Architecture Technique Trade-Offs Using Mean Memory Delay Time

Chung-Ho Chen and Arun K. Somani, *Senior Member, IEEE*

Abstract—Many architecture features are available for improving the performance of a cache-based system. These hardware techniques include cache memories, processor stalling characteristics, memory cycle time, the external data bus width of a processor, and pipelined memory system, etc. Each of these techniques affects the cost, design, and performance of a system. We present a powerful approach to assess the performance trade-offs of these architecture techniques based on the equivalence of mean memory delay time. For the same performance point, we demonstrate how each of these features can be traded off and report the ranking of the achievable performance of using them.

Index Terms—Bus width, cache hit ratio, memory cycle time, performance trade-off, pipelined memory, read-bypassing write buffer.

1 Introduction

Architecture techniques that affect memory latency are the major factors that determine the performance of a computing system. Using a larger cache memory is usually suggested to increase the memory bandwidth [1]. Improving the cache hit ratio reduces the mean memory delay time. Increasing the width of a processor external data bus increases the performance of a cache-based system. Alternatively we can say that increasing the data bus width reduces the mean memory delay time for a given application. Other architecture features such as read-bypassing, write buffers, and pipelined memory systems also improve the performance by reducing the mean memory delay time.

In a cache-based system, the mean memory delay time (t_{mean}) can be represented by $HR + (1 - HR) \times t_{main}$ where HR is the hit-ratio for the given cache and t_{main} is the (main) memory latency normalized with the cache hit cycle time. At first glance, the representation of t_{mean} seems to be straightforward; however, many important architecture issues are involved. Architecture features such as external data bus width, pipelined structures, cache stalling features, or read-bypassing write buffers affect the memory cycle time. Thus, the effect of all these factors must be included in determining the t_{mean} for a given cache-based system. It has been chaotic that one system using a wider data bus claims better performance over a system using a larger cache or vice versa. Each of these hardware features has performance trade-offs. This paper presents a unified trade-off methodology to compare the achievable performance improvement due to the use of each of the architecture techniques that affect the mean memory delay time. (An earlier version of this paper appeared in the Proceedings of the 21st Annual International Symposium on Computer Architecture, April 18-21, 1994, Chicago.) The performance of an architecture feature is related to a common metric, the hit ratio, based on the equivalence of mean memory delay time.

Our analysis shows the amount of hit ratio that yields the same performance improvement as obtained by doubling the data bus width. Caches with different types of stalling features or memory design trade differently with the data bus width. When we refer to a processor data bus, we mean the external data bus of a processor. Our approach allows to study the impact of other architecture features which affect the memory delay time and thus a unified comparison of the architecture performance can be done. For instance, we can rank the effectiveness of different architecture techniques. Except for the pipelined memory system, to improve the performance of a system using non-pipelined memories, doubling the data bus width is the best choice. Using read-bypassing write buffers is the second best choice, while the use of a cache that allows cache access when a cache is filling due to a load miss is the third best choice.

We also determine the cross-over point of the memory cycle time where the use of a pipelined memory system is most advantageous.[1] The memory cycle time when the performance of a pipelined memory 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 or burst read memory system can trade a lot of hit ratio (cache size) and should be seriously considered in the design of cache-based systems.

In verifying the methodology, we apply our approach to determine the optimal line size and obtain the exact same results as in Smith's work [2]. We further show that the trade-off methodology can be generalized for composite cache hit ratio and is a powerful approach in evaluating the performance of architecture techniques. The rest of the paper

- *C.-H. Chen is with the Department of Electronic Engineering, National Yunlin Institute of Technology, Taiwan, Republic of China. E-mail: chen@el.yuntech.edu.tw.*
- *A.K. Somani is with the Department of Computer Science and Engineering and the Department of Electrical Engineering, University of Washington, Box 352500, Seattle, WA 98195. E-mail: arun@shasta.ee.washington.edu.*

Manuscript received November 1994; revised November 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96151.

1. The pipelined memory model considered in this paper is addressed in Section 4.4.

is organized as follows. We discuss related research in Section 2. In Section 3, we give the notation and the hardware characteristics under study. We develop the trade-off methodology in Section 4. The trade-off analysis and verification are presented in Section 5 and is generalized for composite hit ratio system and burst read memories in Section 6. This paper is concluded in Section 7.

2 Related Research

The design of a cache memory involves many issues such as line fetch algorithm, replacement policy, write handling protocol, split or unified cache, coherency, virtual or real address tag, and 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 find the line size which minimizes the *mean memory delay per memory reference* or the *mean read time*. Mean memory delay time is not the only factor that determines the performance, but it can serve as a measure to evaluate the performance of architecture techniques that affect the memory request latency.

Alpert and Flynn showed that using a larger line size reduces the overhead of storing address tags and other cache control information and thus leads to more a cost-effective cache design [6]. Chen and Baer examined the effectiveness of using nonblocking caches, prefetching caches, and read bypassing write buffers in reducing memory latency [9]. 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 and optimize a particular cache implementation [10]. From the system design point of view, optimizing the design space around hit ratio may only result in a little performance improvement, for instance, if the memory latency is small. In addition to cache memories, architectural features such as data bus width, memory latency, pipelined memories, and how they are used may have significant effects on the performance of a cache-based system. Since each of the architecture techniques considered here affects the mean memory delay time, the mean memory delay time can be used to evaluate the performance of these architecture techniques including cache design.

3 Representation of Execution Time

The relationships between the hardware architectural techniques and the characteristics of programs are derived by computing the CPU execution time considering various techniques that affect memory latency. The notation of the parameters is described in Table 1. These parameters specify the characteristics of an architecture feature, application, and the relationship between them. An application here can be a task, a subroutine, or any phase of a computation.

The following baseline system model is used.

1) A RISC processor is considered, which has an on-chip write-back and write-allocate data cache and an instruction 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. All of the memory references are first directed to the on-chip caches.
3) Both a nonload/store instruction and a load/store instruction that hits in the cache effectively take one cycle (due to pipelining) for execution.
4) The memory system has the same memory cycle time for read and write requests. This approximation of memory model is only used for a baseline system with a nonburst memory design. Therefore, it does not apply to current design that uses burst memories; nevertheless, this basic model provides a base for comparing the effectiveness of higher performance memory systems such as pipelined or burst memory design with the cache systems.

In a cache-based system, the number of bytes reads, R (R_I), can be related to the miss ratio of a cache. Parameter α is used to specify the amount of traffic due to the write-back of dirty lines while the parameter ϕ is associated with the processor (cache) stalling feature as will be defined later. An application in a uniprocessor system with specific architectural features is characterized by $\{E, R_I, R, \alpha, \phi\}$.

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 with line size L of the on-chip data cache and is given by $\lambda_m = \frac{R}{L}$. Since every read/write miss results in loading a full line into the cache, R is divided by L to represent the number of load/store instructions that miss in the data cache.

