A Unified Architectural Tradeoff Methodology *

Chung-Ho Chen

Department of Electronic Engineering
National Yunlin Institute of Technology
Touliu, Yunlin
R.O.C. on Taiwan

Arun K. Somani

Dept. of EE and Dept. of CSE
University of Washington
Seattle, WA 98195
U.S.A.

Abstract

We present a unified approach to assess the tradeoff of architecture techniques that affect mean memory access time. The architectural features we consider include cache hit ratio, processor stalling features, line size, memory cycle time, the external data bus width of a processor, pipelined memory system, and read bypassing write buffers. We demonstrate how each of these features can be traded off to achieve the desired performance. The performance of an architecture feature is quantified in terms of cache hit ratio based on the equivalence of mean memory delay time. This paper investigates the implication of architectural tradeoffs on the pin count, memory system design, and on-chip cache area for microprocessor systems.

1 Introduction

Architecture techniques that affect mean memory access time are the major factors that determine the performance of a computing system. Using a larger cache memory is usually suggested to improve the memory performance [1]. Increasing the width of a processor external data bus improves the performance but increases as the pin count of a microprocessor. An on-chip cache memory requires a considerable amount of area of the microprocessor chip. Read-bypassing write buffers increase the complexity of the processor design and so does a pipelined memory system. Each of these hardware features has a different performance/cost tradeoff. This paper presents a unified tradeoff methodology to compare the achievable performance of each of these architectural features. We use cache hit ratio, which is dependent on the cache architecture and its size, as a common measurement unit. We evaluate the improvement in performance from changing an architectural feature. We then identify the change required in the hit ratio to achieve the same performance improvement. This is accomplished by deriving expressions for mean memory access delay time for the two cases and establishing an equivalence between the two.

We show that the reduction in performance due to reducing a hit ratio (hence, a cache size) can be compensated by improving or by providing other architectural features. For instance, our analysis shows that the performance loss due to reducing the hit ratio (i.e., reducing the cache size) of a blocking cache from HR to $2HR-1$ to at most $2.5HR-1.5$ can be compensated by doubling the data bus width. For example, the performance loss due to reducing cache hit ratio from 0.95 to 0.9 ($= 2\times 0.95-1$) or from 0.98 to 0.96 can be compensated by doubling the external data bus of a processor. Equivalently, increasing the hit ratio HR of a blocking cache by $0.5(1-HR)$ to $0.6(1-HR)$ improves the performance by an amount that is obtainable by doubling the data bus width. When we refer to a processor data bus, we mean the external data bus of a processor. Other types of stalling features in cache memories trade differently with the data bus width.

Our model allows us to study the impact of other architectural features. For instance, we obtained exactly the same results as in Smith's paper for determining the optimal lines [2]. The application of our research extends beyond determining the best line size for cache memories. We can now rank the performance of each of the features. Except for the pipelined memory system, doubling the data bus width is the best choice. Use of read-bypassing write buffers is the second best choice. The use of a cache which allows cache access when a load miss is in progress is the third best choice. For pipelined memories, we also determine the cross-over point for memory cycle time where the use of a pipelined memory system is most advantageous. Doubling the data bus width or using the read-bypassing write buffers has a limited performance contribution in systems that have a relatively long memory cycle times. The memory cycle time, when the performance of a pipelined system surpasses that of doubling the data bus width, is not large at all, especially when a large cache line is used. A pipelined memory system impacts the hit ratio (or correspondingly cache size) considerably, and therefore, should be seriously considered in the design of microprocessor systems.

The rest of this paper is organized as follows. We discuss the related research in Section 2. In Section 3, we give the notation and the assumptions of the hardware under study. We develop the tradeoff model in Section 4. The tradeoff analysis is presented in Section 5. We summarize our findings in Section 6.

*This research in part was supported by the NSF grant MIP-9224462. The authors thank the referees, Philip Leung, Ted Hamlin, and Ing MacDuff for their valuable comments.

2 Related Research

The design of a cache memory involves many issues such as the choice of line fetch algorithm, the replacement policy, the write handling protocol, split vs. unified cache, coherency, virtual vs. real address tag, and the blocking characteristics on a miss [3]. Cache line size is one of the critical parameters that affect cache performance. Smith and Przybylski used the cache miss ratio obtained from trace-driven simulations to study the factors for choosing a cache line size [2, 4, 5]. Their criterion in selecting the best line size is to minimize *the mean memory delay per memory reference* or *the mean read time*. Alpert and Flynn pointed out that using a larger line size reduces the overhead of storing address tags and other cache control information and thus leads to more cost-effective cache design [6].

Chen and Baer examined the effectiveness of using non-blocking caches, prefetching caches, and read bypassing write buffers in reducing memory latency [9]. They found that prefetching caches often have a higher performance than non-blocking caches, and that read bypassing write buffers can reduce the write miss penalty significantly. Tullsen and Eggers found that software-controlled prefetching offer limited benefit in a bus-based multiprocessor system [10]. Since the design space for cache memories is so diverse, it is very natural for a computer designer to focus on only a limited number of parameters to optimize a particular cache implementation [11]. From the system design point of view, optimizing the design space around hit ratio or memory traffic may not produce a cost-effective system. In addition to cache memories, architectural features such as data bus width, pipelined memories, and how they are used within a range of memory cycle times are also critical in determining performance.

3 Model of CPU Execution Time

In this section, we establish the relationships between the architectural features of hardware and the characteristics of programs. The CPU execution time is derived by considering various techniques that affect memory latency.

3.1 Parameters and Assumptions

The notation for the parameters is described in Table 1. These parameters specify the characteristics of an architectural feature, application, and the relationship between the two. An application here can be a task, a subroutine, or a phase of computation. The following assumptions are used for our study:

1. We use a RISC processor model with on-chip instruction and write-back data cache. The processor has a separate external address bus and data bus.

