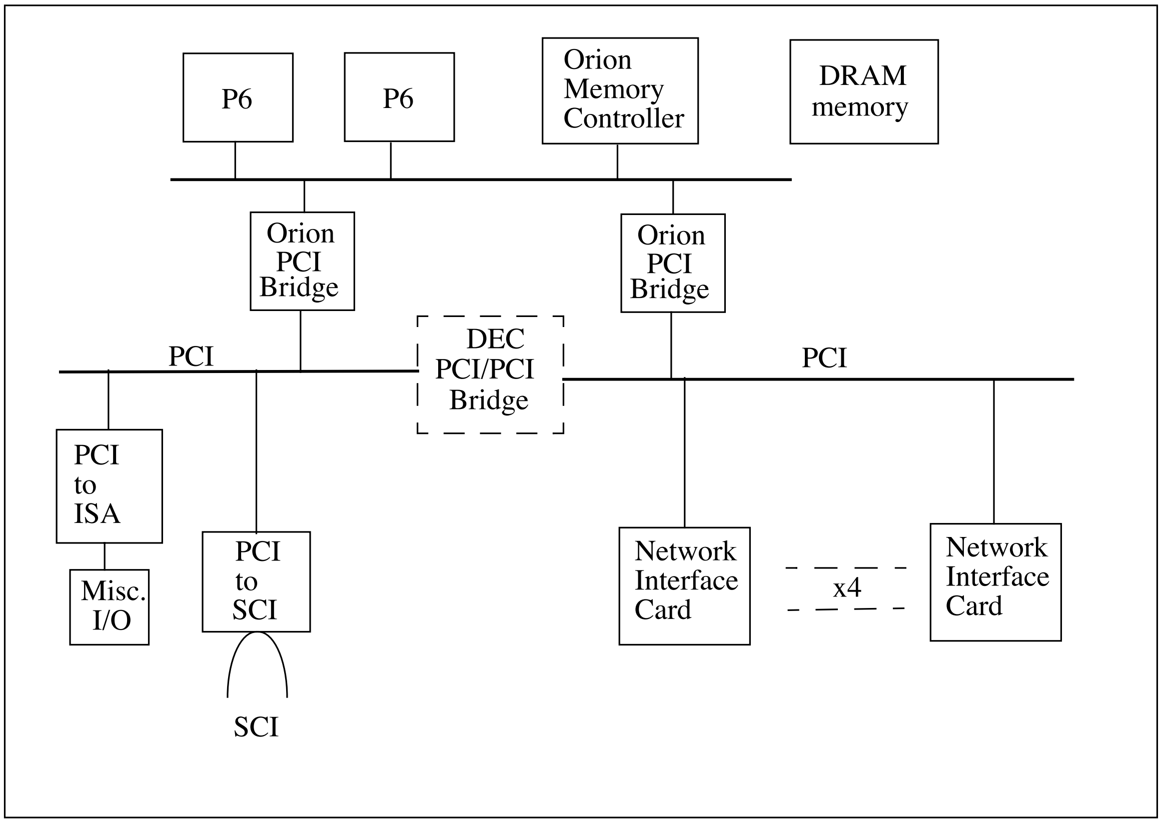


This appears to be a very efficient implementation for the FSP. User data can be transferred from the SCSI devices across the two PCI buses to the SCI interconnect without going onto the CPU's memory bus. Metadata, on the other hand, can go directly through the Orion PCI Bridge (the OPB) on the right hand side to the CPU's memory data. So the data that the CPU needs to see, i.e. the metadata, can be readily accessed, whereas the data it doesn't care about, i.e the user data, bypasses it.

The DEC PCI/PCI bridge is present mainly for electrical loading reasons. There are concerns about the performance implications this bridge may have. Only having one write buffer in the bridge limits the benefits we can achieve from the buffers in the PCI/SCI bridge. On the other hand, it does give us 2 PCI busses and increases the potential performance of the I/O system. The write buffer straddles the two PCI busses such that write data can be transferred from the SCI bus on one PCI bus, and out to the SCSI device on the other. This ensures that the write data only appears on each bus once. Messages from other subsystems can be accessed on the left hand side PCI bus, and may not interfere with transfers on the right hand side PCI bus. We believe, then, that the benefit of the two PCI busses should compensate for the penalty incurred by the bridge.

