AREA AND PERFORMANCE ANALYSIS OF PROCESSOR CONFIGURATIONS WITH SCALING OF TECHNOLOGY

Steve Fu and Michael Flynn

Technical Report: CSL-TR-94-605

March 1994

This work was supported by NSF under contract MIP88-22961, using facilities supported by NASA contract NAG2-842.

AREA AND PERFORMANCE ANALYSIS OF PROCESSOR CONFIGURATIONS WITH SCALING OF TECHNOLOGY

by

Steve Fu and Michael Flynn

Technical Report: CSL-TR-94-605

March 1994

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

Abstract

As integrated circuit density increases, computer architects face the interesting problem of how best to utilize the available die size given cost and performance constraints. Traditionally, area partitioning and floor-planning have been done in an ad hoc fashion based on intuition and experience of the designers. This paper proposes a systematic methodology for correlating area and performance as the designer increases the transistor count of a given sub-unit. Specifically, we investigate the performance of three possible processor configurations, and present performance results as the mimimun feature size is reduced.

Key Words and Phrases: cache, technology scaling, bus traffic, Area modeling, superscalar, multiprocessor

Contents

List of Figures

List of Tables

1 Introduction

Advances in technology have allowed recent microprocessors to have more than 3 million transistors, operating frequencies of 200MHz, and over 500 pins on a single chip. The architect faces the complex problem of how to balance performance against complexity and design time. Since the life cycle of a microprocessor is about two years, the architect must examine ways to use the advances in IC technology and design a microprocessor that fits the needs of the target market in terms of performance, power consumption, and cost.

The importance of die size with regard to fab costs [1], design cost, and power consumption has been determined empirically through statistics. The design cost of a microprocessor has increased drastically with density, and power consumption has scaled up with *die size* $*$ *clock frequency*. With recent industrial design examples, it is obvious that early partitioning of die area, cycle time, and power budget leads to a successful microprocessor design. Traditionally, tools for evaluating performance have evaluated the TIME resource only. Time aspects include such parameters such as latency and throughput of operations. This paper proposes techniques to correlate performance to another key resource in the design space: AREA.

Another growing problem is the ever-widening gap between CPU cycle time and main memory access time. This paper proposes an analytical approach to incorporate the effect of bus contention and bus saturation in the overall performance of the processor. Different cache management and memory interface schemes are analyzed with respect to memory traffic and resulting CPI.

In this paper, three processor configurations with different instruction and data bandwidth requirements are mapped to a technology-independent area model. Under the constraints of fixed cycle time and die area, an optimal cache and memory interface design is selected for each processor configuration. Repeating this sequence for the feature sizes of interest, an evolution path in terms of processor configuration and performance is derived.

2 Processor Specifications

The instruction set is to remain consistent across the three configurations. The configurations also share common modules with the only modifications allowed being the data path widths and controls. The three configurations are:

1. Single issue processor with a branch adder and expanded cache. The high level block diagram is shown in Figure 1, and the processor organization is shown in Figure 2.

2. Superscalar processor that inspects 4 instructions and issues up to 2 instructions every cycle.

3. Two baseline processors, each possessing its own Icache and sharing one Dcache. The high level block diagram is shown in Figure 3.

An optimal cache and memory organization for each configuration is selected given the area constraint, and a 256KB off-chip secondary cache is optional.

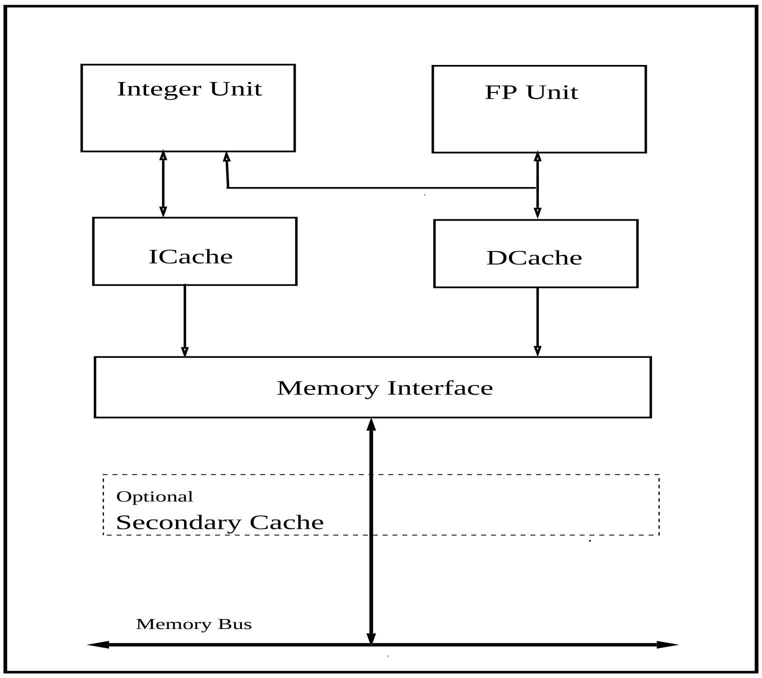

Figure 1: Baseline Block Diagram

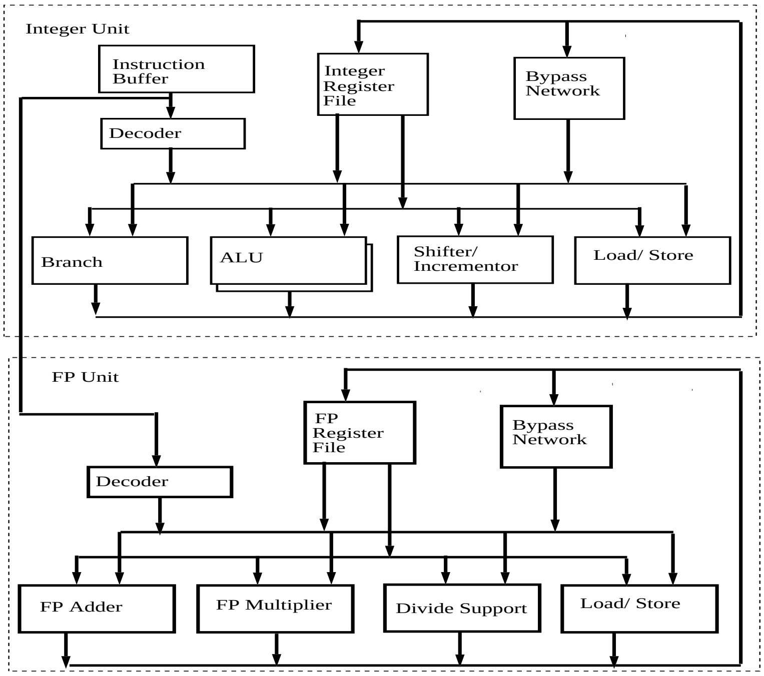

Figure 2: Processor Organization

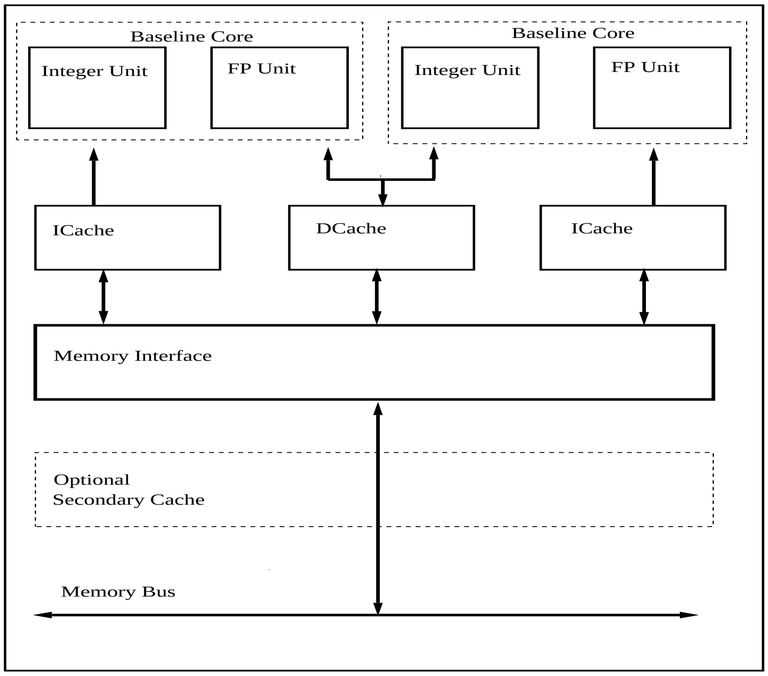

Figure 3: MP Block Diagram

Table 1: Pipeline Specifications

Instr. Type				
ALU	IF	D/RF	EX	PA
LD/ST	IF	D/RF	AG	PA
BR/BC	IF	D/AG	TIF	

Table 2: Given Conditions

Area	
Total Chip Area	230mm^2
I/O Pad Overhead	20%
Latch Overhead	10% of Integer and FP units
Bus Overhead	40% of Integer and FP units
Aspect Ratio Mismatch Overhead	10% of cache area
Base CPI	
Implementation Type	Base CPI
Baseline Processor	1.47
Superscalar Processor	0.833
Multiprocessor Processor	0.795
Reference Per Instruction	
Instruction Reference	1
Data Read Reference	0.33
Data Write Reference	0.235
Timing Specification	
Cycle time	8 ns
Memory T_{access}	96 ns
Memory T_{pagemode}	16 ns
Memory T_{bus}	8 ns
On-chip Cache Access	8 ns
L2 Cache T_{access}	24 ns
Memory Specifications	
Memory Size	64MB
Types	4M x 8b
Fast Sequential Page Mode	Available
Interleaving	2 or 4
TLB Specifications	
Number of Entries	128
Associativity	2-way
Page Size	4KB
TLB Miss Rate	0.0065
TLB Miss Penalty	20 cycles
Secondary Cache Specifications	
Size	256KB or 4x First Level Cache Size
Organization	Direct-mapped

