A Cached System Architecture Dedicated for the System IO Activity on a CPU Board

Michael M. Hsieh, Tek C. Wei, and William Van Loo

Sun Microsystems Inc., 2550 Garcia Avenue, Mountain View, California 94043

ABSTRACT

This paper describes the architecture of a cached IO subsystem on a cpu board of a high performance workstation. The cached IO subsystem is intended to reduce the memory latency for IO activity and minimize the number of processor cycles stolen by IO traffic to achieve a better balanced computer system in terms of both the cpu computation power and the system IO bandwidth. The model of the cached IO subsystem is shown to illustrate the system architecture. The discussion covers arbitration of system IO activity, the architecture of the virtual address accessed IO cache, the cache coherency algorithm, as well as the comparison of theoretical bounds of IO bandwidth and cpu degradation between different caching schemes.

1. Introduction of the CPU Board Architecture

A general-purpose high-performance workstation is usually configured with a central processing unit(cpu) board, a few memory expansion boards, some disk storage, a display monitor and the network communication interface [1]. A virtual memory operating system is assumed for the computer system in this article. The memory expansion boards typically reside on a memory bus different from the main system bus. Special function devices, including mass storage, communication hardware, as well as hardware accelerators, such as the floating point accelerators are connected with the cpu board via the system bus. The industry standard VMEbus is cited as the system bus in the following discussion for illustration purposes. All the devices interfacing with the system bus usually share a direct memory access path provided by the cpu board.

The cpu board contains the core elements of the computer [1] which include the microprocessor as the central processing unit, cpu cache memory, memory management unit (MMU), floating point coprocessor, system bus interface, memory interface, on-board Ethernet interface and other standard on-board IO devices such as Boot PROM, time-of-day clock, interrupt control, serial ports, mouse and keyboard etc.. Altogether, they provide the basic system functions. The Sun-4 virtual memory architecture supports virtual input/output procedures as well as direct virtual memory access from system IO devices such as on-board Ethernet interface and VMEbus slave devices [2]. The interconnection model of logical devices in the system is illustrated in Figure 1 [2].

Presented at International Conference on Computer Design — 10/02/89

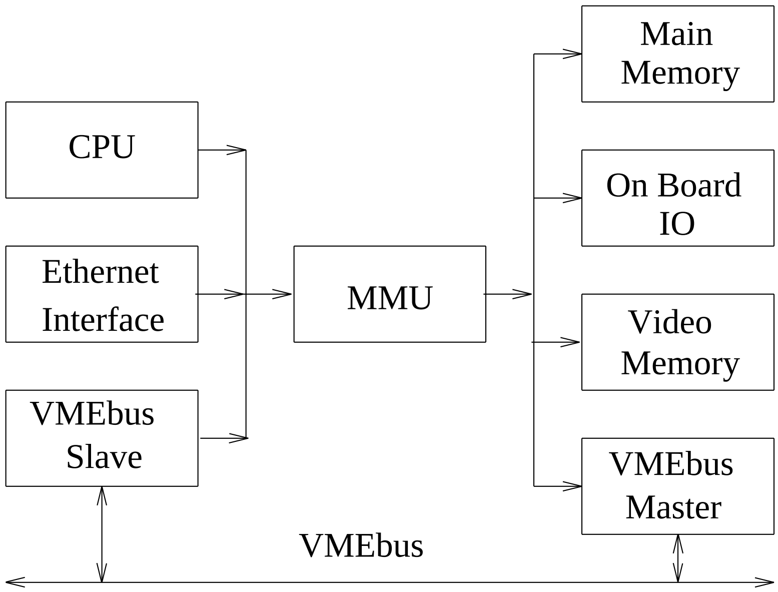

Figure 1. Interconnection Model of Logical Devices

As the speed of microprocessors increases, cache memory has been used to match the cpu speed in order to achieve the highest system performance. The Sun-4/470 workstation will be cited for illustration purposes. The cpu of the Sun-4/470 delivers 20 MIPs computing power running at 33 MHz. The cpu cache in this machine is a virtual address based, direct mapped, copy-back cache of 128K bytes with data block size of 32 bytes. The copy-back algorithm is selected for its better performance [3]. A direct mapped cache is picked because of its fast access time and simplicity [4].

