A Partitioned Translation Lookaside Buffer Approach to Reducing Address Bandwidth

Matthew Farrens, Arvin Park
Rob Fanfelle, Pius Ng, Gary Tyson

Division of Computer Science
University of California
Davis, CA 95616
tel: (916) 752-9678 fax: (916) 752-4767
email: farrens@cs.ucdavis.edu, park@cs.ucdavis.edu

Abstract

This paper presents a simple modification of a computing system's virtual memory hardware that can sharply reduce the number of pins required to transmit address information between a single chip processor and off-chip memory. By partitioning the virtual memory system's translation lookaside buffer (TLB) so that the virtual page numbers are stored in a cache on the processor chip and corresponding real page numbers are stored in a set of registers at the memory, it becomes possible to transmit a small index to the real page number from processor to memory instead of the real page number itself. Trace driven simulations indicate that this technique can significantly reduce the number of pins required to transmit address information between processor and memory. Because this technique makes use of existing virtual memory hardware, it can eliminate address pins without significantly changing computing system design, cost, or performance.

Keywords: Base Registers, Caching, Compression, Microprocessor Design, Translation Lookaside Buffer, Virtual Memory, VLSI.

1. Introduction

Fueled primarily by advances in VLSI technology that have reduced the minimum feature size by well over an order of magnitude, the cycle time of a VLSI processor is now approaching that of the CRAY-1. In addition, the move away from CISC instruction sets has allowed computer architects to exploit the performance-enhancing technique of pipelining. Since extensively pipelined VLSI processors generate memory requests at a much higher rate than non-pipelined machines, the demands on the I/O bandwidth of a processor are intensified, and the penalty paid for off-chip requests quickly becomes the dominant bottleneck to high performance.

While technological advances have greatly increased the density and speed of VLSI circuits, they have done little to increase the number of I/O pins available. As the minimum feature size of a VLSI process decreases by a factor of two the number of devices that can be fabricated in a unit area increases by a factor of four. However, the number of I/O pins that can be fabricated in a unit area remains a constant, since there are other external physical constraints (such as minimum area for wire bonding) that determine I/O pad size.

The fact that the amount of computing power per I/O pin has grown tremendously means that I/O pins are now a precious commodity, and must be used carefully. It also means that certain brute force techniques employed by supercomputers to increase the I/O bandwidth (such as providing extremely wide busses between memory and processor) cannot be used in VLSI. Many other techniques remain available, however - techniques such as providing separate I/O pins for addresses and data, employing separate uni-directional busses for both instruction addresses and data, and using on-chip instruction and data caches. It is up to the VLSI designer to carefully select the ones that will benefit the processor performance the most.

As VLSI processor address spaces move inexorably from 16 bits towards 64 bits, other factors will make even these techniques more difficult to use. One major problem in VLSI I/O design is controlling switching transients. If 32 address lines all change simultaneously, for example, a significant amount of noise is generated. One standard technique used to minimize such transients is to alternate active I/O and ground pins. Such a technique is feasible for 16 or 32 active pins, but a dedicated address bus of 64 bits would require 128 pins to implement, and providing separate instruction and data address lines would require 256 pins. Obviously, such an arrangement rapidly becomes untenable.

This also assumes only one processor per silicon die. As circuit densities continue to increase, it is inevitable that multiple processors will be fabricated on a single die. This will create an even greater demand on the I/O system, and necessitate a judicious use of the I/O pins.

In this paper a method for reducing the number of address lines necessary to transmit address information from processor to memory will be presented. This can be accomplished by making minor modifications to a processor's virtual memory system, which will allow up to a 75% decrease in the number of address lines necessary (in the case of a 64-bit address space). Section 2 presents a background of how this is done, section 3 provides an overview of the proposed technique, section 4 describes the simulation environment used, section 5 contains the proposed modifications to the virtual memory system necessary to implement this proposal, section 6 contains to example implementations, and sections 7 and 8 presents our conclusions and summary.

2. Background

Since I/O pins are a precious commodity, it is vital to increase the amount of *useful* information transferred per I/O pin per unit time. This can be accomplished (in the case of address streams) by noting that instruction address streams tend to be sequential, with references generally differing only in the low order bits. Since the higher order bits remain unchanged, they contain essentially *redundant* information. By recoding the information to be transferred, this redundant and/or useless information can be largely eliminated, and a much smaller number of I/O pins can be used.

In order to eliminate the redundant information in the address stream (and reduce the address pin count), we developed a technique we refer to as *Dynamic Base Register Caching* (DBRC) [FaPa91a, PaFa90]. This technique will be briefly described below.

2.1. Dynamic Base Register Caching

In Dynamic Base Register Caching (DBRC) hardware divides each address word into two components, a high order component (a base) and a low order component (an offset). The low order component is transmitted directly from processor to memory, while commonly referenced high order components are stored in matching sets of base registers located in both the processor and the memory.

Each time a new address is generated by the processor, the set of dynamically allocated base registers in the processor is searched for a register whose contents match the high order component of the address. In order to facilitate a fast search for a new index, the set of base registers *in the processor* is configured as a cache. If the search is successful, the location of the register in the cache is transmitted to the memory, instead of the high order component itself. The memory uses this index to extract the proper high order component of the address from its base register file. (Note that the set of registers at the memory are configured as a register file,

not as a cache). The complete address is then regenerated by concatenating the transmitted low order component with the extracted high order component. A diagram of the scheme appears in Figure 1.

Since the base register index transmitted between processor and memory is typically much smaller than the high order component it replaces, a significant reduction in the number of address lines is achieved. We refer to the combined index and the low order component as the *reduced* address.

If the high order component of an address is not found in one of the processor base registers, a fault occurs. One of the base registers at the processor is immediately overwritten with the new high order component, and memory is informed of this fault by the transmission of a reserved index bit pattern. The new high order component is then transferred by the processor across the reduced address or data lines, and loaded into a corresponding base register at the memory.