5.2 Network Processor

This is the block diagram of the Network Processor



The DEC PCI/PCI bridge is in dotted lines because in the first implementation this bridge is unused. It is simply there because we are using the same motherboard as the FSP.

The Network Processor presents some different challenges. In this first implementation the DRAM on the processor bus will be used for packet data, user data cache, and CPU instructions. This is obviously not ideal, and looks in many ways like the IOP architecture of the NS7000, with the same problems. This approach is being used in Euclid 1, however, to constrain the engineering effort required. We suspect that, with the fact that we have two PCI buses to divide the data paths, the contention for the DRAM memory will limit the potential performance of this design. It should be noted, too, that there is no hardware assist for checksumming - this will all be done in software.

5.3 Hardware/Software Interface

Many of the lessons learned in the previous designs are being used in Euclid. We are continuing in our efforts to keep the processor off the peripheral bus as much as possible. As in the NS7000, we will continue to use polling, rather than interrupts, to schedule the work, and the events register itself has been implemented such that it is the main memory of the processor itself. The message descriptor FIFO has also been changed from a hardware FIFO to a circular buffer in main memory. This had the double benefit of saving cost, as well as moving the FIFO itself much closer to the CPU. The CPU no longer has to read the FIFO from the main data bus, in this case, PCI.

5.4 Future Improvements

From an architectural view, then, the FSP design seems optimal. There will undoubtedly be some deficiencies in the chip implementations, and we will be looking closely at opportunities to improve these. We may also support multiple processors in the future, but at the moment this is thought to be unnecessary to reach our performance goals.

The NP, on the other hand, offers some substantial room for improvement. There have been many different discussions about this, but for the sake of this paper, we'll mention two.

5.4.1 Memory on Blink

In both the IOP of the NS7000 and the Network Processor, the data cache memory has been part of the memory on the Network Processor. There are good reasons why the cache should be managed by the Network Processors and placed as closely to the network as possible. Demand for access to this data cache, however, severely impacts the ability of the Network Processor CPU itself to access it's instruction and own data space.

Architecturally we would benefit by moving back to the original NS5000 architecture where the data cache was directly attached to the interconnect. In the future we may have requirements for very high speed access to sequential data. If we move the data cache to the interconnect, we may be able to more easily achieve this, using such things as interleaving. These notions lead us to the idea of implementing a memory design on Blink. Blink is the bus on the other side of the SCI Link Controller IC, and in Euclid 1 is present in the PCI/SCI boards as well as in any SCI switch. Moving our data cache to one of these spaces could significantly relieve the contention for the Network Processor CPU's memory bus.

