PAGED CACHE: AN EFFICIENT PARTITION ARCHITECTURE FOR REDUCING POWER, AREA AND ACCESS TIME

Yen-Jen Chang, and Feipei Lai

Dept. of CSIE
National Taiwan University
No.1, Sec. 4, Roosevelt Road, Taipei, Taiwan
Phone:886-2-33667569, Fax: 886-2-23637204,
Email:d88017@csie.ntu.edu.tw

Dept. of CSIE & EE
National Taiwan University
No.1, Sec. 4, Roosevelt Road, Taipei, Taiwan
Phone:886-2-33665003, Fax:886-2-23637204,
Email: flai@cc.ee.ntu.edu.tw

ABSTRACT

Power consumption is an increasingly pressing problem in high performance processors, and the caches usually consume a significant amount of power. This paper presents a new cache partition architecture, called *paged cache*, which is beneficial for area, power and performance. In the *paged cache*, we divide the entire cache into a set of partitions, and each partition is dedicated to only one page cached in the TLB. By restricting the range in which the cached block can be placed, we can eliminate the total or partial tag depending on the partition size. Furthermore, by accessing only a single partition, instead of accessing the entire cache, both the power consumption per cache access and the average access time can be reduced largely. We use *SimpleScalar* to simulate the *SPEC2000* benchmarks and perform the *HSPICE* simulations (with a 0.18 μm technology and 1.8V voltage supply) to evaluate the proposed architecture. Experimental results show that the *paged cache* is very efficient in reducing both power consumption and tag area of the on-chip L1 caches, while the average access time of cache can be improved.

1. INTRODUCTION

The on-chip caches have been shown to be one of the major power and area consumers. To allow high clock frequencies to be used, these on-chip caches are implemented using arrays of densely packed *SRAM* cells. The number of transistors devoted to the on-chip caches is often a significant fraction of the total transistor budget for the entire chip. As the on-chip cache size keeps increasing, the power dissipated by the on-chip caches become significant (e.g., 25% of the total chip power in the DEC 21164 [1], 43% of the total power in the SA-110 [2]), but also it occupies a large portion of the chip-area [3]. Since this trend will likely continue as processors become more sophisticated, both area and power savings resulting from a cost-effective on-chip cache design can have a significant impact on the overall system performance.

Clearly, cache is one of the most attractive targets for both power and area reduction. There have been several techniques for reducing the cache power consumption [4][5][6][7] and the cache area cost [8][9][10].

In this paper, we investigate a cache architecture from three different perspectives: area, power and performance (i.e., access time). To reduce both the area cost of implementing the tag array and the power consumption per cache access, we took a closer look at the relationship between the *translation lookaside buffer (TLB)* and the cache. The TLB is a small associative memory that maps virtual page number to physical page number. As we know, in most conventional architectures, the cache access follows the TLB access. The cache hit or miss is independent of the result of TLB access. In contrast, we propose a new cache architecture, called *paged cache*, in which the cache memory holds only blocks belonging to the pages that are contained in the TLB. We further divide the entire cache into a set of partitions. All partitions have the same size and the number of partitions is the same as the entry number of TLB, i.e., the number of pages cached in TLB. Each partition is dedicated to one and only one page, and thus the tag area can be dramatically decreased. In addition, by triggering a single partition per access, both the power consumption and the access time can be reduced largely.

The rest of this paper is organized as follows. Section 2 identifies the problems of the conventional cache architecture, and we describe the details of the proposed paged cache architecture. In Section 3, we give a detailed power estimation model and timing simulation for the proposed architecture. Experimental results are given in Section 4, and Section 5 offers some conclusions.

2. PAGED CACHE ARCHITECTURE

2.1 Conventional Cache

In most computers, caches are accessed with the physical address. After TLB translation, the physical

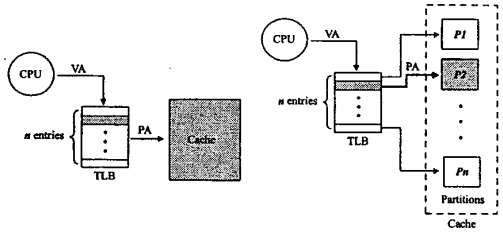
0-7803-7690-0/02/$17.00 ©2002 IEEE

address issued to the cache consists of three parts: tag, index, and offset. Suppose that the cache is organized as a collection of $S=C/(B\times A)$ sets, where A, B, and C represent associativity, block size and cache size, respectively. The index part has length of $log_2(S)$ bits, which is used to index the set that potentially contains the desired data. The offset part has length of $log_2(B)$ bits, and is used to select the appropriate byte within a block. Finally, if the address space is 32-bit, the tag part has length of $32-log_2(S)-log_2(B)$ bits, which is used to check whether the access is hit or miss.