Extensive trace driven simulations of Dynamic Base Register Caching [FaPa91a, FaPa91b] have shown that DBRC is highly effective, and that on average over 99% of all address

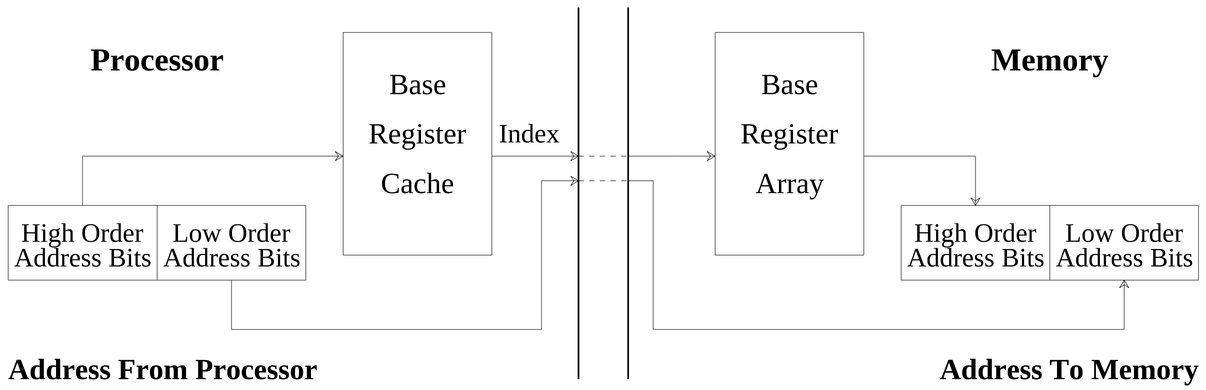

Figure 1

Overview of DBRC

references can be handled with a reduced address size of 16 bits. This also implies that 50% of a processors address bandwidth (25% of the total I/O bandwidth) is used less than 1% of the time, which is a significant waste of I/O resources.

3. Integrating DBRC into a Virtual Memory System

Dynamic Base Register Caching is effective because it takes advantage of the locality exhibited by memory reference streams. This same locality is also exploited by a virtual memory system's *Translation Lookaside Buffer*. In this section we present a way to capitalize on the fact that a Translation Lookaside Buffer and a DBRC perform very similar functions, by integrating a DBRC into a virtual memory system's TLB.

3.1. Basic Partitioned TLB Structure

A Translation Lookaside Buffer (TLB) stores common translations from virtual to real page numbers, which are merely the high order components of both virtual and real addresses. The page offsets, or lower order address components, are the same for corresponding real and virtual addresses, and are sent directly to the memory without passing through the TLB. (see Figure 2a).

A virtual memory system's TLB is typically located as a single unit on a CPU chip, or on a separate Memory Management Unit (MMU) chip. If the MMU is on a separate chip, the entire virtual address generated by the CPU must be sent to the MMU in order to calculate the requisite physical address. If the MMU resides on-chip with the CPU, then the virtual to physical translation is calculated locally and the proper physical address must then be sent off-chip to the memory system.

In order to realize a reduction in the required number of address pins, we propose to partition a standard TLB (figure 2a) into two separate functional units (figure 2b). One portion of the TLB will reside on the processor chip, and will be referred to as the *processor TLB.* This TLB will be configured as a cache, and will contain only virtual page numbers. The other portion of the TLB will be located external to the CPU (either on a separate MMU chip, or in the address decoding logic of the main memory) and will be referred to as the *memory TLB.* The memory TLB contains the real page numbers corresponding to the virtual page numbers stored in the processor TLB, and is organized as a register file. Note that each virtual page number entry in the processor TLB cache corresponds to a unique entry in the memory TLB register file.

This partitioned TLB functions as follows: During a memory access, the virtual address generated by the CPU is divided into a (virtual page number, page offset) pair. The page offset is transferred directly to memory, while the processor TLB is searched for the virtual page number.