TABLE 1
Architecture Parameters

D	processor's external data bus width in bytes. D is a number like 4, 8, 16, or 32.
L	cache line size in bytes.
β_m	memory cycle time per read/write memory cycle.
E	number of instructions executed for an application.
R	number of data bytes read in full bus width upon misses for E instructions executed. R does not include instruction fetch.
R_I	number of instruction bytes read in full bus width upon misses for E instructions executed.
α	cache line flush ratio for E instructions executed. Given $0 \le \alpha \le 1$, the number of bytes of cache dirty lines which are copied-back (flushed) for E instructions executed is represented by αR.
ϕ	stalling factor.

3.1 Stalling Features

In this paper, we distinguish processor (cache) stalling features that are related to various cache implementations with the mechanism of processor bus interface control that may determine how the memory system is designed. A cache miss may be satisfied in different ways that are determined both by the cache implementation as well as the memory system design. Memory system design determines how soon a cache line can be moved into the cache while the stalling features determine when the CPU may use the available cache data.

For the stalling features addressed in the following, the nonpipelined memory model is used. Table 2 lists the cache stalling features that are considered in this study. Fig. 1 depicts the actual delays incur when various stalling features are used. 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 the full-stalling, $(L/D)\beta_m$ cycles contributes to the execution time for each cache miss. Therefore, the stalling factor is L/D as indicated in Table 2.

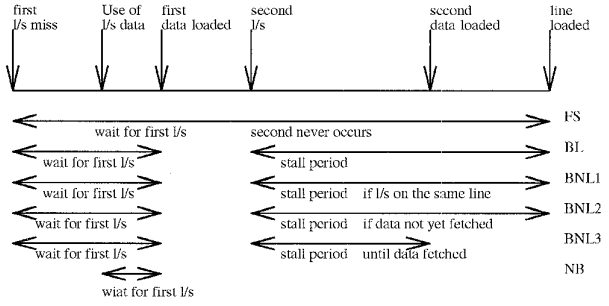


Fig. 1. Cache stalling features and delays incurred.

TABLE 2
PROCESSOR STALLING FEATURE

	features	stalling factor
FS	full-stalling	$\phi = \frac{L}{D}$
BL	bus-locked	$1 \le \phi \le \frac{L}{D}$
BNL	bus-not-locked	$1 \le \phi \le \frac{L}{D}$
NB	nonblocking	$1 \le \phi \le \frac{L}{D}$

In a cache-bus-locked feature, for a miss cycle, the cache first requests the missed data from the memory. As soon as the requested data arrive, the processor continues execution. This feature is the "out-of-oder" fetch property as introduced in [13]. The cache fetches the rest of the line and the cache bus remains locked up. If any load/store occurs during this period, that load/store is stalled until the line is completely fetched. We call this stalling feature bus-locked (*BL*). The minimum value of the stalling factor ϕ due to *BL* stalling is one where no subsequent cache access occurs while the cache bus is locked up; however, the maximum value could still be up to L/D.

To reduce the stalling delay, an alternative is to allow the processor to access other cache lines, and the cache bus is not locked up. This stalling feature is bus-not-locked (*BNL*). In such a case, for a second access on the line being fetched currently, the processor may be stalled for a variable period depending on the implementation. We consider three possible scenarios and assume that a line is filled using multiple bus transfers. In scenario BNL_1, the processor is stalled by the second access for the entire duration until the line is completely fetched even though the second access was requesting data which were just brought in. In scenario BNL_2, the processor is stalled only if the second access happens to be on that part of the line which has not been yet fetched. If stalling occurs, the processor is delayed for the time until the entire line is fetched completely. In scenario BNL_3, a stall occurs only if the data have not been yet fetched. Otherwise, an access can be satisfied by a partially filled line. The value of ϕ for different cases of the *BNL* feature can vary between 1 and L/D as indicated in Table 2.

A cache may be designed with the nonblocking (*NB*) feature, that is, the processor is allowed to access all cache lines while the cache is filling a missing line. The execution of a processor may be stalled by a nonblocking cache as well if the data being fetched need to be used [9]. The minimum possible stalling factor for the *NB* feature is zero where no subsequent instruction uses an operand being loaded by a previous cache miss process. The *NB*, or the *BL*, or the *BNL*, is partial-stalling (*PS*) in contrast to the full-stalling feature. Parameter ϕ which quantifies the performance of various cache implementations also affects the mean memory delay time in a cache-based system.

3.2 Execution Time for the Baseline System

Two elements contribute to the CPU execution time. First, the time includes the instruction fetching cycles from the memory due to instruction cache misses. Instruction caches with 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 which have two buses to access two large instruction and data caches, respectively [12]. The dominating part of the CPU execution time is the time to execute the instructions. For a RISC processor, the execution time X is represented by the following expressions:

$$X = (E - \lambda_m) + \frac{R}{L}(\phi\beta_m) + \frac{\alpha R}{D}\beta_m. \quad (1)$$

The $(E - \lambda_m)$ 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/store instructions that miss in the cache 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. With advanced technology, 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 represents the memory references whose miss penalty cannot be hidden. Or alternatively, β_m can be scaled down to represent the average miss penalty.

3.3 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 instruction pipelining, the instruction hit cycles are overlapped with the execution time X in (1). When instruction cache hit ratio is high, 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 (1). In either case, the CPU execution time is always represented in the same form as in (1).

4 Derivation of Performance Equivalent Point

When comparing the performance trade-offs for different architecture features, we improve the system with each architecture feature and measure the performance improvement. This performance enhancement can be achieved by doubling the data bus width, or changing from a full-stalling cache to a partial-stalling one, or providing the read-bypassing write buffers, or using a pipelined or burst read memory system. For the same performance (mean memory delay time), the original system must use either a larger cache or must use a more powerful architecture feature. Hence, a relationship between the difference of hit ratio for the two systems and the performance-improving technique used exists at the same performance equivalent point while the application and other architecture features remain unchanged. In the following, we derive the performance trade-offs between hit ratio and various architecture features by establishing a performance equivalent point.

4.1 Data Bus Width versus Cache Hit Ratio

To begin, we consider the trade-off between cache hit ratio and data bus width for the baseline model where the non-pipelined memory cycle time is used and no write buffers are provided. The following two expressions denote the execution time of using data bus width D and $2D$, respectively.

$$X_D = \left(E - \frac{R}{L}\right) + \frac{R}{L}(\phi\beta_m) + \frac{\alpha R}{D}\beta_m \tag{2}$$

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

Here $\{R', \alpha', \phi'\}$ and $\{R, \alpha, \phi\}$ are parameters for the system using data bus width $2D$ and D, respectively. The maximum value of ϕ' is $L/2D$ assuming that $L \geq 2D$. 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 and $2D$, respectively, have the same execution time. (Shortly we will show that this equality is in fact based on the equivalence of mean memory delay time.) In the two systems, the number of instructions (E) executed, the cache line size, the memory cycle time, and the stalling feature are all the same. Solving $X_D = X_{2D}$, yields

$$\frac{R'}{R} = \left(\frac{\left(\phi + \left(\frac{L}{D}\right)\alpha\right)\beta_m - 1}{\left(\phi' + \left(\frac{L}{2D}\right)\alpha'\right)\beta_m - 1}\right). \tag{4}$$

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

$$MR_1 = \frac{\lambda_m}{\lambda_m + \lambda_h} = \frac{1}{s+1}. \tag{5}$$

We use the hit (miss) ratio in the D width system as a base, namely, given a hit (miss) ratio for the D width system. 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 same application is considered, 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 due to a smaller cache size so that they result in the same performance.