The cache consists of the tag array and the data array. Each cached block in the data array has only one address tag corresponding to it in the tag area. It is clear that the tag array is the storage overhead needed to determine whether the corresponding cached data is what we want. For example, in a 32KB 2-way cache with block size of 32 bytes, the area proportion of tag array to data array is about *1:15*. If a 64-bit wide address space were used, the tag length would be *50* bits and then the area ratio of tag array to data array would be *1:5*. From this example, it is clear that the tag area takes a notable fraction of space as compared to data area. It will take more area as the memory address space increases.

2.2 Paged Cache

Figure 1(a) shows a possible access flow of the conventional cache. Normally, the virtual address *(VA)* generated by the CPU is fed into the TLB to generate the physical address *(PA)*, after that a cache lookup is done. It is very important that we only consider a physically addressed cache in this study. That is, the cache cannot be accessed until the generation of physical address. Note that there is no relationship between the TLB access and the cache access in the conventional cache architecture, and the result of cache access is independent of the TLB access result.



(a) Conventioanl cache. **(b) Paged cache.**

Figure 1: Cache architecture. (The gray blocks symbolize active component)

We now propose a new cache architecture, called *paged cache*, for parallel address translation and area cost reduction. In the paged cache shown in Figure 1(b), we divide the entire cache into several pieces, named *partitions*, and limit the partition size to be not larger than the page size. Note that the number of partitions must be the same as the TLB entry number. In contrast to the conventional cache, each partition is dedicated to only one page that is cached in TLB. There is one-to-one correspondence between partitions and TLB entries. Consequently, we connect the cache access with the TLB access, that is, the cache hit implies the TLB hit (or the TLB miss implies cache miss). By restricting the range in which the cached block can be placed, we can map one page into the specific partition. The tag length of the paged cache is determined by the partition size and given by:

$$tag\ length = log\left(\frac{page\ size}{partition\ size}\right).$$

Because the page size is fixed at 4KB throughout this paper, the reasonable partition sizes used in this paper are 1KB, 2KB or 4KB, and thus the tag length are 2-bit, 1-bit or 0-bit, respectively.

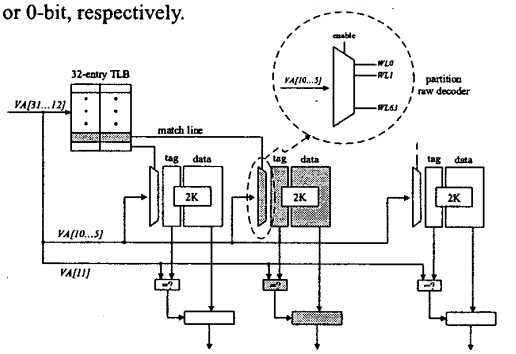


Figure 2: A paged cache example with a 32-entry fully-associative TLB and a 64KB cache. (The gray blocks symbolize active partition.)

Consider a 64KB paged cache with a 32-entry fully-associative TLB as shown in Figure 2. Throughout this paper, we fix the address space to be 32-bit wide, the page size 4KB and the block size 32B. Because the number of partitions must equal the TLB entry number, the size of partition is 2KB. For mapping one page to one partition, we have to use 1-bit as tag to determine whether the access is hit or not. The address format is shown in Figure 3(b). Compared to the conventional one shown in Figure 3(a), which is a one-way 64KB cache architecture, we can observe the tag length is reduced from 16-bit to 1-bit. Hence, the tag area can be largely reduced. The access flow in the paged cache architecture is described as follows:

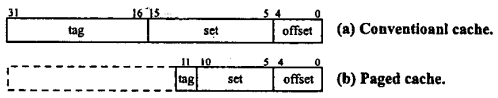


Figure 3: Address format.