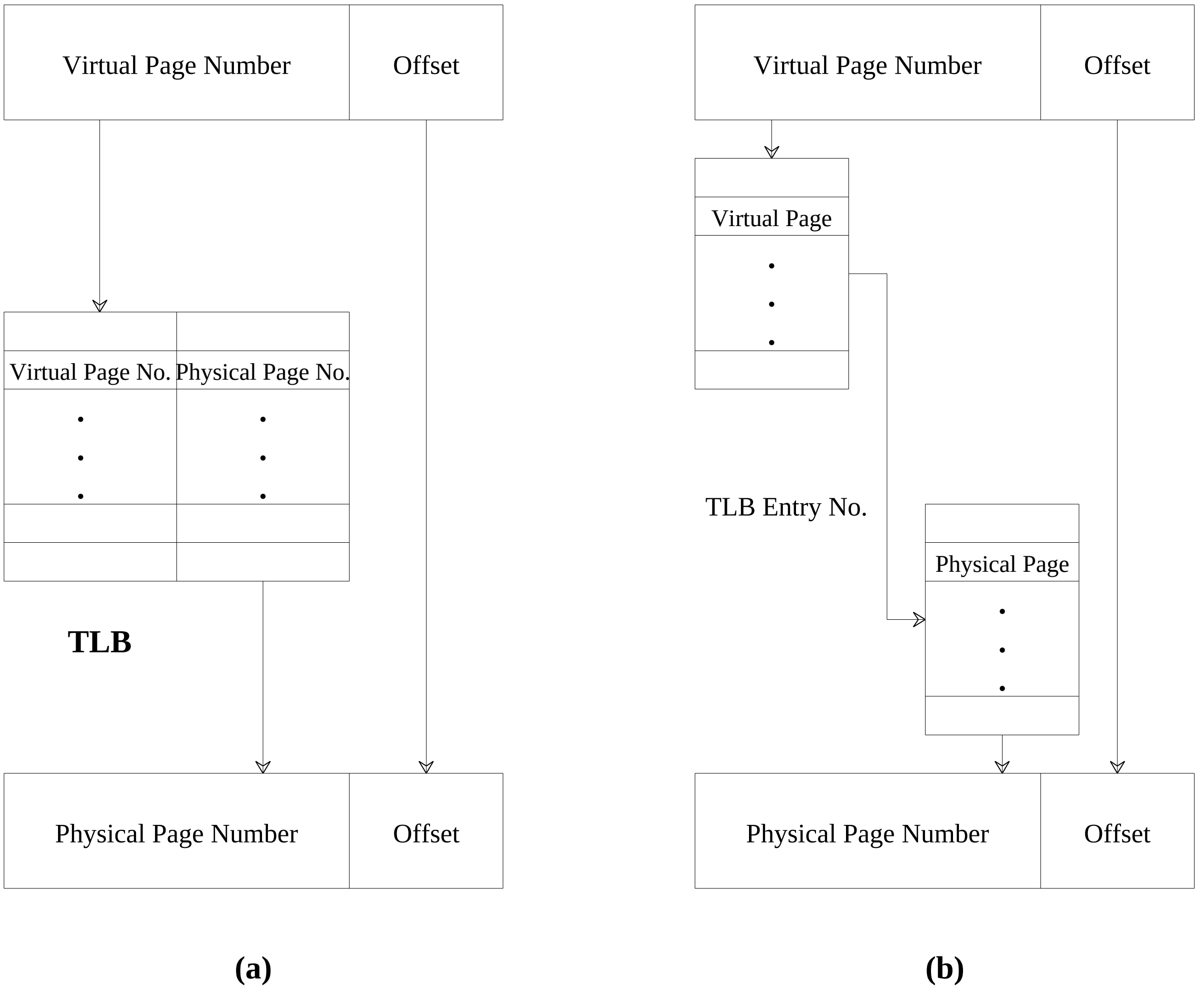

Figure 2

Possible TLB Configurations

If the virtual page number is found in the processor TLB, the index specifying the virtual page number's location in the processor TLB is transmitted to the memory TLB. (Since there is a one-to-one correspondence between the virtual and corresponding real page numbers, this index is all that is necessary to identify the correct real page number in the memory TLB). The complete real address is then formed from the concatenation of the page offset and the real page number.

This partitioning of the TLB makes it possible to transfer a small virtual page number index from processor to memory instead of the entire virtual page number itself. Such an approach can provide a significant reduction in required address lines between processor and memory.

Virtual page numbers not found in the processor TLB will cause a TLB miss. During a TLB miss, a walk through the levels of a hierarchically organized process page table is performed to locate the proper real page number. This table walk proceeds in the same manner as a table walk for a normal unified TLB, and is shown in figure 3. The *Base Page Table Register* (BPTR), configured as part of the memory TLB, contains the physical address of the base page table which resides in main memory. When a TLB miss occurs, the *Base Page Table Offset* (BPTO) bits of the virtual address are transferred from the CPU to the memory TLB via the offset lines and are used by the memory TLB as an offset into the base page table. The selected

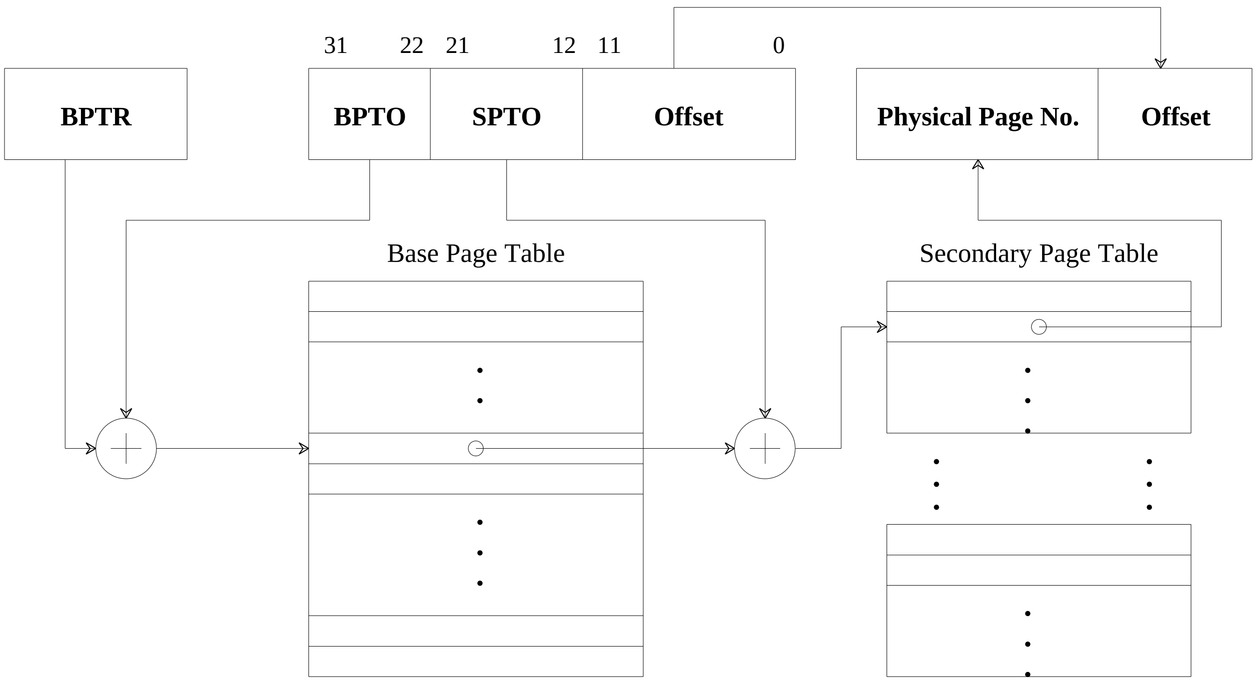

Figure 3

Overview of Table Walk

base page table entry contains the base address of a secondary page table for this region of virtual addresses. The *Secondary Page Table Offset* (SPTO) is then sent over the offset lines to the memory TLB, and is used as an index into the secondary page table. At this point the offset bits from the original virtual address are sent across the offset lines and are combined with the physical page number extracted from the secondary page table to generate the complete real address for the memory reference.