2. For a cache miss, a whole line is brought into the cache from the memory.

3. All memory references are first directed to the on-chip caches.

Table 1: Architectural Parameters

D	processor's external data bus width in bytes. D can be any number $\in \{4, 8, 16, 32\}$.
L	cache line size in bytes.
β_m	memory cycle time for a D-byte read/write cycle.
E	number of instructions executed for an application.
R	number of data bytes read in full bus width by a processor upon read miss for E instructions executed. R does not include the portion for instruction fetch.
R_I	number of instruction bytes read in full bus width by a processor for E instructions executed.
W	number of write-around miss instructions that use the external data bus of a processor for E instructions executed. In particular, $W = W_{1b} + \frac{W_{2b}}{2} + \cdots + \cdots + \frac{W_{Db}}{D} + \cdots + \frac{W_{jD}}{jD}$ $= \sum_{i=1}^{D} \frac{W_{ib}}{i} + \sum_{j=1}^{\omega/D} \frac{W_{jD}}{jD}$, $i = 1, 2, 4, \cdots D$, $j = 1, 2, \cdots \omega/D$. ω is the maximum write operand size which exceeds the bus width D. W_{ib} (W_{jD}) is the number of data bytes written using i (D) bytes of the processor data bus respectively.
α	cache line flush ratio for E instructions executed, $0 \le \alpha \le 1$. The number of bytes of cache dirty lines which are copied-back (flushed) for E instructions executed is represented by αR.
ϕ	stalling factor.

4. A load/store instruction effectively takes one cycle when it hits in the cache. Because instructions are pipelined, the throughput is effectively one instruction per cycle.

5. The memory system has the same memory cycle time for read and write requests.

In a cache-based system, the number of bytes of read R (R_I) and write around miss W can be related to the hit ratio of a cache. An application in a uniprocessor system with specific architectural features is characterized by $\{E, R_I, R, W, \alpha, \phi\}$. For E instructions executed, let λ_m be the number of load/store instructions that cause cache misses, and λ_h be the number of load/store instructions that hit in the data cache. Then, λ_m is related to R, W and line size L of the on-chip data cache by the following relation:

$$\lambda_m = \frac{R}{L} + W. \tag{1}$$

Since every read miss results in loading a full line into the cache, R is divided by L to represent the number of load instructions that miss in the data cache using the write-around mode for handling write misses. For a cache with write-allocate mode R/L becomes the sum of the load and store instructions that miss in the cache because a write miss reads in a cache line. For this case W equal to zero. For both of the write miss handling modes, λ_m always represents the number of load/store instructions that cause cache misses.

3.2 Stalling Features

The execution of a processor may be delayed by various stalling cycles due to a cache miss. Table 2 lists the processor stalling features that are considered in this study. For a full blocking cache, a processor waits for the requested data until the entire cache line is brought into the cache. This is full-stalling (FS). In case of full-stalling, $(L/D)\beta_m$ time contributes to the execution time for each cache miss. The stalling factor is L/D as indicated in Table 2.

With a cache-bus-locked feature, for a miss cycle, the cache first requests the missed data from the memory. As soon as the data arrive, the processor continues execution. While the cache fetches the rest of the line, the cache is locked up. If any load/store occurs during this period, that load/store is stalled until the line is completely fetched. We call this scenario as bus-locked (BL) stalling feature. The minimum value of ϕ due to BL stalling is one.

To reduce the stalling delay, an alternative is to allow the processor to access other cache lines without locking up the cache bus. This stalling feature is bus-not-locked (BNL). In BNL case, the processor may be stalled when the current line being fetched is accessed a second time. The period of this stall varies depending on the implementation. We consider three possible scenarios for BNL and assume that a line is filled using multiple bus cycles. In scenario BNL_1, the processor is stalled by the second access until the line is completely fetched. In scenario BNL_2, the processor is stalled only if the second access happens to be on the part of the line which has not yet been fetched. If stalling occurs, the processor is delayed until the entire line is fetched completely. In scenario BNL_3, a stall occurs only if the data have not been fetched. Otherwise, an access can be satisfied by a partially filled line. The minimum ϕ for the BNL feature for eah scenario is indicated in Table 2.

A cache may be designed with the non-blocking (NB) feature. The execution of a processor may also be stalled by a non-blocking cache as well [9]. The minimum possible stalling factor for this NB feature is zero. The NB, or the BL, or the BNL, is partially-stalling (PS) in contrast to the full-stalling feature.

3.3 Execution Time

The CPU execution time consists of two factors. First, the time includes the instruction fetching cycles from the memory due to instruction cache misses. Instruction caches with a full blocking feature can be found in most of the current processors. However, since most of the instructions are resident in the instruction cache and the execution is pipelined, the time consumed for the instruction fetching is relatively small, especially for processors that have two separate buses to access instruction and data caches respectively [13]. The second element which dominates the execution time is the time to execute the instructions. We use X to represent this part of the CPU execution time. For a RISC processor, the execution time X is represented by the following expressions assuming that all non load/store instructions as well as load/store instructions with a hit in cache memory take one clock cycle.

Table 2: Processor Stalling Feature

	features	stalling factor
FS	full stalling	$\phi = \frac{L}{D}$
BL	bus-locked	$1 \leq \phi \leq \frac{L}{D}$
BNL	bus-not-locked	$1 \leq \phi \leq \frac{L}{D}$
NB	non-blocking	$0 \leq \phi \leq \frac{L}{D}$

$$\begin{aligned} X &= (E-\lambda_m) + \frac{R}{L}(\phi\beta_m) + \frac{\alpha R}{L}(\frac{L}{D})\beta_m + W\beta_m. \\ &= (E-\lambda_m) + \frac{R}{L}(\phi\beta_m) + \frac{\alpha R}{D}\beta_m + W\beta_m. \quad (2) \end{aligned}$$

The $(E-\lambda_m)$ term accounts for the time the program spent in executing the non-load/store instructions and the load/store instructions that are hits in the data cache. Load instructions that miss in caches stall the execution of the processor by $\frac{R}{L}(\phi\beta_m)$ cycles. For instance, when loading R bytes of data, a full-blocking cache stalls the execution of the CPU by $\frac{R}{D}\beta_m$ cycles with $\phi = \frac{L}{D}$. When no write buffers are provided, the flushes stall the CPU by $(\frac{\alpha R}{D})\beta_m$ cycles. The last term $W\beta_m$ represents the cycle times due to write-around misses. We assume that the size of data being written is smaller or equals to bus width D. When a cache uses write-allocate to handle write misses, the missing data are first read into the cache before the write is performed. In this case, R includes the data read for the write misses, and W is zero.

As technology advances, techniques such as cache line prefetching, or register preloading can be used to hide or reduce the penalty of some read misses [8]. In these cases, R will represent the memory references whose miss penalty cannot be hidden. Alternatively, β_m can be scaled down to represent the average miss penalty.

