AN AREA-UTILITY MODEL FOR ON-CHIP MEMORIES AND ITS APPLICATION

Johannes M. Mulder
Nhon T. Quach
Michael J. Flynn

Technical Report: CSL-TR-90-413

FEBRUARY 1990

This work was supported by NSF contract No. MIP88-22961 using Nasa facilities under contract NAGW 419.

AN AREA-UTILITY MODEL FOR ON-CHIP MEMORIES AND ITS APPLICATION

by

Johannes M. Mulder, Nhon T. Quach, and Michael J. Flynn

Technical Report CSL-TR-90-413
February 1990

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305-4055

Abstract

Utility can be defined as quality per unit of cost. The utility of a particular function in a microprocessor can be defined as its contribution to the overall processor performance per unit of implementation cost. In the case of on-chip data memory (e.g., registers, caches) the performance contribution can be reduced to its effectiveness in reducing memory traffic or in reducing the average time to fetch operands. An important cost measure for on-chip memory is occupied area. On-chip memory performance, however, is expressed much more easily as a function of size (the storage capacity) than as a function of area.

Simple models have been proposed for mapping memory size to occupied area. These models, however, are of unproven validity and only apply when comparing relatively large buffers ($\geq$ 128 words for caches, $\geq$ 32 words for register sets) of the same structure (e.g., cache versus cache). In this paper we present an area model for on-chip memories. The area model considers the supplied bandwidth of the individual memory cells and includes such overhead as control logic, driver logic, and tag storage, thereby permitting comparison of data buffers of different organizations and of arbitrary sizes. The model gave less than 10% error when verified against real caches and register files.

Using this area-utility measure ($\frac{Performance}{Area}$), we first investigated the performance of various cache organizations and then compared the performance of register buffers (e.g., register sets, multiple overlapping sets) and on-chip caches. Comparing cache performance as a function of area, rather than size, leads to a significantly different set of organizational tradeoffs. Caches occupy more area per bit than register buffers for sizes of 128 words or less. For data caches, line size is a primary determinant of performance for small sizes while write policy becomes the primary factor for larger caches. For the same area, multiple register sets have poorer performance than a single register set with cache except when the mamory access time is very fast (under 3 processor cycles).

Key Words and Phrases: Area-utility model, on-chip buffers, cache, registers, multiple-register set, register window, global register allocation, memory traffic, cycle time, traffic ratio, timing model.

Contents

1 **Introduction** 1

2 **Performance measures** 2

Performance models 3

Model parameters and cycle-ratio calculations 5

Limitation of the Performance-model 6

3 **An improved area model** 7

Area unit 8

Register-set and memory areas 8

Cache areas 10

Set-associative caches (s.a.c.) 10

Fully associative caches (f.a.c.) 12

Limitation of the Area-model 15

Area-model verification 18

4 **Application of area and performance models** 19

Cache performance 19

Cache-organization tradeoffs as a function of area utility 24

Cache register-set tradeoffs as a function of area utility 26

5 **Conclusion** 27

6 Acknowledgements 28

List of Figures

1 Architecture design influenced by performance requirements and cost constraints. 1
2 Proposed area model relative to simple models. 2
3 Reference stream models for four different buffering organizations. 3
4 Area of on-chip data memory as chip cost function. 8
5 Data- and tag-area model. 9
6 Relative area for set-associative caches. 13
7 Fully associative cache layout. 14
8 Relative area of fully-associative caches. 16
9 Aspect ratio and area change as a function of layout. 17
10 Traffic ratio for a combination of a (small) register set and a set-associative cache. 22
11 Best-case performance for register-cache combination. 23
12 Worst-case performance for register-cache combination. 23
13 Performance as a function of set associativity, area, and size. 24
14 Full versus set associativity. 25
15 Performance as a function of line size, area, and size. 25
16 Traffic ratio as a function of buffer area for baseline, SRS, MRS, and SRS with Cache organization. 26
17 Cycle ratio as a function of buffer area for baseline, SRS, MRS, and Cache organization (2/4 cycle memory). 27

List of Tables

1 Allocation of Cycles and Equations for Cycle Ratio Calculation 6
2 Comparison of Actual and Predicted Cache Areas. 20
3 Comparison of Actual and Predicted Register File Areas. 21

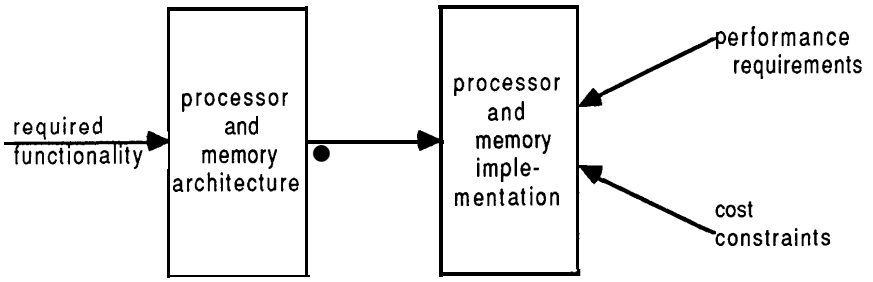


Figure 1: Architecture design influenced by performance requirements and cost constraints.

1 Introduction

Performance requirements and cost constraints placed on an implementation directly influence processor and memory architecture design decisions (see Figure 1). In the design of an architecture, it is necessary to abstract these implementation details to architecture measures for making tradeoffs.

The two most obvious performance-related architecture measures for data buffering are hit ratio and traffic ratio. The former models the buffer quality in capturing the requests from the processor and the latter models the buffer response in satisfying the demands placed on the memory system. Although both measures indicate the relative differences when comparing buffers of a similar structure (e.g., cache vs cache), they do not suffice when the structures of the buffers to be compared are significantly different (e.g., register vs cache). To compare different data buffers, it is necessary to use a measure which models the timing of the buffer as well as its behavior. The *cycle ratio* presented in Section 2 combines these architecture measures with the timing characteristics of the buffer or memory, yielding the total time spent in the memory system.

On an implementation level, cost could be represented in terms of power consumption, I/O bandwidth, or chip area. Because a balanced distribution of chip area among different functions is our main focus, we equate cost to area in this paper. Since the performance of data buffers are primarily characterized by their sizes (storage capacity), a mapping from size to area is needed. Hill [1] and Alpert [2] have used simple area models for such a purpose. The simple models account for tag and line-status bits in addition to the data bits; the difference in area between CAM cells and normal storage cells is also included [2]. These simple models, however, are of unproven validity. Moreover, the models only apply when comparing large caches of the same structure. When comparing small caches or comparing buffers of different structures (e.g., cache vs register) these simple area models do not suffice. In small caches the area overhead

dominates, but is not included in the simple models. The difference between the simple model and the present model is shown in Figure 2 for a two-way set associative cache, a fully-associative cache, and a register set. For small caches, the differences in area predicted by the models are significant.

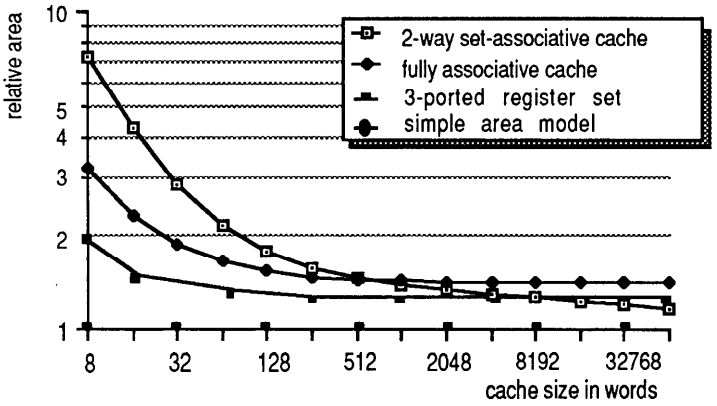


Figure 2: Proposed area model relative to simple models.

When comparing buffers of different structures, it becomes important to consider the supplied bandwidth of the individual cells in the area model. A register set, for example, often supplies two to four times the bandwidth of a cache. This bandwidth difference shows up in the additional area occupied by a register bit as compared to a cache bit.

The area model presented in Section 3 corrects these deficiencies by including data bits, tag bits, and overhead logic (i.e., drivers and comparators), and by considering the bandwidth of the individual memory cells in the model. Measuring the utility ($= \frac{performance}{chip\ area}$) of a particular function allows comparison of independent and different functions. Although applicable in a wider sense we restrict our utility measure to on-chip data buffering (e.g., registers and caches). In section 4, we use this utility measure (cycle *ratio* and the area model) to compare cache and register set performance. The comparison is done for various cache and register set organizations as a function of both area and size.

2 Performance measures

Extending the concept of hit ratio and traffic ratio requires splitting the reference streams into those with identical timing characteristics. By associating a penalty to the various reference streams, we calculate the number of cycles a processor spends in the whole buffering/memory hierarchy. The measurement tools must be able to record statistics for these different streams.