Superscalar Specifications

- 2 Integer ALUs
- Pipelined FP Unit
- 1 Branch Adder
- 1 L/S Unit
- Register File with 2 Read and 2 Read/Write Ports
- Additional Bypassing Muxes
- Doubled Instruction Buffer Size
- Doubled Decoder Hardware Size

Dual Issue Combinations

- 2 Integer ALU Ops
- 1 FP and 1 Integer Op
- 1 FP Multiply and 1 FP Addition
- 1 Branch
- 1 L/S

MP Specifications

- Two Baseline Processors
- Two I Caches
- Two I TLBs
- One Shared D Cache
- One Shared D TLB

Assumptions

- For copyback cache, assume a dirty line ratio of 50% (w).
- For the MP shared data cache, the cache miss ratio is based on a multiprogram level of two and a task switch quantum of 100. This is intended to account for cache collision effects.
- Secondary cache implementations increase the cost of the system by 20%.

- The processor's intended environment is scientific.
- Physical address space is 64MB.
- Each cache tag includes an additional 4-bit control field (for cache coherency and line replacement purposes).
- Two memory interface widths to choose from for all possible implementations due to pin limitations. Either 32b or 64b physical words can be utilized, although the 64b implementation is assumed to add 5% to system cost.

3 Methodology

The methodology can be divided into six steps:

- Map each processor configuration to a technology independent area model of processor units.
- Explore all possible cache configurations given remaining area.
- Instruction and write buffer design.
- Cache design.
- Memory design.
- Evaluate performance of each configuration.

The above steps are applied to each processor configuration, and repeated for minimum feature sizes of 1μ, 0.75μ, 0.6μ, 0.5μ, 0.4μ, and 0.3μ.

4 Area Modeling

The area model used in this paper is based on the work done by Mulder [3]. *Register bit equivalent* (**rbe**) is the basic unit of area. An **rbe** is a six-transistor static cell with input/output isolation, and has been determined to be about 2700 λ^2. (λ is the fundamental resolution unit proposed by Mead and Conway, and f is the actual feature size in μ.) In this paper, static ram cells are used for cache storage bits, and these are estimated to be 0.6 **rbe**. A larger area unit "A" is used throughout this paper. This is the size of a 3-ported 32 entry register file.

In Table 4, each module is shown with its corresponding area utilization in "A," where "A" is defined as 1481 **rbe** and each **rbe** is 675 f^2. The various units that belong in the baseline processor are summed up. After obtaining the total area for both integer and FP unit, the overhead for both latches and busses for communication is added. These are 10% and 40% of the units, respectively. It is given that the die area is 230 mm^2. However, to account for peripheral area such as IO Pads, 20% of the die area is subtracted, which gives

Table 3: Given Area Specifications

Feature Size	0.75μ
Defect Density	1 per cm^2
Yield	10%
Control Ovrhd	20%
Aspect Ratio Ovrhd	10%
Latch Ovrhd	10%
Bus Ovrhd	40%

us 184 mm^2. To convert the available area from mm^2 to units of A, the following formula is used:

$$\text{Area}(A) = (\text{area})\ \text{mm}^2/(1481 * 675 * f * f * 0.001 * 0.001).$$

Subtracting the baseline area and deducting 10% for aspect mismatch gives the area available for cache + directory. A similar analysis is used for superscalar and MP implementations of the chip. The basic parameters are given in Table 3. A sample calculation is shown in Table 4 for the case of a baseline processor with 0.75 μ feature size. Similar analysis can be done with superscalar and MP, but is not shown for brevity. The total area without cache is summarized in Table 5.

5 Cache Configuration Exploration

Tag Memory Modeling

A physical address is broken into the following:

Tag	Index	Offset

Direct Map Cache

$$\begin{aligned} \text{Tag bits} &= \log(\text{Mem size/Cache size}) \\ \text{Index bits} &= \log(\text{Number of lines}) \\ \text{Offset bits} &= \log(\text{Number of bytes/ line}) \end{aligned}$$

Associative Cache

$$\begin{aligned} \text{Tag bits} &= \log(\text{Mem size/Cache size}) + \log(\text{Degree of associativity}) \\ \text{Index bits} &= \log(\text{Number of lines/Degree of associativity}) \\ \text{Offset bits} &= \log(\text{Number of bytes/line}) \end{aligned}$$

Table 4: Baseline Processor Area Calculation

Baseline Area		Conversion To KB	
Integer ALU	1.00A	Die Area	$230mm^2$
Integer register	1.00A	Area Available	$180mm^2$
Shifter	0.50A	Area Available in A	335.11A
Incrementor	0.40A	$-$ Total Baseline	82.35A
PC unit	1.00A	Remaining Area	335.11A
2 TLBs	6.00A	-10% Aspect Ratio Mismatch	25.28A
Decode/Control	1.00A	Total Cache Area	227.48A
Cache Controller	1.00A	Cache Bits	561116.96b
Bus Logic	2.00A	Cache KB	68.5KB
Stored Buffer+Bypass	1.00A		
Load/store	0.20A		
Clock generator	1.00A		
Branch Adder	1.00A		
Total Integer Unit	17.10A		
FP RegFile	1.00A		
FP Adder	13.5A		
FP Multiplier	20.30A		
Divide Support	3.00A		
Total FP Unit	37.80A		
Latch Ovhd	5.49A		
Bus Ovhd	21.96A		
Integer Unit	17.10A		
Total Baseline	82.35A		

Table 5: Area Summary

Baseline	82.35A
Superscalar	105.93A
MP	160.20A

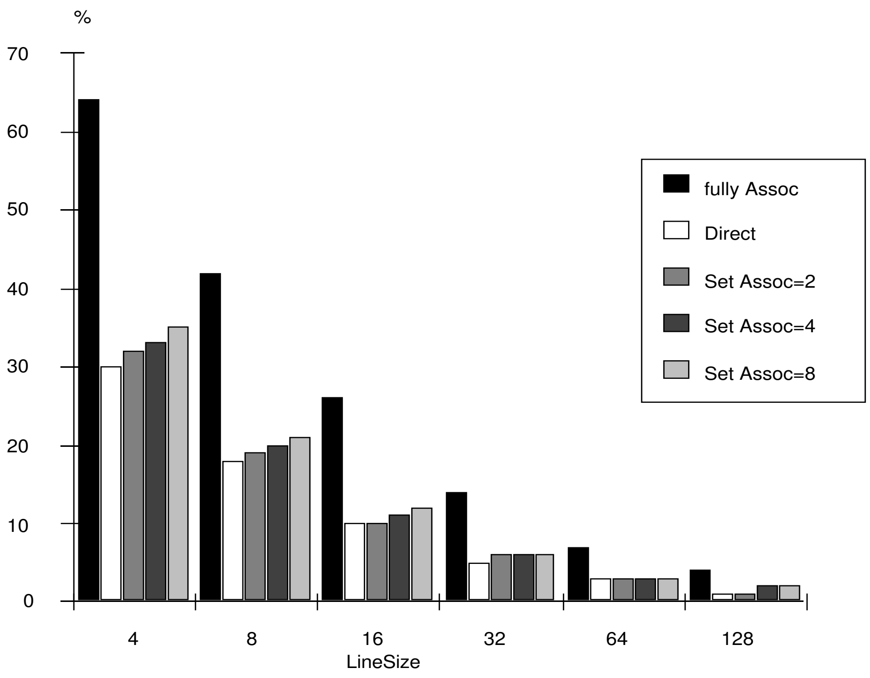

Figure 4: Percent total cache area used for cache tags.

Fully Associative Cache

$$\begin{aligned} \text{Tag bits} &= \log(\text{Mem size}/\text{Cache size}) + \log(\text{Number of lines}) \\ \text{Index bits} &= 0 \\ \text{Offset bits} &= \log(\text{Number of bytes/line}) \end{aligned}$$

For tag memory, four bits are added as control bits. In addition, the direct-mapped cache requires only one comparator, while the set-associative requires as many comparators as its degree of associativity. The fully-associative cache requires a content-addressable memory (CAM) as its directory, since every line is a candidate. These memory bits are modeled as 2 **rbe** each.

Figure 4 shows the tag memory area for different configurations and line sizes. The fully-associative cache is prohibitively expensive for small line sizes. Given a certain target data cache size, the smaller line size implementations simply do not fit in the given area. Set associativity of 8 gives us the greatest reduction in cache miss ratio, yet provides a relatively low tag memory overhead. Increasing line sizes has the greatest effect in reducing tag memory overhead.