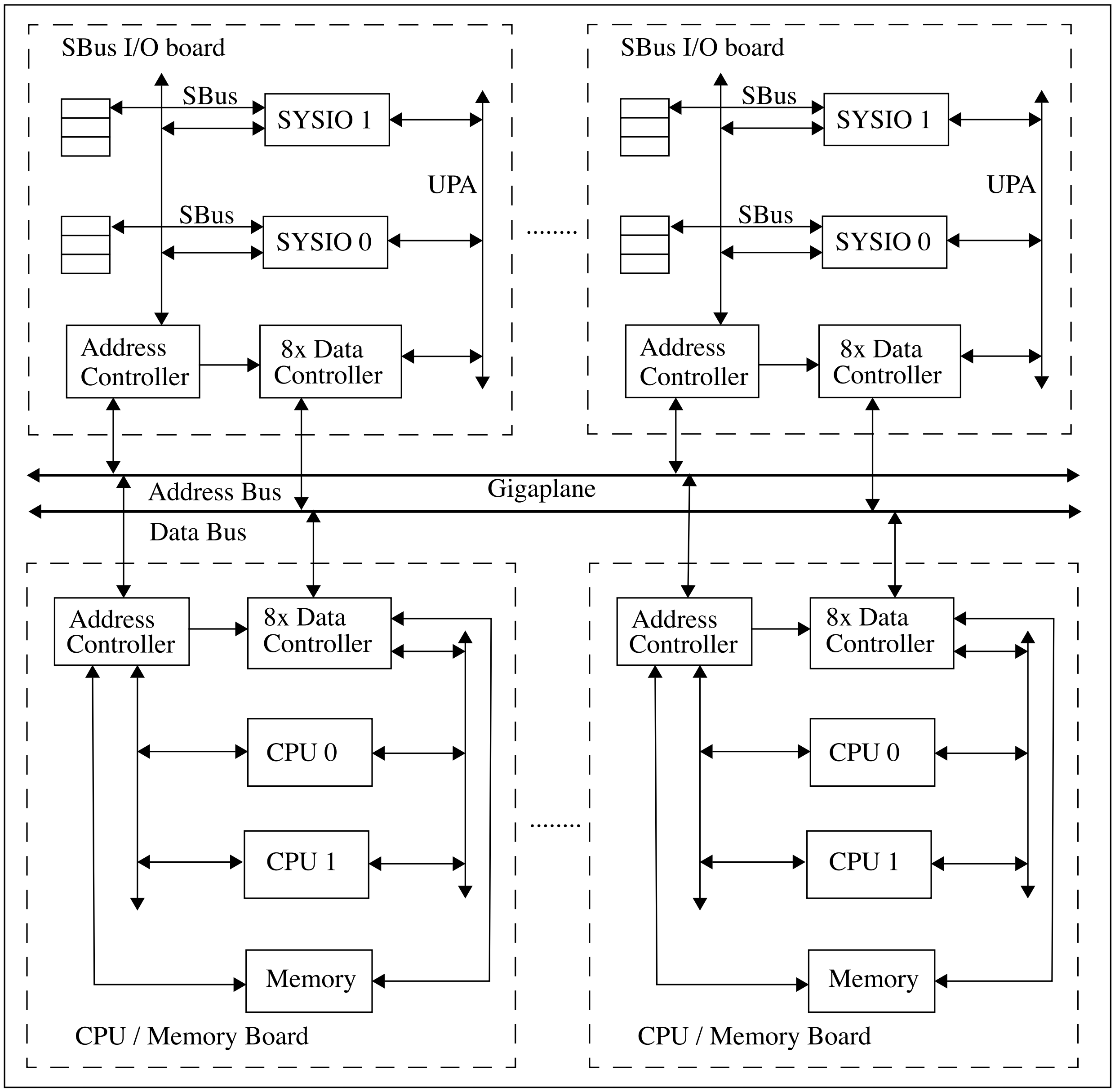
5.4.2 Network Helper

The other data we would like to avoid having on the CPU memory bus is the network packet data. Unfortunately this packet data is normally composed of two parts, a header and a data field. While the CPU needs in general to look at the header, it really has little use for the data field. Various schemes have been discussed to separate these two data fields, so that the header may be easily read or composed by the CPU, but that the data goes directly to the data cache. There are also opportunities for hardware development to support the Network CPU by performing some basic functions such as checksumming. Checksumming can account for as much as 10% of the CPU activity if done in software, whereas it is a relatively simple thing to implement in hardware. It is these discussions that have led to the working specifications of an IC design that we call Cygnus.

6.0 Competing architectures

6.1 Sun Ultra Enterprise Server

This is the block diagram of the Enterprise system architecture



This is a fairly classic SMP kind of design. In order to be able to effectively support multiple processor boards, all using a snooping cache coherency protocol, a very fast system interconnect is required.