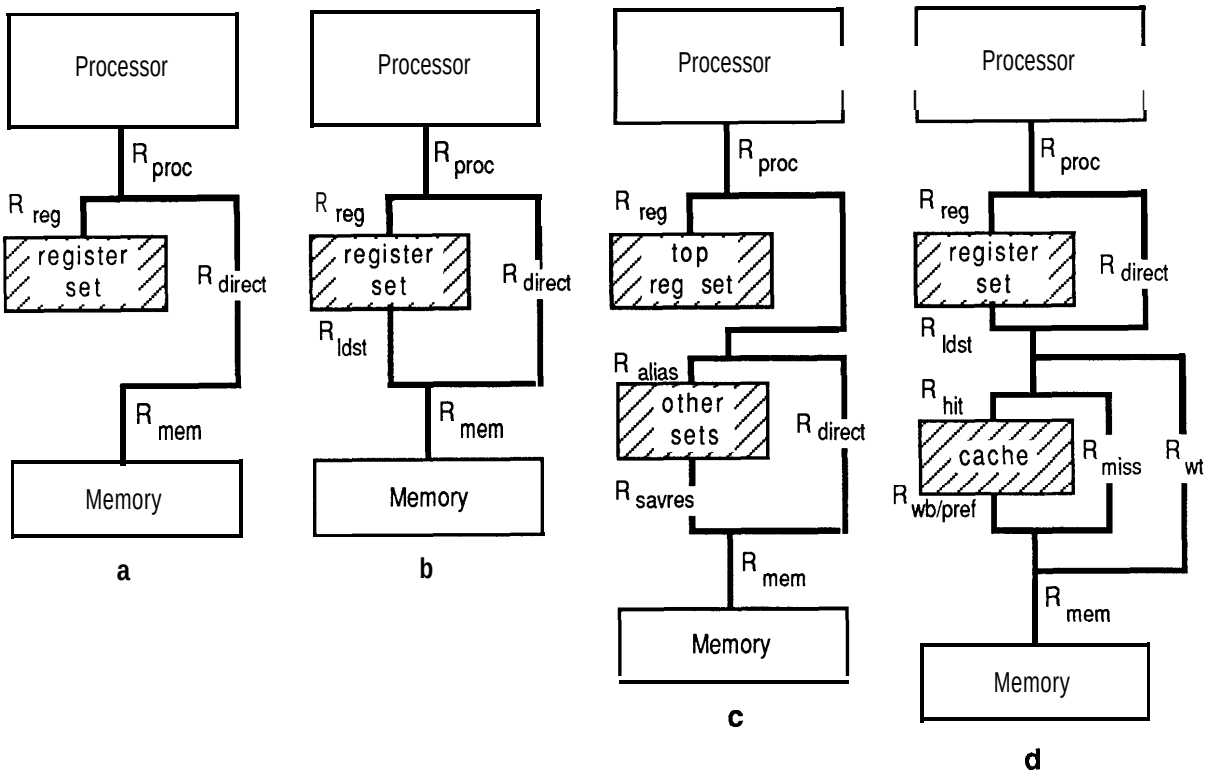


Figure 3: Reference stream models for four different buffering organizations.

Performance models

In this paper, we use the timing characteristics for four different buffer models: baseline, single register set, multiple register set, and single register set with cache. Each stream is modelled in terms of *R*, a baseline reference stream. This is a measure of a sequence of references made by a processor to the memory system, R_{proc}, or the number of references made by the processor to the register set, R_{reg} or to other elements in the memory hierarchy such as the memory, R_{mem}. The baseline reference stream is created by a sequence of instructions, *I*. So long as all measurements of the various R_i are with respect to the same *I*, the results are consistent.

Baseline: The baseline model establishes a reference point for all cycle ratio calculations. This model only buffers intermediate results in expression evaluation (Figure 3a), i.e., the model assumes that only temporaries used in expression evaluation reside in a register. The processor fetches all other operands directly from memory. In this model, the reference streams from the processor to the off-chip memory can be described as

$$R_{proc} = R_{reg} + R_{direct},$$

$$R_{\mathbf{mem}} = R_{direct} .$$

The model assumes availability of sufficient temporary registers. Since in this paper, we only look at register based architecture, the baseline is best interpreted as a register set machine with 8 general purpose registers. This allows sufficient registers for temporaries plus others for stack control, indexing, etc. Other architectures discussed below have registers *in addition to* this baseline of 8 registers.

Single register set (SRS): Besides expression temporaries, an SRS architecture buffers any object to which a register can be allocated. Because the number of objects eligible for register allocation can be easily far greater than the number of register available, a reference stream exist for saving and restoring registers to and from memory. The two reference stream equations become (see Figure 3b):

$$\begin{aligned} R_{proc} &= R_{reg} + R_{direct}, \\ R_{\mathbf{mem}} &= R_{direct} + R_{ldst}. \end{aligned}$$

Note that R_{reg} is now different from R_{reg} of the baseline model.

Multiple overlapping register sets (MRS): An MRS resolves the issue of register shortage in SRS organizations by having multiple register sets with an overlap between sets for passing parameters. We distinguish two different access modes: to the top or current set and to any other set in use. Because the number of sets required to eliminate register shortage may still be too large, MRS organizations save and restore complete sets when a set shortage occurs. Access other than to the top set, therefore, requires a check if the set exists in on-chip or off-chip memory. Such a mechanism is easily implemented when all potential sets have a memory address. This mechanism is called aliasing and is described elsewhere in more detail [3, 4]. Because of the two-level nature of the buffer, the stream equations contain an additional component:

$$\begin{aligned} R_{proc} &= R_{reg} + R_{direct} + R_{alias}, \\ R_{\mathbf{mem}} &= R_{direct} + R_{savresr} . \end{aligned}$$

Note that R_{reg} in the SRS model is not necessarily the same as R_{reg} in the MRS model.

Single register set and cache (SRS with cache): Including a cache as second-level buffer complicates the model because a cache model requires two or three individual reference streams entering and leaving the cache. The stream equations now become slightly more complicated:

$$\begin{aligned} R_{proc} &= R_{reg} + R_{direct}, \\ R_{cache} &= R_{hit} + R_{miss} + R_{wt}, \\ R_{\mathsf{mem}} &= R_{miss} + R_{pref} + R_{wt} + R_{wb}. \end{aligned}$$

Where R_{hit} contains those references satisfied by the cache and R_{miss} those requiring off-chip access. The references contained in R_{miss} are only those corresponding directly to a reference in R_{cache}. Those indirectly caused by a reference in R_{miss} (e.g., fetching objects in the same cache line, or prefetching a line) are included in R_{pref} . All caches simulated in this paper use subblock placement [1, 2] where the subblock size equals that of the transfer unit between cache and memory. Because of subblock placement the caches do not require prefetch reference streams (R_{pref} = 0). Both write strategies, write-back and write-through, were simulated. R_{wb} is the *write back* traffic and R_{wt} the *write through* traffic.

The SRS architecture modelled by SRS with cache roughly corresponds to an architecture with 16 general purpose registers (with only eight registers available to the register allocator).

Model parameters and cycle-ratio calculations

Parametrization of the different models now consists of determining the (relative) cycle count. Table 1 lists the number of cycle accounted for by the various storage hierarchies in Figure 3 and the cycle count equations for the models. In the table, an entry of "1" means that each reference in *R* requires one cycle. We begin with the notion of a base cycle count C_{base}. This approximates the number of cycles required by the memory system (including registers) to execute the reference stream. C_{base} is determined specifically for the baseline processor.

Other cycle counts are taken relative to C_{base}. Most instructions that creat the reference stream take one or two register reads and one register write (ALU type, load, store). By allocating one cycle for each write (and none for reads), we effectively allocate one cycle to each instruction that uses the register set. Instructions that access memory require additional

Table 1: Allocation of Cycles and Eauations for Cycle Ratio Calculation

	access						
	all reg	mrs alias	cache hit	cache miss	cache writeback	cache writethrough	all memory
read	0	1	1	$1 + P_{mem}$	P_{mem}	P_{mem}	P_{mem}
write	1†	1	1	$1 + P_{mem}$	P_{mem}	P_{mem}	P_{mem}

†An entry of "1" means that, each reference in R requires one cycle.

$$C_{base} = R_{reg_{write\ (temporaries\ only)}} + R_{mem} \cdot P_{mem}$$

$$Cr_{srs} = \frac{R_{reg_{write}} + R_{mem} \cdot P_{mem}}{C_{base}}$$

$$Cr_{mrs} = \frac{R_{reg_{write}} + R_{alias} + R_{savres} \cdot P_{mem} + R_{direct} \cdot P_{mem}}{C_{base}}$$

$$Cr_{cache} = \frac{R_{reg_{write}} + R_{hit} + R_{miss} \cdot (1 + P_{mem}) + R_{wb} \cdot P_{mem} + R_{wt} \cdot P_{mem}}{C_{base}}$$

cycles corresponding to the memory access time, P_{mem}. Overlapping of P_{mem} with instruction execution is discussed in the next section. The models presented here are determined relative to C_{base}. Presumably, the addition of a more elaborate buffering scheme to the baseline processor reduces the resulting reference stream. Thus the expected cycle ratios are less than one.

C_{base} estimates the time the memory system is occupied (without overlap of its functions). It underestimates the total cycle count for program execution by ignoring branch instructions completely and also ignoring the various pipeline breaks. (see [4] for different models).

Limitation of the Performance-model

As indicated earlier, performance models driven solely by reference streams require the individual references in a stream to be independent of time and of each other. Both requirements may cause inaccuracies and complicate refinement of the models.

Individual independence

Individual independence of reference implies that only one reference accesses a data buffer at a time. This complicates modeling of multi-port buffers. Within our architectural model this is solved for first-level buffers with two read ports and one write port. Multiple write ports and concurrent access of second-level buffers cannot be modelled.

Time independence (overlapping of operations)

Dependence of time occurs when implementing (partially) overlapped access for references in a particular stream. The part of reference which is overlapped and therefore invisible for the cycle ratio, shows up in the performance when this reference blocks a visible reference. A cache prefetch, which is supposedly invisible, for example, may block the bus to memory also required by a subsequent cache miss. The effect of overlapping or time dependence can be bounded or approximated in several ways:

Best case: no interference at all, overlap is maximal. Best case is a good approximation for large buffers (small traffic ratios).

Worst case: all overlapped references interfere. Worst case may be a good approximation for small buffers (large traffic ratios).