Let MR_2 (HR_2) be the miss (hit) ratio associated with the $2D$ width system and $\lambda'_m = r\lambda_m$ where $\lambda'_m = \frac{R'}{L}$ then

$$MR_2 = \frac{\lambda'_m}{\lambda'_h + \lambda'_m} = \frac{r\lambda_m}{\lambda_h + \lambda_m} = \frac{r}{s+1}. \tag{6}$$

Let HR_1 be the hit ratio associated with the D width system. Between the two systems, the difference of hit ratios equals the difference of miss ratios. Then the cache hit ratio that trades the performance of a D-byte width is

$$HR_1 - HR_2 = MR_2 - MR_1 = \frac{r-1}{s+1} \tag{7}$$

where $s = \frac{HR_1}{1-HR_1}$ (from $\lambda_h = s\lambda_m$) and $r = \frac{R'}{R}$ (from $\lambda'_m = r\lambda_m$). Equation (7) is only valid for the physical system where $HR_2 \geq 0$. Row one of Table 3 lists the execution time for the $2D$ width system, ratio r, and stalling factors for which full stalling caches are considered.

TABLE 3
Ratio of Data Cache Misses (R) and Stalling Factors (Write Allocate)

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

THEOREM 1. *The hit ratio to bus width performance trade-off is based on the equivalence of mean memory delay time and is independent of the non-memory reference instructions such as multiple cycle floating point operations.*

PROOF. With $\lambda_h + \lambda_m = \lambda'_h + \lambda'_m$ (the total number of data memory references) and $X_D = X_{2D}$, we obtain

$$\frac{\frac{R(1+\alpha)}{D}\beta_m + \left(E - \frac{R}{L} - N_{LS}\right)}{\lambda_h + \lambda_m} = \frac{\frac{R'(1+\alpha')}{2D}\beta_m + \left(E - \frac{R'}{L} - N_{LS}\right)}{\lambda'_h + \lambda'_m} \quad (8)$$

where N_{LS} is the number of the nonload/store instructions and $\phi = L/D$, $\phi' = L/2D$ are used as an example. Since $\lambda_h = E - \frac{R}{L} - N_{LS}$ and $\lambda'_h = E - \frac{R'}{L} - N_{LS}$, therefore

$$\frac{\overbrace{\frac{R(1+\alpha)}{D}}^{\text{memory cycles}}\beta_m + \overbrace{\lambda_h}^{\text{hit cycles}}}{\lambda_h + \lambda_m} = \frac{\overbrace{\frac{R'(1+\alpha')}{2D}}^{\text{memory cycles}}\beta_m + \overbrace{\lambda'_h}^{\text{hit cycles}}}{\lambda'_h + \lambda'_m}$$

As shown, both sides of the above expression are the mean memory delay time per (data) memory reference. Hence, the trade-off results are based on the equivalence (balance) of mean memory delay times. □

From Theorem 1, the assumption that a nonload/store instruction is executed in one clock cycle can be relaxed for nonsuperscalar processors.

THEOREM 2. *If the baseline system A and B both use write-allocate and full-stalling data caches with the same line size and have an average copy-back ratio of* 0.5, *and system A have data bus width D bytes and line size L = 2D, then system A with a design hit ratio HR_1 has the same performance as obtained from system B which uses a 2D data bus and a cache with a hit ratio of* $2.5HR_1 - 1.5$.

PROOF. The fastest possible nonpipelined memory system, $\beta_m = 2$, is used to find the lower bound of hit ratio of system B.[2] With $\alpha = \alpha' = 0.5$ for the same copy-back ratio, based on the equivalence of mean memory delay time, we find $r = \frac{R'}{R} = 2.5$ from (4). Substituting the value in (7) yields $HR_2 = 2.5HR_1 - 1.5$. □

Note that the results in this section are applicable to systems with nonpipelined memory design. The above analysis considers the situation where the fastest nonpipelined memory system is used. Thus, it represents one end of the design spectrum. The other end of the design space is considered in the following theorem.

THEOREM 3. *If the baseline system A and B both use write-allocate and full-stalling data caches with the same line size and have the same copy-back ratio, and system A has data bus width D bytes and line size L ≥ 2D, then system A with a design target hit ratio HR_1 has the same performance as obtained from system B which uses a 2D data bus, and uses a cache with a hit ratio of* $2HR_1 - 1$.

PROOF. Applying the L'Hospital's rule in (4) for a relatively large β_m, we find $r = \frac{R'}{R} = 2$ and $HR_2 = 2\,HR_1 - 1$. □

The above results state that the performance loss due to reducing the hit ratio of a blocking data cache from HR to $2HR - 1$ or at most to $2.5HR - 1.5$ can be compensated by doubling the data bus width. The hit ratio traded between $2HR - 1$ and $2.5HR - 1.5$ is determined by the line size and the memory cycle time.

We can also use the hit (miss) ratio in the $2D$ width system as a base, i.e., given HR_2 (MR_2). 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

$$HR_1 - HR_2 = \frac{1-r}{s+1} \quad (9)$$

where $s = \frac{HR_2}{1-HR_2}(\lambda'_h = s\lambda'_m)$ and $r = \frac{R}{R'}$.

COROLLARY 1. *Given the base hit ratio (HR_2) of the 2D bus width system, L = 2D, $\beta_m = 2$, and $\alpha = \alpha' = 0.5$, the amount of hit ratio that trades the performance of a D-byte bus is* $0.6(1 - HR_2)$.

PROOF. This result can be obtained from (9) directly or from the result of Theorem 2, that is, $HR_1 - HR_2 = 0.6(1 - HR_2)$. □

COROLLARY 2. *Given the base hit ratio (HR_2) of the 2D bus width system, $\alpha = \alpha'$, L ≥ 2D, and a relatively large β_m, the amount of hit ratio that trades the performance of a D-byte bus is* $0.5(1 - HR_2)$.

PROOF. $HR_1 - HR_2 = 0.5(1 - HR_2)$ from (9). □

The above results state that given $L \geq 2D$ and $\alpha = \alpha' = 0.5$, the performance improvement due to increasing the data cache hit ratio at HR by $0.5(1 - HR)$ to at most $0.6(1 - HR)$ is the same as that obtained by doubling the data bus width for systems designed with nonpipelined memories.