1. The virtual address generated by the CPU is concurrently fed into the TLB and the decoder of each partition. The bus widths of input for TLB and partition decoder are 20-bit (i.e., *VA[31...12]*) and 6-bit (i.e., *VA[10...5]*), respectively.

2(a). In case of TLB hit, the match lineis used to enable the corresponding partition decoder to decode the index part of the reference address.

2(b). In case of TLB miss, it implies this reference would miss in cache access. We must reload the demand page from the page table. It is important to note that if this reloading induces the TLB replacement, the partition corresponding to the replaced page must be flush. By resetting the valid bit of all cache blocks, the partition can be easily flushed.

3. To determine a cache hit, we must concurrently check the valid bit and compare the tag with that of selected block. If the valid bit of selected block is 0, it means this block is not available.

4(a). In case of a cache hit, the desired data is returned to the processor.

4(b). In case of a cache miss, the physical address generated by the TLB can be used to load the desired block from the lower level of memory hierarchy, and then the corresponding valid bit and tag must be updated.

3. EVALUATION FOR AREA AND POWER

3.1 Area Cost Analysis

An important cost measure for the on-chip cache is its occupied chip area. In this paper, we use the simplest area cost model, the number of tag cells, for both the conventional cache and the paged cache. Because the tag array of paged cache and that of conventional cache have the same structure and implementation, the reduction in terms of the number of tag cells in the paged cache architecture can directly reflect the actual savings in chip area. Note that the sizes of data array for both architectures are identical.

Table 1: Tag length for various cache configurations.

	1-way	2-way	4-way	P=1K	P=2K	P=4K
16K	18	19	20	2	1	0
32K	17	18	19	2	1	0
64K	16	17	18	2	1	0
128K	15	16	17	2	1	0

Based on the cache configuration, the lengths of tag for both conventional cache and paged cache are summarized in Table 1. We observed that the tag length of the conventional cache is dependent on the cache size and associativity. By contrast, the tag length of the paged cache is only dependent on the partition size. For example, suppose that the partition size is 1K (P=1K). Because the page size is 4K, there are four possible blocks that can be mapped to the same set. Consequently, we use two bits as tag to check whether the selected set contains the required data. Because we do not take into account the valid bit, the tag length is 0 for partition size of 4K. In Figure 4, we depict the number of tag cells in the caches for the various configurations of the paged cache architecture.

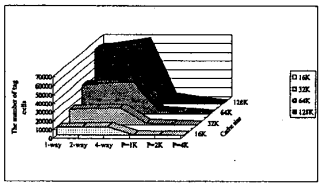


Figure 4: The number of tag cells for various cache configurations.

3.2 Power Consumption Per Access

Based on the cache access model described in [11], we perform the *HSPICE* simulation and summarize the power consumption per access for various cache configurations in Table 2. The key observation is that with the use of the paged cache architecture, a large reduction in the power consumption per access can be easily achieved. In the conventional cache, the power consumption is proportional to the cache size and the degree of associativity. By contrast, the power consumption per access of the paged cache architecture is independent of the cache size, but only dependent on the partition size.

Table 2: Power consumption per access for various cache configurations.

Power (mW)	1-way	2-way	4-way	P=1K	P=2K	P=4K
16K	64.55	106.15	193.53	42.13	42.92	44.88
32K	85.50	129.11	212.30	42.13	42.92	44.88
64K	136.49	171.01	258.22	42.13	42.92	44.88
128K	254.25	272.98	342.01	42.13	42.92	44.88

3.3 Estimation of Access Time Spent in the Bitlines

Up to this point, we have investigated the impact of both area cost and power consumption of the proposed architecture. Another important factor is the cache access time. Because the size of partition in the proposed architecture is very small compared to the conventional cache and only one partition would be active in each cache access, we can infer the access time of the paged cache is shorter than that of the conventional cache. To measure the access time of the two architectures more accurately, we performed timing simulations of the two implementations. In this simulation, we compare the smallest cache studied in this paper (i.e., 16K direct-mapped cache, the best case in the conventional cache) and the paged cache with 4K partition size (the worst case in our page cache) to show the gain of access time. Note that here we do not consider the access time spent in other components, e.g., decoder, comparator, multiplexer, output driver, etc. As illustrated in Fig. 5, the access time spent in the bitlines of the paged

cache architecture is about 0.2ns faster than that of the conventional cache.