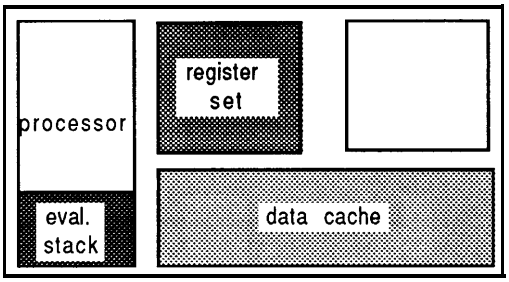
Probabilistic: all references are assumed to be statistically distributed over time. The visibility of an overlapped reference now becomes $Pr(\ interference)\ .\ E(inter\ f\ erencelength)$. $Pr(interf\ erence)$ is the probability that a visible reference runs into an overlapped reference (and must wait) or $\frac{C_{overlapped}}{C_{bestcase}}$, where $C_{overlapped}$ is the total number of hidden cycles of all (partially) overlapped references and $C_{bestcase}$ is the number of best-case cycles according to the performance model. $E(\ inter\ f\ erencelength)$ is the expected number of interference cycles or $\sum_{n=1}^{P_{max}} (1 - Pr(int.))^{P_{max}-n}\ .\ Pr(int.) \cdot n$.

Examples of references which may be (partially) overlapped are writes in general, write-through and write-back references in particular, and cache prefetches (transfer units within a line, or complete lines).

As this paper is primarily concerned with relatively small on-chip buffers, the performance measures assumed worst case behavior unless indicated otherwise. The benchmarks used here consist of 5 medium sized programs (dynamic size 2.5 to 35 million bytes) generally representative of a workstation environment (non scientific). For additional information, see [5].

3 An improved area model

In this paper, data-buffer area is modelled according to Figure 4. The total amount of area occupied by a combination of buffers is simply the sum of the individual areas. We ignore the wiring overhead necessary to combine the buffers for modeling simplicity.



$$A_{total} = A_{e\text{-}stack} + A_{r\text{-}set} + A_{d\text{-}cache}$$

Figure 4: Area of on-chip data memory as chip cost function.

Area unit

Although the most obvious unit for area is μm^2, the unit for this model is a technology independent notion of a register-bit equivalent or rbe. The advantage of this is the relatively straightforward relation between area and size, which facilitates interpretation of area figures. One rbe equals the area of a bit storage cell. Because not all storage cell designs occupy the same area, a suitable cell has to be selected as the area unit. Static storage cells occupy more area than dynamic ones, and the area of both static and dynamic cells depends on the bandwidth required. A higher bandwidth potentially implies more bit and control lines; more lines which cross a cell increase the area. A higher bandwidth can also imply an increased transistor size to increase the speed of driving the bus lines.

The following buffer area model uses three different storage cells: a six-transistor static cell with high bandwidth, a six-transistor static cell with lower bandwidth, and a three-transistor dynamic cell [6] (henceforth referred to respectively as register cell, static cell, and dynamic cell). The area unit, rbe, equals the area of the register cell.[1] We have empirically determined that the static cell area is 0.6 rbe and the dynamic cell area is 0.3 rbe. Dynamic cells are sometimes used to reduce the area of on-chip caches at the expense of bandwidth.

Register-set and memory areas

Register buffers are generally an integral part of the data path. These buffers use high bandwidth register cells consisting of a read port and a port which can be used for either reading or writing. These register cells can support two reads and a time-multiplexed write per access cycle. Throughout the remainder of this paper, we refer to such register cells as "3-ported cells", though they actually have less hardware overhead than ones with two read ports and a

[1] MIPS-X [7] is used as the basis for certain empirical parameterizations. This experimental microprocessor was implemented in CMOS technology with 2μm minimum geometry. Its register cell was 37x55μm and its cache storage cell (static) was 30x40μm.

separate write port.[2] Besides storage cells, register buffers have bit-line sense amplifiers and control line drivers, which occupy additional area. The overhead for sense-amplifiers and drivers on all four sides of the bit array totals approximately six rbe. Figure 5a shows the area model of a register buffer or on-chip memory. The total area measured in rbe for a single array is

$$area = (registers_w + L_{sense\_amp})(datawidth_b + W_{driver}) \quad (1)$$

where $registers_w$ is the number of registers in words, L_{sense_amp} the length of the sense amplifiers, $datawidth_b$ the width of the word line drivers, all in unit of rbe. (The subscripts "b" and "w" are used in this paper to denote a quantity in bit and in word, respectively. A word is equal to four bytes or 32 bits.) From MIPS-X data, L_{sense_amp} and W_{driver} equal to 6 rbe. Equation (1) then becomes

$$area_{register\_set} = (registers_w + 6)(datawidth_b + 6) \text{ rbe} \quad (2)$$

In this study, $datawidth_b$ is assumed to be 32 bits for all register buffers (register files) unless otherwise stated.

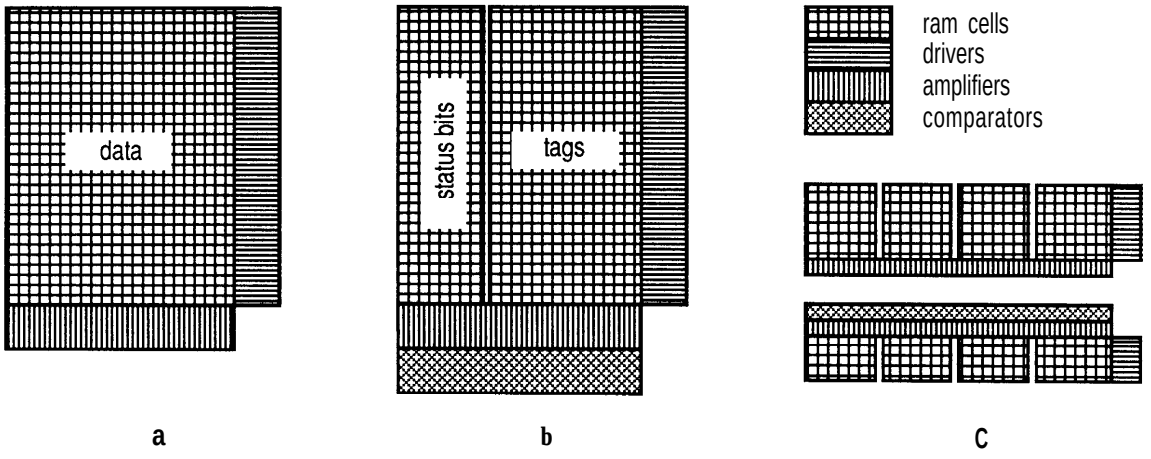


Figure 5: Data- and tag-area model.

Large on-chip buffers, other than register buffers, are generally associated with cache or a similar structure [8]. The bandwidth requirements of these buffers or memories are significantly

[2] Register buffer designs often differ in the way the read and write ports are used. For example, a 3-ported register buffer may have two read ports and a separate write port, requiring a total of 4 bit lines, or two read ports and a time-multiplexed write port, requiring only 2 bit lines. The write port may share the decoder or the bit lines with the read port, or both.

lower than that of a register set. These buffers usually support only one read or write at a time, and have more time to complete these operations than a register set. The storage cells used for these buffers can be either static or dynamic ones. Relaxed timing constraints allow use of smaller drivers and amplifiers. As for static cell area, we scale the equation for register area model (i.e., equation (2)) by 0.6 for static memory area model. For a static-memory array of $size_w$ words each of $line_b$ bits long, for example, the area is

$$area_{static\_memory} = 0.6(size_w + 6)(line_b + 6) \text{ rbe}$$

The area equation for dynamic-memory can be derived similarly, scaling equation (2) by 0.3. The size of the drivers in a dynamic-memory, however, does not scale in the same manner as the storage cells and is comparable to the static-memory one [6]. The area of dynamic memory is approximated as

$$area_{dynamic\_memory} = 0.3(size_w + 12)(line_b + 6) \text{ rbe}$$

Cache areas

The area occupied by caches is more complicated. Besides data (or array) bits, a cache consists of area for address tags, dirty, and valid bits, comparators, and control logic. The control logic is usually implemented in programmable logic array (PLA). Generally the cache divides into two relatively independent sections, one for the data bits and one for the tags, dirty, and valid bits. Both require additional area for drivers and amplifiers and the tag section also includes address comparators. The tags and the address comparators have two fundamentally different implementations. Set-associative caches generally store the tags in static cells (and sometimes in dynamic cells) using one bank of cells for each degree of associativity and one comparator per bank. Fully associative caches store tags in content addressable memory (CAM) cells, each cell consisting of storage and a comparison circuit. These two cache organizations have different area models. Caches are able to use static-cells or dynamic-cells because of their relaxed bandwidth requirements compared with registers.

Set-associative caches (s.a.c.)

The tag area for a set-associative cache is the tag-bit area plus the overhead for status bits, amplifiers, drivers, and comparators. The area of the comparators is largely determined by the routing of the address lines to the tag comparators. If the address lines run perpendicular to the bit lines of the tag cell, an area of at least the address line pitch times the number of lines

is necessary. MIPS-X comparators are $300x30\mu m^2$, mainly to allow 24 metal wires with $10\mu m$ pitch to cross. Based on these figures the area model assumes a comparator area of 6 x 0.6 rbe.