4.2 Stalling Feature versus Hit Ratio

To improve performance, a partial-stalling cache can be designed instead of a full-stalling cache. Using a partial-stalling (PS) cache reduces the mean memory delay time. The performance trade-off between using a partial-stalling cache and cache hit ratio is obtained as follows. Let HR_1 be the hit ratio for the full-stalling cache system as in (2) with $\phi = L/D$ and HR_2 be the hit ratio for not using the full-blocking feature, or equivalently for having a partial-stalling feature. For the same performance, HR_1 must be greater than HR_2. The execution time X_{PS} for a *PS* stalling feature is shown in Table 3 row 2. The cache hit ratio difference traded for using the partial-stalling feature is the same as in (7). Note that the ratio r is obtained by solving (2) = X_{PS}. Moreover, ratio r is also based on the equivalence of mean memory delay time.

To evaluate the effectiveness of various stalling features, we use trace-driven simulations to obtain the stalling factor ϕ'. 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. In case of the BNL_1 stalling feature stalling occurs in two situations. First, if a

2. The reason is that with fast memories hit ratio has less weight than with slow memories.

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 distance between the ith and $(i+1)$th load/store instruction that miss in the cache. 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.

$$\phi' = \left(\sum_{i=1}^{\lambda_m}\left((\frac{L}{D}-1)\beta_m - \Delta C_i\right)\right)(\frac{1}{\lambda_m \beta_m}) + 1 \qquad (10)$$

where λ_m is the number of the load/store instructions that are misses. The last term, a constant 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 Fig. 2, we show the average stalling factors obtained from the trace-driven simulation of the SPEC92 programs nasa7, swm256, wave5, ora, doduc, and hydro2d. The programs as shown in Table 4 are used to generate traces with defaults on a DECstation 5000 with the *pixie* utility. The cache system we simulated is 8K bytes, two-way set associative with a line size of 32 bytes. These parameters are chosen based on a typical on-chip cache implementation. The simulation has run enough length of the traces considering the size of the on-chip caches to minimize the transient effect. The purpose of this simulation is to compute the stalling factor ϕ' as used in row 2 of Table 3.

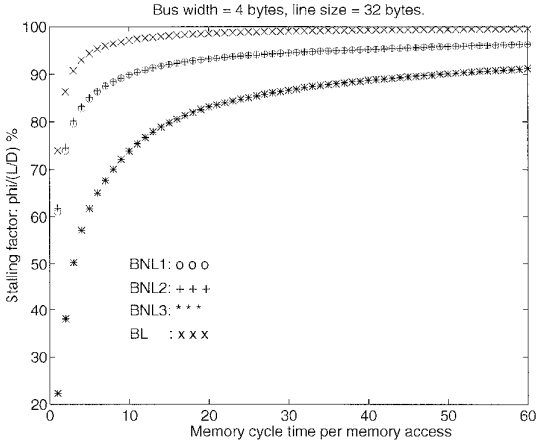


Fig. 2. Stalling factor (average from six SPEC92 programs with 50M instructions executed each, write-allocate, 8K bytes, two-way set associative).

The stalling factor is illustrated in the percentages of L/D. A partial-stalling feature 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 a 20-30% reduction in the read miss latency of a full-blocking cache can be achieved for a memory cycle time which is about smaller than 15 processor clock cycles.

TABLE 4
BENCHMARK DESCRIPTIONS AND DATA READ-WRITE STATISTICS FOR 50 MILLION INSTRUCTIONS EXECUTED

Benchmark	R/W (%)	Descriptions
nasa7	41.8	7 application routines that do matrix, manipulations, FFTs, Gaussian elimination, create vortices.
hydro2d	34.7	An astrophysics application.
swm256	29.5	A shallow water model that solves shallow water equations.
wave5	18.3	A two dimensional electro-magnetic particle-in-cell simulation.
ora	22.5	An optical system application.
doduc	35.6	A Monte Carlo simulation for a nuclear reactor component.

R/W % = percentage of load/store instructions

4.3 Read-Bypassing Write Buffers versus Hit Ratio

With an appropriate memory cycle time, the maximum efficiency of using read-bypassing write buffers is achieved when the cache flush latency can be completely hidden. The execution time X_{WB} described in Table 3 row 3 represents the best possible performance of using read-bypassing write buffers where the α portion is zero. Let HR_2 be the hit ratio for having the read-bypassing write buffers with the execution time X_{WB}, and HR_1 be the hit ratio for not having read-bypassing write buffers with the execution time as shown in (2). In both cases, the stalling factor is L/D. The ratio r in Table 3 row 3 is obtained by solving (2) $=X_{WB}$. The cache hit ratio difference traded for the performance of using the read-bypassing write buffers is determined by (7) with $s = \frac{HR_1}{1-HR_1}$.

4.4 Pipelined (Burst Read) Memory System versus Hit Ratio

Performance improvement can be achieved by changing the nonpipelined memory into a pipelined one where the next memory cycle can be started without waiting for the previous memory cycle to finish. In this paper, the pipelined operation considered is limited to initiate for the requests of fetching (or writing back) an entire cache line. We have not modeled the pipelined operations across multiple-line fetches in this study. The pipelined cycle for fetching a cache line is described as follows. Let q be the clock cycles for the memory system to be ready for receiving a new address request and beginning the next pipeline cycle. The pipelined memory cycle time β_p per L-byte request is given by

$$\beta_p = \beta_m + q\left(\frac{L}{D} - 1\right). \qquad (11)$$

With the pipelined memory system, the execution of a processor is stalled by the pipeline latency 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 it needs for an L-byte line, and the pipelined memory 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 memory system. Because of the full-blocking feature, if the data bus width is equal to the line size, then the latency for a line is same both for pipelined and non-pipelined memory systems as indicated in (11) by setting $L = D$. For full-blocking write-allocate caches, we can relate the nonpipelined memory cycle time β_m with the pipelined memory cycle time β_p.

In trading the performance of a pipelined memory system with hit ratio, let HR_2 be the hit ratio for using the pipelined memory system with the execution time X_{PM} in Table 3 row 4 and HR_1 be the hit ratio for not using the pipelined memory system with the execution time as shown in (2). Again, in both cases, the stalling factor is L/D. The ratio r in Table 3 row 4 is obtained by solving (2) = X_{PM}. The cache hit ratio difference traded for using the pipelined memory system is the same as in (7) with $s = \frac{HR_1}{1-HR_1}$. This pipelined memory model is similar to that of the burst read mechanism where the first word of a cache line is read usually with more cycles than the subsequent words.

5 Results of Architecture Trade-Offs

We present the results obtained by using the above trade-off approach. The comparisons made are based on a system using a full-blocking cache, D-byte data bus, and the non-pipelined basic memory model.

5.2 Data Bus Width and Hit Ratio

Fig. 3 illustrates the performance trade-off between a 32-bit bus width and the hit ratio of a full-blocking cache with base hit ratio of 98% and 90%, respectively. When the data bus and memory width are increased from 32 bits to 64 bits, the hit ratio in the 64-bit case could be smaller than the base hit ratio of the 32-bit system for both systems to have the same performance. The amount of the hit ratio traded is shown on the y axis in Fig. 3. The design limit is reached when the memory cycle latency β_m is two. We assume that the flush ratio α is 0.5 although the other value of α can also be used. In [2], [11], Smith also used 50% in describing the copy back traffic.