After calculating the area available for cache, we proceed with the selection of the cache configuration that results in the optimal CPI. One of the problems is that the area may not fall squarely into $2n$KB boundaries, where n is a positive integer. Industry implementations use different associativities to accommodate this problem, but this greatly increases the

Table 6: Cache Configurations(KBytes)

Feature Size	Baseline	Superscalar	MP
1.0u	24-U 16I/8D 8I/16D	16-U 8I/8D	2I/2I/2D
0.75μ	64K-U 32I/32D	48K-U 16I/32D 32I/16D	8I/8I/16D 16I/16I/8D
0.6μ	64I/48D 48I/64D	64I/32D 32I/64D	32D/32I/32I 48D/16I/16I
0.5μ	96I/64D 128I/32D	128I/32D 96I/64D	48D/48I/48I 64D/32I/32I
0.4μ	256-U 128I/128D	256-U 128I/128D	128D/64I/64I
0.3μ	512-U 256I/256D	512-U 256I/256D	256D/128I/128I

number of configurations that we must look at. An example is the SuperSparc with its 20KB Icache and a associativity of 5.

The effect of non-binary associativity is better utilization of the available area, but it should have only a second-order effect on CPI. The cache configurations that were considered are shown in Table 6.

6 Cache Design

After examining the possible cache sizes with respect to area constraints, we can consider other aspects of on-chip cache design. The cache miss data is based on a series of design target miss rates (DTMR) from the work of Smith [6]. In comparisons with SPEC92 cache performance data produced by Hill [4], DTMR tends to be a more conservative estimate of cache performance.

The following aspects of cache design were explored:

- Unified versus Split.
- Line sizes and associativity.
- Traffic generated by different cache management schemes.
- Different buffer management and memory interface schemes.

6.1 Unified versus Split

Given a fixed area to implement the cache, we first qualitatively consider whether to use the unified or the split cache scheme. Unified cache offers a lower overall miss rate, but we must consider the amount of contention delay due to conflicting data and instruction accesses. A split cache offers higher bandwidth, since separate data paths exist for instruction and data accesses. However, we must provide the additional area required for the data paths and a higher overall miss rate. As a rough approximation, we assume that for a split cache 10% of the cache area is utilized by data paths and overhead.

In calculating the contention, one needs to take the actual CPI into consideration. Many of the instruction requests are from branches that degrade the CPI of the processor. Only in-line reference stream and executed instructions need to be considered, since an instruction that is not executed cannot result in contention. We can model the contention using probabilistic models. Furthermore, the analysis is divided into Copy Back Write Allocate (CBWA) and Write Through No Write Allocate (WTNWA) cases. The abbreviations used are Instruction (I), Instruction Fetches (IF) , Data Fetches (DF), and Data Stores (DS).

CBWA

For CBWA, the data traffic needs to include both data fetch and data store, since, unlike WTNWA, both access the cache.

$$\begin{aligned}\text{Prob (Contention/I)} &= \text{(Inline IF/I)} * (\frac{DF + DS}{I}) \\ &= (0.5) * (0.235 + 0.33) = 0.283\end{aligned}$$

Since the on-chip cache can be accessed every cycle, we should take CPI into account and calculate the probability of contention per cycle:

$$\text{Prob (Contention/Cycle)} = \frac{\text{(Inline IF/I)} * (\frac{DF+DS}{I})}{\text{CPI}^2} = 0.131.$$

Assuming the penalty due to contention is one cycle, the additional CPI is 0.131

WTNWA

$$\begin{aligned}\text{Prob (Contention/Cycle)} &= \frac{\text{(Inline IF/I)} * \text{(DF/I)}}{CPI^2} = 0.0764 \\ \text{Additional CPI} &= 0.0764\end{aligned}$$

Based on the preceding penalty estimation, we can compare the performance lost due to contention versus the performance lost due to smaller split caches and higher miss rates.

6.2 Line Size and Associativity

Line size is the unit of transfer between cache and memory. The selection of line size is critical for processor performance. Altering the line size affects the following:

- Cache Miss Ratio.
- Memory Traffic.
- Tag Memory Size for cache directory.

Increasing or decreasing the line size is a decision based on balancing cache miss ratio, memory traffic, and tag memory size. Increasing the line size decreases the miss rate due to the enhanced spatial locality of the cache (sequential address request). However, the decrease in miss ratio becomes smaller as we increase the line size further, since the cache is losing its temporal locality (fewer lines). The optimal point is reached when the effect of decreasing the miss ratio no longer compensates for the increase in T_{line}. Increasing the line size decreases the number of lines in the cache, thus decreasing the overhead of tag memory associated with each line. With small caches, if we increase the line size too much, we hurt the temporal locality of the cache, since we tend to replace lines too often due to the small number of lines available.

Since the focus of this paper is *AREA*, not *TIME*, we assume that the cache implementation is not the critical path of the processor. Empirical data of recent industry implementations indicates that 8ns is sufficient for accessing most cache configurations. With this in mind, we assume that increasing the associativity does not penalize cycle time. What we can consider is the area cost due to associativity, along with the cost of comparators. We consider the cases of direct mapped, 2,4,8-way set associative, and fully associative. More associativity reduces the miss rate, but also results in larger tag memory comparator and muxing cost.

6.3 Traffic Analysis

To provide a more quantitative view of memory traffic differences between different cache management schemes, simple models are presented for four different schemes. The following parameters are utilized in these models :

- T_{line} is the time required to complete a line access. It is dependent on the line size (L), the memory interleaving factor (M), the memory bus width, the sequential page mode access time (T_{pagemode}), and the bus cycle time (T_{bus}):

$$T_{\text{line}} = T_{\text{access}} + T_{\text{pagemode}} \lceil L/M - 1 \rceil + T_{\text{bus}}((L-1) \bmod M).$$

- T_{word} is the word access time(T_{access}).

- Cache Miss Ratio (MR).

- % of total lines that are dirty (W).

- Reference Traffic Patterns.

The schemes considered are:

- Write Through No Write Allocate (**WTNWA**)
- Copy Back Write Allocate (**CBWA**)
- Write Through with a Write Assembly Cache (**WAC**)
- Write Through with CBWA Secondary Cache (**WTWSC**)

WTNWA

$$\begin{aligned}
\text{Line Accesses/I} &= \text{IF/I} * (\text{ICache.MR}) + \text{DF/I} * (\text{DCache.MR}) \\
\text{Word Access/I} &= \text{DS/I} \\
\text{Total Bus Time/I} &= \text{DF/I} * \text{Dcache.MR} * \text{T}_{\text{line}} + \text{DS/I} \\
&\quad * \text{T}_{\text{access}} + \text{IF/I} * \text{ICache.MR} * \text{T}_{\text{line}} \\
\text{Total Offered Occupancy} &= (\text{Total Bus Time/I})/\text{Cycle time}
\end{aligned}$$

CBWA

$$\begin{aligned}
\text{Line Accesses/I} &= \text{IF/I} * (\text{ICache.MR}) + (\text{DF} + \text{DS})/\text{I} \\
&\quad * (1 + \text{W}) * (\text{DCache.MR}) \\
\text{Word Access/cycle} &= 0 \\
\text{Total Bus Time/I} &= \text{LineAccesses/I} * \text{T}_{\text{line}} \\
\text{Total Offered Occupancy} &= (\text{Total Bus Time/I})/\text{Cycle time}
\end{aligned}$$

WAC

$$\begin{aligned}
\text{Line Accesses/I} &= \text{IF/I} * (\text{ICache.MR}) + \text{DF/I} * (1 + \text{W}) \\
&\quad * (\text{DCache.MR}) + \text{DS/I} * (1 + \text{W}) * (\text{WAC.MR}) \\
\text{Word Access/cycle} &= 0 \\
\text{Total Bus Time/I} &= \text{IF/I} * (\text{ICache.MR}) * \text{T}_{\text{line}} \\
&\quad + \text{DF/I} * (1 + \text{W}) * (\text{DCache.MR}) * \text{T}_{\text{line}} \\
&\quad + \text{DS/I} * (\text{WAC.MR}) * \text{T}_{\text{line.wac}} \\
\text{Total Offered Occupancy} &= (\text{Total Bus Time/I})/\text{Cycle time}
\end{aligned}$$

WTWSC

The secondary cache is assumed to be CBWA, and has a fixed line size of 128 bytes. The sizes of the secondary cache vary across different feature sizes from a minimum of 256KB to 2MB in order to maintain a reasonable size ratio between the first-level cache and the secondary cache. The first-level cache associated with the secondary cache is WTNWA.

$$
\begin{aligned}
\text{Line Accesses/I} &= \text{IF/I} * \text{Secondary Cache.MR} \\
&\quad +(\text{DF}+\text{DS})/\text{I} * (1+\text{W}) \\
&\quad *(\text{Secondary Cache.MR}) \\
\text{Word Access/cycle} &= 0 \\
\text{Total Bus Time/I} &= \text{Line Accesses/I} * T_{\text{line}}(128B) \\
\text{Total Offered Occupancy} &= (\text{Total Bus Time/I})/\text{Cycle time}
\end{aligned}
$$

6.4 Write Buffer and Memory Interface Schemes

Different combinations of CBWA and WTNWA and different buffer management schemes have different implications for the overall CPI. For the analysis below, a "perfect write buffer" is assumed, since the effect of write traffic is analyzed separately. With this assumption, no write buffer contention can occur.

Before proceeding, a few terms need to be introduced:

$$
\begin{aligned}
T_{\text{m.miss}} &= \text{Time memory is busy due to a read request.} \\
T_{\text{c.miss}} &= \text{Time processor is stalled due to a read miss.} \\
T_{\text{write.mem}} &= \text{Time memory is busy due to a write miss.} \\
T_{\text{write.c}} &= \text{Time processor is stalled due to a write miss.} \\
T_{\text{busy}}(\text{Read}) &= T_{\text{m.miss}} - T_{\text{c.miss}}.
\end{aligned}
$$

With the *WTNWA* configuration, we consider two buffer management schemes.