The number of tag bits per line equals the number of address bits used to address the cache minus the bits used to index the transfer units and lines. The present calculation uses 30 address bits, which implies an address space of one Giga word covered by the cache. The number of status bits per line depends on the transfer-unit[3] size and on the write strategy. Every line has one line-validity bit and every transfer unit has one validity bit and possibly one dirty bit. The dirty bit is present if the write strategy is write back; if the write strategy is write through, there is no need for a dirty bit. Area data presented in this and the later sections use one bit per line and two bits per transfer unit. Besides data and tags, caches require PLAs for control. Only for small caches does this influence the overhead noticeably. The size of the controller depends strongly on the write and prefetch strategies. The assumed size of the PLA is 130 rbe [9], a fairly low estimate.

Figure 5a shows the layout of a cache array (data), and Figure 5b shows the layout of a directory area. Figure 5c shows the floorplan of a four-way set-associative cache; the four data areas are placed side by side and driven by one set of drivers. The four directory areas are also placed side by side across from the four data array areas.

Excluding the space taken by the address and data busses, the total area of a cache is

$$area_{s.a.c.} = pla + data + tags + status.$$

The area of the different items are functions of the storage, $size_b$, the degree of associativity, *assoc,* the line size, $line_b$, and the size of a transfer-unit, $transfer_b$. The number of transfer units in a line, *tunits,* the total number of address tags, *tags,* and total number of tag and status bits, tsb_b, are

$$tunits = \frac{line_b}{transfer_b},$$

$$tags = \frac{size_b}{line_b},$$

$$tsbits_b = tsb_b \; . \; tags = (1 + \gamma \cdot tunits + log_2 \frac{2^{30} \cdot assoc}{size_b}) \cdot tags$$

Where γ equals 2 for a write-back cache and 1 for a write-through cache. The area of a set associative cache using static cells is

[3]This is because of the assumption of subblock placement with subblock size equals the size of the transfer unit between cache and memory.

$$area_{s.a.c.} = 130 + 0.6(lines \cdot assoc + 6)(\frac{tags}{assoc} + 6) + 0.6(tsb_b \cdot assoc + 6)(\frac{tags}{assoc} + 6 + 6) \text{ rbe}$$
$$= 195 + 0.6 \cdot overhead_1 \cdot size_b + 0.6 \cdot overhead_2 \cdot tsbits_b \text{ rbe}$$

where

$$overhead_1 = 1 + \frac{6 \cdot assoc}{tags} + \frac{6}{line_b \cdot assoc}$$

and

$$overhead2 = 1 + \frac{12 \cdot assoc}{tags} + \frac{6}{tsb_b \cdot assoc}$$

The area of a set associative cache using dynamic cells can be derived similarly

$$area_{s.a.c.} = 195 + 0.3 \cdot overhead3 \cdot size_b + 0.3 \cdot overhead4 \cdot tsbits_b \text{ rbe}$$

where

$$overhead3 = 1 + \frac{6 \cdot assoc}{tags} + \frac{12}{line_b \cdot assoc}$$

and

$$overhead4 = 1 + \frac{12 \cdot assoc}{tags} + \frac{12}{tsb_b \cdot assoc}$$

Figure 6a shows the effect of different line sizes on the cache size relative to the storage provided ($\frac{area_{rbe}}{size_b}$). The area reduction is rather small, 32%, when moving from a 4-word line to a 64-word line since the tag-area reduction is partially compensated by an increase in transfer-unit status bits. Figure 6b shows the effect of the associativity on the cache area per data bit. As soon as the area becomes dominated by data array bits the associativity has little effect on the cache area per data bit. For small caches, however, the tag comparators determine the differences among cache organizations. Figure 6c shows the area of three caches relative to the area of a three-ported register set. Caches occupy more area than registers for 128 words or less. For these sizes the cache overhead dominates the cache area. For larger sizes, the smaller storage cells in the cache provides a total cache area smaller than the register set. A four-way set-associative cache of size 1024-word with two-word lines only takes 75% of the area of 1024 word register.

Fully associative caches (f.a.c.)

The tag area of a fully-associative cache is only a function of the number of address bits. The tag bits, however, are not stored in static or dynamic cells but in CAM cells. Alpert [10] assumed CAM cells to be twice the size of a static cell (1.2 rbe). His assumption is based on data for the 280,000. Our tag-area model assumes the same ratio. Figure 7 shows the layout

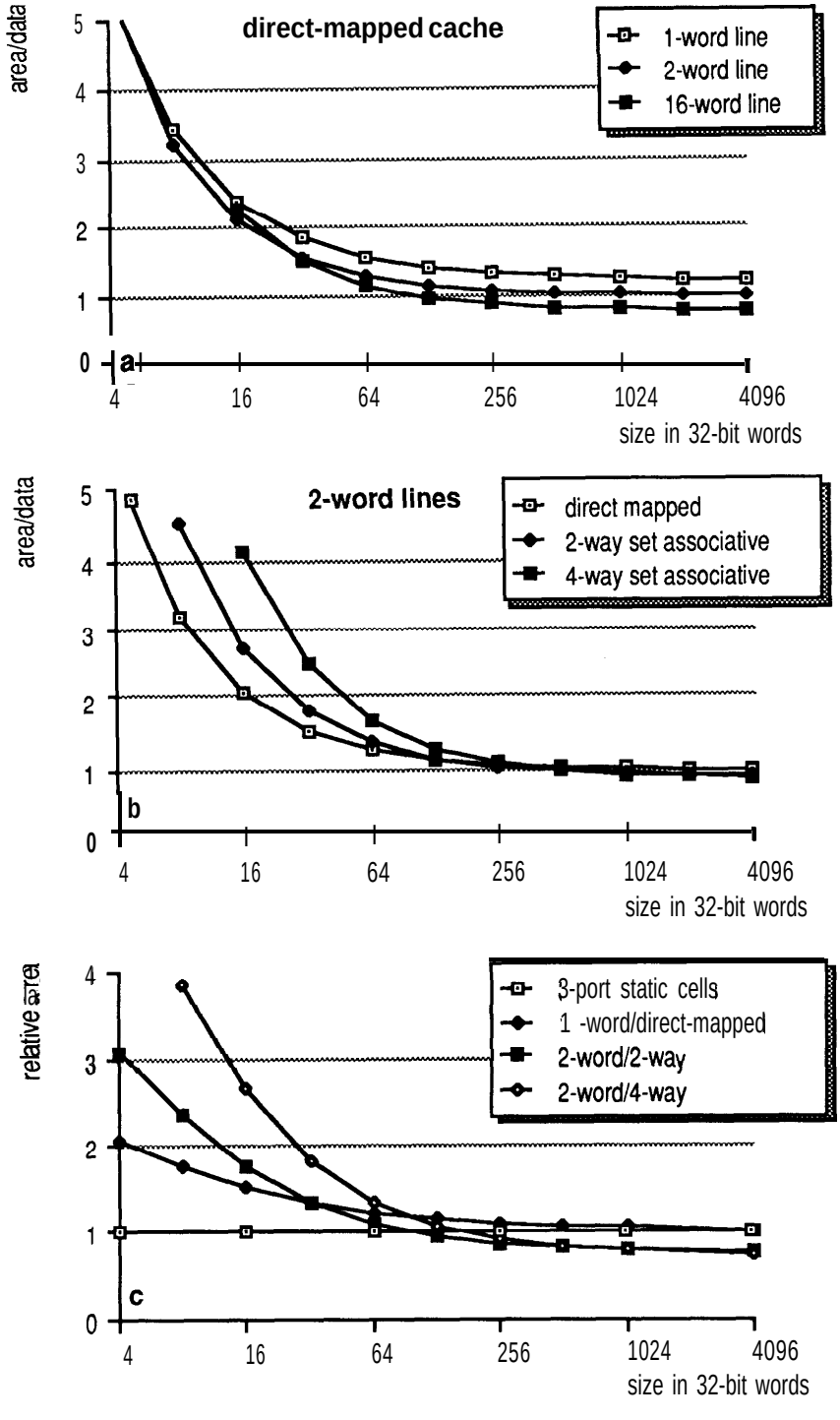


Figure 6: Relative area for set-associative caches as a function of line size, associativity, and provided storage.

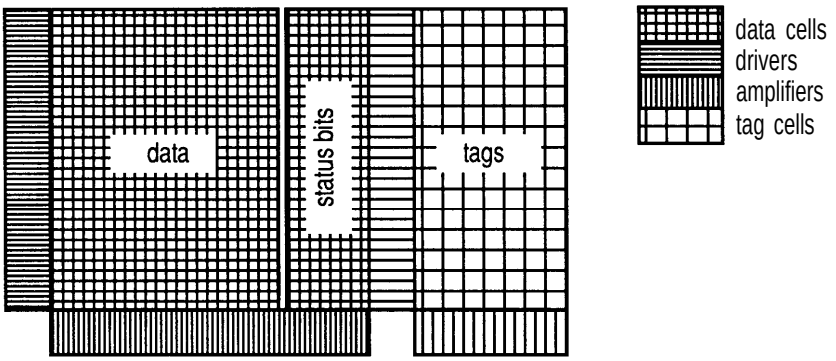


Figure 7: Fully associative cache layout.

of a fully-associative cache. If the associative search through the tags yields a hit, then the corresponding status bits are examined and the data array is indexed. Generally the status bits are combined with the tags to get the status early, which is useful if the tags and data are not placed immediately next to each other. The status bits, however, can be data type cells. The occupied area of a fully-associative cache is:

$$\begin{aligned} area_{f.a.c.} &= pla + (data + status) + CAM \\ &= 130 \cdot \mathrm{k}\ 0.6(tags+6)(\beta \cdot line_{b\_data}+6) + \\ &\quad 0.6(\sqrt{2} \cdot tags + 6)(\sqrt{2} \cdot line_{b\_CAM} + 6) \end{aligned} \tag{3}$$

Where $\beta = 1 \cdot \mathrm{l}\text{-} \frac{\gamma}{transfer_b}$ and $line_{b\_CAM} = 30 - log_2(line_w)$. The derivation of the equation follows the static memory one. The CAM cells are assumed to have an aspect ratio of 1. Defining $size_{b\_CAM} = tags \ . \ line_{b\_CAM}$, expanding, and rearranging, we rewrite equation (3) as

$$area_{f.a.c.} = 175 + 0.6. \ \beta \cdot overhead:, \ . \ size_{b\_data} + 1.2 \ . \ overhead6 \ . \ size_{b\_CAM}$$

where

$$overhead_5 = 1 + \frac{6 \cdot \beta}{tags} + \frac{12}{\beta \cdot line_b}$$

and

$$overhead6 = 1 + \frac{8.5}{tags} + \frac{8.5}{line_b}$$

The effect of organization on the area of fully-associative caches is shown in Figure 8a. Increasing the line size has significantly more effect for fully-associative caches than for direct-mapped ones (Figure 6a). Moving from l-word lines to 16-word lines, for example, reduces the cache area by 60%; the same move for a direct-mapped cache results in 35% less cache area. Figures 8b

shows the area of various cache and register configurations relative to the area occupied by a fully-associative cache of indicated sizes in 32-bit words. Generally, the fully-associative cache occupies the most per bit area, for sizes in excess of 64 words and registers the next most area per bit with set associative caches occupying the least area per bit over the same range. Similarly from Figures 8c, fully-associative caches occupy more area than 4-way set associative caches at large sizes with the crossover point depending on the line size.