In total, it takes three cycles to transfer the BPTO, the SPTO, and the page offset bits from the CPU to the memory TLB:

cycle 1 - transfer BPTO
cycle 2 - transfer SPTO
cycle 3 - transfer OFFSET

However, since a page table lookup must be performed after each transfer, the cost of requiring multiple cycles to transfer the virtual address may be completely subsumed by the multiple page table lookups required by the virtual memory system. Since these table lookups are required on TLB misses for *any* virtual memory scheme, the partitioned TLB approach does not introduce any significant delay in TLB fault processing.

4. Simulations

In order to determine the optimal configuration for the partitioned TLB a series of trace-driven simulations were performed. The focus of these simulations was to investigate the effects of TLB size, page size, and index pattern reservation on TLB hit rate. These simulations are similar to the ones performed in order to validate the concept of Dynamic Base Register Caching (DBRC) [FaPa91a]. However, the DBRC cache sizes studied in the previous simulations were much smaller than the size of a typical TLB. In addition, unlike a DBRC miss, a TLB miss requires multiple main memory accesses to retrieve page table entries. Because a TLB miss is so time-consuming, TLBs are made larger, and page sizes are increased to ensure that TLB misses occur less frequently.

Simulations were also performed in order to determine the performance degradation due to reducing the number of TLB entries by one or two. This was necessary because one possible implementation of the partitioned TLB uses a reserved index pattern to indicate a TLB miss. Reserving this index pattern reduces the number of usable entries in the TLB by one, and the effects of this reduction had not yet been investigated.

The effects of reserving a second index pattern was examined because it may be advantageous to use a reserved index pattern to indicate an I/O transaction. By using the offset lines in conjunction with this reserved I/O index, a separate I/O address space can easily be supported.

4.1. The Simulation Model

A total of 17 traces were used in the simulations. Nine of the traces were 500,000 reference ATUM traces [AgSH86], and eight were 3,000,000 reference traces of the SPEC benchmark programs gathered using Pixie on a DECstation 5000/200. The SPEC traces were constructed by randomly selecting 50,000 reference samples of the SPEC benchmark program in execution. It would be preferable to use entire traces instead of samplings, but time and hardware constraints prevented us from doing so.

The PTLB simulations covered a range of page sizes from 512 bytes to 8192 bytes, and a binary range of TLB sizes from 32 to 256 entries. Both fully associative and 4-way set associative cache organizations were simulated across the entire range of TLB and page sizes.

In order to assess the performance effects of reserving an index pattern, we also performed simulations that decreased the selected TLB size by one and by two. Since 4-way set associative caches require an even multiple of 2 entries, when an index pattern is reserved one (or two) of the sets is left with only 3 way associativity.

4.2. Discussion of the Simulation Results

In the simulation results presented in this section, the TLB miss rate is measured as a function of the size of the *reduced address*. The reduced address is determined by adding the number of bits reserved for the page offset and the number of bits needed to represent the entry in the TLB. For example, a virtual system with a 4K page size and a 64 entry TLB will have a 12 bit page offset and a 6 bit index, which translates into an 18 bit reduced address.

Figure 4 shows the effects of TLB size on TLB performance for the SPEC traces and for the ATUM traces. It is apparent that there is a significant difference between the way a fully associative TLB performs on the different traces, and is due to the existence of OS calls in the ATUM traces. Simulations were done on the ATUM traces with the OS calls removed, and the resulting graphs looked virtually identical to figure 4a. Since the overwhelming preponderance of computers *have* Operating Systems, it is apparent that the SPEC traces are of little use in determining the ideal TLB size and configuration.

In figure 5 fully associative TLB's are compared to 4-way set associative TLB's. (It should be noted that figure 5a is a repeat of figure 4b with a different scale.) As can be seen in figure 5, the number of entries in the TLB has the greatest effect on the miss ratio (up to a point). However, no significant performance increase is observable once the number of TLB entries goes beyond 64. Furthermore, the choice of page size does not appear to effect the relative