A virtual address cache overcomes some of the shortcomings of physical address cache but also comes with some other concerns of its own [5]. A virtual address cache contains blocks of data from the virtual address space. The processor can access the virtual address cache without translation delay. Address translation is performed by the MMU only when the processor experiences a cache miss condition which calls for a system memory access. The memory access operation is completed using the physical address. The size of a virtual address cache scales with the technology as there is no inherent limit to the size of each cache set. The issues associated with the virtual address cache include the requirement of various cache flushing mechanisms and alias address handling. Engineering solutions to these problems usually involve combined effort between system software and hardware. Different cache flush operations can be initiated by the software with the hardware executing the command [2,5]. The synonyms can also be easily detected and processed by hardware if all aliased addresses are made modulo the cache size [2].

2. IO Subsystem Architecture

The system IO transfer in the following discussion uses virtual addresses with supervisor privilege. All the IO devices are represented as individually mapped virtual objects. These references are translated, protected and checked in identical fashion as those in the cpu. The special attributes of IO transfers compared with that of cpu transfers include single and sequential accesses with predominance of writes over reads into system memory. Making the fragmentation of the physical memory transparent to the kernel code handling IO activity saves the effort of doing scatter and gather when the allocated memory crosses the physical page boundary.

The MMU and the slow main memory access are the major bottlenecks for operations in the system under discussion. Non-cached IO transfers are slow and represent poor matches with the cpu speed. The cpu cache has been used [1] to reduce the memory latency and minimize the number of address translations in the system IO transfer at the expense of the stalled cpu. Bandwidth needed for supporting the high performance devices severly cuts the cpu cache utilization. Tables 1. and 2. illustrate the differences for cases of memory read and memory write activities caused by IO transactions using the Sun-4/470 cpu board as an example.

Table 1. IO Device Write to CPU Board			
CPU-Cached IO vs. Non-Cached IO			
Operation Activity	Write Hit	Write Miss	Write Through
IO Transfer	9	26	15
Stolen Cycles	6	23	10
Device cyc/32 Bit	11.125		15
Stolen cyc/32 Bit	8.125		10
Device cyc/16 Bit	10.0625		15
Stolen cyc/16 Bit	7.0625		10

Table 2. IO Device Read from CPU Board			
CPU-Cached IO vs. Non-Cached IO			
Operation Activity	Read Hit	Read Miss	Read Direct
IO Transfer	8	24	21
Stolen Cycles	6	21	21
Device cyc/32 Bit	10		21
Stolen cyc/32 Bit	7.875		21
Device cyc/16 Bit	9		21
Stolen cyc/16 Bit	6.9375		21

The disk technology presently supports disks using SMD interfaces at 2.4 MBytes/sec transfer rate [1]. Products using IPI-2 interface will have transfer rates between 3 MBytes/sec and 10 MBytes/sec. The Ethernet interface is operating at 10 MBits/sec using a 16 bit interface with the cpu. Assuming all IO transfers will be buffered in the cpu cache. There are 33 million machine cycles per second available from a cpu running at 33 MHz. The percentage of stolen cpu cycles for such a system, when one SMD disk and one on-board Ethernet both writing to cpu board in parallel at their peak rate, can be as high as 28.2%. A high performance disk using IPI-2 interface with 10 MBytes/sec transfer rate alone can cause up to 61.6% of cpu cycles to be stolen by the IO activity.

In addition to the cpu cache interference issue, allowing IO data to go through the cpu cache will also cause extra cache misses due to random IO activities, need more memory bus bandwidth to do write-allocation for supporting the partially modified cache lines, as well as require additional cache flush operations for the IO modified cache blocks. Altogether, they could severly degrade cpu's utilization of the processor cache.