3.4 Effect of Instruction Cache Misses

The instruction cache misses contributing to the CPU execution time can be represented by $\frac{R_I}{L}\phi\beta_m$ where ϕ has a minimum of one. Due to the instruction pipelining, the instruction hit cycles are overlapped with the execution time X in Eq. (2). From trace-driven simulations, we and other researchers [15] have observed that without process switching, instruction cache hit ratio is usually very high. Then, the CPU execution time will be dominated by the X. In a multiprogramming case, a higher instruction miss ratio is expected. In this case, the miss portion cannot be neglected in the CPU execution time, and $\frac{R_I}{L}\phi\beta_m$ should be added to Eq. (2). In either case, the CPU execution time is always represented in the same form as in Eq. (2).

4 Derivation of Architectural Trade-off

When comparing the architectural tradeoffs for two systems, we improve one system by including adding an architectural feature which has a higher performance. The result is that the mean memory delay time is reduced by the use of that feature. This performance enhancement can be achieved, for instance, by doubling the data bus width, by changing from a full stalling cache to a partially stalling one, by providing the read-bypassing write buffers, or by using a pipelined memory system. To maintain the same performance on a given application, the original system without that feature must have a higher hit ratio (implying larger cache) than the new system. In other words, for the same performance, the original system must use a larger cache than the new system which uses a more powerful architectural feature. Hence, there is a relationship between the difference in the hit ratio for the two systems and of the performance-improving feature used.

4.1 Data Bus Width versus Hit Ratio

We now consider the tradeoff between cache hit ratio and data bus width. The following two expressions denote the execution time of using data bus widths D and $2D$, respectively.

$$X(D) = E - (\frac{R}{L} + W) + \frac{R}{L}(\phi\beta_m) + \frac{\alpha R}{D}\beta_m + W\beta_m$$

and

$$X(2D) = E - (\frac{R'}{L} + W') + \frac{R'}{L}(\phi'\beta_m) + \frac{\alpha' R'}{2D}\beta_m + W'\beta_m.$$

Here $\{R', W', \alpha', \phi'\}$ and $\{R, W, \alpha, \phi\}$ are parameters for the system using data bus width $2D$ and D, respectively. The maximum value of ϕ' is $L/2D$ and $L \geq 2D$. To determine data bus width and cache hit ratio trading, we let $X(D) = X(2D)$ so that the two cases using data bus width D or $2D$ have the same execution time. In the two systems, the number of instructions (E) executed, and the cache line size, the memory cycle time, and the stalling feature are all the same. For a write-around cache, $W = W'$. If the cache uses write allocation mode, then $W = W' = 0$ since the data read due to write misses are included in R or R', respectively. Solving $X(D) = X(2D)$, yields

$$R' = \left(\frac{(\phi + (\frac{L}{D})\alpha)\beta_m - 1}{(\phi' + (\frac{L}{2D})\alpha')\beta_m - 1} \right) R. \tag{3}$$

Let $\lambda_h = s\lambda_m$. The miss ratio MR_1 of the data cache for the case of D width is given by

$$MR_1 = \frac{\lambda_m}{\lambda_m + \lambda_h} = \frac{1}{s+1}. \tag{4}$$

We use the hit (miss) ratio in the D width system as a base. To achieve the same performance, the $2D$ width system could afford a lower hit ratio than the hit ratio in the D width system. Equivalently, the $2D$ width system can use a smaller cache to have the same performance. Since the program characteristics remain unchanged, therefore $\lambda_h + \lambda_m = \lambda'_h + \lambda'_m$. Only some load/store instructions that hit in the cache of the D width system become misses in the cache of the $2D$ width system so that they have the same performance.

Let MR_2 be the miss ratio for the $2D$ width system and $\lambda'_m = r\lambda_m$ where $\lambda'_m = \frac{R'}{L} + W'$ then

$$MR_2 = \frac{\lambda'_m}{\lambda'_h + \lambda'_m} = \frac{r\lambda_m}{\lambda_h + \lambda_m} = \frac{r}{s+1}. \tag{5}$$

Let HR_2 be the hit ratio for the $2D$ width system, and HR_1 be the hit ratio for the D width system. Notice that between the two systems, the difference of hit ratios equals to the difference of miss ratios. Then cache hit ratio that trades a D width is

$$\begin{aligned} HR_1 - HR_2 &= MR_2 - MR_1 \\ &= \Delta HR \\ &= \Delta MR \\ &= \frac{r-1}{s+1} \end{aligned} \tag{6}$$

where $s = \frac{HR_1}{1-HR_1}$ (from $\lambda_h = s\lambda_m$) and $r = \frac{\frac{R'}{L}+W'}{\frac{R}{L}+W}$ (from $\lambda'_m = r\lambda_m$). Eq. (6) is only valid for the physical system where $HR_2 \geq 0$.

Two limits in Eq. (6) are of interests. First, with $L = 2D$, $\beta_m = 2$, and $\alpha = \alpha' = 0.5$, we find $R' = 2.5R$. Using Eq. (6), we obtain $HR_2 = 2.5HR_1 - 1.5$. Secondly, with $\alpha = \alpha' = 0.5$, we apply the L'Hospital's rule in Eq. (3) for a relatively large β_m. We find $R' = 2R$ and $HR_2 = 2HR_1 - 1$. This result states that the performance loss due to reducing the hit ratio of a blocking cache from HR to a value in the range from $2HR - 1$ to $2.5HR - 1.5$ can be compensated by doubling the data bus width.

We also can use the hit (miss) ratio in the $2D$ width system as a base. The D width system must have a higher hit ratio than that of the $2D$ width system for both of them to have the same performance. The hit ratio difference is given by

$$\Delta HR = \frac{1-r}{s+1}. \tag{7}$$

where $s = \frac{HR_2}{1-HR_2}$ and $r = \frac{\frac{R}{L}+W}{\frac{R'}{L}+W'}$.

Similarly, we find $R = 0.4R'$ and $\Delta HR = 0.6(1 - HR_2)$ with $L = 2D$, $\beta_m = 2$, and $\alpha = \alpha' = 0.5$. With $\alpha = \alpha' = 0.5$ and a relatively large β_m, we find $R = 0.5R'$ and $\Delta HR = 0.5(1 - HR_2)$. This result states that given $L \geq 2D$ and $\alpha = \alpha' = 0.5$, the performance improvement obtained by increasing the cache hit ratio at HR by a value in the range from $0.5(1 - HR)$ to $0.6(1 - HR)$ is the same as that obtained by doubling the data bus width.