Scheme 1: In a read miss, the entire line must be fetched in prior to the resumption of processing.

Scheme 2: In a read miss, wraparound is used so that the first word is returned first, and processing continues while memory finishes loading the rest of the line into cache.

For *scheme one:*

$$
\begin{aligned}
T_{\text{m.miss}} &= T_{\text{line}} \\
T_{\text{c.miss}} &= T_{\text{line}} \\
T_{\text{write.mem}} &= \max(T_{\text{access}}, T_{\text{cycle}}) \\
T_{\text{write.c}} &= 0. \\
T_{\text{busy}}(\text{Read}) &= 0.
\end{aligned}
$$

Since $T_{\text{busy}}(\text{Read}) = 0$, no memory contention can occur due to a read blocking another read. However, a write that is being serviced can block a read request, and is accounted for in $T_{\text{w.interference}}$.

$$T_{\text{w.interference}} = (1 - (1 - (\text{IF+DF})/\text{I})^{\frac{T_{\text{write.mem}}}{\text{cycle time}}})(DS/I)(T_{\text{write.mem}}/2).$$

The first term, $(1-(1-(\text{IF+DF/I})^{T_{write.mem/cycletime}})$, is the probability of a read request during the number of cycles it takes to process the write request. The second term, (DS/I), is the probability of a write request during any cycle. The third term, $(T_{write.mem/2})$, is the average waiting time of a read request.

After including the write interference time, the time to process a read and the resulting CPI are:

$$\begin{aligned}
T_{\text{c.miss}} &= T_{\text{line}} + T_{\text{w.interference}} \\
\text{CPI} &= \text{CPI}_{\text{base}} + (\text{DF/I} * \text{DCache.MR} * \text{T}_{\text{c.miss}})/\text{Cycle time} \\
&\quad +(\text{IF/I} * \text{ICache.MR} * \text{T}_{\text{c.miss}})\ /\text{Cycle time.}
\end{aligned}$$

For *scheme two:*

$$\begin{aligned}
T_{\text{m.miss}} &= T_{\text{line}} \\
T_{\text{c.miss}} &= T_{\text{access}} \\
T_{\text{write.mem}} &= \max(T_{\text{access}}, T_{\text{cycle}}) \\
T_{\text{write.c}} &= 0
\end{aligned}$$

For this case, a read in progress can block an upcoming read, since $T_{\text{busy}} = T_{\text{line}} - T_{\text{access}}$. As a result, we must also include $T_{\text{r.interference}}$ in the final read processing time of memory. The read interference time and the resulting CPI can be approximated as follows:

$$\begin{aligned}
T_{\text{r.interference}} &= \frac{1}{2}\Big((\text{DCache.MR} * (\text{DF/I/Cycle Time}) \\
&\quad +\text{ICache.MR} * (\text{IF/I/Cycle Time})) * \text{T}_{\text{busy}}{}^2\Big)\,. \\
CPI &= \text{CPI}_{\text{base}} + (\text{DF/I} * \text{DCache.MR} + (\text{IF/I}) * \text{ICache.MR}) \\
&\quad *(\text{T}_{\text{c.miss}} + \text{T}_{\text{w.interference}} + \text{T}_{\text{r.interference}})/\text{Cycle time.}
\end{aligned}$$

For CBWA, we consider three buffer management schemes.

Scheme 1: In a read miss, the dirty line is written to memory, followed by the entire line being fetched and loaded into the cache, prior to the resumption of processing.

Scheme 2: In a read miss, the dirty line is transferred to the write buffer simultaneously with the initiation of the fetching of the line. The processor resumes when the entire line has been loaded into the cache.

Table 7: Cache/Memory Interface Options

Option 1	Option 2	Option 3
Write dirty line to memory	Fetch new line	Fetch missing word first
Fetch new line	Start processor	Start processor
Start processor	Write dirty line to buffer	Finish line fetch and write dirty line to buffer
$T_{\text{m.miss}} = (1+w)T_{\text{line}}$	$T_{\text{m.miss}} = (1+w)T_{\text{line}}$	$T_{\text{m.miss}} = (1+w)T_{\text{line}}$
$T_{\text{c.miss}} = (1+w)T_{\text{line}}$	$T_{\text{c.miss}} = T_{\text{line}}$	$T_{\text{c.miss}} = T_{\text{access}}$
$T_{\text{busy}} = 0$	$T_{\text{busy}} = wT_{\text{line}}$	$T_{\text{busy}} = (1+w)T_{\text{line}} - T_{\text{access}}$

Scheme 3: In a read miss, the dirty line is transferred to the write buffer simultaneously with the initiation of the fetching of the line. The processor resumes when the first word is returned from memory (wrap -around fetch).

Instead of repeating the analysis, the required terms are summarized in Table 7.

For *CBWA*, both read and write interferences have been incorporated into $T_{\text{interference}}$, since the $T_{\text{m.miss}}$ includes both read and write reference traffic of the cache. For all three schemes, the interference time and the resulting CPI can be approximated as follows:

$$\begin{aligned} T_{\text{interference}} &= (\text{DCache.MR} * (\text{DF/I} + \text{DS/I}) + \text{ICache.MR} * \text{IF/I}) \\ &\quad * \frac{1}{\text{Cycle time}} * \frac{\text{T}_{\text{busy}}{}^2}{2}. \end{aligned}$$

$$CPI = \text{CPI}_{\text{base}} + (\text{DCache.MR} * \text{DF/I} + \text{ICache.MR} * \text{IF/I}) * \frac{\text{T}_{\text{c.miss}} + \text{T}_{\text{interference}}}{\text{Cycle time}}.$$

7 Results

The effects of different choices in cache and memory interface design on performance are presented in the following sections. Similar analysis is carried out for all possible minimum feature sizes, but in most cases only the results from the 0.75μ case are presented, for brevity. With the goal of achieving an optimal CPI, variations in performance with respect to the following design choices are shown:

- Different write buffer management and memory interface schemes.
- Doubling the memory bus width.
- Addition of a secondary cache.
- The effect of memory bus traffic.

The last section presents the CPI versus feature size of the different configurations.

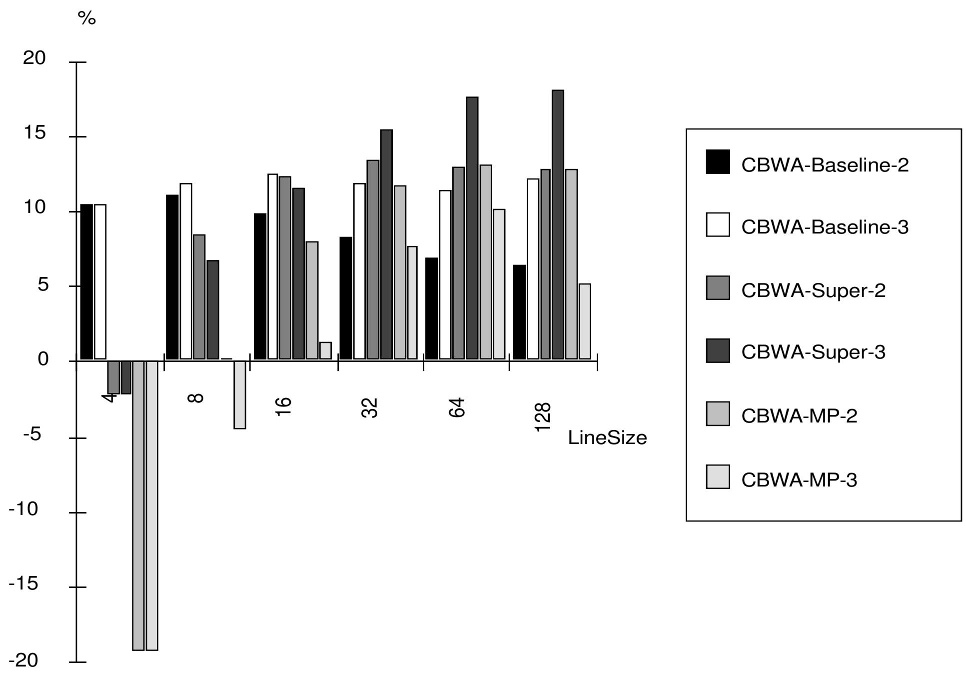

Figure 5: Percent CPI improvement of CBWA buffer management scheme 2 and 3 relative to scheme 1 for 0.75μ (Write Policy-Configuration-Buffer Management Scheme).

7.1 Write Buffer Management and Memory Interface

The CPI versus line size of each scheme is generated and summarized. Figures 5 and 6 detail the performance improvement of schemes two and three relative to buffer management scheme one. Two observations can be made based on the data:

- For the CBWA data, at low line sizes, there is no benefit in using the more complicated schemes due to the higher miss ratios and the inability to amortize the memory access time.
- For the CBWA-MP data, at higher line sizes, the benefit in using the more complicated schemes is reduced. This reduction is because the decrease in cache miss ratio cannot compensate for the increase in bus contention time.

7.2 Physical Word Size

The effects of increasing the memory bus for both CBWA and WTNWA are displayed in Figures 7–9.