Devoting a separate virtual address based cache for system IO transfers can reduce the memory latency and the frequency of address translation. Furthermore, the dedicated IO cache can minimize the cpu cache interference since IO transfers experiencing IO cache hit will not disturb cpu operation. Figure 2 describes the architecture of the cached IO system in relation to the rest of the cpu board.

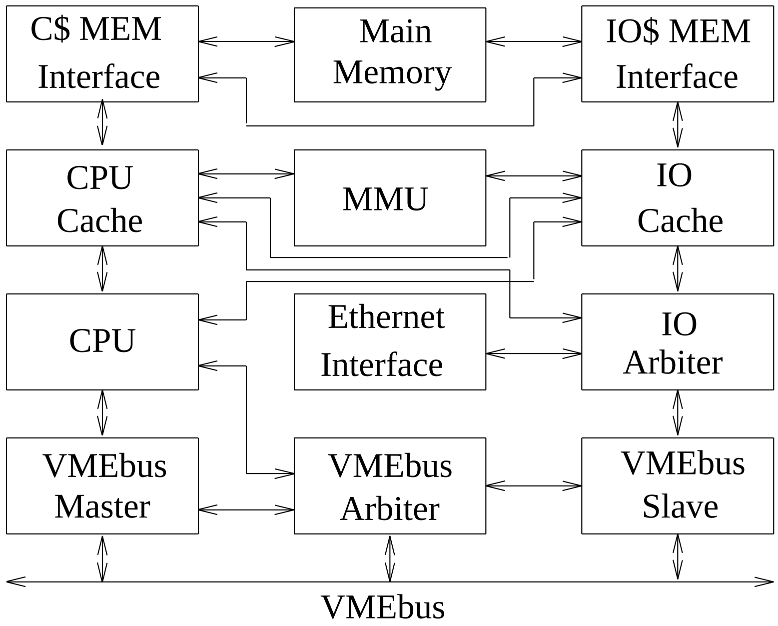

Figure 2. Major Subsystem Architecture

3. IO Cache Architecture and Operation

The virtual memory space of each context is divided into two partitions, the kernel and user space. The system IO transfers discussed herein are mapped into the top 1M bytes of kernel space [2]. The kernel treats memory in units of 8 KByte pages [1]. The base address for the kernel space is a fixed value such that the virtual memory area allocated for IO transactions occupies a common address range throughout all processes. The attributes of pages used for IO activities can thus be established independent of context and the hardware for distinguishing the associated context of different IO transfers can also be eliminated.

The kernel assigns to each IO device a unique set of virtual pages, up to 64 Kbytes at a time for each IO transfer. Figure 3. shows an example of the virtual memory mapping. Given the 1M Bytes of the system IO area, a minimum of 16 different IO transactions can be queued up at one time. By indexing the IO cache mod the virtual page, no two devices can interfere each other's cache entries regardless of their transfer sequence. The total number of virtual memory pages supported by the IO cache is thus selected as 128 [6,7] which requires only a few entries in MMU for the address translation.