Table 3: Ratio of Cache Misses (r) and Stalling Factors (write allocate)

Metric	parameters
Doubling bus	$X = E - \frac{R'}{L} + \frac{R'}{L}(\phi'\beta_m) + \frac{\alpha' R'}{2D}\beta_m$ $r = \frac{(\phi+(\frac{L}{D})\alpha)\beta_m - 1}{(\phi'+(\frac{L}{2D})\alpha')\beta_m - 1}$ $\phi = \frac{L}{D},\ \phi' = \frac{L}{2D}$
Partially stalling (BL, BNL)	$X = E - \frac{R'}{L} + \frac{R'}{L}\phi'\beta_m + \frac{\alpha' R'}{D}\beta_m$ $r = \frac{(1+\alpha)(\frac{L}{D}\beta_m) - 1}{(\phi'+(\frac{L}{D})\alpha')\beta_m - 1}$ $\phi = \frac{L}{D}$, ϕ': simulation
Write buffers	$X = E - \frac{R'}{L} + \frac{R'}{L}(\phi'\beta_m)$ $r = \frac{(\phi+(\frac{L}{D})\alpha)\beta_m - 1}{\phi'\beta_m - 1}$ $\phi = \frac{L}{D},\ \phi' = \frac{L}{D}$
Pipelined memory	$X = E - \frac{R_p}{L} + \frac{R_p(1+\alpha_p)}{L}\beta_p$ $r = \frac{(1+\alpha)(\frac{L}{D}\beta_m) - 1}{(1+\alpha_p)\beta_p - 1}$ $\phi = \frac{L}{D},\ \phi' = \frac{L}{D}$

4.2 Stalling Feature versus Hit Ratio

Similarly, we obtain how much hit ratio can be traded by switching from using a full-stalling cache to a partially-stalling cache to achieve the same performance. The execution time X and the ratio r for a PS stalling feature are shown in Table 3. The comparison is based on using a full-stalling cache with hit ratio HR_1. Let HR_2 be the hit ratio for not using the full-blocking feature, or equivalently for having the partially-stalling feature. For the same performance, HR_1 must be greater than HR_2. The cache hit ratio difference traded for using the partially-stalling feature is the same as in Eq. (6).

We obtain the stalling factor ϕ' from trace-driven simulations. We assume that each instruction is executed in one cycle except for the load/store misses and access stalling. This implies that an infinite instruction cache or an instruction cache with a very high hit ratio is used. The latter can be achieved relatively easily and is usually the assumption in cache related studies [10].

With the BNL_1 stalling feature, stalling occurs in two situations. First, if a load/store accesses the line which is being fetched, that load/store is stalled until the entire line is brought into the cache. Second, if another cache miss occurs while a previous load miss is in progress, then the new miss is stalled until the previous missed line is brought into the cache. Let ΔC_i be the difference of the instruction numbers of the two load/store misses. Then, the second load/store access is stalled by

$$\text{Max}\left\{ (\frac{L}{D} - 1)\beta_m - \Delta C_i,\ 0 \right\}$$

cycles. The stalling factor ϕ' for the BNL_1 model is computed as follows.

$$\phi' = \left(\sum_{i=1}^{\lambda_m} \left((\frac{L}{D} - 1)\beta_m - \Delta C_i \right) \right) (\frac{1}{\lambda_m \beta_m}) + 1 \quad (8)$$

where λ_m is the number of the load/store instructions that are misses (write-allocate cache is assumed). One is added for the basic read miss time in the representation of X for the BNL_1 stalling feature. The stalling factors for the BNL_2, BNL_3, and BL features can be obtained in a similar way and we will not discuss them here.

In Figure 1, we show the average stalling factors obtained from the SPEC92 programs nasa7, swm256, wave5, ora, doduc, and hydro2d. The stalling factor is illustrated in the percentages of L/D. A partially-stalling behaves like a full-stalling feature when its stalling factor reaches 100%. As expected, a longer memory latency has more stalling occurrences. We observe that the stalling factors are very high for the BL, BNL_1, and BNL_2 features. If subsequent load/store accesses are only stalled by the latency for the requested data to arrive, i.e., the BNL_3 feature, about 20-30% reduction in the read miss latency of a full blocking cache can be achieved for a memory cycle time smaller than 15 processor clock cycles.

4.3 Read-Bypassing Write Buffers versus Hit Ratio

With an appropriate memory cycle time, the read-bypassing write buffers can completely hide the latency of cache flushes. Execution time described in Table 3 represents the best possible performance of using the read-bypassing write buffers. Let HR_2 be the hit ratio with the read-bypassing write buffers, and HR_1 be the hit ratio with no read-bypassing write buffers. The cache hit ratio difference traded for the performance of using the read- bypassing write buffers is determined by Eq. (6) with $s = \frac{HR_1}{1-HR_1}$, and the ratio of cache misses (r) is shown in Table 3 for the write allocate cache.

4.4 Pipelined Memory System versus Hit Ratio

Instead of waiting for a complete memory cycle, the next memory cycle can be started as soon as possible through pipelining. Let q be the clock cycles for the memory system to be ready for receiving a new address request and beginning the next pipelined cycle. The pipelined memory cycle time β_p per L-byte request is given by

$$\beta_p = \beta_m + q(\frac{L}{D} - 1). \tag{9}$$

With the pipelined system, the execution of a processor pipeline is stalled by the duration β_p for a cache miss in a full blocking cache. The assumption for β_p is that the processor can issue the consecutive memory requests that it needs for an L-byte line, and the pipelined system can accommodate each request within q clock cycles. For instance, if $q = 2$, we may deem β_p as the best possible implementation of a pipelined system. The pipelined operation is initiated for the requests within the entire cache line. Because of the full blocking feature, if the data bus width and the line size are the same, then the pipelined stall cycles for the entire line is the same with the non-pipelined stall cycles as indicated in Eq. (9) by setting $L = D$. Within the scope of using full blocking write-allocate caches, we can relate the non-pipelined memory cycle time β_m with the pipelined memory cycle time β_p.