These figures exhibit an interesting point. The increase in bus width brings a large return in CPI reduction for the CBWA case. but not for the WTNWA case. It is somewhat intuitive that increasing bus width reduces T_{line} ,but does nothing for T_{access}. Since WTNWA

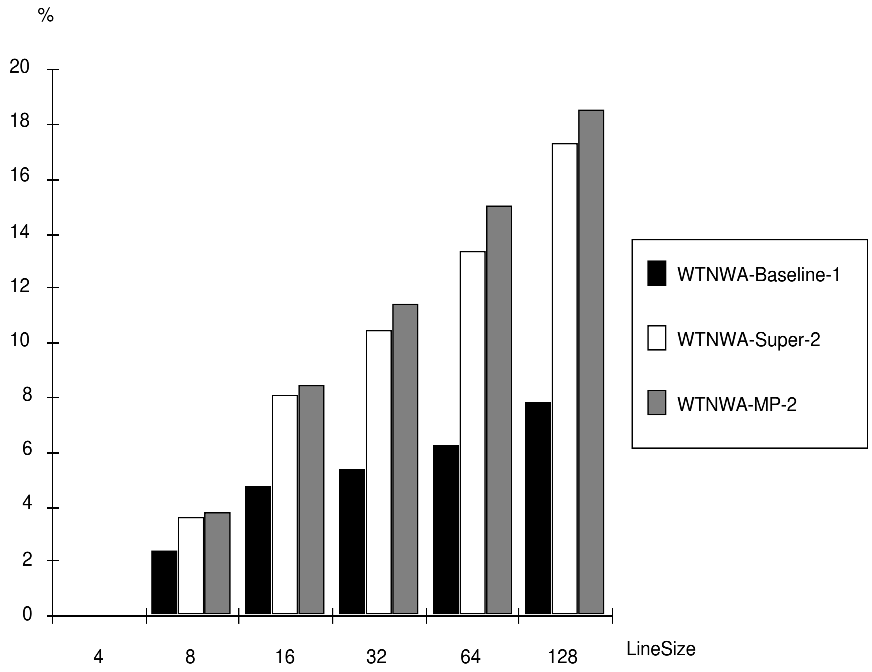

Figure 6: Percent CPI improvement of WTNWA buffer management scheme 2 relative to scheme 1 for 0.75μ (Write Policy-Configuration-Buffer Management Scheme).

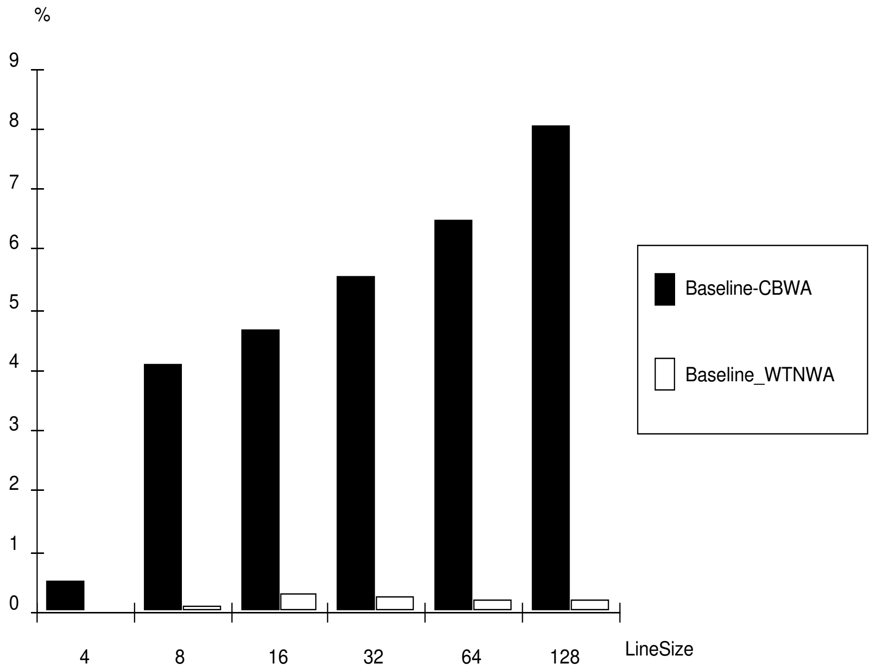

Figure 7: Percent CPI improvement of 64b memory bus relative to 32b memory bus for a 0.75μ Baseline processor.

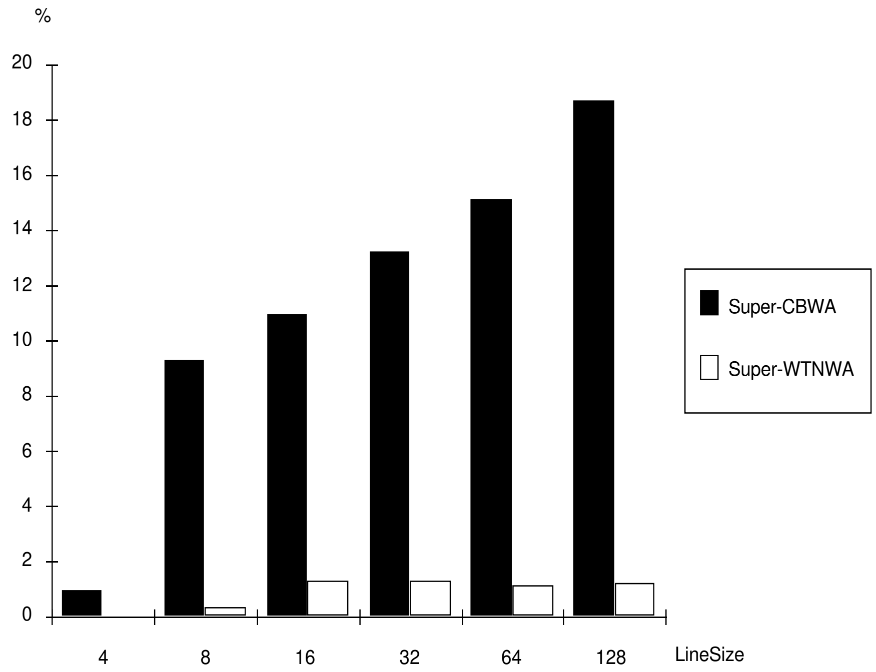

Figure 8: Percent CPI improvement of 64b memory bus relative to 32b memory bus for a 0.75μ Superscalar processor.

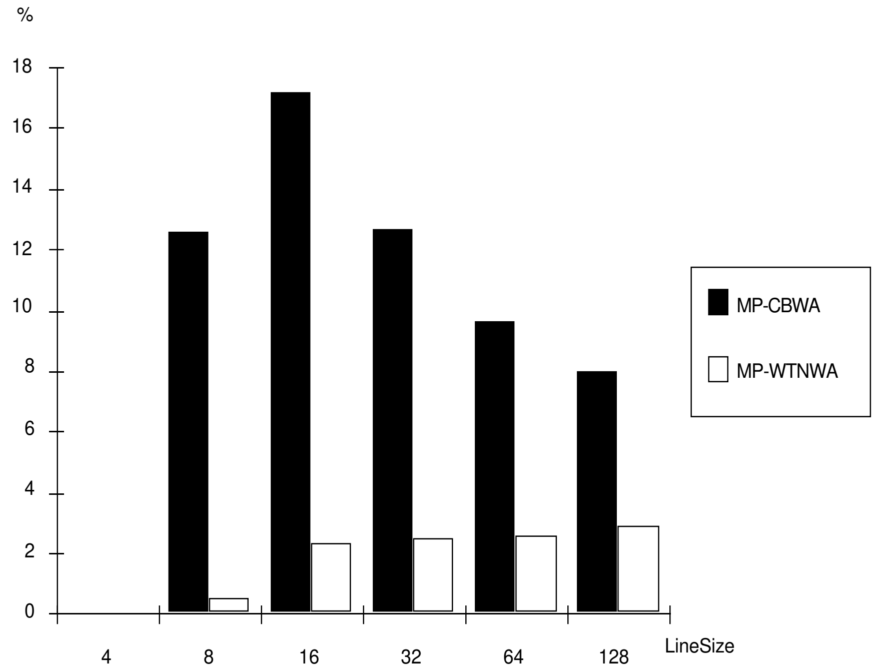

Figure 9: Percent CPI improvement of 64b memory bus relative to 32b memory bus for a 0.75μ MP processor.