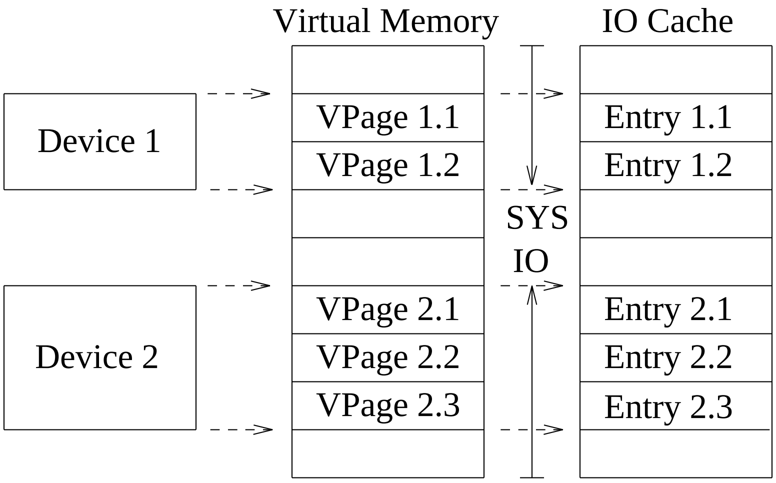

Figure 3. IO Device Mapping Example

Typical copy-back cache hardware can handle partially modified cache block by either allocating a data block on a write-miss [3] or doing the data merge at the write-back time using byte marks to identify the modified data [6]. The sequential nature of virtual addresses for IO data suggests that IO-cached transactions can be made cache-block-based transfers. Doing the cache-block-based transfers will eliminate memory latency and hardware complexity for handling the partially modified cache block. The system software needs to flush the last line of the page to complete the transfer. The cache-block-based transfer scheme also leads to further reduction in hardware by allowing the IO cache to support each page with one data block. The simple cache block replacement algorithm such as replace-on-miss scheme can thus be applied to the IO cache control mechanism. The data block size is chosen at 32 bytes, identical to that of the cpu cache. The data storage capacity is thus 4K bytes for 128 blocks.

The implementation of the IO cache is a direct mapped, virtual address accessed copy-back cache completely constructed with discrete components. In the storage area intended for the system IO data transfers, there are 128 tags corresponding to each of 128 data blocks. Each tag contains the virtual address for the page, the page offset for the buffered line as well as statistics bits like the Valid and Modified bits. The tags and data reside in the same static RAM component which results in cost savings without performance degradation.

The IO cache is used to buffer the high throughput VMEbus activities, the data transfers via Ethernet interface, and the data buffer descriptors for the Ethernet interface. All the data are buffered and supplied by the IO cache when the IO transfer causes a IO cache hit. The cpu cache and system memory are checked only if a miss in the IO cache is encountered. This arrangement allows the cpu to access the processor cache while IO devices are accessing the IO cache. It results in a great reduction in cpu cache interference as illustrated in Table 3 and Table 4.

Table 3. IO Device Write to CPU Board

IO-Cached IO vs. Non-Cached IO

Operation Activity	Write Hit	Write Miss	Write Through
IO Transfer	6	24	15
Stolen Cycles	0	13	13
Device cyc/32 Bit	8.25		15
Stolen cyc/32 Bit	1.625		13
Device cyc/16 Bit	7.125		15
Stolen cyc/16 Bit	0.8125		13

Table 4. IO Device Read from CPU Board			
IO-Cached IO vs. Non-Cached IO			
Operation Activity	Read Hit	Read Miss	Read Direct
IO Transfer	8	51	45
Stolen Cycles	0	13	13
Device cyc/32 Bit	13.375		45
Stolen cyc/32 Bit	1.625		13
Device cyc/16 Bit	10.6875		45
Stolen cyc/16 Bit	0.8125		13

The IO cache lines supporting the VMEbus slave devices are indexed with the virtual page numbers. Ethernet interface has no access to those blocks devoted to the VMEbus interface. This organization guarantees that the Ethernet transfers will never interfere with the VMEbus activities. The number of virtual memory pages assigned to the VMEbus slave devices is less than that of all the available data block sets in the IO cache. The IO cache supports the Ethernet data transfers using lines not allocated for the VMEbus slave devices. Those Ethernet data blocks are indexed with a unique set of pointers independent of the virtual address of the transfer. However, the virtual addresses of the transferred data are used to determine the hit/miss condition in the IO cache as done in the cases for the VMEbus slave devices. The Ethernet transactions which read from the system memory use different IO cache blocks from that for writing into the memory. Separating the cache blocks used in different directions of memory access greatly simplifies the IO cache flush at the end of the transaction because the read buffer will never have its Modified bit set. Flushing a valid but not modified cache line only involves tag invalidation.