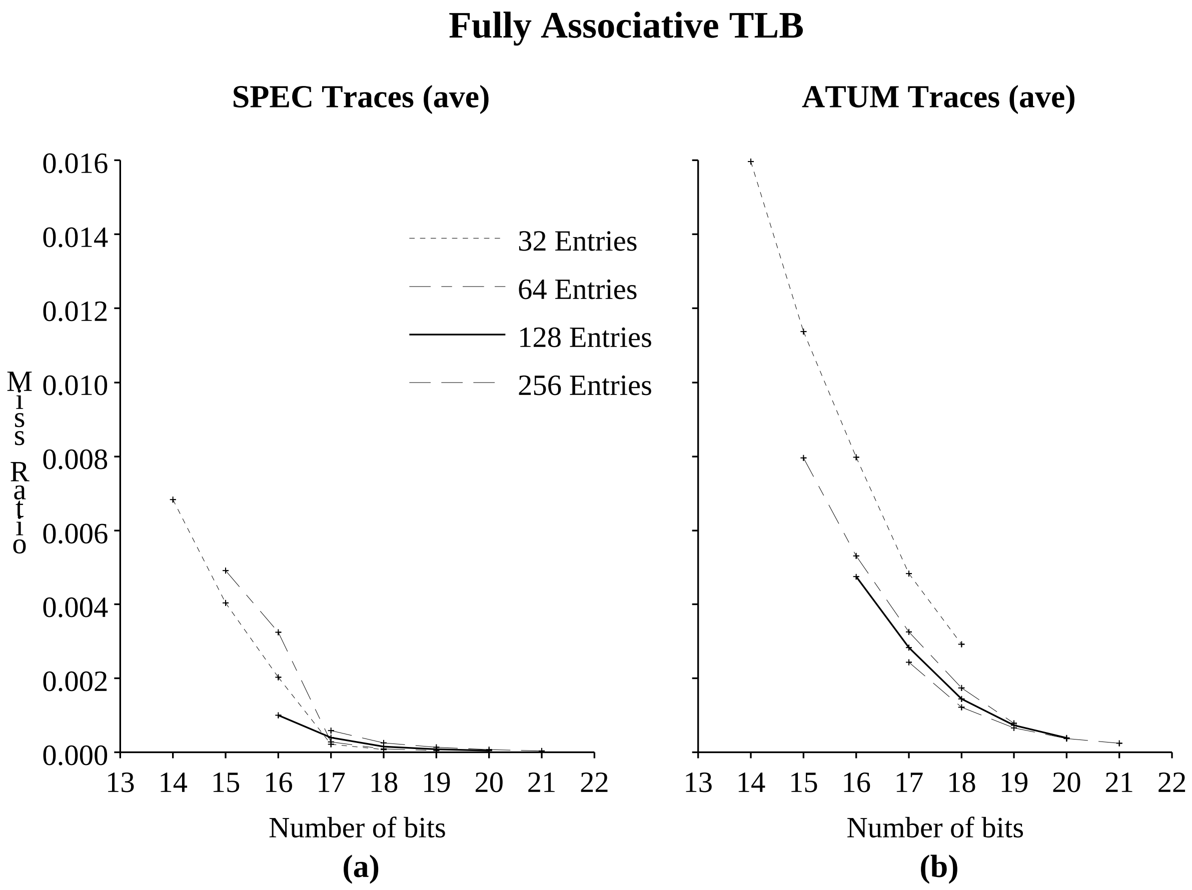

Figure 4

Comparison of SPEC vs. ATUM Traces (ave)

performance of the various TLB sizes. For a fully associative TLB (figure 5a) without reserved slot, for example, the miss rate is decreased by .3% to 1% (absolute scale) with the increase from 32 entries to 64 entires. Any further increase in the number of TLB entries decreases the miss rate by less than 0.05%.

Figure 5b shows how the use of reserved slots in a 4-way set associative TLB results in a minor increase in the miss rate. In the 64 entry 4-way set associative TLB, the increase in miss rate for the first reserved slot was less than .05%, and .19% for the second slot. For the 32 entry 4-way set associative TLB, the increase in miss rate was much more dramatic; approximately .18% for the first reserved slot and .32% for the second. While this may seem an acceptable increase in miss rate, the relatively high TLB miss cost necessitates a miss rate as small as practical. (The fully associative TLB exhibits a similar degradation of performance due to the existence of reserved slots, but at a much reduced level.)

These figures indicate that there is little advantage in increasing TLB size above 64 fully-associative entries. However, selecting the ideal reduced address size is not as straightforward. With a reduced address size of 16 bits (64 entry PTLB and a 1K page size) the TLB miss rate is just over .53%. By using 1 more bit (increasing the page size to 2K) the TLB miss rate drops to .32%. In fact, figure 5a shows that the decrease in miss rate is nearly linear with respect to

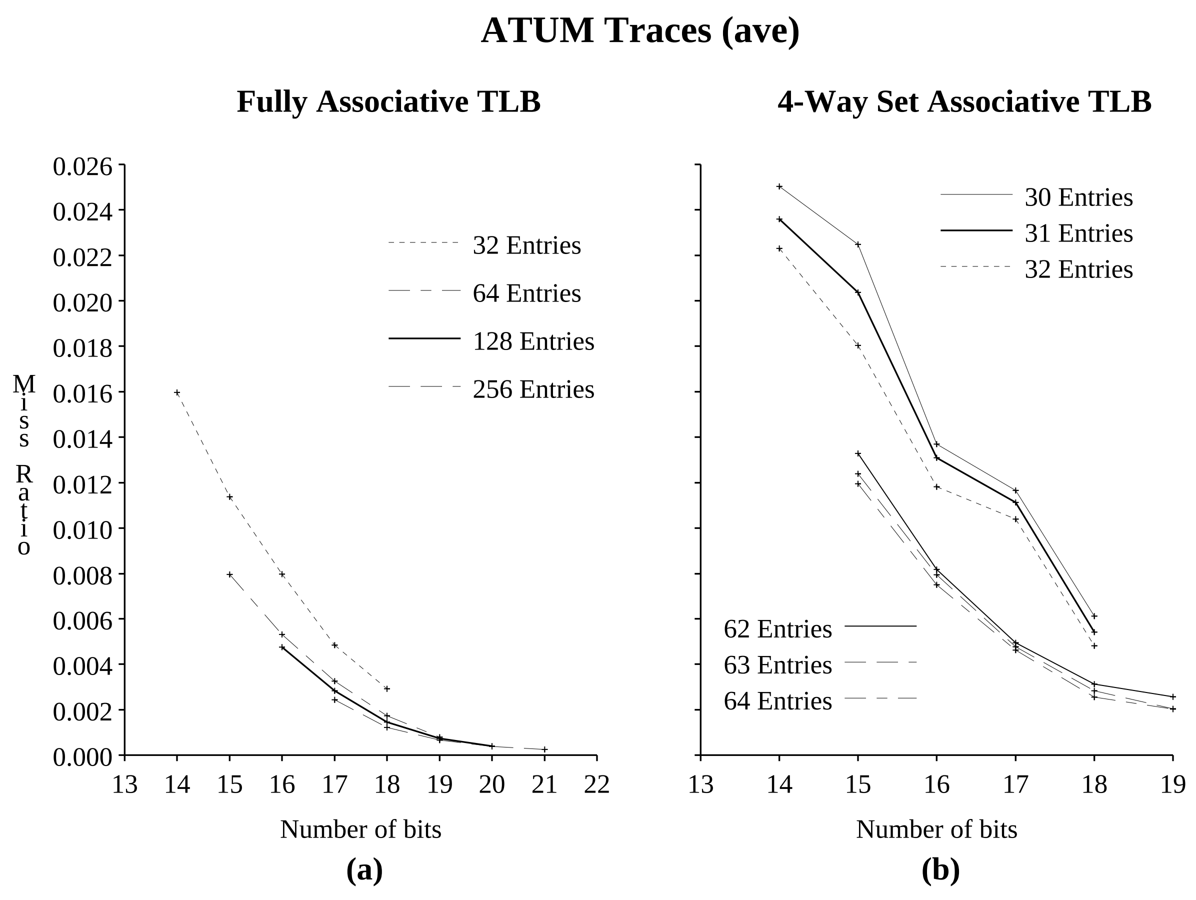

Figure 5

Comparison of Fully Associative vs. Set Associative TLB's

number of bits from a reduced address size of 16 up to a reduced address size of 19. Selecting the reduced address size to implement for a given system will not be an easy decision, and will depend on a number of other factors (such as scarcity of I/O pins and type of workload.)