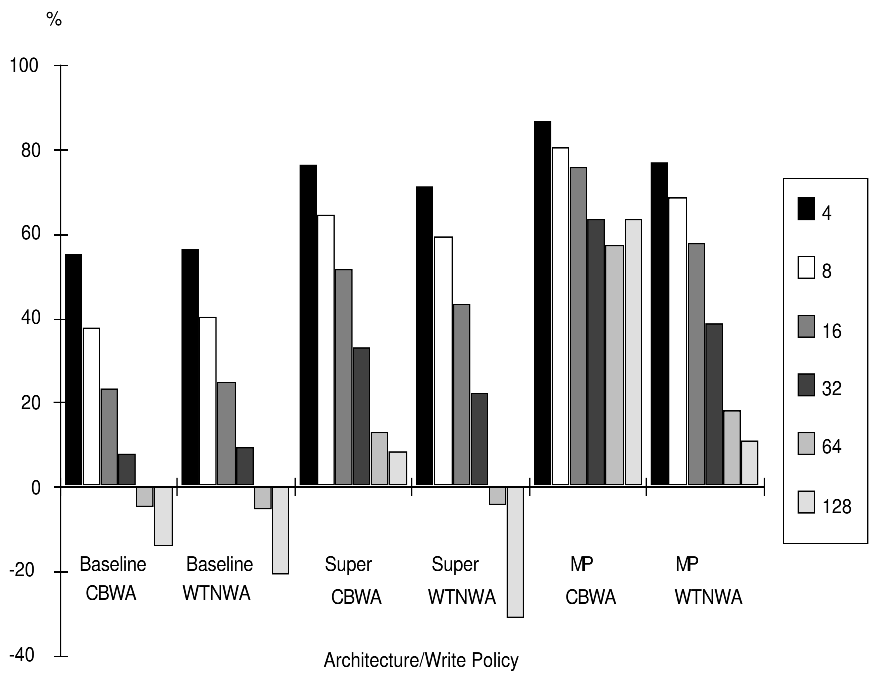

Figure 10: Percent CPI improvement of implementations with secondary cache relative to implementation without secondary cache for 1.0μ across different line sizes.

traffic is dominated by the write traffic as the miss ratio is reduced, its performance is not enhanced by a reduction in T_{line}.

Another interesting question is why the improvement reaches a peak for the MP-CBWA case at line size of 16B, and drops after that. The reduction is due to the small cache area for the MP implementation. CPI reaches a peak at smaller line sizes.

For CBWA implementation, it is worthwhile in all cases to implement a wider memory interface, since the performance benefit is greater than the 5% additional cost for most line sizes.

7.3 Secondary Cache

The effect of the addition of a secondary cache to CPI across the different line sizes, write policy, and architecture is shown for the 1.0μ and 0.3μ cases. These cases are selected in order to show the benefits of secondary cache as feature size is shrunk. The size of the secondary cache is either 256KB or a minimum of 4x the size of the first level cache.

From the data in Figure 10, it is beneficial to add a secondary cache only with shorter line sizes, since it is assumed that the secondary cache adds 20% to system cost.

Figure 10 compares the best CPI without secondary cache to the CPI implemented with a secondary cache. The negative improvement with longer line sizes is due to the fact that the implementation with secondary cache uses a simple write buffer management scheme. As a result, for the CBWA case, $T_{\text{c.miss}} = (1 + w)T_{\text{line.128}}$ (we first need to write back

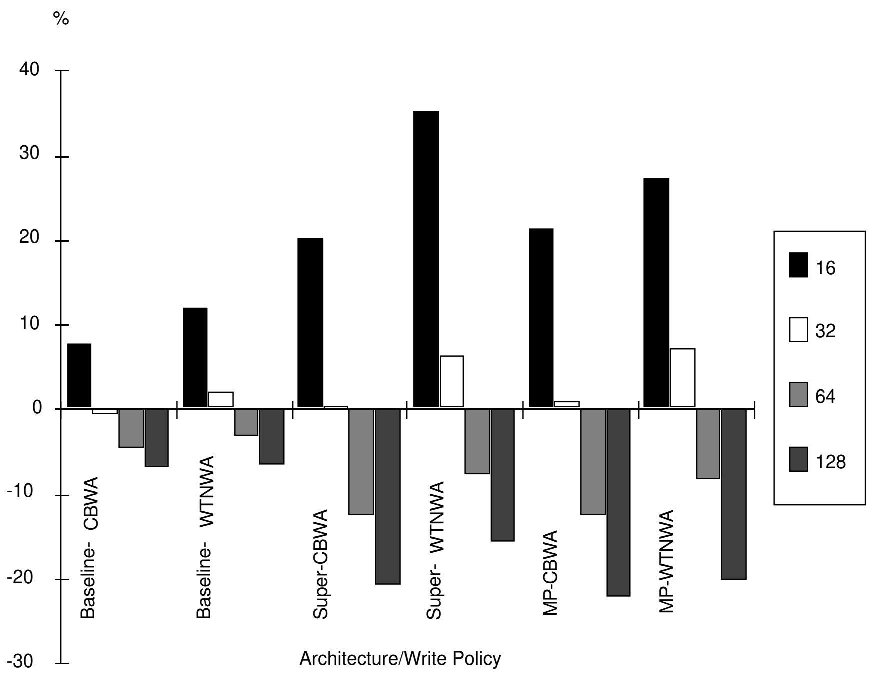

Figure 11: Percent CPI improvement of implementations with secondary cache relative to implementations without secondary cache for 0.3μ across different line sizes.

the dirty line and wait for the entire read line to be returned prior to the resumption of processing).

Therefore, for the baseline case (which has the biggest level-1 cache and smallest reference traffic), as the line size is increased, the best CPI without secondary cache is lower than with secondary cache.

The addition of secondary cache brings no improvement for the 0.3μ case, since the miss rate of the level-1 cache is now even lower (a much larger on-chip cache), and the ratio of size between the first-level and secondary caches is only four. This conclusion is shown in Figure 11 for the 0.3μ case.

Based on these data, the addition of a secondary cache is not warranted, since the performance improvement does not exceed 20% for higher line sizes. However, we have not taken memory traffic into account, and secondary cache has a great effect in reducing memory traffic.

7.4 Traffic and Line Size

Traffic on the memory bus is critical to overall system performance. When the offered bandwidth exceeds the achieved bandwidth, the memory bus dictates the performance of the overall system. Four main implementations are considered:

- Write Through No Write Allocate cache.

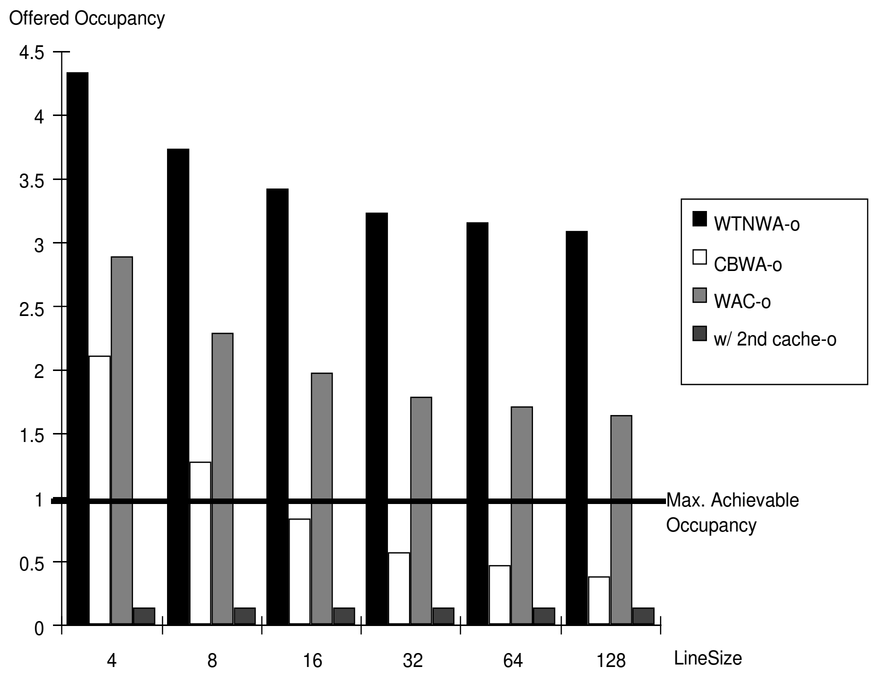

Figure 12: Baseline cache management policy/traffic analysis for 0.75μ.

- CopyBack Write Allocate cache.
- Write Through No Write Allocate cache with a Write Assembly cache.
- WTNWA with the addition of a secondary cache.

A total of 8 ns of bus time is available to each instruction, and the ideal offered bus time has to be below 8ns to avoid significant performance degradation. The data is presented in two ways. We first compute the total offered occupancy assuming an ideal bus (bus is able to supply the required bandwidth) for each implementation, and next, we break down the traffic of each implementation into different reference streams to get a better idea of the types of traffic on the bus and how it changes with increasing line size. When the offered traffic with an ideal bus exceeds that of available bus time, the base CPI is scaled to account for the effect of bus saturation.

The results indicate that the WTNWA implementation has the highest offered occupancy among the four implementations. Even at a line size of 128 bytes, the offered traffic is three times more than the available bandwidth. These data rule out the use of a WTNWA cache, since the bus has become the limiting resource even without multiple processors on the bus. In a multiprocessor system, we can select either the CBWA with large line size or the WTNWA implementation with a secondary cache. Figure 13 shows that traffic is higher for the MP implementation. In the MP implementation, it is even more important to have a secondary cache to reduce bus traffic.