In the upper part of Fig. 3, 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 another words, increasing the hit ratio from 96% to 98% (2% increase), we can reduce the bus width (processor data bus and memory bus) from 64 bits to 32 bits while retaining the same performance. When L = 8 bytes and $\beta_m = 2$, increasing hit ratio 3% (from 95% to 98%), a 32-bit bus width can get the same performance as a 64-bit system with 95% hit ratio. The lower part of Fig. 3 shows the

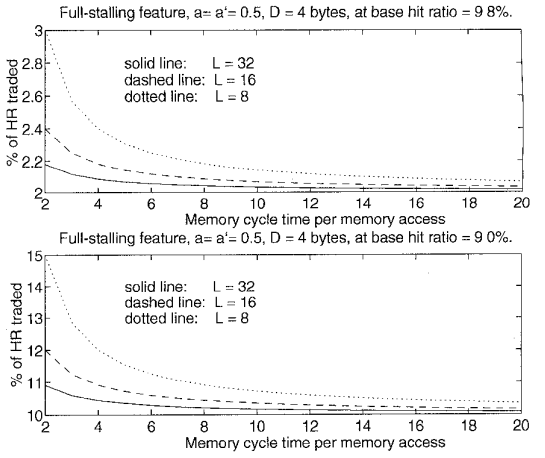


Fig. 3. Effect of memory latency on the hit ratio and bus width trading (use $D = 4$ bytes as the base system, $\alpha = \alpha' = 0.5$).

case where a different base hit ratio is used.

This illustration agrees with the previous limit analysis for (9). 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 at most $0.6(100 - HR)$% has the same performance as obtained by doubling the data bus. To relate performance to cost, we can map a hit ratio to cache size and trade-off the cache size with the data bus width at the performance equivalent point. For instance, a processor with a 64-bit data bus and an 8KB cache (hit ratio = 91% from the simulation results of [14]) and a processor with a 32-bit data bus and a 32KB cache (95.5%) establish a performance equivalent point. That is, the (64-bit, 8KB) system and the (32-bit, 32KB) system has the same performance. The (32-bit, 32KB) system has significantly reduced the package size and pin count of the processor. In other words, increasing the size of a small on-chip cache can easily achieve the same performance level that is achievable by doubling the data bus width. In this context, a relatively smaller amount of chip area is increased in the cache memory to trade for the processor pin count and memory data bus width.

To consider a larger cache size, we use the hit ratio and cache size data from Short and Levy's work again [14]. For instance, if we choose a (32-bit, 128KB) system as a base, the 64-bit system that has the same performance is one that has a 32KB cache. The conclusion made from this mapping example is that based on a system with a higher hit ratio, increasing the bus width is more advantageous for trading the chip area because bus width trades a larger cache size (die area) for a large cache.

If we map the hit ratio to cache size with other simulation data, such as those in [13 (p. 424)] and then trade-off with the bus width, the results are similar.

5.2 Unified Comparisons Based on Nonpipelined Memories

Figs. 4, 5, and 6 illustrate the trade-offs among hit ratio, pipelined memory 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 ($\phi = L/D$) and the nonpipelined memory system. (See (2).) The nonpipelined memory cycle β_m is plotted on the x axis. These curves serve two purposes. First, they show how each individual architecture feature is traded with the hit ratio to achieve the same mean memory delay. 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 does not make the difference because $q = 2$. This is shown in the figures where the solid lines meet with the x axis.

In Fig. 4, a fast pipelined memory 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 by using the read-bypassing write buffers because there are some reads that can not bypass on-going writes. This situation is similar to the one where a request is stalled in a partial-stalling cache. From the simulations, we found that an application makes very frequent consecutive requests to a cache line which just missed. That is, 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. We also found that it is much easier to hide the cache flush latency successfully by using the write buffers because

1) the flushed cache line is written after the missing line is filled, and
2) the processor will spend some time using the data on the line just fetched from the main memory.

We notice that for $L/D = 2$ in Fig. 4, using a high speed pipelined memory system does not show any performance advantage over doubling the bus width even for a large memory cycle time. When the line size to data bus width ratio is increased, the advantage of using a pipelined memory system is shown in Fig. 5. The performance improvement due to the BNL_1 feature is quite limited (see Figs. 4 and 5). The BNL_3 feature has a higher performance improvement when the memory cycle time is small (see Fig. 6).

From Figs. 4, 5, and 6, we rank the performance of each of the features except the pipelined memory system as follows. In general, doubling the data bus width is best, providing the read-bypassing write buffers is the second best while using a cache with a bus-not-locked feature is the third most useful. This observation is generally good for a wide range of memory latencies and is not sensitive to line sizes. The stalling factor for a nonblocking cache was not evaluated from the simulation. However, because many subsequent load/store accesses are directed to the line just missed and even if a processor does not stall the execution for the first miss, a subsequent load/store access will be stalled unless the processor include mechanism to support multiple load/store misses.

The use of a pipelined memory system is most advantageous when the memory latency reaches the cross over points in the curves. Doubling the data bus width, using the read-bypassing write buffers, and a cache with a bus-not-locked feature has a limited performance contribution over

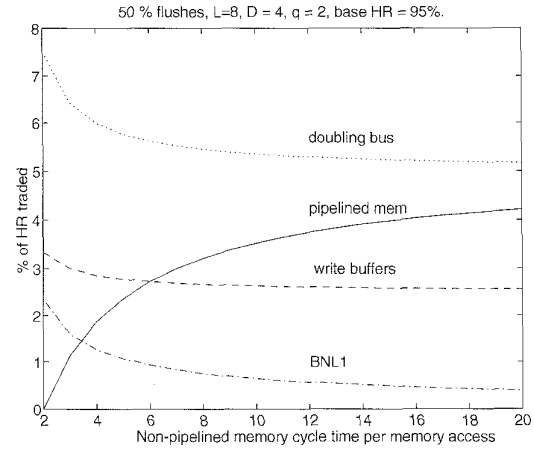


Fig. 4. Architecture trade-off for L = 8 bytes.

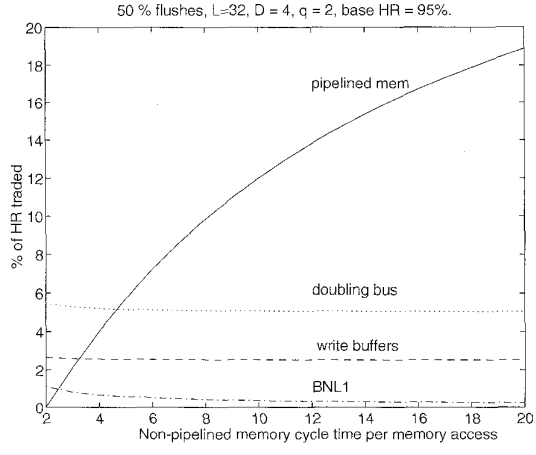


Fig. 5. Architecture trade-off for L = 32 bytes.