Limitation of the Area-model

The area model is based on three assumptions. The first and most important assumption is that the access time of a buffer is independent of the storage provided. Second, the area only depends on the buffer organization and not on the layout specifics. Third, the aspect ratio is not significant for modeling purposes.

Access-time dependencies

To maintain the same access time while increasing the buffer size generally means that the storage cells, the drivers, and amplifiers also grow in size. This implies that the model is accurate for buffer sizes about which we have parametrized the model. These sizes are approximately 32x32 bits for register buffers and 2 Kbytes for caches.

The influence of layout on area

In any implementation, the amount of wasted area depends on the actually layout of the buffer. Our model allows for some wasted area because it abstracts both tag and data area to rectangles. Further, a circuit can be laid out in several ways, requiring slightly different amounts of area.

Aspect ratio

Figures 9a, b, and c illustrate the relation between layout and both size and aspect ratio (defined here as the width to height ratio of a geometry). If small caches with high degrees of associativity are laid out according to the area model (Figure 5c), the aspect ratios may become large. Figure 9a shows a four-way set-associative cache laid out according to our model. Although the area is optimal, the aspect ratio may be impractical for wiring purposes. Folding the cache twice (Figure 9b) and four times (Figure 9c) improves the aspect ratio from 7 to 2 and 0.6 but increases the area by 15% and 40%, respectively. Ignoring aspect ratio then can introduce an error of $\pm 20\%$ (over the aspect ratios considered, with model centered on an aspect ratio of about 2). The area increase is caused by two reasons. First, every fold requires its own drivers

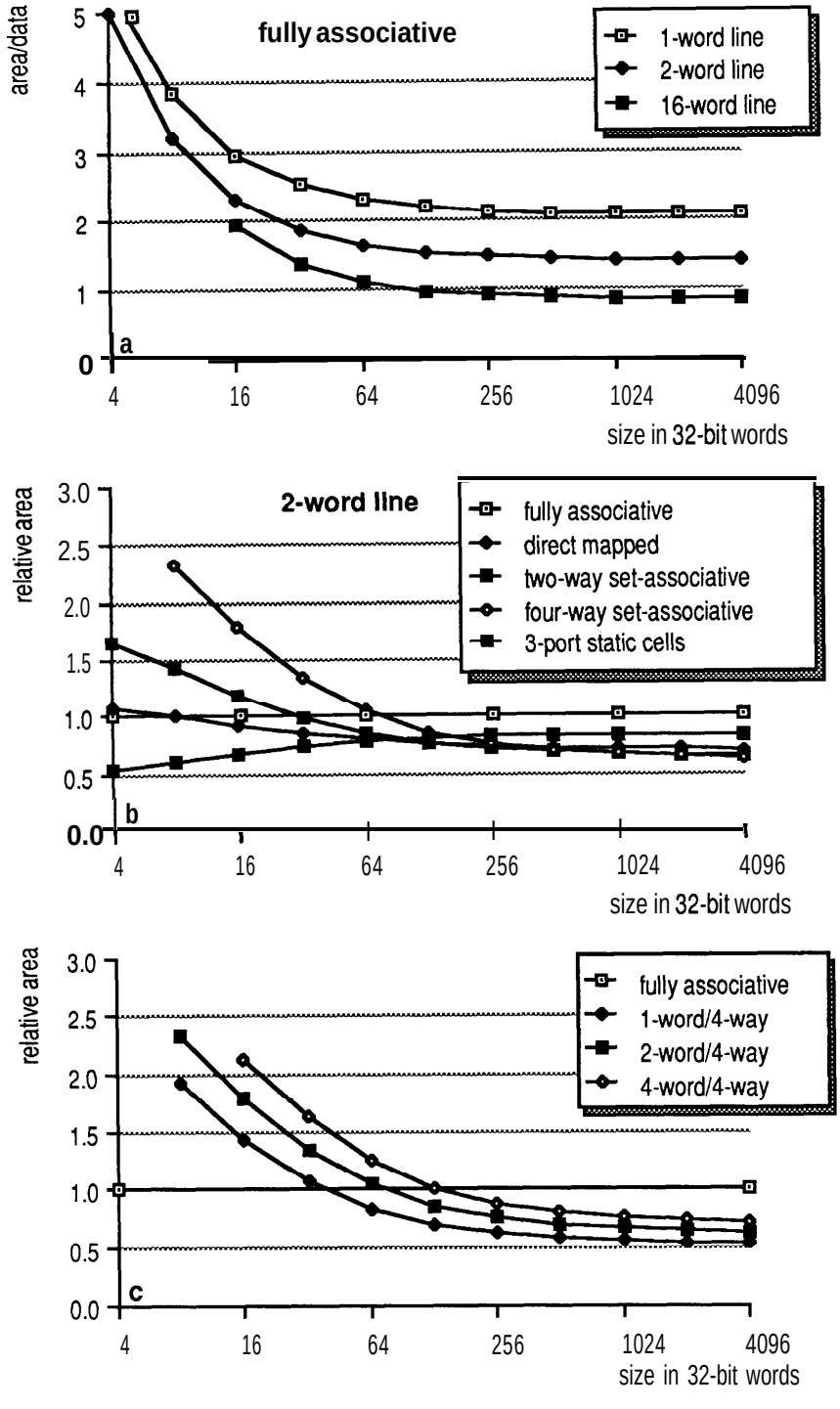
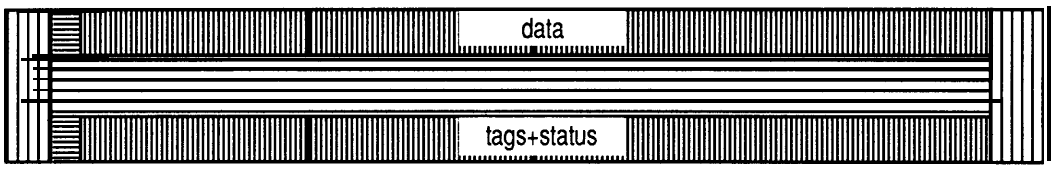
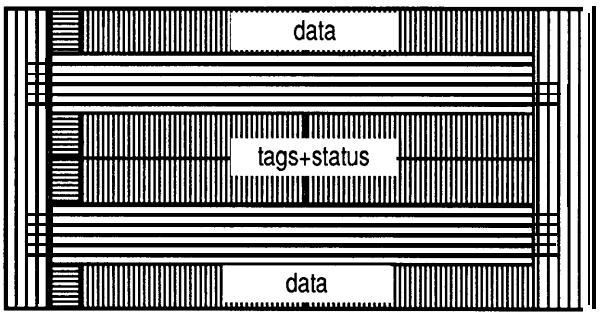


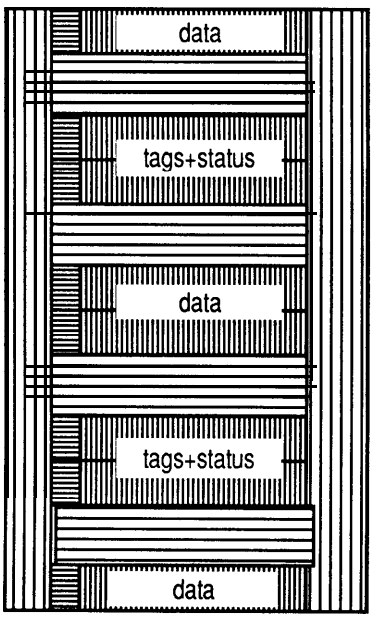
Figure 8: Relative area of fully associative caches as a function of line size and provided storage.



a size: 1, aspect ratio:7



b size: 1.15, aspect ratio: 2



c size: 1.4, aspect ratio: 0.6

Figure 9: Aspect ratio and area change as a function of layout.

for both tag and data arrays and second, every fold increases the area for both address and data busses supplying the cache. While aspect ratio in cache design can be important [11], we chose to ignore it to simplify modeling. This necessarily limits the achievable accuracy of our model.