In trading the performance of a pipelined system with hit ratio, we say that HR_2 is the hit ratio for using the pipelined system, and HR_1 is the hit ratio for not having the pipelined system. The cache hit ratio difference traded for using the pipelined system is the same as in Eq. (6), and the ratio r is shown in Table 3 for the write allocate cache.

4.5 Equivalence of Mean Memory Delay Time

In this section, we show that the tradeoff model is based on the equivalence of the mean memory delay time and independent of the non-memory reference instructions such as some multiple cycle floating point operations. To explain this, we use the tradeoff between doubling the data bus width and the hit ratio of a full blocking cache as an example. Using $\lambda_h + \lambda_m = \lambda'_h + \lambda'_m$ (the total number of data memory references) and $X(D) = X(2D)$, we obtain

$$\frac{\frac{R(1+\alpha)}{D}\beta_m + (E - \frac{R}{L} - W - N_{LS})}{\lambda_h + \lambda_m} = \frac{\frac{R'(1+\alpha')}{2D}\beta_m + (E - \frac{R'}{L} - W' - N_{LS})}{\lambda'_h + \lambda'_m} \tag{10}$$

where N_{LS} is the number of the non-load/store instructions.

Since $\lambda_h = E - \frac{R}{L} - W - N_{LS}$ and $\lambda'_h = E - \frac{R'}{L} - W' - N_{LS}$, then

$$\frac{\frac{R(1+\alpha)}{D}\beta_m + \lambda_h}{\lambda_h + \lambda_m} = \frac{\frac{R'(1+\alpha')}{2D}\beta_m + \lambda'_h}{\lambda'_h + \lambda'_m}$$

Both sides of the above expression are the mean memory delay time per (data) memory reference. Since the mean memory delay time of an instruction cache, or a unified cache can also be represented in the same form as a data cache. Therefore, the tradeoff model can also be applied to an instruction cache or a unified cache.

5 Results of Architectural Tradeoffs

We examine the results obtained from the tradeoff model, first, the tradeoff between data bus width and hit ratio for full-blocking caches. Then, we compare all of the cases based on a full blocking and non-pipelined memory system.

5.1 Data Bus Width and Hit Ratio

Figure 2 illustrates the performance tradeoff between a 32-bit width and the hit ratio of a full-blocking cache with base hit ratio of 98% and 90%, respectively. We assume that the cache memory employs the write allocate mode for handling write misses so that $W = 0$. The base hit ratio is associated with the $X(D)$ where D is 32 bits. When the data bus and memory width are increased from 32 bits to 64 bits, the hit ratio in the 64-bit case must be smaller than the base hit ratio so that the performance is the same in both cases. The amount of the hit ratio traded is shown in the y axis. The design limit is reached when the memory cycle time β_m is two. We assume that the flush ratio α is 50% of R although the other value of α can also be used. In [2, 12], Smith also used 50% in describing the copy back traffic.

In the upper part of Figure 2, given L = 32 bytes and a relatively long memory cycle time, a 32-bit bus system using a cache with a hit ratio of 98% has the same performance as a 64-bit bus system using a cache with a hit ratio of about 96% (98 - 2). In other words, by increasing the hit ratio from 96% to 98% (that is by increasing the cache size so that the hit ratio increases by 2%), we can reduce the bus width (processor data bus and memory bus) from 64 bits to 32 bits while maintaining the same performance. When L = 8 bytes and $\beta_m = 2$, increasing the hit ratio by 3% (from 95% to 98%), we can trade a 64 bit bus with a 32-bit bus width. The lower part of Figure 2 depicts the cases where a different base hit ratio is used.

This illustration agrees with the previous limit analysis in Eq. (7). That is, given $L \geq 2D$ and $\alpha = \alpha' = 0.5$, increasing the hit ratio at HR% of a full-blocking cache by $0.5(100 - HR)$% to $0.6(100 - HR)$% has the same performance as that obtained by doubling the data bus width. As the memory cycle time increases, the traded hit ratio is reduced as indicated in the figure. This is because when the memory cycle time increases, hit ratio becomes *more precious* and thus bears more weight in affecting the mean memory delay time. With the same base hit ratio, the hit ratio traded for a large line size is smaller than that of a smaller line size. This indicates that hit ratio is more important in affecting the mean memory delay time when a large line size is used.

5.2 Implication of Data Bus Width and Hit Ratio Tradeoff

We use the following example to address the implication of bus width and cache memory tradeoff.

Example 1 Case 1: Increasing the hit ratio of a full blocking cache from 91% to 95.5% requires increasing cache size from 8K to about 32K for a trace driven simulation reported by Short and Levy in [14]. The same performance improvement can be achieved by increasing the bus width from 32 bits to 64 bits in the same processor while keeping the cache size at 8K. Thus, a processor with a 64-bit bus and an 8K cache and a processor with a 32-bit bus and a 32K cache have the same execution time in a uniprocessor system.

Case 2: To consider a larger cache size, we again use the hit ratio and cache size from [14]. Using our results, a processor with a 64-bit bus and a 32K cache and a processor with a 32-bit bus and a 128K cache have the same performance in a uniprocessor system. □

From the above example, we observed that by doubling or quadrupling a small cache such as 8 Kbytes, the performance can in general be increased to the level as that obtained by doubling the data bus width. In this context, we can increase a relatively smaller amount of chip area in the cache memory to trade for the processor pin counts and memory data bus width. On the other hand, increasing the bus width is more advantageous for trading the chip area when the cache is large (i.e., has high hit ratio) because bus width trades a larger cache size (chip area) for a large cache.

5.3 Unified Comparisons

Figure 3–5 illustrate the tradeoffs among hit ratio, pipelined system, bus width, processor stalling feature, and read-bypassing write buffers. The comparisons for these architecture features are based on the same grounds, i.e., a full-blocking cache and non-pipelined memory system. The non-pipelined memory cycle β_m is plotted on the horizontal axis. The curves serve two purposes. First, they show how each individual architecture feature is traded with the hit ratio. Second, the curves show the comparisons among the features themselves. The solid line shows the amount of hit ratio difference which is required to trade the performance of using the pipelined bus and memory system. If the memory cycle β_m is two, pipelining or not does not make the difference because $q = 2$. This is shown in Figure 3–5 where the solid lines meet with the x axis.