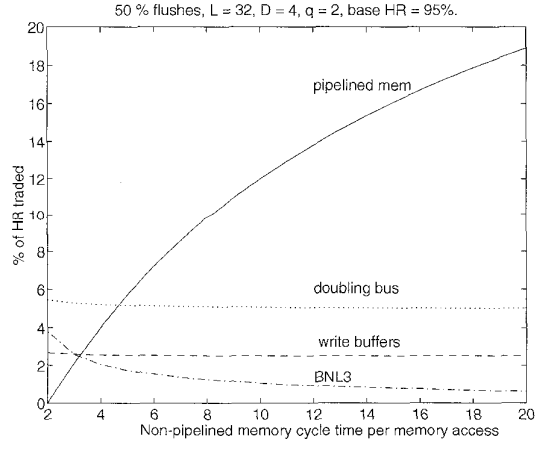


Fig. 6. Architecture trade-off for BNL_3.

a relatively large range of memory latency. The memory cycle time is less than about five or six clock cycles for $q = 2$ ($L > 2D$) when the performance of a pipelined memory system surpasses that of doubling the bus width. Notice that a large hit ratio (cache size) is traded with the usage of the pipelined memory system. For instance, in a 32-bit bus and 32-byte cache line size system, changing a 10-cycle per 32 bits nonpipelined memory system into a pipelined memory system can trade about 12% of the hit ratio for a given 95% hit ratio cache system as shown in Fig. 6. Thus, the 10-cycle per 32 bits nonpipelined memory system with a 95% hit ratio cache for some application has the same performance as a system with pipelined memory and a cache with only an 83% hit ratio for the same application. The huge amount of hit ratio traded for the use of a pipelined memory system shows that pipelined memories or burst read memories are probably the most cost-effective technique to use for performance.

5.3 Model Verification with Line Size and Hit Ratio Trade-Off

In this section, we verify the trade-off methodology by developing a performance trade-off between line sizes and the hit ratio, and compare our results with previously published results. Given a cache size, a larger cache line size (up to a certain range) usually results in a higher hit ratio than a smaller line size for the same application [2], [4]. Smith determined the best line size by finding the line size which minimizes the mean memory (cache miss) delay per memory reference [2].

As shown later, the trade-off approach presented in this paper obtains the exact same results as those of Smith. In addition, the trade-off approach can be used to quantify the inter-relationship in line size, cache hit ratio, and memory cycle times. To do this, we use $c + \beta(L/D)$ for the time it takes to fill 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 delay time. We find the trade-off by setting the simplified execution time to be equal, i.e., $X_{FS} = X^*_{FS}$, where

$$X_{FS} = (E - \frac{R}{L_0}) + \frac{R(1+\alpha)}{L_0}(c + (\frac{L_0}{D})\beta) \tag{12}$$

and

$$X^*_{FS} = (E - \frac{R^*}{L^*}) + \frac{R^*(1+\alpha^*)}{L^*}(c + (\frac{L^*}{D})\beta) \tag{13}$$

and obtain

$$\frac{R^*}{R} = \left(\frac{(1+\alpha)\left(c + \left(\frac{L_0}{D}\right)\beta\right) - 1}{\left(1+\alpha^*\right)\left(c + \left(\frac{L^*}{D}\right)\beta\right) - 1}\right)\left(\frac{L^*}{L_0}\right). \tag{14}$$

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 equivalence of mean memory delay time is

$$\Delta EHR_{L^*} = EHR - HR = MR - EMR = \frac{1-r}{s+1} \tag{15}$$

where $s = \frac{HR}{1-HR}$ and $r = \frac{\frac{R^*}{L^*}}{\frac{R}{L_0}}$.

ΔEHR_{L^*} is the minimum hit (miss) ratio difference required for using a larger line size L^* to have the same performance as using a smaller lines size L_0. Next, we use the hit ratio and line size trade-off relationship combined with Smith's approach to determine the optimal line size [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 $\{L_i \mid 1 \le i \le y\}$. For $1 \le i \le y$, the optimal line size is determined by the following minimum operations:

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

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

$$\text{Min}\left\{\left(1 - HR_{L_i}\right)(c' + \beta\frac{L_i}{D})\right\} \tag{17}$$

where $c' = c - 1$ in relation to (16). 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 case for the comparison of mean memory delay with the set of y line sizes represented by $\{L_i \mid 1 \le i \le y\}$ 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 when $HR_{L_i} \ge HR_{L_0}$. Based on the minimum mean memory delay approach, the best line is determined by the following equivalent maximum operations:

$$\text{Max}\left\{\left(\left(1 - HR_{L_0}\right) - \left(1 - HR_{L_i}\right)\right)(c + \beta\frac{L_i}{D}) + HR_{L_0} - HR_{L_i}\right\}$$

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

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:

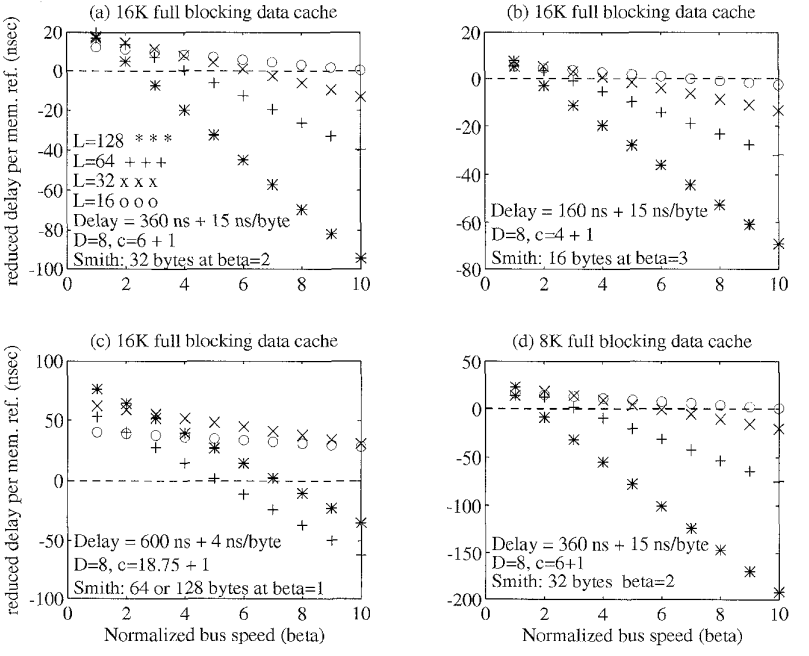


Fig. 7. Validation with Smith's design target hit ratio for data caches.