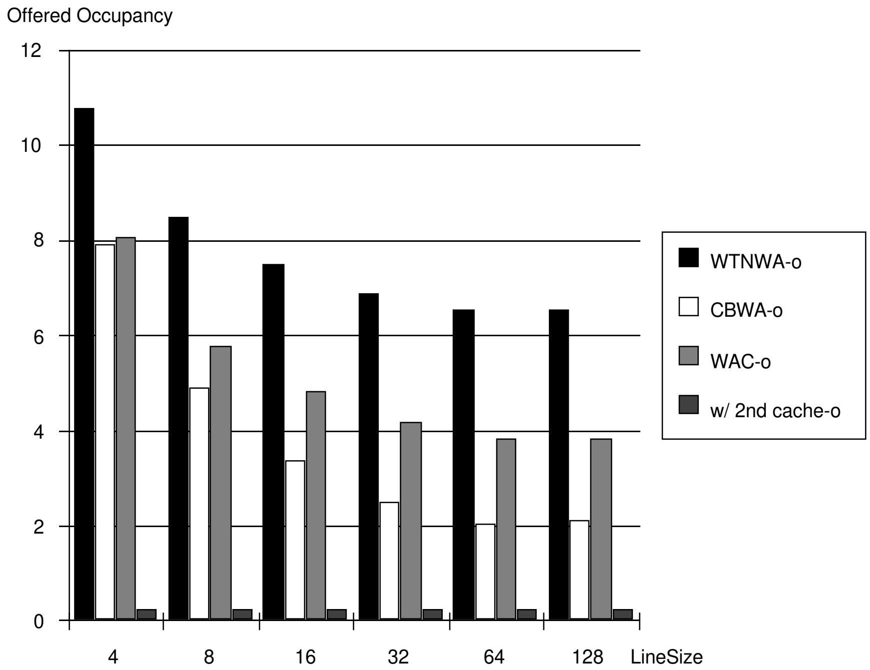

Figure 13: MP cache management policy/traffic analysis for 0.75μ.

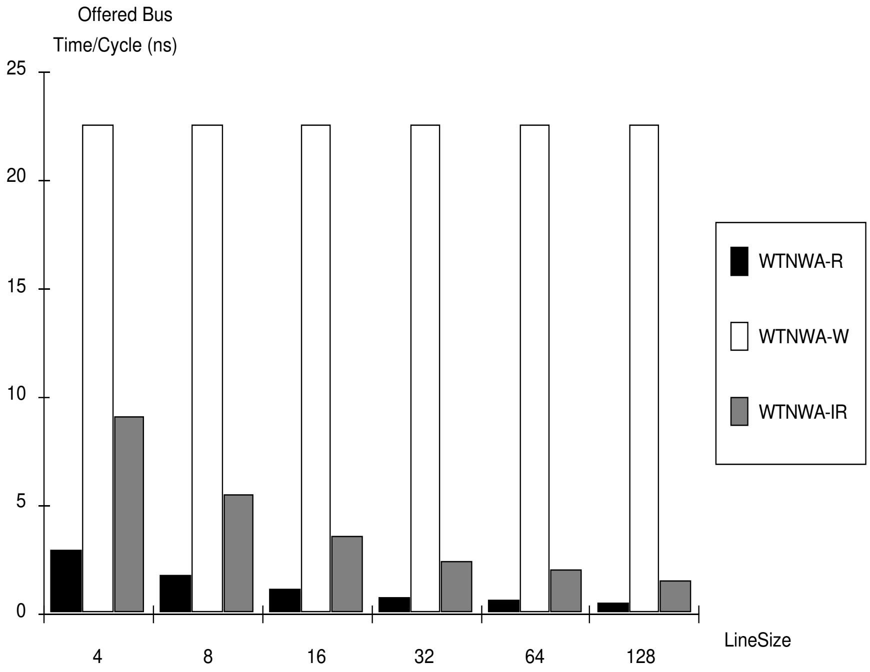

Figure 14: Baseline WTNWA traffic breakdown for 0.75μ(Configuration- Reference Type).

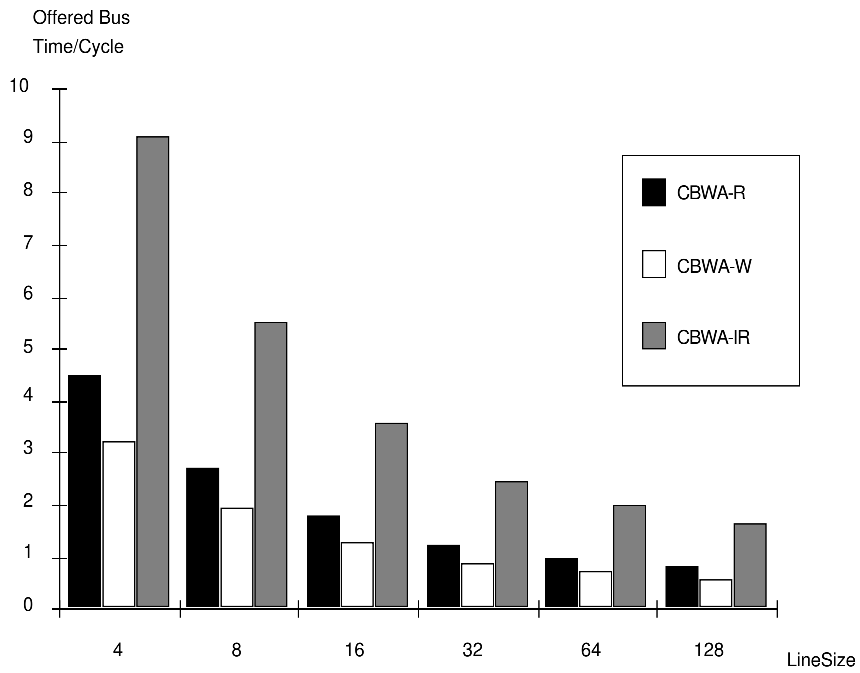

Figure 15: Baseline CBWA traffic breakdown for 0.75μ (Configuration- Reference Type).

As Figure 14 for WTNWA shows, when line size is increased, the traffic from both data and instruction read references is reduced, but the data write traffic remains constant. In this implementation, the CPU performance is bus saturated since offered bus time is greater than the maximum achievable bus time.

Figure 15 gives the traffic breakdown for the CBWA implementation. One can see that as line size increases, reducing the cache miss ratio, a significant reduction in all three streams of traffic occurs. At line size of 128 bytes, the bus occupancy is kept below 0.5, and processor performance degradation due to the bus contention among the three reference streams is taken into account by the read/write interference time, as described earlier.

Figure 16 shows the traffic breakdown of the WAC case. In using the write assembly cache, the data write traffic is converted from word traffic to the more efficient line traffic. Data from [7] show that the WAC can filter out approximately 70% of the write traffic. With WAC, the write traffic is reduced, but still dominates the overall reference traffic.

In order to prevent the bus from becoming the system bottleneck, we can pick either the CBWA or the implementation with secondary cache. The total offered occupancy has to be kept below the maximum achievable occupancy. As we decrease the feature size, the situation is improved because larger caches reduce the bus traffic.

The CPI models that we used previously assumed a "perfect write buffer." These models for CPI are valid as long as bus occupancy is kept below one, and a sufficiently large write buffer exists. However, when bus occupancy is above one, the bus saturates and we must consider total traffic and degrade the base CPI accordingly. Figure 17 shows the final CPI for the 0.75μ implementations , with bus traffic taken into account by scaling the base CPI

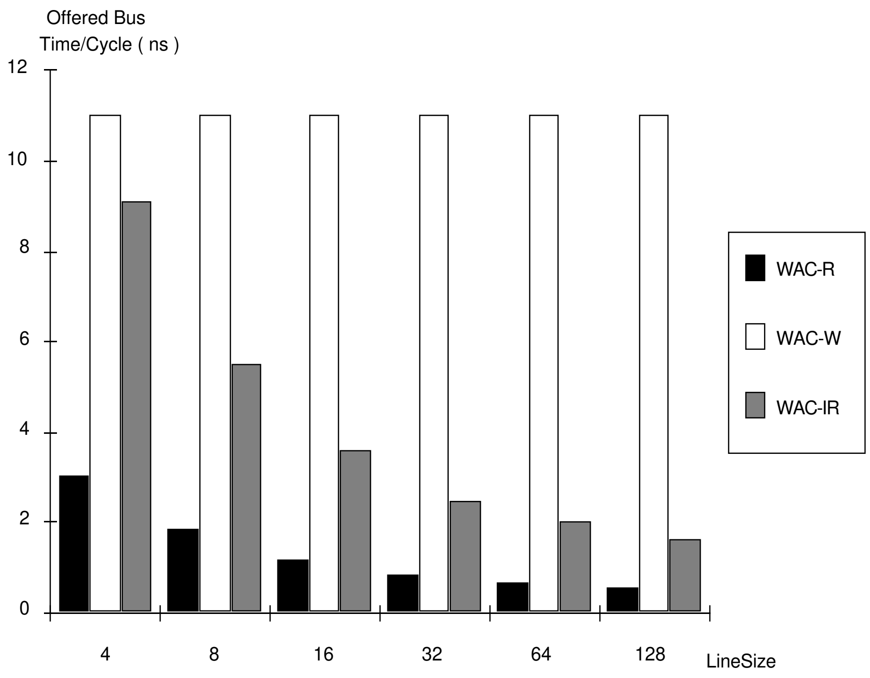

Figure 16: Baseline WAC traffic breakdown for 0.75μ(Configuration- Reference Type).

with respect to the offered occupancy.

Some useful conclusions can be drawn from Figure 17. The best-performing configurations are the superscalar and the baseline implementation with CBWA cache at a line size of 128 bytes. The fact that the baseline implementation comes out relatively even with the superscalar can be attributed to its larger cache and smaller need for bandwidth. Another interesting point is that the worst performing implementations are all WTNWA, with WTNWA-MP being the worst. It is also notable that increasing the line size has less beneficial effect on WTNWA than on the other implementations. This is due to the fact that most of its traffic is word writes, and the initial access time cannot be amortized, as in the case of line accesses.