In Figure 3, a fast pipelined system is used with $q = 2$, and the line size is eight bytes. The flush ratio is assumed to be 0.5 considering the average situation. The BNL_1 stalling-feature is evaluated with the average stalling factor obtained from the simulations. The dashed curve represents the best performance of using the read-bypassing write buffers because there are some reads that can not bypass the on going writes. This situation is similar to the request stalling in using a partially-stalling cache. From the simulations, we found that a processor has very frequent consecutive requests made on the same missing line. When a load miss is in progress, the occurrences of another miss for the other cache lines are much fewer than the load/store accesses on the line which is being filled. It is much easier to hide the cache flush cycles successfully by using the write buffers because (1) the flushed cache line is posted after the missing line is filled, and (2) the processor will spend some time using the data on the line just retrieved.

We notice that for $L/D = 2$ in Figure 3, using a high speed pipelined system does not display any performance advantage over doubling the bus width even for a large memory cycle time. When the line size to bus width ratio is increased, the advantage of using a pipelined system materializes as shown in Figure 4. The performance improvement due to the BNL_1 feature is quite limited (ref. Figure 3 and 4). The BNL_3 feature has a higher performance improvement when the memory cycle time is small (ref. Figure 5).

From Figure 3–5, we rank the performance of each of the features except the pipelined memory as follows. In general, doubling the bus width is the best choice. The second best choice is providing the read-bypassing write buffers. Finally, using a cache with a bus-not-locked feature is the third best choice. This observation is generally good for a wide range of memory cycle time and is not sensitive to the line sizes. The stalling factor for the non-blocking cache was not evaluated from the simulation. However, because many subsequent load/store accesses are directed to the missing line, even a load miss does not stall the execution of a processor, subsequent load/store accesses will be stalled unless the mechanism for supporting multiple load/store miss is provided.

The use of a pipelined system is most advantageous after the memory cycle time reaches the cross over points in the curves. Doubling the bus width, using the read-bypassing write buffers, and a cache with a bus-not-locked feature ($BNL1$ or $BNL3$) has constant performance improvement over a relatively large memory cycle times range. The memory cycle time is less than about five or six clock cycles for $q = 2$ ($L > 2D$) when the performance of the pipelined system surpasses that of doubling the bus width. Notice that a large hit ratio (cache size) is traded with the usage of the pipelined system.

5.4 Validation of the Line Size and Hit Ratio Tradeoff

It is widely believed that given a cache size, using a larger cache line size (up to a certain range) has a higher hit ratio than using a smaller line size for the same application [4, 2]. Smith determined the best line size by finding the line size which minimizes the mean memory (cache miss) delay per memory reference [2]. Our goal here is to quantify the relationship among the line size, the cache hit ratio, and the memory cycle times and also validate the tradeoff methodology. To do this, we use $c + \beta(L/D)$ for the time in filling a cache line as used in [2]. The constant c accounts for the memory access latency, and β is the transfer time of the bus when D bytes are transmitted per bus cycle. We pose the question as how much of a hit (miss) ratio difference between using a larger line and a smaller line is necessary to justify the advantage of using a large line size in terms of mean memory access delay? We find the tradeoff by solving $X_{FS} = X^*_{FS}$ where

$$X_{FS} = (E - \frac{R}{L_0} - W) + \frac{R(1+\alpha)}{L_0}(c + (\frac{L_0}{D})\beta) + W(c+\beta) \quad (11)$$

and

$$X^*_{FS} = (E - \frac{R^*}{L^*} - W^*) + \frac{R^*(1+\alpha^*)}{L^*} (c + (\frac{L^*}{D})\beta) + W^*(c+\beta) \quad (12)$$

and obtain

$$R^* = \left(\frac{(1+\alpha)\left(c + (\frac{L_0}{D})\beta\right) - 1}{(1+\alpha^*)\left(c + (\frac{L^*}{D})\beta\right) - 1} \right) (\frac{L^*}{L_0})R. \qquad (13)$$

To serve the context of trading line size with hit ratio, let EHR be the hit ratio for using a larger line size L^*, and HR be the hit ratio for using a smaller line size L_0. Then, the difference of cache hit ratio for the same execution time is

$$\begin{aligned} \Delta EHR_{L^*} &= EHR - HR \\ &= MR - EMR \\ &= \frac{1-r}{s+1} \end{aligned} \qquad (14)$$

where $s = \frac{HR}{1-HR}$ and $r = \frac{\frac{R^*}{L^*}+W^*}{\frac{R}{L_0}+W}$.
In Eq. (14), r is smaller than one so that ΔEHR_{L^*} is greater than zero.

5.4.1 When to Use a Larger Line Size

In Eq. (14), we have found the minimum hit (miss) ratio difference ΔEHR_{L^*} required for using a larger line size L^* to have the same performance as using a smaller lines size L_0. Let the actual hit ratio difference between using L^* and L_0 for a given application be denoted by ΔHR_{L^*}. Given the same cache size, ΔHR_{L^*} is a constant. Using a larger line size which has a higher hit ratio does not improve performance if $\Delta HR_{L^*} < \Delta EHR_{L^*}$. Generally, our study shows that larger line sizes are better to be used in larger caches. This result conforms with those in [15]. Interested readers can refer to the work in [16] for further details.

5.4.2 Design Space of Bus Speed and Model Validation

Next, we determine the range of bus speeds or memory access time for which using a larger line size is beneficial. We use the hit ratio and line size tradeoff model combined with Smith's approach in determining the optimal line [2].

An optimal line size can be determined by finding the least average memory delay per memory reference. Suppose that we want to determine the optimal line size from the set of y line sizes represented by $\{L_i | 1 \leq i \leq y\}$. For $1 \leq i \leq y$, the optimal line size is determined by the following equation.

$$\text{Min}\left\{(1 - HR_{L_i})(c + \beta\frac{L_i}{D}) + HR_{L_i}\right\} \qquad (15)$$

The hit cycle time is assumed to be one here. Latency c and bus speed β are normalized with the hit cycle time. Smith multiplied $(c' + \frac{L}{D}\beta)$ by the corresponding miss ratio to find the optimal line size. He uses the following equation to determine the optimal line size.

$$\text{Min}\left\{(1 - HR_{L_i})(c' + \beta\frac{L_i}{D})\right\} \qquad (16)$$

where $c' = c - 1$ in relation to Eq. (15). What he minimized is actually the minimum mean cache miss delay time per memory reference. Since hit cycle times are the same for the comparison, the minimum of the cache miss delay can also determine the optimal line size.