There is a separate set of 128 IO cache lines assigned for storing the Ethernet data buffer descriptors. The tag for each of the corresponding line contains the virtual address of the descriptor, the Valid and the Modified bit. From the system software's point of view, the descriptor portion of the IO cache is a linearly addressed storage area which can be pre-loaded as part of the IO cache initialization. The tags for the descriptor lines are marked as Valid and Modified at the descriptor pre-loading time. From the Ethernet interface point of view, the descriptors are always available in the IO cache. This arrangement not only unifies the IO cache interface mechanism with the Ethernet hardware, but also speeds up the data buffer descriptor fetching operation.

4. Data Consistency Between CPU Cache, IO Cache and Main Memory

The cpu cache and IO cache operating together form a multiple cache system which can lead to cache data coherence problems. Hardware based cache coherence scheme such as that using dual cache directory along with the multiple master bus snooping technique have been proposed [8]. However, the effort and cost for implementing dual directory caches in discretes on a space-tight cpu board of a workstation was beyond practical consideration for this project. Since the address translation is required on every IO cache miss for all the IO transfers, the MMU is chosen to be the focal point for hardware based cache coherency scheme. When IO transactions causing a miss in the IO cache, the missed virtual address will be sent to both MMU for translation and cpu cache for coherency checking. The cpu cache is required to execute a line flush if the main memory needs to be updated with the data from the cpu cache. The main memory always supplies data to the requesting devices via the IO cache. The cache coherency scheme guarantees the consistency between the IO data and the data operated on by cpu. Tables 5. and 6. highlight the key elements of the cache coherency algorithm.

Table 5. IO Device Write to CPU Memory via IO Cache					
Page Status		CPU $ Status		Action	
IO $ $able	C $ $able	Line Val	Line Mod	CPU Cache	IO Cache
Yes	Yes	Yes	Yes	Line invalidated	Line WB, invalidated
Yes	No	X	X	No Action	Line WB, invalidated
No	Yes	Yes	Yes	Line WB, invalidated	Data WT
No	Yes	Yes	No	Line invalidated	Data WT
No	Yes	No	X	No Action	Data WT
No	No	X	X	No Action	Data WT
Yes	Yes	Yes	No	Line invalidated	Line WB, invalidated
Yes	Yes	No	X	No Action	Line WB, invalidated
WB = Written Back			WT = Written Through		

Table 6. IO Device Read from CPU Memory via IO Cache					
Page Status		CPU $ Status		Action	
IO $ $able	C $ $able	Line Val	Line Mod	CPU Cache	IO Cache
Yes	Yes	Yes	Yes	Line WB, stay Valid	Line Fill
Yes	Yes	Yes	No	No Action	Line Fill
Yes	Yes	No	X	No Action	Line Fill
Yes	No	X	X	No Action	Line Fill
No	Yes	Yes	Yes	Line WB, stay Valid	Data Fetch
No	Yes	Yes	No	No Action	Data Fetch
No	Yes	No	X	No Action	Data Fetch
No	No	X	X	No Action	Data Fetch
WB = Written Back					

5. System IO Arbitration

The arbitration for access to the IO cache is optimized for the VMEbus slave devices by giving them the default privilege. The Ethernet interface is granted highest priority when its request is competing with other pending devices. Once the Ethernet Interface gains access to the IO cache, it will be allowed to fill up or unload its on-chip FIFO to minimize the possibility of packet underrun or overrun. The IO cache also allows the kernel to preload all the buffer descriptors for the network activities in the IO cache to further reduce the possible memory latency and cpu disturbance. The access to IO cache initiated by the cpu is interleaved with VMEbus slave transfers because of its infrequent occurrence and fast turnaround.

6. Comparison Between CPU Cached Transfer and IO Cached Transfer

The average IO traffic generated by a single user on a 1-MIP engineering workstation is about 2.5 - 10 KBytes/sec [9,10]. Studies also showed that about one third of disk accesses are for disk writes [10]. The extrapolated amount of IO transactions on a 20-MIP machine could be in the range of 50 KBytes/sec in workstation applications to 4.2 MBytes/sec in a server configuration serving 35 clients with 120 KBytes/sec bandwidth required per client. The figure of merit of the overall performance for such a computer system should include its IO bandwidth in addition to its intrinsic computing power. A cost effective design IO cache can improve the IO bandwidth and significantly reduce the cpu hold-up rate compared with using the cpu cache for IO transfers. Table 7. compares the IO bandwidth and the stalled cpu rate based on the linear extrapolation of the effective IO traffic for a 20-MIP system.