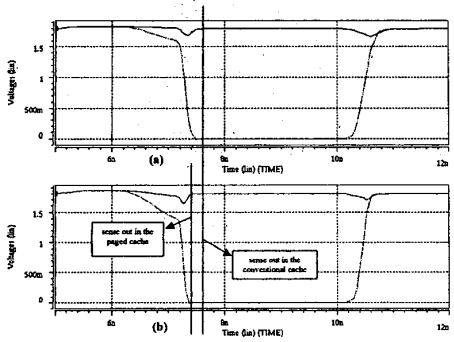


Figure 5. Bitlines signal waveforms in (a) the conventional 16KB direct-mapped cache, and (b) the paged cache with 4KB partition size.

4. EXPERIMENTAL RESULTS

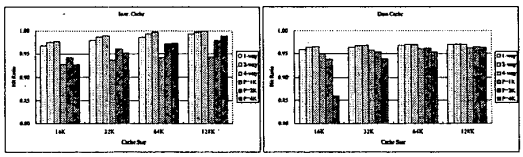
In this paper, we use *SimpleScalar* [12] in conjunction with the *HSPICE* simulations (with a 0.18 μm technology and 1.8V voltage supply) to evaluate the proposed architecture. The numbers of cache accesses were obtained from the simulation of a set of *SPEC2000* benchmarks. To get a good mix of CPU-intensive and memory-intensive loads, we randomly chose eight CINT2000 benchmarks (164.gzip, 175.vpr, 176.gcc, 181.mcf, 197.parser, 253.perlbmk, 255.vortex, 256.bzip2) and four CFP2000 benchmarks (177.mesa, 179.art, 183.equake, 188.ammp).

4.1 Configurations Studied

For the results presented here, we use a two level on-chip cache hierarchy, with split L1 caches and unified L2 cache. To avoid an explosion in the number of results, the page size is fixed as 4KB, so the partition size can be 1KB, 2KB or 4KB. For our base case, we assume a 64KB, direct-mapped L1 I-cache and a 64KB 4-way L1 D-cache. The block sizes for both caches are set to 32B. The L2 cache is assumed to be a 256KB 4-way unified on-chip cache with a block size of 64B.

4.2 Results and Discussions

Figure 6 shows the impact of using the paged cache architecture on the hit ratio. In the conventional architecture, the hit ratio increases with the degree of associativity. The decrease in hit ratio with the use of the paged cache architecture is as expected; this is because the paged caches hold only blocks belonging to the pages that are contained in the TLB. An interesting variation in hit ratio is observed between the I-caches and the D-caches. For the I-caches, which has superior spatial locality, increasing the partition size to 4KB for the smaller cache (i.e., 16K and 32K in Figure 6(a)) does not result in hit ratio improvement as in the case of larger cache (i.e., 64K or 128K). This is because the number of partitions is too small to cache sufficient pages. By contrast, for the D-caches which has poor locality, decreasing the partition size to 1KB (i.e., enlarging the number of partitions) would be beneficial for all cases.



(a) Instruction cache. **(b) Data cache.**

Figure 6: Hit ratio for various conventional caches and paged caches.

In the following discussions, we use area cost, power consumption and access time as the evaluation criteria to compare the base case implemented in conventional cache architecture with that implemented in paged cache architecture.

Area Cost: In Figure 7, we depict how hit ratio in our base case is reduced by using the paged cache architecture with various partition sizes. From these results, we decide to use the page cache with 4KB partition size for I-cache, because the hit ratio decrease is the smallest (i.e., from 0.98 to 0.97). Similar to the D-cache, the partition size that we use in the proposed architecture is 1KB.

	Base	P=1K	P=2K	P=4K
I-cache	0.9862	0.9417	0.9725	0.9734
D-cache	0.9697	0.9614	0.9616	0.9545

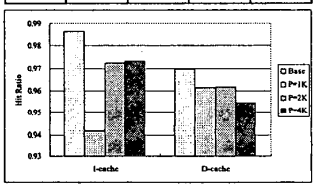


Figure 7: Hit ratio for base case and paged cache architecture.