Area-model verification

Clearly, the best way to establish the validity of the model is to compare the model prediction with actual caches and register buffers (or register files). For this purpose, we introduce a technology factor, *TF*, for both caches and register files. *TF* arises because our model was derived based on the MIPS-X data which is built with a 2μm technology. The use of *TF* permits comparison of caches and register files across generations of technologies (e.g., 1μm vs 2μm). Since *TF* is an area scale factor, it can be obtained simply as

$$TF = (\frac{minimum\ geometry\ in\ pm}{2})^2$$

For register files, the situation is more complicated because not all register files have the same number of read and write ports as the MIPS-X does. Also, read and write methods vary among processors. A read or a write port needs a decoder (and a word line) and one to two bit lines depending on the accessing methods. Single-ended ports require only one bit line; differential ports require two. To account for the different numbers of ports, we modify equation (1) as follows

$$area = (registers, + L_{sense\_amp})(datawidth_b + W_{dec} \ . \ N_{dec}) \ . \ PF \text{ rbe} \qquad (4)$$

Where W_{dec} is the width, and N_{dec} the total number, of the decoder. *PF* is an empirical factor accounting for the number of register ports in the register file. W_{dec} and *PF* are modelled as

$$W_{dec} = \alpha \ . \ datawidthb \qquad (5)$$

and

$$PF = [1 \text{ t } 0.25(N_{bit\_lines} - 2)] \qquad (6)$$

Incorporating equations (5) and (6) and rearranging, equation (4) becomes

$$area = datawidth_b(1 + a \ . \ N_{dec})(registers_w + L_{sense\_amp})[1 + 0.25(N_{bit\_lines} - 2)] \text{ rbe}$$

where N_{bit_lines} is the number of bit lines in the register file. For register files with only single-ended ports, N_{bit_lines} equals N_{dec}. In general, $N_{bit\_lines} = N_{dec} + N_{differential\_ports}$. In words, equation (5) states that the size of a decoder in a register file is proportional to $datawidth_b$, the number of bits it has to drive. MIPS-X data indicate that this proportionality constant a is

0.1. Equation (6) models the effect of each bit line in excess of two as increasing the register file area by 25% over a register file that has 2 bit lines (specifically, over the MIPS-X register file).

Table 2 compares the actual cache sizes with the model prediction. The cache areas in the "AREA„ column are in thousands of μm^2, obtained from the micrographs or the designers of the processors. The "MODEL" column contains the predicted cache areas, scaled appropriately by the *TF* factor. The absolute average error (AAE) is about 8.9%. The average error is -6.5% with a standard deviation of around 8.6%. The M68020 and DEC μVAX processors use one-transistor cells in the cache arrays. This has been modelled using the area equation for dynamic memory. The DEC2 processor uses four-transistor cells in the cache, which are about 10% smaller than the six-transistor static cells assumed in the present study [12]. The data and error given in Table 2 include this adjustment. The DEC μVAX processor also uses a folded bit line sensing scheme to reduce the size of the cache, the actual cache size and error should have been larger than those indicated in the table.

A similar set of data is presented in Table 3 for register files. The area data are obtained with the same procedure. The AAE is about 7.1%. The average error centers at 1.4% with a standard deviation of 9.9%. The MIPS-X register file includes the double-bypass logic, which occupies roughly 40% of the total area as estimated by visual inspection of the micrograph. The register file in the HP RISC processor drives the bus lines directly, requiring register cells that are 50% (1.5 rbe) larger than the conventional ones [13]. The register files in GE1 and GE2 processors use bigger cells than necessary because of the requirements of low soft-error rates. The actual cell size is 37x100 μm^2 in a 1.2μm process. We accounted for this by using this size as the area unit (instead of rbe). The data presented in Table 3 include all these adjustments.

4 Application of area and performance models

In this section, we consider data-cache performance and tradeoffs in cache parameters. We apply both models (area utility) to a comparison of cache organizations with large register-oriented buffering (MRS).

Cache performance

Small on-chip caches have attracted significant attention in recent years because of increasing circuit integration density. The Zilog 280,000 [29], DEC's microvax [18], and the Intel i486 [30] incorporate a on-chip mixed instruction and data cache; the Motorola M68020 [31], Bell-lab's CRISP microprocessor [32], Stanford's MIPS-X [21], Matsushita's research microprocessor [22], and HP RISC processor [15] include an instruction cache; and the Motorola M68030 and

Table 2: Comparison of Actual and Predicted Cache Areas.

μP	TECH. (μm)	TYPE♭	SIZE (Bytes)	AREA‡ (Kμm^2)	MODEL ERROR (Kμm^2)	ERROR (%)†	REF.
M68020	2.0	I,1w	256	*4,449*	4,048	*-9.0*	[14]
M68030	1.2	I,1w	256	2,445	2,184	-10.7	[14]
	1.2	D,1w	256	2,345	2,184	-6.9	
HP RISC	1.6	I,1w	256	2,775	3,134	12.9	[15]
NS32532	1.25	I,2w	512	3,776	3,246	-14.0	[16]
	1.25	D,1w	1K	7,699	6,153	-20.1	
NEC	1.2	I,1w	1K	9,448	8,596	-9.0	[17]
DEC1 (μVAX)	2.0	I/D,2w	1K	8,750	8,705	-0.5	[18]
DEC2	1.5	I,S	1K	9,448	9,858	4.3	[19, 12]
	1.5	D,1w	2K	20,125	16,935	-15.9	
DEC3	1.5	I,1w	2K	18,463	15,773	-14.6	[20]
MIPS-X	2.0	I,S	2K	27,517	27,545	0.1	[21]
Matsushita	1.0	I,2w	2K	11,188	10,448	-6.6	[22]
i860	1.0	I,2w	4K	13,347	12,805	-4.1	[23]
	1.0	D,2w	8K	26,977	23,904	-11.4	
i486	1.0	I/D,4w	8K	26,000	26,500	1.9	[24]

Legend:

♭ I - I-cache; D - D-cache; I/D - Mixed cache or cache that can be used either as an I-cache or as a D-cache; lw - Direct-mapped; 2w - 2-way set-associative; 4w - 4-way set-associative; S - Sector cache.

‡ Measured or reported areas.

† Percent error is calculated as:

$$\%error = \frac{Model - Actual}{Actual} \cdot 100$$

Table 3: Comparison of Actual and Predicted Register File Areas.

μP	TECH. (μm)	PORTS R/W/(R/W)	N_{bit_lines} ♯	TYPE ♭	SIZE (bits)	AREA ‡ ($K\mu m^2$)	MODEL ($K\mu m^2$)	ERROR † (%)	REF.
MIPS-X	2.0	2/0/1*	2	I	32x32	3,330	3,217	-3.4	[21]
DEC3	1.5	1/0/1	4	I	48x32	3,534	3,523	-0.3	[20]
HP1	1.5	2/2/0	4	I	31x32	3,450	3,727	8.0	[25]
GE1	1.2	2/1/1	4	FP	8x64	4,760	4,558	-4.2	[26]
GE2	1.2	2/1/1	4	FP	21x32	3,734	4,396	17.7	[27]
i860	1.0	3/2/0	5	FP	8x128	2,581	2,343	-9.2	[28]

Legend:

♯ N_{bit_lines} is the total number of bit lines in the register file; it is equal to the number of ports if only single-ended ports are used. In general, $N_{bit\_lines} = N_{decoders} + N_{differential\_ports}$ (see text).

♭ FP - floating point registers, I - integer registers.

‡ Measured or reported areas.

† %Error is calculated as:

$$\%Error = \frac{Model - Actual}{Actual} \cdot 100.$$

* MIPS-X's register file has 3 set of decoders but has only 2 bot lines (see text).

M68040 [33, 34], the Intel i860 [35], and the National Semiconductor NS32532 [36] have separate instruction and data caches.

Mixed on-chip caches may not offer sufficient combined instruction and data bandwidth to be implemented in high speed microprocessors (especially RISC type). Instruction-only caches are not always as area efficient as split caches for a given area [4]. Additionally, certain memory organization and processor-architecture combinations may perform better with a single data cache over a single instruction cache. The recent attention given to small caches has mainly been directed towards instruction and mixed caches, while fewer studies have been done on data-only caches.

In this subsection we investigate the performance of four data-cache register-set combinations, all with SRS architecture and (eight) registers set allocated globally[4] [37, 38] and a two-way set-associative cache organization with a line size of two words. The combinations differ on the write (writeback or writethrough) and prefetch[5] (with or without) strategies. All caches use subblock placement, where a subblock is the width of a transfer unit between cache and memory (one word).

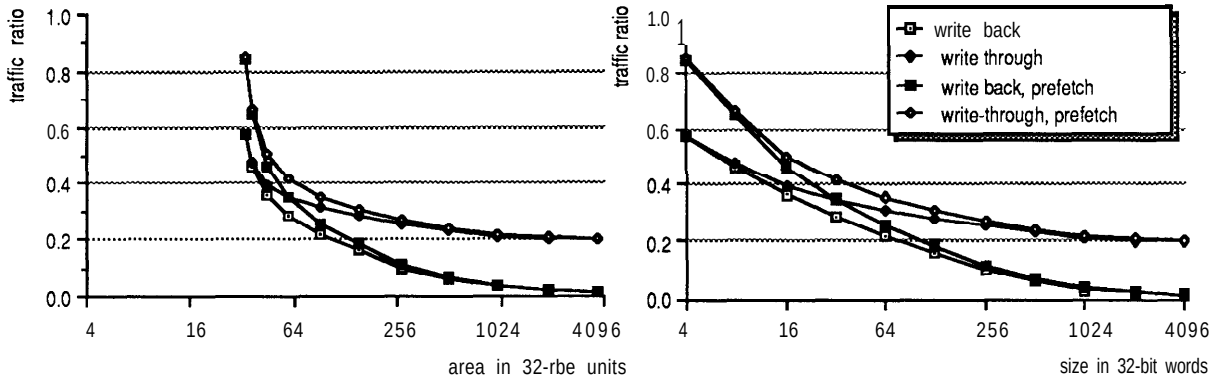


Figure 10: Traffic ratio for a combination of a (small) register set and a set-associative cache.