We use the line size L_0 as a base for the comparison of mean memory delay with the set of y line sizes represented by $\{L_i | 1 \leq i \leq y\}$ where $L_i > L_0$. In this setting, we can examine whether a larger line size offers a performance advantage or not due to its higher hit ratio. The hit ratio of line L_i is denoted as HR_{L_i} and HR_{L_0} for L_0. We consider the range of line sizes where $HR_{L_i} \geq HR_{L_0}$. Based on the minimum mean memory delay approach, the best line is determined by the following equivalent maximum operations.

$$\begin{aligned} & \text{Max}\{((1 - HR_{L_0}) - (1 - HR_{L_i}))(c + \beta\frac{L_i}{D}) \\ & + HR_{L_0} - HR_{L_i}\} \\ = \; & \text{Max}\left\{(\Delta HR_{L_i})(c - 1 + \beta\frac{L_i}{D})\right\} \end{aligned} \qquad (17)$$

It becomes maximum operation because we choose the largest difference of the mean memory delay between line size L_0 and each of the other line sizes respectively. The largest difference means the smallest mean memory delay of the corresponding line size L_i. The following maximum operation determines the optimal line size and indicates the beneficial range of bus speed or memory access time for using that line size.

$$\text{Max}\left\{(\Delta HR_{L_i} - \Delta EHR_{L_i})(c - 1 + \beta\frac{L_i}{D})\right\} \qquad (18)$$

where ΔEHR_{L_i} is specified by Eq. (14) replacing L^* with L_i for $1 \leq i \leq y$ respectively. The above operation can also be represented as

$$\text{Max}\left\{(\Delta MR_{L_i} - \Delta EMR_{L_i})(c - 1 + \beta\frac{L_i}{D})\right\} \qquad (19)$$

Since the difference of hit ratio equals to the difference of miss ratio. Line size L_i is justified by its sufficient lower miss ratio being a large size when the above maximum has a value greater than zero. The value represented by Eq. (19) is the reduced (from the memory delay of using size L_0) memory delay per memory reference.

We compare the optimal line size determined by Eq. (19) with the results of Smith's work. Our comparison results are presented in Figure 6. The optimal line sizes determined by Eq. (19) exactly match with those of Smith's work. This result validates our tradeoff methodology. In addition, our analysis shows the range of the bus speeds to be used when using a larger line size is beneficial. These bus speeds must have a positive reduced memory delay. The bus speeds where the negative reduced memory delay occurs are too slow to be useful for a larger line size to take the advantage of its larger hit ratio.

6 Summary

We investigated the performance tradeoffs among the external data bus width, cache hit ratio, processor stalling features, read-bypassing write buffers, and pipelined memory systems. In addition, we also studied the inter-relationship among the line size, the hit ratio, and the memory cycle times, and thereby validated our tradeoff approach. The impact of these architectural features on performance was evaluated by developing a CPU execution time model. Our tradeoff model is based on the equivalence of the mean memory delay time in two systems using different architectural features and thus is independent of the non-load/store instructions. Using this model, we obtained the following specific results:

- By increasing the size of a small on-chip cache or by doubling the data bus width, identical performance improvements can be achieved. For example, for $L \geq 2D$ and $\alpha = \alpha' = 0.5$, the performance improvement achieved by increasing the cache hit ratio at HR by a value in the range $0.5(1-HR)$ to $0.6(1-HR)$ is the same as that obtained by doubling the data bus width.
- In non-pipelined memory system, the three best architectural features in order of priority to improve performance are doubling the bus width, providing the read-bypassing write buffers, and the use of a cache with a bus-not-locked.
- A cache that allows other cache lines to be accessed while a line is being filled has a very limited performance advantage. If, however, subsequent load/store accesses are only stalled by the latency of the requested data, then the read miss latency of a full blocking cache can be reduced by 20-30% for a memory cycle time of less than 15 clock cycles.
- The pipelined memory system helps to improve performance most when the memory cycle time is larger than about five clock cycles (for $L/D > 2$ and $q = 2$). The performance improvement by doubling the bus width, using read-bypassing write buffers, and using caches with a bus-not-locked feature is limited when the memory cycle time is relatively large. We have shown that the availability of a pipelined memory and bus system has significant impact on the performance.

Currently, we are studying the effects of multiple instruction issues on the results presented here. Our approach to this future research is similar to the one we developed above. We will develop a CPU execution time model for systems where the throughput could be more than one instruction per clock cycle.

References

[1] J. R. Goodman, "Using Cache Memory to Reduce Processor-memory Traffics," *Proceeding of the 10th Intl. Symp. on Comput. Arch.*, pp. 124- 131, 1983.

[2] A. J. Smith, "Line (Block) Size Choice for CPU Cache Memories," *IEEE Transactions on Computers,* Vol. C-36, No. 9, pp. 1063-1075, September 1987.

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

[4] S. Przybylski, M. Horowitz, and J. Hennessy, "Performance Tradeoffs in Cache Design," *Proceeding of the 15th International Symposium on Computer Architecture,* pp. 290- 298, May 1988.

[5] S. Przybylski, "Performance Impact of Block Sizes and Fetch Strategies," *Proceeding of the 17th International Symposium on Computer Architecture,* pp. 160-169, May 1990.

[6] D. B. Alpert, and M. J. Flynn, "Performance Trade-offs for Microprocessor Cache Memories," *IEEE Micro,* pp. 45- 54, August 1988.

[7] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proceeding of the 17th International Symposium on Computer Architecture,"* pp. 364-373, 1990.

[8] W. Y. Chen, S. A. Mahlke, and W. W. Hwu, "Tolerating First Level Memory Access Latency in High-Performance Systems," *Proceeding of the International Conference on Parallel Processing,* pp. I36-I43, August 1992.

[9] T. F. Chen and J. L. Baer, "Reducing Memory Latency via Non-blocking and Prefetching Caches," *Technical Report 92-06-03,* Dept. of Computer Science and Engineering, University of Washington, June 1992.

[10] D. M. Tullsen and S. J. Eggers, "Limitations of Cache Prefetching on a Bus-Based Multiprocessor," *Proceeding of the 20th International Symposium on Computer Architecture, 1993*

[11] A. J. Smith, "Second bibliography on cache memories," Comput. Architecture News, Vol. 19, No. 4, pp. 154-182, June 1991.