For simplicity, we use the number of tag cells to estimate the area of tag array in the paged cache architecture. The tag length is *16*-bit for the base case of L1 I-cache, and it is *18*-bit for the base case of L1 D-cache. Because the partition size is 4KB, the tag length is *0*-bit for the paged L1 I-cache. In the paged L1 D-cache, the tag length is *2*-bit. Note that we do not take into account the status bits, such as valid bit, dirty bit, etc. For a 64KB cache, there are $2^{11}=2048$ blocks in the data array. As we know, each cache block in the data area has only one corresponding tag. For the L1 I-cache, the number of tag cells for the conventional cache and the paged cache are $2^{11}\times 16$ and *0*, respectively, i.e., the tag area savings is

100%. In the case of L1 D-cache, the number of tag cells was reduced from $2^{11}\times18$ to $2^{11}\times2$, representing the tag area savings of about *88%*.

Power Consumption: According to the memory hierarchical model, the average power consumption per access for the conventional cache and our proposed architecture can be expressed by the following equations:

$$P_{Conv\_ave} = P_{L1\_Conv} + (1\text{-}HR_{L1\_Conv})\times P_{L2}, \quad (1)$$
$$P_{PC\_ave} = P_{L1\_PC} + (1\text{-}HR_{L1\_PC})\times P_{L2}, \quad (2)$$

where P_{L1_Conv} and P_{L1_PC} are the power consumption per access of the *L1* conventional cache and *L1* paged cache, respectively, , HR_{L1} is the hit ratio of the L1 cache, and P_{L2} is the power consumption of the *L2* on-chip cache. Based on the analysis model of power consumption described in Section 3, we obtain the values of P_{L1_Conv}, P_{L1_PC} and P_{L2} for both I-cache and D-cache, as shown in Table 3(a).

Combine Equations (1), (2) and the results illustrated in Table 3(a), the average power consumption per cache access measured in *mWatts* for both conventional cache and paged cache are shown in Table 3(b). Clearly, for I-cache, if the paged cache with 4KB partition size was employed, the average power consumption per access can be reduced from *144.03mW* to *59.39mW*, representing a power savings of about *58%*. Similarly, in the D-cache, if the paged cache with 1KB partition size was used, the average power consumption per access can be reduced from *274.76mW* to *63.22mW*, representing a power savings of about *77%*.

Table 3: Power consumption per access (in *mW*).

	P_{L1_Conv}	P_{L1_PC}	P_{L2}
I-cache	136.49	44.88	545.96
D-cache	258.22	42.13	

(a) Power consumption per access for L1 and L2 on-chip cache.

	P_{Conv_ave}	P_{PC_ave}
I-cache	144.03	59.39
D-cache	274.77	63.22

(b) Average power consumption per access for both conventional cache and paged cache.

Access Time: A simple rule of hardware design: *Smaller is faster*. This simple principle is particularly applicable to memories built from the same technology. Because the partition in our proposed architecture is smaller than the conventional L1 cache, it should have a faster access time. We use the tool *Cacti*, described in [11], to estimate the access time of the conventional on-chip caches, as well as the paged cache that was proposed to reduce mainly the area cost and power consumption. Note that the cache models used in this tool is not precisely equal to the paged cache. Actually, the access time of the partition is slightly faster than the results shown here. This is because the tag length of our proposed architecture is far shorter than that of the conventional cache. In our experiments, the partition in the paged cache is implemented as a direct-mapped organization, with size ranging from 1KB to 4KB, and block size is 32 bytes. Figure 8 shows the access time of base case with various partition sizes, based upon the cache model described in [11] using 0.18 μm technology.

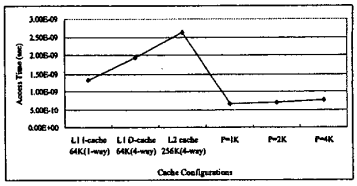


Figure 8: The access time of base case with various partition

Similar to the average power consumption per access, the average access time for the conventional cache and our proposed architecture can be expressed by the following equations:

$$T_{Conv\_ave} = T_{L1\_Conv} + (1\text{-}HR_{L1\_Conv})\times T_{L2}, \quad (3)$$
$$T_{PC\_ave} = T_{L1\_PC} + (1\text{-}HR_{L1\_PC})\times T_{L2}, \quad (4)$$

where T_{Conv_ave} and T_{PC_ave} are the access time of the L1 conventional cache and L1 paged cache, respectively, and T_{L2} is the access time of the L2 on-chip cache. Combine Equations (3), (4) and the results illustrated in Table 4(a), the average cache access time measured in nanosecond for both conventional cache and paged cache are shown in Table 4(b). The key observation is approximately a *40%* reduction in access time when using the paged L1 I-cache with 4KB partition size, and *61%* for the paged L1 D-cache with 1KB partition size can be obtained. Consequently, our proposed cache architecture can improve the cache access time, and this improvement in L1 cache access time may presents an opportunity to increase the fundamental processor clock.