Figure 18 shows the same analysis for the 0.3μ implementation. In this case, the best performance is MP with CBWA. This is a reversal from the 0.75μ implementation, where MP generally performs the worst. This graph does not include the WTNWA cases, since we have concluded that those are not feasible implementations. However, the secondary cache(2$) implementations are shown.

The main difference between the 0.3μ and the 0.75μ implementations is that the first-level cache is now sufficiently large to supply the bandwidth demands of the MP implementation. There is also much less degradation of CPI due to bus saturation (except in the case of WTNWA implementation).

7.5 CPI versus Feature Size

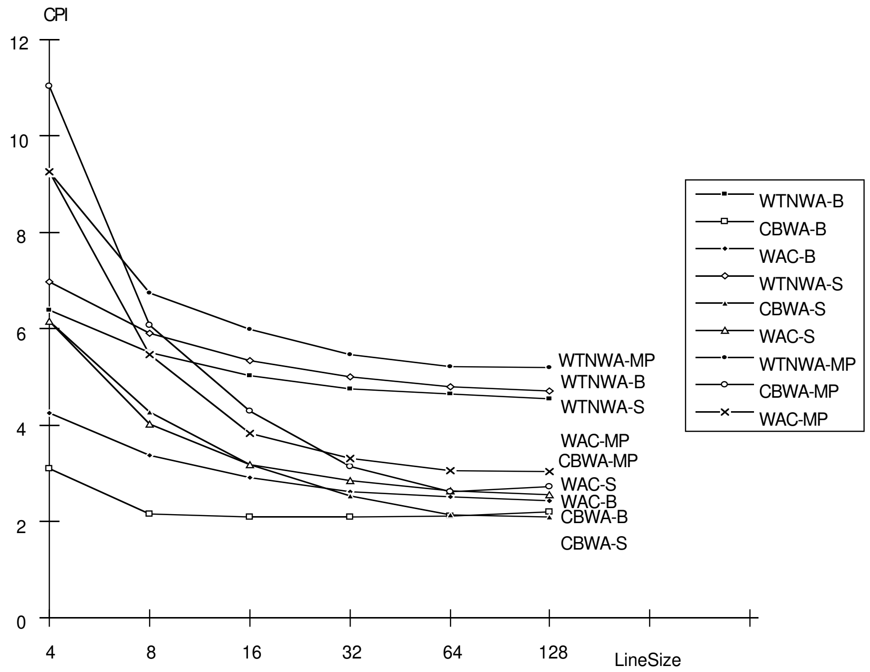

Figure 17: CPI/cache management policy summary for 0.75μ (Write Policy-Configuration).

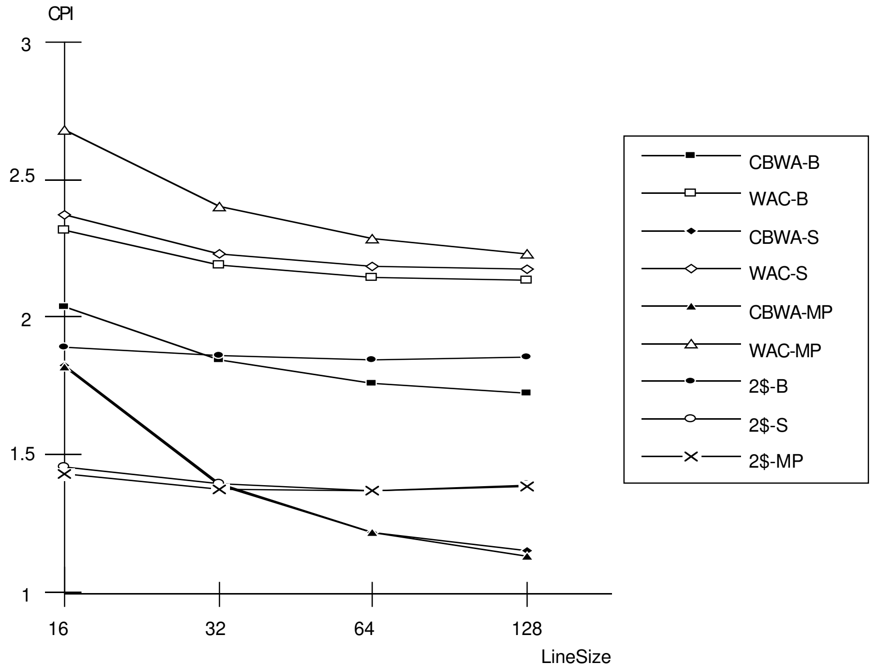

Figure 18: CPI/cache management policy summary for 0.30μ (Write Policy-Configuration).

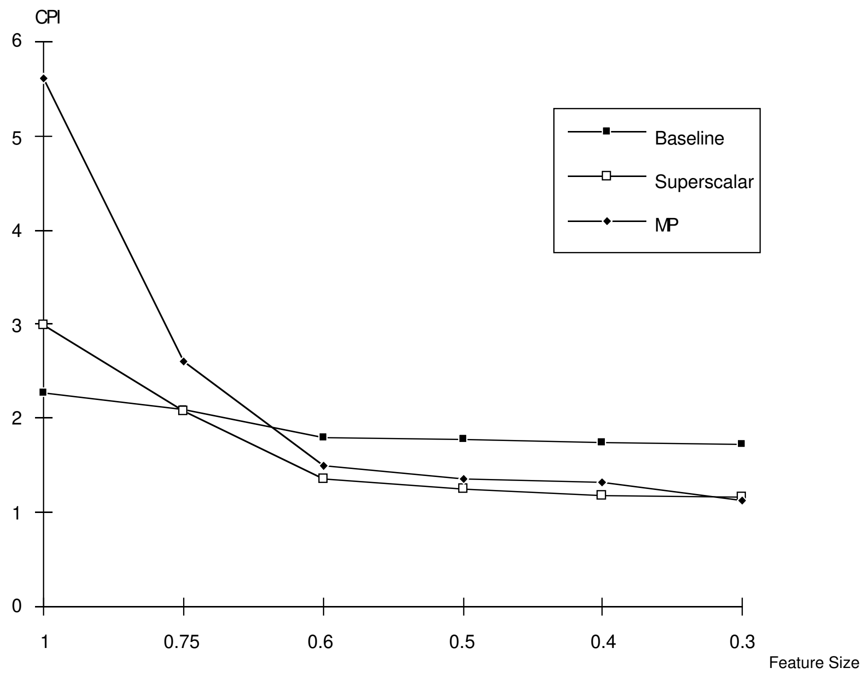

Figure 19: CPI versus feature size summary.

Figure 19 shows the best CPI across feature sizes. At the 1μ feature size, the bandwidth supplied is not sufficient to match that of the MP implementation. This results in a dramatic drop in CPI for the MP case as we go from 1μ to 0.75μ. It can also be observed that the increasing cache size does not help the baseline CPI much after 0.75μ. With this observation, one should consider moving to either superscalar or MP architecture. This coincides with what is happening in the industry. To decide between the superscalar and MP cases is a different story. The MP implementation has a higher demand for bandwidth, and it is at a disadvantage due to the requirement for split ICache. The CPI of the two implementations is roughly equalized at 0.3μ. The slow decrease of CPI after 0.6μ indicates that architects should consider other architectures at this point, such as higher-number issue superscalar or higher MP implementations. However, steps must be taken to provide memory bandwidth so that the memory bus does not become the limiting factor.

8 Conclusions

This paper presents a technique to correlate area and performance as technology is scaled for three processor configurations. Performance gain due to scaling of technology levels out for the three implementations at different points. For the baseline case, insignificant gains are achieved as one scales below 0.75μ. For the superscalar and MP implementations, the reduction of gains occurs at 0.6μ. This should give designers a hint for the future. As technology scales to below 0.6μ, architects need to explore the design of higher-issue superscalars and more than two multiprocessors on a single chip. The benefits of single level on-chip caches decline as these caches reach 128KB and beyond. The desire to scale cycle time also prevents the continued expansion of the first level cache. One alternative that should be considered is on-chip secondary cache.

This paper also addresses the widening gap between processor speed and memory latency. Based on our analysis, only the CBWA and secondary cache implementations are feasible. A simple model is presented which scales the base CPI as the offered traffic exceeds that of the available bus bandwidth. The results show that a simple bus-based memory interface quickly becomes the system bottleneck, requiring higher-bandwidth implementations such as crossbar switches to force down contention.

References

[1] S. K. Ghandi. VLSI Fabrication Principles: Silicon and Gallium Arsenide. Interscience Series. Wiley, New York, 1983.

[2] M. J. Flynn. Computer Architecture: Concurrent and Parallel Processor Design. Jones and Bartlett, Boston, 1994.

[3] J. M. Mulder, et al. An area model for on-chip memories and its application. *Journal of Solid State Circuits* 26(2) , February 1991.

[4] M. D. Hill. Cache Performance of the SPEC92 Benchmark Suit. *IEEE Micro*, Vol. 13, Number 4, pages 17-27, August 1993.

[5] M. Johnson. Superscalar Microprocessor Design. Innovative Technology Series. Prentice Hall, 1990.

[6] A. J. Smith. Line size choices for CPU cache memories. *IEEE Transactions on Computers*, C-36(9):1063-1075, September 1987.

[7] B. K. Bray and M. J. Flynn. Write caches as an alternative to write buffers. Technical Report CSL-TR-91-470, Stanford University, April 1991.