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

where ΔEHR_{L_i} is specified by (15) replacing L^* with L_i for $1 \le i \le y$, respectively. The above operation can also be represented as

$$\text{Max}\left\{\left(\Delta MR_{L_i} - \Delta EMR_{L_i}\right)(c - 1 + \beta\frac{L_i}{D})\right\} \qquad (20)$$

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. We compare the optimal line size determined by (20) with the results of Smith's work [2]. They are presented in Fig. 7. For instance, consider Fig. 7a. Given 360 ns + 15 ns/byte for the delay time and bus width D = 8 bytes, normalizing with 60 ns (a chosen processor cycle time), we obtain c = 6 + 1 and $\beta = (8 \times 15) \div 60 = 2$. At $\beta = 2$, the optimal line size determined by (20) for a 16KB data cache is 32 bytes which has the maximum reduced delay time per memory reference (see Fig. 7a). This result is exactly the same one as in Smith's work. The order of line size to choose is 32, 64, 16, and 128 bytes, which also matches with Smith's work. Consequently, our performance trade-off methodology is verified.

6 Generalization of the Trade-Off Methodology

We have shown the trade-off relationship between an architecture feature and the data cache hit ratio based on the nonpipelined memory model. In Section 6.1, we further generalize the trade-off methodology considering the composite hit ratio. In Section 6.2, we use a burst read memory model for the execution time equation and find the trade-off between data path and hit ratio.

6.1 Data Bus Width and Compositive Cache Hit Ratio

Considering the trade-off between composite hit ratio and data bus as an example, we rewrite the expression for the execution time. The following expression denotes the execution time of using data bus width D with I-cache miss penalty.

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

where ω is the ratio of instruction cache misses to data cache misses for the given system, i.e., $\omega = \frac{R_I}{R}$. Similarly for the $2D$ case, the execution time

$$X_{2D} = (E - \frac{R'}{L}) + \frac{R'}{L}(\phi'\beta_m) + \frac{(\alpha' + \omega')R'}{2D}\beta_m$$

with $\omega' = \frac{R'_I}{R'}$. Here $(R', R'_I, \alpha', \phi')$ and $\{R, R_I, \alpha, \phi\}$ are parameters for the system using data bus width $2D$ and D bytes, respectively.[3] The $\frac{\omega R}{D}\beta_m$ and $\frac{\omega' R'}{2D}\beta_m$ are used to quantify the time contributed by instruction cache miss assuming that full-stalling I-caches are used. The I-cache is assumed to have the same line size as the D-cache (data cache). The maximum value of ϕ' is $L/2D$ and $L \ge 2D$. Let $E = E_h + E_m$ where E_h (E_m) is the number of instructions that hit (miss) in the instruction cache used. Solving $X_D = X_{2D}$ yields

3. ω and ω' can be derived from (R, R_I) and (R', R'_I), respectively.

$$\frac{R'}{R} = \left(\frac{\left(\phi + \left(\frac{L}{D}\right)(\alpha + \omega)\right)\beta_m - 1}{\left(\phi' + \left(\frac{L}{2D}\right)(\alpha' + \omega')\right)\beta_m - 1} \right). \quad (21)$$

Let $E_h + \lambda_h = s(E_m + \lambda_m)$. The composite miss ratio MR_1 of the instruction and data cache for the case of D width is given by

$$MR_1 = \frac{E_m + \lambda_m}{E_h + E_m + \lambda_m + \lambda_h} = \frac{E_m + \lambda_m}{s(E_m + \lambda_m) + (E_m + \lambda_m)} = \frac{1}{s+1}. \quad (22)$$

Since the program characteristics remain unchanged, therefore $E_h + E_m + \lambda_h + \lambda_m = E'_h + E'_m + \lambda'_h + \lambda'_m$. Let

$$r = \frac{E'_m + \lambda'_m}{E_m + \lambda_m} = \frac{R'_I + R'}{R_I + R} = \frac{(1+\omega')R'}{(1+\omega)R}. \quad (23)$$

Let MR_2 be the composite miss ratio for the $2D$ width system,

$$MR_2 = \frac{E'_m + \lambda'_m}{E + \lambda'_h + \lambda'_m} = \frac{r(E_m + \lambda_m)}{E + \lambda_h + \lambda_m} = \frac{r}{s+1}. \quad (24)$$

Therefore, the composite hit ratio difference that trades a D width is

$$HR_1 - HR_2 = MR_2 - MR_1 = \frac{r-1}{s+1} \quad (25)$$

where r is specified by (23) and $s = \frac{HR_1}{1-HR_1}$.

COROLLARY 3. *The composite hit ratio to bus width performance trade-off is also based on the equivalence of the mean memory delay time.*

PROOF. Assuming that instruction cache hit cycle times are overlapped with the CPU execution time and do not contribute to the total memory reference cycle times, the proof is similar to that given for Theorem 1. □

It is noted that when I-cache is not considered, i.e., $\omega = \omega' = 0$, the ratio r is the same as in Theorem 2 and Theorem 3, respectively. For most applications have small ω (i.e., small fraction of instruction reference misses), the results of Theorem 2 and Theorem 3 can be applied to composite hit ratio with negligible difference from that determined by (25).

6.2 Trade-Off of Data Path and Hit Ratio in Burst Read Memories

In this section, we show that the trade-off methodology can be easily used to examine the performance trade-off between data path and hit ratio based on a burst read memory model instead of the nonpipelined memory model used previously.

A general form of memory cycles for the burst memory system reading a line can be represented by $c + \beta(\frac{L}{D})$ as in [2] where c is the constant time and β is the per-bus transfer cycles. The first data request of refilling a line takes $c + \beta$ processor cycles while each of the subsequent requests for the rest of the line takes β processor cycles. To use this memory model, (2) and (3) are rewritten as follows.

$$X_D = (E - \frac{R}{L}) + \frac{R(1+\alpha)}{L}(c + \beta\frac{L}{D}) \quad (26)$$

$$X_{2D} = (E - \frac{R'}{L}) + \frac{R'(1+\alpha')}{L}(c + \beta\frac{L}{2D}) \quad (27)$$

The above execution time assumes the use of a full-blocking cache and both the D and $2D$ system have the same line size and the same constant time for the first access. Again, solving (26) = (27), we have

$$r = \frac{R'}{R} = \frac{(1+\alpha)\left(c + \beta\frac{L}{D}\right) - 1}{(1+\alpha')\left(c + \beta\frac{L}{2D}\right) - 1}. \quad (28)$$

As before, the hit ratio that trades the performance of a D-byte bus is described as (7) with the r specified by (28) and its bound in a transfer time dominated memory design is found as follows.

THEOREM 4. *If system A and B both use a burst read memory with $\beta \gg c$ and full-stalling write-allocate caches with the same line size ($L \geq 2D$), and have the same copy-back ratio, then system A with a design target hit ratio HR_1 and D-byte data bus has the same performance as obtained from system B which uses a 2D-byte data bus and uses a cache with a hit ratio of $2HR_1 - 1$.*

PROOF. Applying L'Hospital's rule in (28) for a relatively large β ($\beta \gg c$), we find $r = \frac{R'}{R} = 2$ and $HR_2 = 2HR_1 - 1$ from (7). □

Fig. 8 illustrates the impact of the constant time on the data path and hit ratio trade-off. As shown, a larger constant time reduces the amount of hit ratio which is traded with the performance of bus width. As the per-bus transfer time (β) is increased on the x axis, the amount of hit ratio traded for the performance of a D-byte data path is increased.