5. Implementation Considerations

There are a number of factors that must be taken into account before deciding on a design for a partitioned TLB. Two of these are presented below.

5.1. Location of TLB Cache Replacement Logic

In a partitioned TLB, both portions of the TLB must know which TLB entry is to be replaced on a cache miss. The logic that determines which entry is to be replaced can either be replicated on both the processor TLB and memory TLB, or it may be restricted to only one TLB portion.

If there are no space limitations on either the processor or the memory TLBs, then the cache replacement logic would ideally be implemented on both portions of the TLB. This would eliminate the need for cache maintenance communication between the two sides.

If the amount of space available for the cache replacement implementation *is* limited, the designer may opt to restrict the implementation of this logic to only one portion of the partitioned TLB. This partition will be responsible for both maintaining the cache replacement policy and informing the other partition which cache entry is to be replaced on a cache miss. This can be accomplished by transmitting the index of the replaced entry to the other partition of the TLB across the index lines (which are unused during the sequential transfer of the virtual page number to the memory TLB). Although implementing the cache replacement logic on only one portion of the TLB will save some chip area, the additional logic and data paths required to transmit this information may significantly complicate system design.

5.2. Interaction between PTLB and System Caches

Another issue to consider is the interaction of instruction and/or data caches with the partitioned TLB system. It has been shown [FaPa91a] that the presence of a virtual-addressed cache on the CPU chip does not impact the performance of the basic DBRC system, since cache and DBRC lookups can occur simultaneously. Analogously, the existence of an on-chip virtual-addressed cache should not affect the performance of a partitioned TLB, since TLB lookups can be performed concurrently with an on-chip cache access, effectively hiding the TLB lookup overhead.

Dealing with an external real-addressed cache is straightforward as well. The virtual address generated by the CPU is translated and moved off chip through the partitioned TLB system as described in section 3, before being presented to the off-chip cache.

Supporting a real-addressed cache on the CPU chip is slightly more difficult. Such a configuration requires the existence of a full TLB on chip with the cache, in order to perform the virtual to real translations before each cache access. However, the reduced address can still be transmitted to the memory TLB, which is essentially a copy of the real address store already residing on the CPU. These two copies must be kept consistent, so on a TLB miss the real page number has to be sent from the memory to the on-chip TLB. This transfer can occur while the main memory access is taking place.

5.3. Indication of TLB miss

There are two different ways to have the processor TLB inform the memory TLB of a miss. One approach is to encode the TLB miss information in the index pattern transmitted between the processor and memory TLB's. Such an approach reduces the number of usable processor TLB entries, since at least one index pattern must be reserved to indicate a TLB miss. However, simulation results have shown that given a relatively large fully associative TLB of size n, the hit rate of a TLB of size (n-1) does not degrade significantly.

A second technique is to use a separate control line to inform the memory TLB of a miss, which eliminates the need for a reserved index pattern. The cache size can now be a straight power of 2, simplifying the implementation of an n-way set-associative cache. Unfortunately, this slightly simplified cache design comes at the expense of an extra I/O pin to indicate a TLB miss. The choice of whether or not to add an extra line to indicate a miss is determined by balancing the cost of an extra I/O pin against the resulting simplification in TLB design.

6. Example PTLB Implementations

In this section two different possible PTLB designs will be presented, one assuming the presence of virtual-addressed cache on chip with the CPU, and a second assuming the more difficult case of an on-chip real-addressed cache and/or an off-chip virtual mapped cache.

6.1. Case 1

Figure 6 shows a block diagram of the partitioned TLB assuming an on-chip virtual-addressed cache and/or an off-chip real-addressed cache. In this design, the processor TLB is configured as a 64-entry fully associative cache, for reasons presented earlier. The TLB uses a dedicated TLB miss control line, and an LRU replacement policy (with the replacement logic is