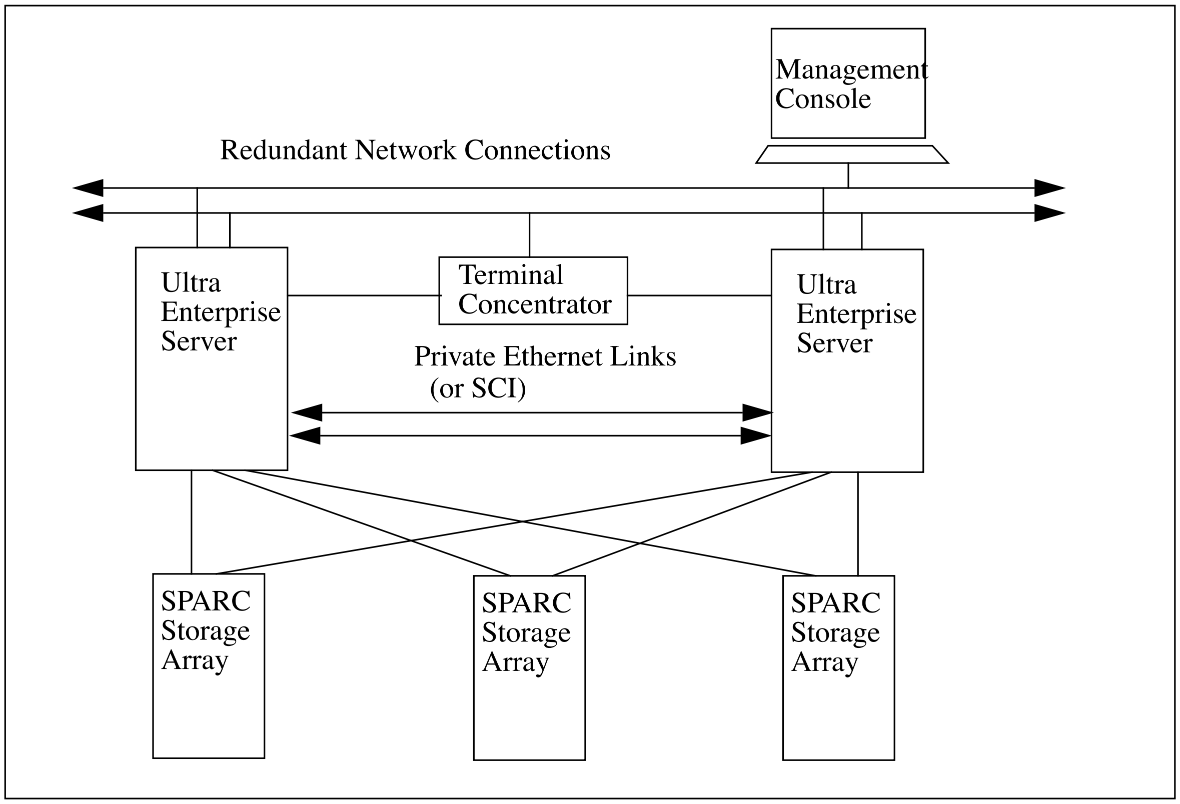
The Gigaplane, as Sun call their interconnect, is very wide and very fast. The data width is 256 bits, or 32 bytes wide, while the address bus is 42 bits wide for a total addressability of 2 TB. It runs at 83.3MHz for a burst transfer speed of 2.6GB/sec. It is split transaction, and the address cycle overhead is eliminated by having separate address and data buses. On the other hand, latency to memory is still of the order of 50 clock cycles for a cache line. In terms of clock cycles this is slightly slower than their old MBus designs, but in absolute time is somewhat faster because of the higher clock speed.

It is interesting to note that this interconnect is, in many ways, simpler than the dual XDBusses found in the previous Sparccenter 2000 design. Sun seems to have determined that having a simpler single bus may actually improve their reliability, despite the fact that dual busses may have offered some redundancy.

We do not believe these kind of architectures perform at all well in file server applications. All data, whether it be user data, meta data, or CPU data, appears on the interconnect. All the emphasis seems to be towards getting everything close to the CPUs. This is ideal for compute server applications.

6.2 Sun Ultra Enterprise Clusters

This is the block diagram of the hardware architecture of the Ultra Enterprise Cluster HA server