[12] A. J. Smith, "Cache Evaluation and Impact of Workload Choice," *Proceeding of the 12th International Symposium on Computer Architecture,* pp. 64-73, June 1985.

[13] *KSR1 Technical Summary*, Kendall Square Research, 1992.

[14] R. T. Short and H. M. Levy, "A Simulation Study of Two- Level Caches," *Proceeding of the 15th International Symposium on Computer Architectures*, pp. 81-88, 1988.

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture, A Quantitative Approach,* Morgan Kaufmann Publishers, Inc. 1990.

[16] C. H. Chen, "High-Performance High-Integrity System Design: Architectural Tradeoff Methodologies and A Cache Error Recovery Protocol," Ph.D. dissertation, Department of Electrical Engineering, University of Washington, 1993.

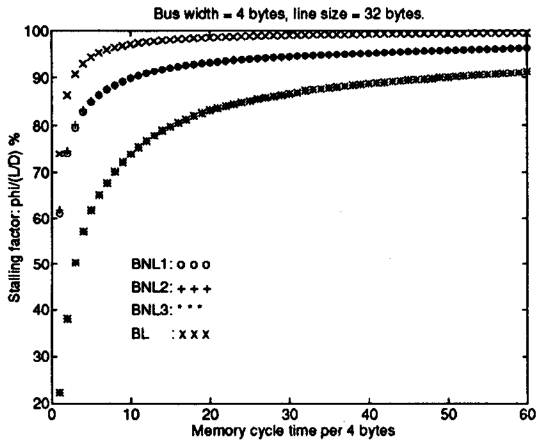

Figure 1: Stalling Factor (average from six SPEC92 programs with 50M instructions executed each, write-allocate, 8 Kbytes, two-way set associative.)

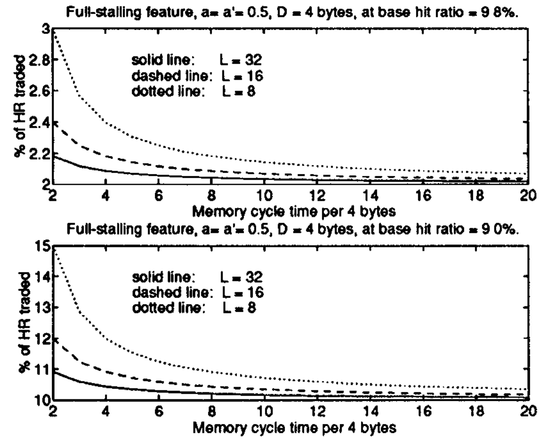

Figure 2: Effect of Memory Latency on the Hit Ratio and Bus Width Trading (Use $D = 4$ bytes as the base system. $\alpha = \alpha' = 0.5$)

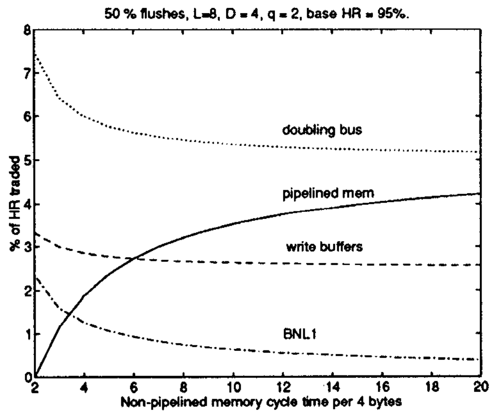

Figure 3: Architectural Tradeoff for L = 8 Bytes

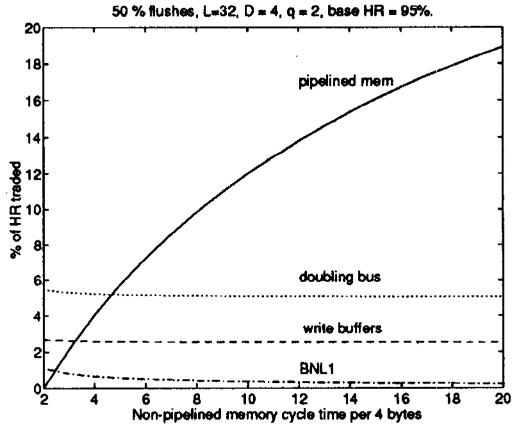

Figure 4: Architectural Tradeoff for L = 32 Bytes

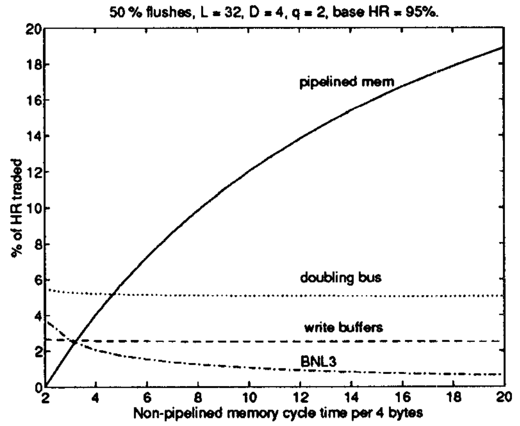

Figure 5: Architectural Tradeoff for BNL_3

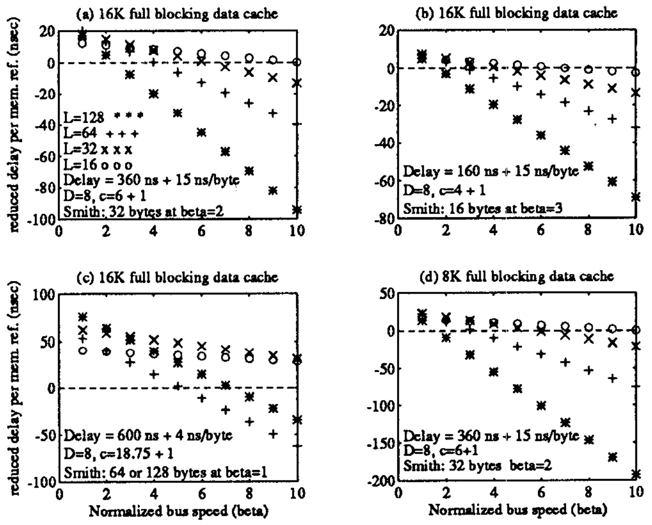

Figure 6: Validation with Smith's Design Target Hit Ratio for Data Caches