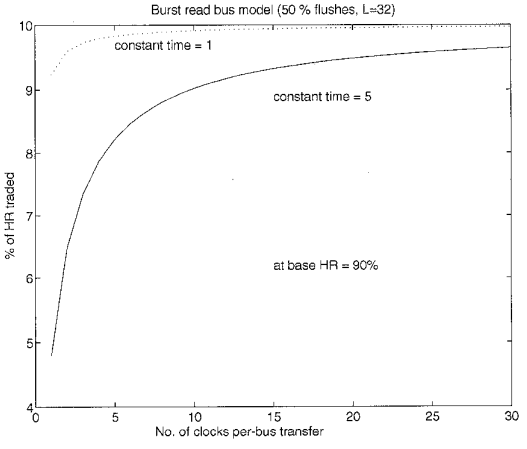


Fig. 8. Data path and hit ratio trade-off based on burst read memories.

7 CONCLUSION

We presented an architecture trade-off methodology and investigated the performance trade-offs among external data bus width, cache hit ratio, processor stalling features, read-bypassing write buffers, and pipelined memory systems. We used the trade-off methodology to determine the optimal line size and verify the approach. We showed that the performance trade-offs are based on the equivalence of

mean memory delay time and can be generalized for composite hit ratio and burst read memories. The performance and cost trade-offs examined are listed as follows:

- Given $L \geq 2D$ and $\alpha = \alpha' = 0.5$, the performance improvement due to increasing the cache hit ratio at HR by $0.5(1 - HR)$ to at most $0.6(1 - HR)$ is the same as that obtained by doubling the data bus width using nonpipelined memories. To consider trade-offs between cost and performance, increasing the size of a small on-chip data cache can easily achieve the performance level of doubling the data bus width. In this case, a relatively smaller amount of chip area is increased in the cache memory to trade for the processor pin count and memory data bus width. However, as the hit ratio and cache size curve flattens out, then increasing the bus width is more advantageous for trading the chip area because bus width trades a larger cache size (die area) for a large cache.
- Except for the pipelined memory system, the best choice to improve the performance of a system using nonpipelined memories doubling the bus width. Using read-bypassing write buffers is the second, while the use of a cache with a bus-not-locked feature is the third. In systems that have already used burst read memory design, the attempt in doubling the data path should be accompanied with the reducing of the constant time. Otherwise, the performance improvement of doubling the data path can be quite limited in constant time dominated systems.
- The study for various processor stalling features showed that a cache allowing other cache lines to be accessed while a missing line is being filled has a very limited performance advantage. However, if subsequent load/store accesses are only stalled by the latency for the requested data to arrive, about 20-30% reduction in the read miss latency of a full-blocking cache can be achieved for a memory cycle time of less than 15 clock cycles.

The pipelined memory system is most advantageous for performance when the memory cycle time is larger than about five or six clock cycles (for $L/D > 2$ and $q = 2$). Doubling the bus width, using read-bypassing write buffers, or using caches with a bus-not-locked feature has a limited performance contribution when a relatively long memory cycle latency presents. For the huge amount of hit ratio that a pipelined memory can trade, it appears that the pipelined mechanism or the burst read memory design is the most cost-effective technique to use.

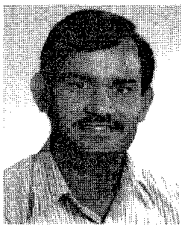
Acknowledgments

This work was supported in part by the U.S. National Science Foundation under grant MIP-9224462 and in part by the funding of NSC-83-0408-E2224-007, Taiwan, Republic of China.

References

[1] J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffics," *Proc. 10th Int'l Symp. Computer Architecture*, pp. 124-131, 1983.
[2] A.J. Smith, "Line (Block) Size Choice for CPU Cache Memories," *IEEE Trans. Computers*, vol. 36, no. 9, pp. 1,063-1,075, Sept. 1987.
[3] A.J. Smith, "Cache Memories," *Computing Surveys*, vol. 14, no. 3, pp. 473-530, Sept. 1982
[4] S. Przybylski, M. Horowitz, and J. Hennessy, "Performance Trade-Offs in Cache Design," *Proc. 15th Int'l Symp. Computer Architecture*, pp. 290-298, May 1988.
[5] S. Przybylski, "Performance Impact of Block Sizes and Fetch Strategies," *Proc. 17th Int'l Symp. 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, Aug. 1988.
[7] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. 17th Int'l Symp. Computer Architecture*, pp. 364-373, June 1990.
[8] W.Y. Chen, S.A. Mahlke, and W.W. Hwu, "Tolerating First Level Memory Access Latency in High-Performance Systems," *Proc. Int'l Conf. Parallel Processing*, pp. I36-I43, Aug. 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 Eng., Univ. of Washington, June 1992.
[10] A.J. Smith, "Second Bibliography on Cache Memories," *Computer Architecture News*, vol. 19, no. 4, pp. 154-182, June 1991.
[11] A.J. Smith, "Cache Evaluation and Impact of Workload Choice," *Proc. 12th Int'l Symp. Computer Architecture*, pp. 64-73, June 1985.
[12] *KSR1 Technical Summary*, Kendall Square Research, 1992.
[13] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, p. 458. Morgan Kaufmann Publishers, 1990.
[14] R.T. Short and H.M. Levy, "A Simulation Study of Two-Level Caches," *Proc. 15th Int'l Symp. Computer Architecture*, pp. 81-88, 1988.

Chung-Ho Chen graduated with honors from the National Taipei Institute of Technology in 1983 and received the MS degree from the University of Missouri at Rolla in 1989 and the PhD degree from the University of Washington at Seattle in 1993, all in electrical engineering. He is currently an associate professor of electronic engineering at the National Yunlin Institute of Technology, Touliu, Taiwan. His current research interests are high-performance processor architectures and implementation of multiprocessor systems.



Arun K. Somani earned his MSEE and PhD degrees in electrical engineering from McGill University, Montreal, Canada, in 1983 and 1985, respectively. Prior to that, he worked as a scientific officer for the government of India in New Delhi from 1974 to 1982. During this period, he designed and developed an anti-submarine warfare system for the Indian Navy. Dr. Somani is currently a professor of electrical engineering and computer science and engineering at the University of Washington at Seattle.

Prof. Somani's research interests are in the areas of fault-tolerant computing, interconnection networks, computer architecture, parallel computer systems, and parallel algorithms. He is currently involved in three major projects: 1) high integrity system design addressing the issue related to cache memory design in redundant computer systems and evaluation tools for such systems, 2) congestion control and fault tolerance in broadband networks, and 3) development of "Proteus" architecture, a multiprocessor system for automated classification of objects based on generalized enhanced hypercube reconfigurable interconnection network exploring coarse grain parallelism.