Figure 10 shows the traffic ratio (R_{mem}/R_{proc}) of the four combinations. For small caches the existence of prefetch determines the difference between the caches. For large caches the write traffic in the write-through organizations dominates the difference. The traffic ratio indicates a significant disadvantage of write-through and prefetch; the cycle ratio indicates the opposite conclusion however.

[4]This corresponds to about 16 general purpose registers when allowance is made for other register functions, e.g., temporaries and control stack pointers, etc.

[5]Prefetch in this context means fetching the transfer unit in a line which was not explicitly requested.

The cycle ratio includes one parameter not yet considered: the memory access time, P_{mem}. When P_{mem} becomes large the cycle ratio approaches the traffic ratio. When P_{mem} equals one, then the cycle ratio reflects the performance when the off-chip memory is ideal. For the cache performance we have chosen a P_{mem} of three cycles which reflects, for example, a 100ns memory for a 33ns processor cycle. Because writeback, writethrough, and prefetch references can be (partially) overlapped our measurements indicate both best- and worst-case cycle ratios.

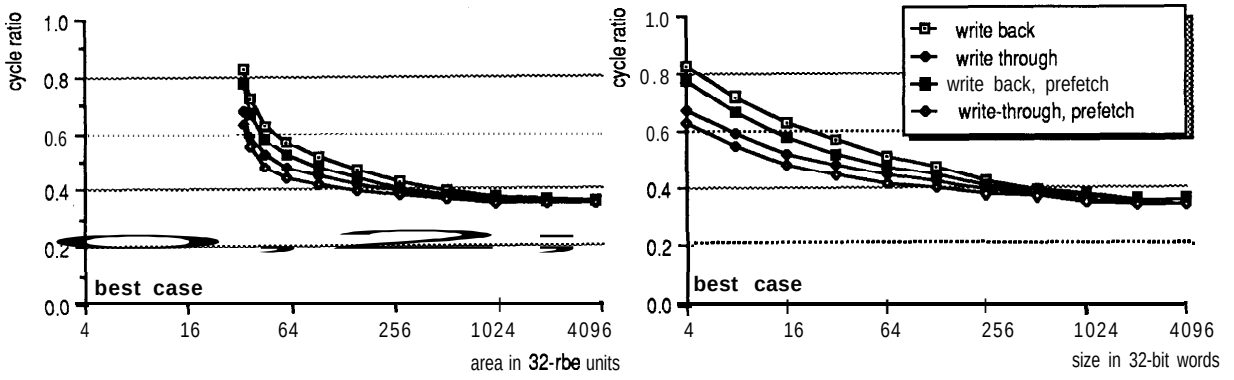


Figure 11: Best-case performance for register-cache combination.

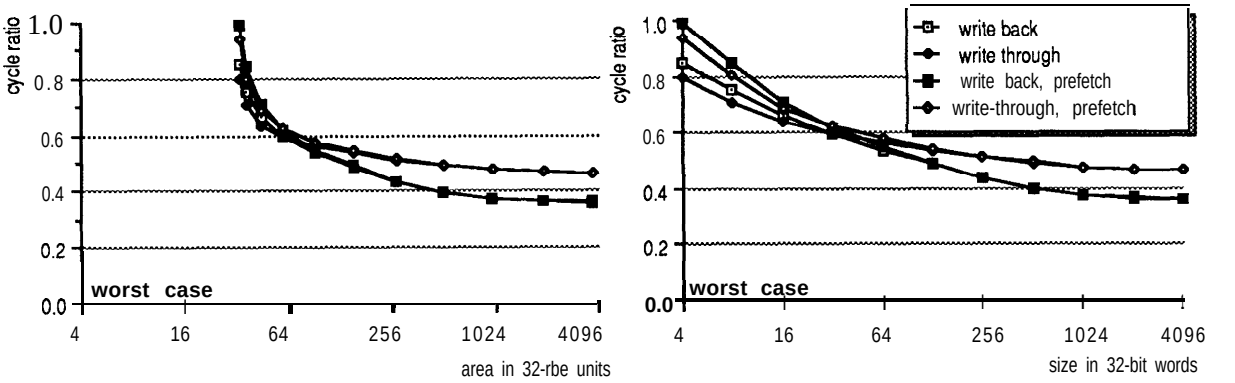


Figure 12: Worst-case performance for register-cache combination.

The best-case figures (Figures 11a and b) show for small caches a minimal difference with respect to prefetch strategies, but favors write-through over write-back. For large caches R_{miss} dominates the ratio because write-throughs, write-backs, and prefetches are invisible. Note that for larger caches the best-case model becomes more realistic.

The worst-case figures (Figures 12a and b) show a completely different picture from the best-case figures, but is similar to the traffic ratio in shape. Prefetching dominates the differences for small caches while the write strategy dominates the difference for large caches.

To interpret the absolute value of the cycle ratio for a particular organization recall that the

cycle ratio models the performance of a non-ideal memory system together with an ideal (zero time for breaks or branches) processor. A small register set combined with a 512-word cache and a 3-cycle memory yields approximately 40% of the memory cycles spent in the baseline model. The performance gain in real processor cycles is smaller and depends on the frequency and penalties of those instructions which leave the data-memory idle (e.g., branches and other pipeline breaks such as run-on instructions).

Cache-organization tradeoffs as a function of area utility

To assess tradeoffs in cache design, we consider the area and size effects with different line size and associativity on traffic ratio. In the following figures the left-hand graph **(a)** always shows the traffic ratio as a function of area and the right-hand graph **(b)** shows the traffic ratio as a function of size (storage capacity). All graphs show traffic relative to one particular organization.

Associativity

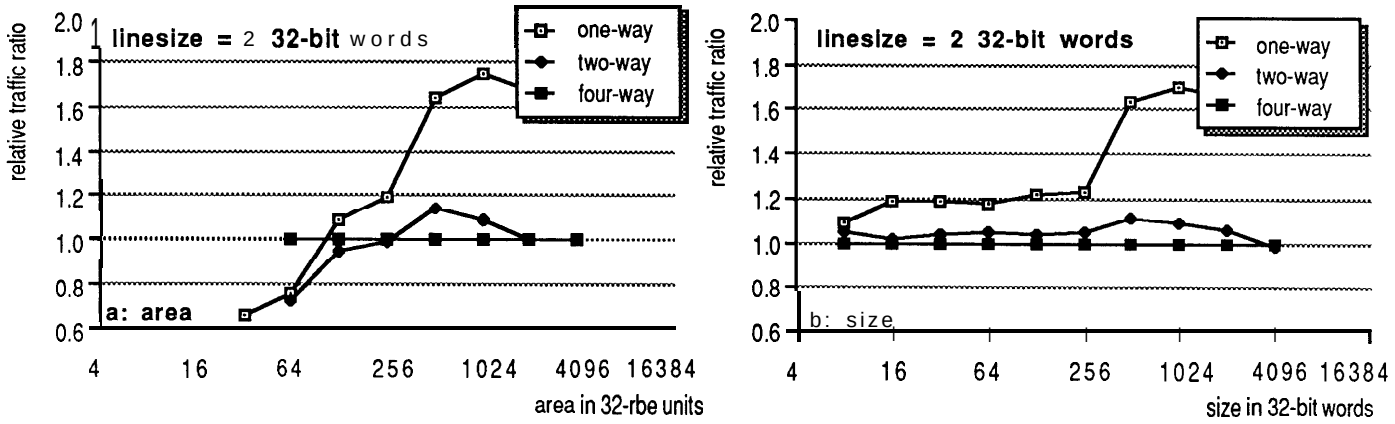


Figure 13: Performance as a function of set associativity, area, and size.

The traffic ratio (Figure 13b) relative to four-way associativity is relatively independent of cache size. Associativity of two-way and four-way performs better than direct-mapped for caches larger than 256 words. For caches larger than 4096 words the associativity differences reduce to zero. Cache traffic as a function of area (Figure 13a) deviates significantly from the traffic as a function of size for small caches (< 256 words). At these sizes, direct-mapped caches perform significantly better as function of area than as a function of size.

Figures 14a and b also show performance as a function of area, size, and associativity, but relative to a fully-associative cache. While for small caches the CAM cells for the tags outweigh the comparators of the set-associative (two-way and four-way) organizations, for larger caches

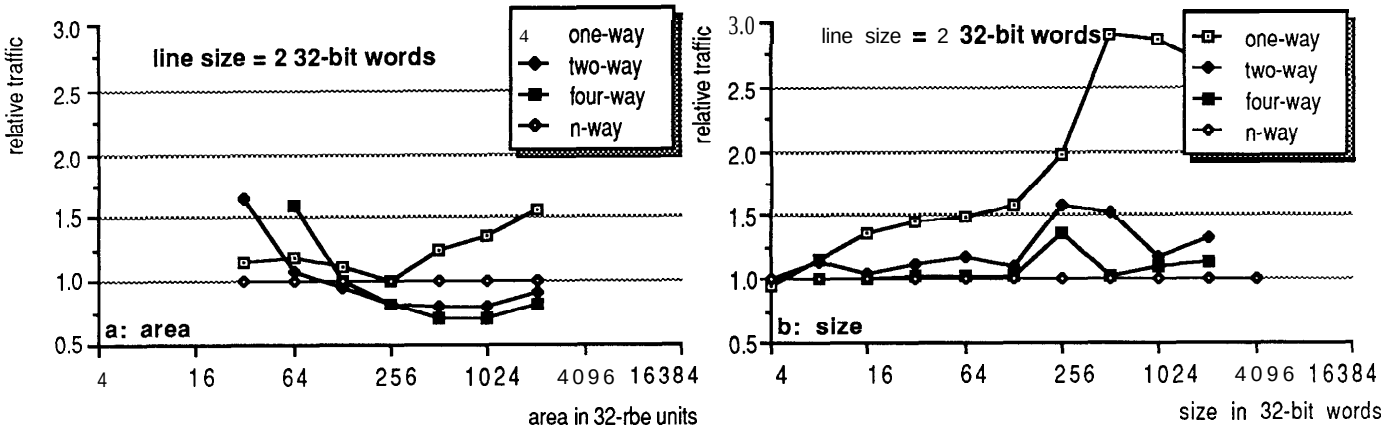


Figure 14: Full versus set associativity.