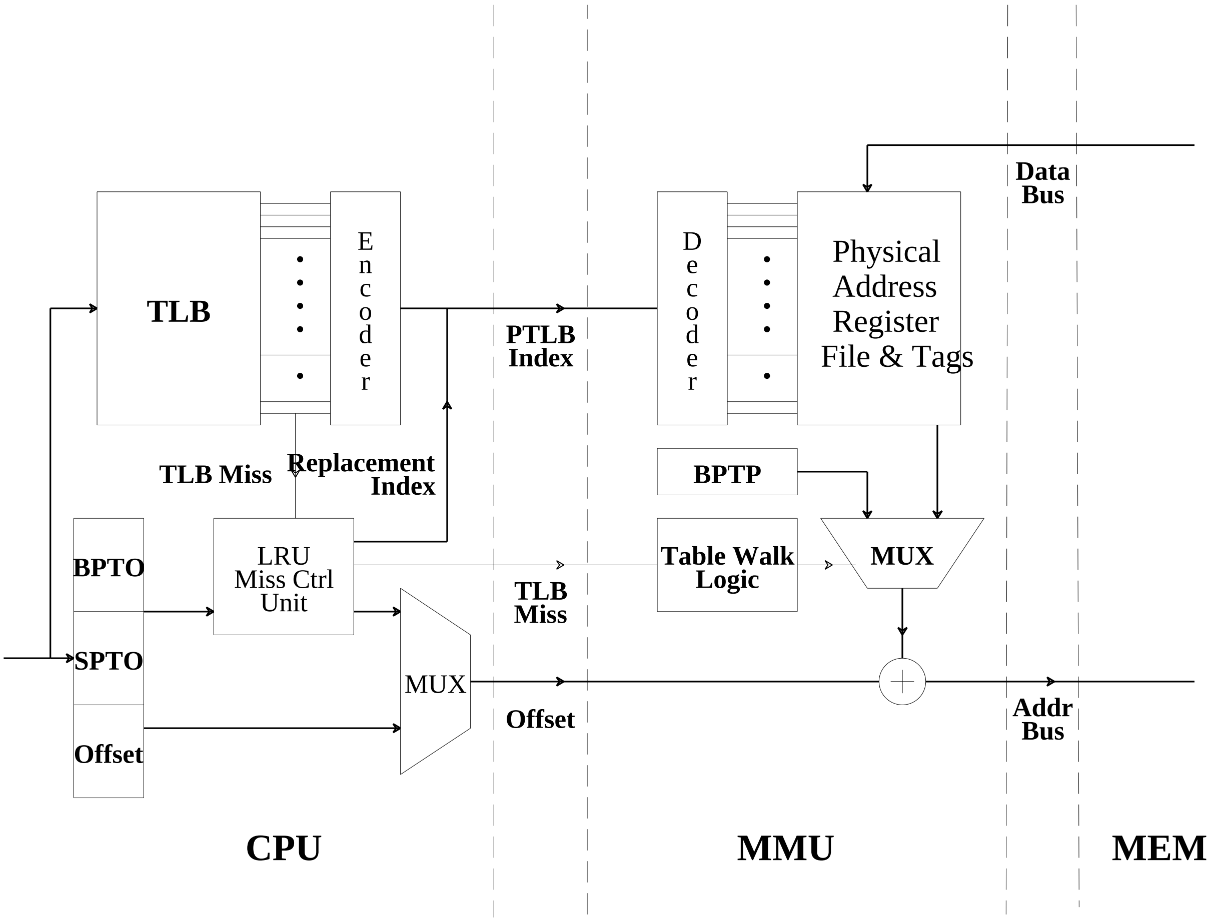

Figure 6

PTLB Design Assuming On-chip Virtual-addressed Cache and/or External Real-addressed Cache

located on the CPU portion of the TLB).

This partitioned TLB implementation operates in the following way: On a memory access, the virtual address is sent to both the on-chip cache and the processor TLB. If the data is available in the cache, the processor TLB result is invalidated and the system is ready to accept the next address. If the requested item is *not* in the cache and a hit occurs in the processor TLB, an index indicating the location of the real page number along with the page offset is sent to the memory TLB. The memory TLB decodes the index and accesses the register file entry containing the correct real page number. This real page number is then concatenated with the incoming offset to form the desired real address. Finally, the LRU maintenance tags in the processor TLB are updated, and the operation is complete.

On a processor TLB miss, the processor TLB notifies the memory TLB of the miss via the TLB miss line. The two offsets that comprise the virtual page number, the base page table offset and the secondary page table offset, are also sent sequentially across the offset lines to the memory TLB. The memory TLB proceeds with a standard page table walk (as described in section 3), combining the separate portions of the virtual page number with the local segment and page table base address registers and traversing the page tables. Once the real page number is obtained it is concatenated with the offset to form the complete real address.

The new real page number must also replace one of the real page number entries in the memory TLB according to the LRU replacement policy. This is accomplished by transmitting the index (indicating which processor TLB entry was replaced) to the memory TLB, and is used to determine which entry in the physical address register file corresponds to the new virtual address entry in the processor TLB.

As pointed out before, replicating the LRU replacement logic at both the processor and memory TLB portions eliminates the need for bidirectional communication between processor and memory TLB partitions. This requires extra logic, but under certain circumstances may be an attractive alternative.

6.2. Case 2

Figure 7 shows a block diagram of the more complicated case of a partitioned TLB assuming an on-chip real-addressed cache and/or an off-chip virtual-addressed cache. This design also assumes PTLB misses will be indicated by a reserved index pattern. Because the cache located on chip with the CPU is real-addressed, all addresses must be fully translated before being sent to the cache. Therefore, an entire MMU must reside on the CPU chip, as shown in figure 7. This figure also shows all the hardware necessary to support both an on-chip real-addressed and an external virtual-addressed cache. The circuitry labeled **<A>** is required only by the on-chip real-addressed cache, and the circuitry labeled **<B>** is only necessary to support the off-chip virtual cache. Thus, in a system with both on-chip and an off-chip real-addressed caches, the circuitry labeled **<B>** could be eliminated. The rest of this discussion will be restricted to the on-chip real-addressed cache configuration, with the understanding that handling an off-chip virtual cache requires only a minor modification to the steps necessary to handle the on-chip real-