Table 4: Access time (in *ns*).

	$Time_{L1\_Conv}$	$Time_{L1\_PC}$	$Time_{L2}$
I-cache	1.3410	0.7649	2.6383
D-cache	1.9237	0.6749	

(a) Access time for L1 and L2 on-chip cache.

	$Time_{Conv\_ave}$	$Time_{PC\_ave}$
I-cache	1.3774	0.8350
D-cache	2.0037	0.7768

(b) Average access time for both conventional cache and paged cache.

Summarize the performance comparisons in power consumption, area cost and access time, the improvements due to the use of paged cache are shown in Table 5. One can see that the paged cache is very effective in reducing both power consumption and area cost for the on-chip L1 caches, while the average access time of cache is also improved.

Tabel 5: Summary of performance improvement.

	power consumption	area cost	access time
L1 I-cache	58%	100%	40%
L1 D-cache	77%	88%	61%

5. CONCLUSIONS

On-chip cache is a major source of both area consuming and power dissipation in most modern processors. In this paper, a new cache architecture, called *paged cache*, was proposed, in which we use a simple partition scheme to divide a large cache into several partitions. The experimental results showed that it is very effective in reducing both power consumption and area cost in on-chip L1 caches, while the average access time of cache is also improved. For the base case of L1 on-chip I-cache (64KB, 1-way), the paged cache with partition size of 4KB can result in roughly *58%* reduction in power consumption, *100%* reduction in tag area and *40%* reduction in average access time. Similarly, for the base case of L1 on-chip D-cache (64KB, 4-way), the paged cache with partition size of 1KB can result in roughly *77%* reduction in power consumption, *88%* reduction in tag area and *61%* reduction in average access time.

REFERENCES

[1] J. F. Edmondson et al., "Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor," Digital Technical Journal, Vol. 7, No. 1, 1995, pp. 119-135.

[2] J. Montanaro et al., "A 160 MHz, 32b 0.5W CMOS RISC Microprocessor," in IEEE ISSCC 1996 Digest of Papers, 1996.

[3] M. Farrens, G. Tyson and A. R. Pleszkun, "A Study of Single-Chip Processor / Cache Organizations for Large Number of Transistors," in Proc. of 21st International Symposium on Computer Architecture, Apr. 1994, pp. 338-347.

[4] K. Ghose and M. B. Kamble, "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation," in Proc. of International Symposium on Low Power Electronics and Design, 1999, pp. 70-75

[5] J. Kin, M. Gupta and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in Proc. of 30th International Symposium on Microarchitecture, Dec. 1997, pp. 184-193.

[6] K. Inoue, T. Ishihara and K. Murakami, "Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," in Proc. of International Symposium on Low Power Electronics and Design, 1999, pp. 273-275

[7] J. L. Cruz, A. Gonzalez and M.Valero, "Multiple-banked Register File Architecture," in Proc. of International Symposium on Computer Architecture, June 2000, pp. 316-325.

[8] A. Seznec, "Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost and Low Miss Ratio," in Proc. of 21st International Symposium on Computer Architecture, Apr. 1994, pp. 384-393.

[9] H. Wang, T. Sun and Q. Yang, "Minimizing Area Cost of On-Chip Cache Memories by Caching Address Tags," IEEE Transaction on computer, Vol. 46, No. 11, Nov. 1997, pp. 1187-1201.

[10] Y. H. Lee, W. Y. Jeong, S. J. Ahn and Y. S. Lee, "Shared Tag for MMU and Cache Memory," Semiconductor Conference, CAS '97 Proceedings, Vol.1, 1997, pp. 77 –80.

[11] S. E. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," DEC WRL Research Report 93/5, July 1994.

[12] D.C. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Computer Architecture News, 25 (3), pp. 13-25, June, 1997. Extended version appears as UW Computer Sciences Technical Report #1342, June 1997.