Table 7. IO Bandwidth and CPU Degradation

IO Scheme	CPU-Cached IO		IO-Cached IO	
IO Interface	32 Bit	16 Bit	32 Bit	16 Bit
IO Bandwidth MBytes/sec	12.4 max.	6.87 max.	13.4 max.	8.02 max.
Stalled CPU Rate @ 50KBytes/sec	0.31% min.	0.53% min.	0.062% min.	0.062% min.
Stalled CPU Rate @ 4.2MBytes/sec	26.1% min.	44.5% min.	5.21% min.	5.21% min.

7. Conclusion

The architecture of a virtual address cache based IO subsystem is introduced. A cost effective implementation of the proposed IO cache for Sun-4/470 workstation/server has also been described along with the estimates in terms of the effective IO bandwidth and the stalled cpu rate caused by IO traffic. A cached IO subsystem significantly reduces the IO induced cpu cache interference, allows IO transfers in parallel with processor cache activities, and saves considerable amount of memory bandwidth by eliminating the block reads for IO writes. A workstation with such a cached IO subsystem can have a better balanced overall performance in terms of computation power and its IO bandwidth, which in turn, in a server configuration, represents the ability to serve more clients beyond that provided by previous generations.

8. Acknowledgements

The authors wish to express their special gratitude to the following colleagues at Sun Microsystems Inc., namely, John Watkins, David Labuda, and Steve Kleiman for their valuable insights and discussions during the definition phase of this virtual cache based IO subsystem. The authors would also like to extend their special thanks to Herb Lopez-Aguado, Trevor Creary, Brad Eltman, Chien Nguyen, Michael Splain, Ravi Subbarao, Shen Wang, and Dock Williams for their fruitful discussions on the implementation of this virtual address cache based system. Hearty thanks is also due to Greg Blanck and Raymond Ng for their effort in reviewing the manuscripts.

References

[1] Michelle Arden, Andy Bechtolsheim and Mark Hall, *Sun-3 Architecture : A Sun Technical Report*, Sun Microsystems Technical Report, 1986.

[2] Ed Kelly, *Sun-4 Architecture*, Version 1.5, Sun Microsystems Inc. Internal Document, Nov. 14, 1986.

[3] A. J. Smith, *Cache Memories*, Computing surveys, Vol. 14, No.3, Sept. 1982, pp. 473-530.

[4] Mark D. Hill, *A Case for Direct-Mapped Caches*, Computer, IEEE, Dec. 1988, pp. 25-40.

[5] William Van Loo, *Maximizing Performance by Choosing Best Memory*, Computer Design, August 1, 1987, pp. 89-94.

[6] Michael M. Hsieh and Tek C. Wei, *Sunray I/O Cache Architectural Specification*, Sun Microsystems Inc. Internal Document, Version 1.4, May 18, 1987.

[7] Michael M. Hsieh and Tek C. Wei, *Sunray I/O Cache Architectural Specification*, Sun Microsystems Inc. Internal Document, Version 4.0, Dec. 10, 1987.

[8] James R. Goodman, *Coherency For Multiprocessor Virtual Address Caches*, Proc. Symposium on Architectural Support for Programming Languages and Operating Systems, IEEE Computer society, 1987, pp. 72-81.

[9] Lazowska, E., et al., *File Access Performance of Diskless Workstations*, ACM Transactions on Computer Systems, Vol. 4, No. 3, August 1986, pp.238-268.

[10] Ousterhout, John., et al, *A Trace Driven Analysis of the UNIX 4.2 BSD File System*, Proc. Tenth Symposium on Operating Systems Principles, December 1985, pp.15-24.