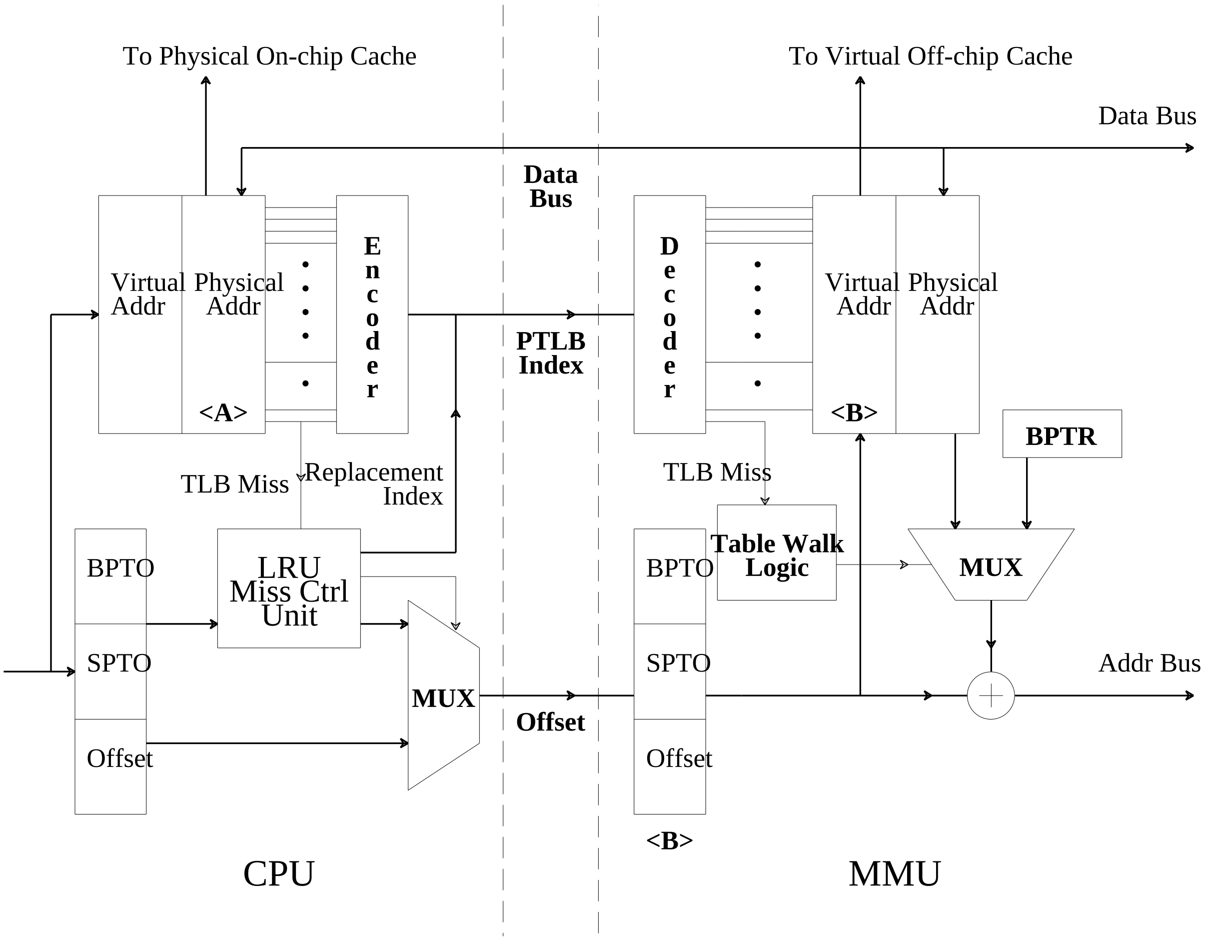

Figure 7

PTLB Design Assuming On-chip Real-addressed Cache and/or Off-chip Virtual-addressed Cache

addressed cache.

The operation of this design differs somewhat from the design used in the case of the on-chip virtual-addressed cache. On all memory accesses, a TLB lookup for the virtual address is performed. If the address is resident in the TLB the corresponding physical address is then sent to the on-chip cache, and a cache lookup occurs. If the desired element is in the cache, then the system is done with this address and ready to accept the next one.

If the element is *not* in the cache, then the offset and PTLB index is sent to the external MMU, where the corresponding physical page number is read out of the physical address file and concatenated with the offset to reconstruct the correct physical address. (Note the similarity between this and DBRC.)

Handling a TLB miss requires 3 cycles, which proceed as follows:

1. During the first cycle, the BPTO is sent over the offset lines, and the reserved index pattern indicating a miss is sent over the PTLB index lines. The memory TLB begins the base page table read.
2. On this cycle, the SPTO is transmitted over the offset lines, and the replacement index (indicating which location is to be replaced) is sent over the PTLB index lines. The memory TLB begins the secondary page table read.
3. Finally, on the third cycle the physical page number returns on the data bus and the processor TLB is updated. The memory TLB also grabs the physical address and places it in the physical address register file.

7. Future Work

In this paper we have only considered implementing the memory TLB portion of a partitioned TLB on a separate MMU chip. The MMU chip takes in a reduced address and constructs the full real address which it sends over a bus to the memory. Although this implementation reduces the number of address pins emanating from the processor, it does not reduce the size of the address bus between processor and memory. An alternative implementation eliminates the separate MMU chip, and locates the memory portion of the TLB in the chip select logic at the memory. This has the advantage of eliminating the delay involved in passing address signals on-to and off-of a separate MMU chip, as well as reducing the number of bus lines between processor and memory.

Moving the memory TLB into the chip select logic has the disadvantage of complicating system design. It becomes necessary to maintain a separate copy of the memory TLB in the chip select logic of each memory board, because the reduced address must be decoded by each memory board to determine if the memory location resides on a particular memory board. Because the memory bus uses only reduced addresses, any DMA device accessing memory directly must also have a copy of the memory TLB to translate a real address into a reduced address.

Although replicating the memory TLB complicates design somewhat, the benefits of reduced bus size may outweigh the cost of this increased complexity. The merits of these implementation options must be investigated in further detail.

8. Conclusions

This paper has shown how a simple modification to a processor's virtual memory system can provide a significant reduction in the number of pins required to transmit address information between procesor and memory. Several different modifications to a standard virtual memory system are presented, in order to show in detail how the modified virtual memory system will function. In addition, the results of trace driven simulations are presented which show that past a certain point the size of a fully-associative TLB has a minimal effect on TLB miss ratio.

Since this strategy makes use of existing memory management hardware, and has almost no impact on system performance, it can provide a very practical method of reducing the I/O bottleneck that is going to occur as processor address spaces grow and as more and more processors are incorporated onto a single silicon die.

9. References

[AgSH86] A. Agarwal, R. L. Sites and M. Horowitz, ''ATUM: A New Technique for Capturing Address Traces Using Microcode'', *Proceedings of the Thirteenth Annual International Symposium on Computer Architecture*, Tokyo, Japan (June 2-5, 1986), pp. 119-127.

[FaPa91a] M. Farrens and A. Park, ''Dynamic Base Register Caching: A Technique for Reducing Address Bus Width'', *Proceedings of the Eighteenth Annual International Symposium on Computer Architecture*, Toronto, Canada (May 27-30, 1991), pp. 198-207.

[FaPa91b] M. Farrens and A. Park, ''Workload and Implementation Considerations for Dynamic Base Register Caching'', *Proceedings of the 24th Annual International Symposium on Microarchitecture*, Albuquerque, New Mexico (November 18-20, 1991), pp. 62-68.

[PaFa90] A. Park and M. Farrens, ''Address Compression Through Base Register Caching'', *Proceedings of the 23rd Annual Symposium and Workshop on Microprogramming and Microarchitectures*, Orlando, Florida (November 27-29, 1990), pp. 193-199.