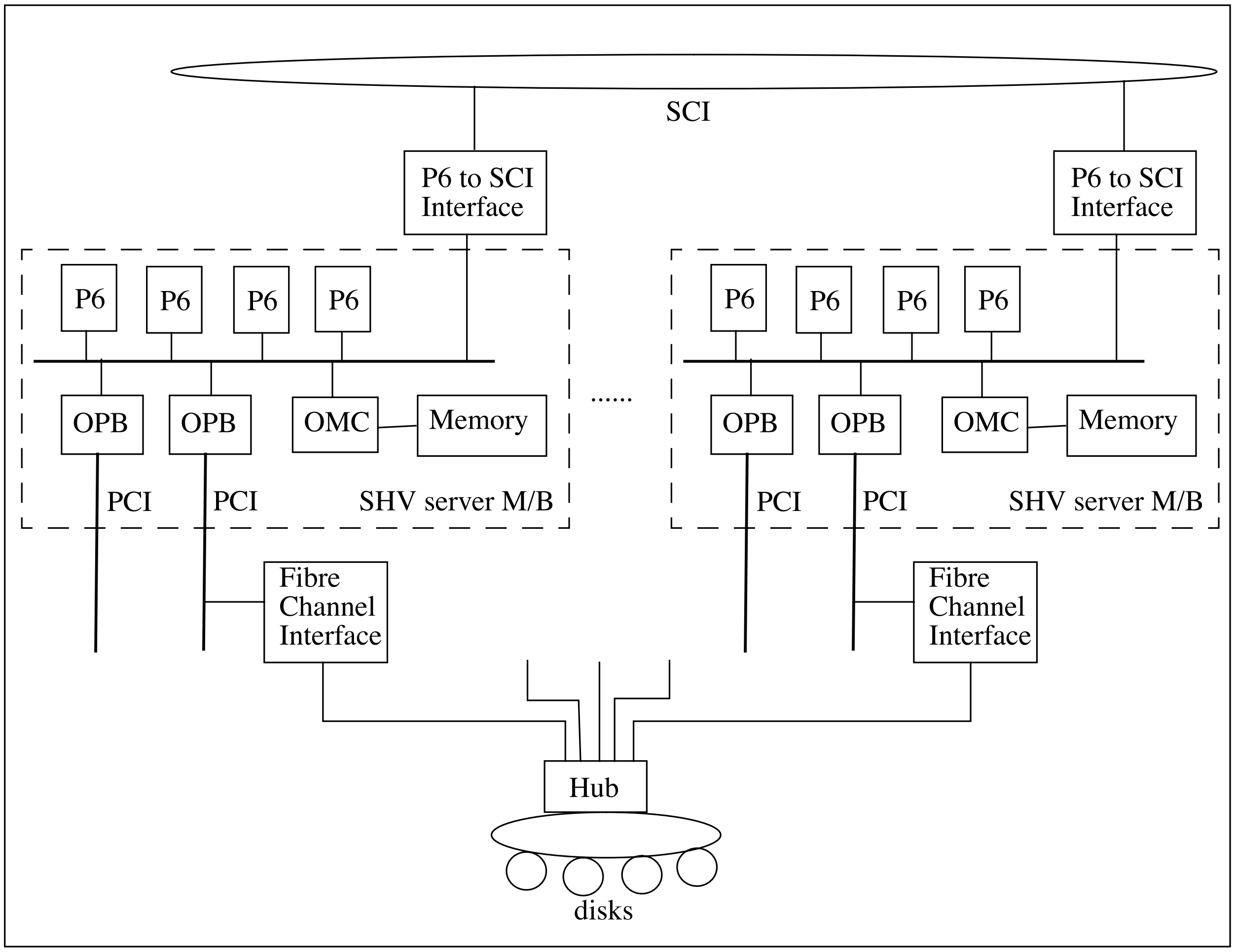
Sun supports this hardware as either one server serving users with the other as a hot standby, or as two servers serving users, each backing the other up should it fail. They also support mixing different classes of servers for more efficient costs of the backup configurations. In the High Availability configuration, data is mirrored on the Sparc Storage Arrays. A third storage array is used to save a third copy of the meta data. During failover, all three copies are checked. If two or more agree, they are assumed to correctly represent the state of the data.

In the original architecture the private links between the servers were dual Ethernets, but Sun has obviously determined that the latencies involved in the protocols required was impacting the performance, and so they have switched to SCI. SCI has latencies which are an order of magnitude lower.

While this is only a 2 node cluster, Sun are preparing to offer multi-node clusters using the same kind of architecture with SCI switches, and Fibre channel switches to support more storage arrays.

6.3 DG ccNuma Servers

This is the block diagram of Data General's ccNuma servers



This is a true SMP architecture. They are connecting Standard High Volume (SHV) 4 way P6 motherboards from Intel together on SCI and implementing the directory based coherency protocol designed as the IEEE SCI standard. Like Sun, they are going to take advantage of Fibre Channel disk drives and switches to connect the disk storage.

There are many technical challenges in this architecture, and doubts about it's true scalability.