(> 128 rbe) the set-associative caches outperform fully-associative caches of the same area. At this line size, direct-mapped cache always produces equal or more traffic than fully-associative cache for all areas considered. The performance variations between fully and set-associative caches are significantly smaller when compared by area rather than by size (-25% - +50% vs +40% - +200%).

Line size

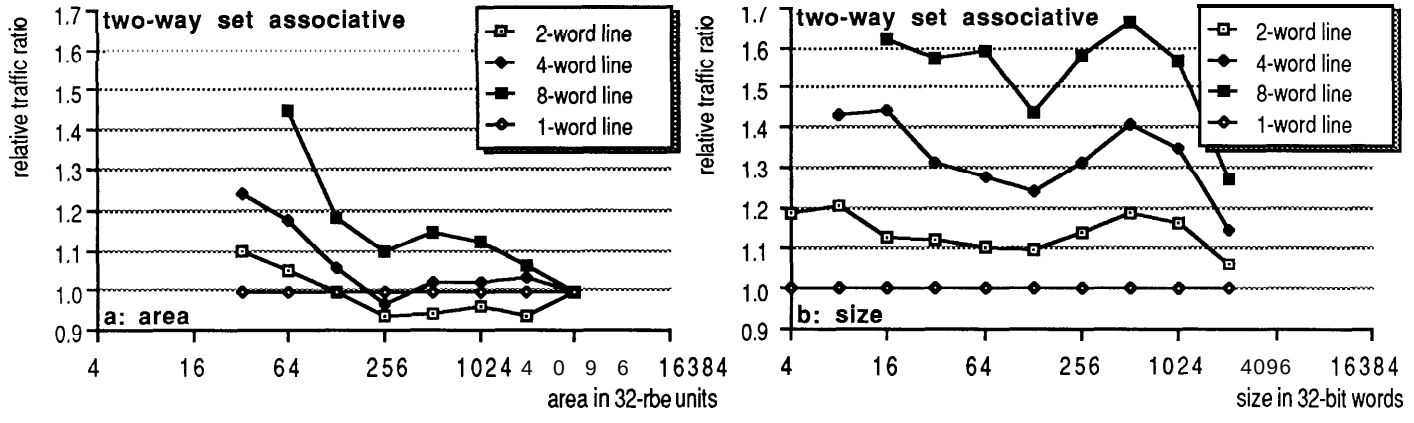


Figure 15: Performance as a function of line size, area, and size.

Figures 15a and b show relative traffic ratio as a function of area and size with line sizes ranging from one to eight words. The traffic ratio is relative to a cache with a line size of one word. The differences in relative traffic ratio among caches are quite large when compared by size (up to 65% for a cache with a line size of eight words. See Figure 15b), but become noticeably smaller

when compared by area, especially for medium-size caches (256 $\leq$ size < 4096 rbe). Figure 15a also shows a different performance order from Figure 15b.

Cache register-set tradeoffs as a function of area utility

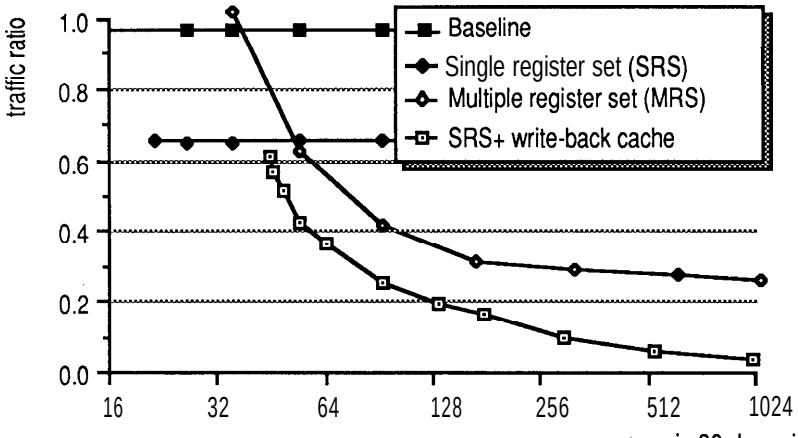


Figure 16: Traffic ratio as a function of buffer area for baseline, SRS, MRS, and SRS with Cache organization.

This section compares the performances of SRS, MRS, and SRS with cache. All register sets are allocated globally, the cache used in the SRS with cache combination is two-way set associative with a line size of two words and a subblocks of one word. The MRS organization is similar to the one found in RISC-II and SPARC, except for the addition of alias detection and resolution (16 registers for locals, 8 registers for input parameters, and 8 registers for output parameters) [3],

Every procedure or function call causes switching of register set. When insufficient sets are available the MRS overflows into memory (R_{ressav} in Section 2).

Figure 16 shows the traffic ratio of the three organizations relative to the baseline. Global register allocation becomes ineffective in reducing the traffic ratio below 65% when more than 8 registers are available for allocation (i.e., 16 general-purpose registers in total). Adding area to MRS decreases the ratio down to approximately 25% of the baseline, when it covers most runtime-stack accesses. Cache covers all accesses and reduces the traffic ratio to less than 5%. SRS with cache outperforms MRS for all areas considered.

Figures 17ab show the cycle ratios of the three organizations relative to the baseline for a two- and four-cycle memory respectively. For small buffers (< 256 **rbe),** the cycle lost during the cache lookup for a reference miss causes SRS with cache to have a slight disadvantage compared to MRS for two-cycle memories. Larger buffer areas, however, always yield an advantage for

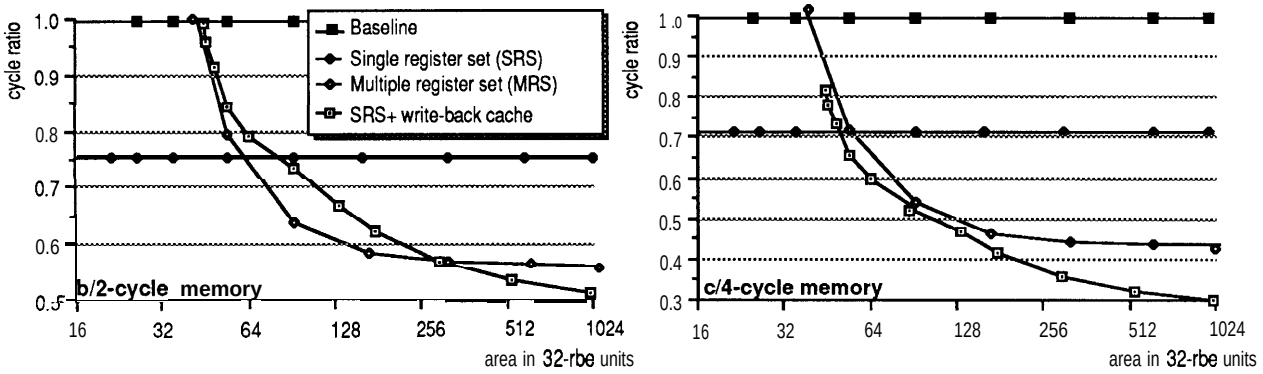


Figure 17: Cycle ratio as a function of buffer area for baseline, SRS, MRS, and Cache organization (2/4 cycle memory).

the SRS with cache organization. The MRS organization performs approximately as well as SRS with cache for three-cycle memories, but is outperformed for memories of four cycles or more for all area. It is important to note, however, that both MRS and SRS organizations perform better when more sophisticated register allocation strategies are used [39]. The SRS with cache combination profits less because only eight registers can be exploited (the SRS with cache combination has a total of 16 registers, but only eight is available to the register allocator).

5 Conclusion

In this paper we have developed an area-utility model for comparing data-buffering schemes. The performance model translates typical architecture measures into a performance measure, cycle *ratio.* The area model incorporates overhead area like drivers, sense amplifiers, tags, and control logic. Data cells are distinguished according to their delivered bandwidth in the model.

Using the utility model and traffic-ratio, we have compared the performance of various cache organizations as a function of area and size and investigated the relative performance of large register sets and caches. We have compared cache traffic and cycle ratio and established worst- and best-case cycle ratios. Both measures are required to get correct insight into cache behaviors. Comparing cache performance as a function of area, rather than size, leads to a significantly different set of organizational tradeoffs. Caches occupy more area per bit than register buffers for sizes of 128 words or less. For data caches, line size is a primary determinant of performance for small sizes while write policy becomes the primary factor for large caches. Multiple register sets ("register windows") have poorer performance than a single register set plus a cache (occupying the same area) except when the cache miss time (memory access) is less than three cycles.

6 Acknowledgements

Donald Alpert of Intel Corporation have kindly provided information regarding the i486 cache. Jonathan Levy of National Semiconductor Corporation, Richard Heye, Norman Jouppi, and Steve Morris of Digital Equipment Corporation, Leslie Kohn of Intel, Karl Molnar and David Lewis of General Electric, and Jeffrey Yetter of Hewlett-Packard have been helpful in clarifying some of the data in their papers. The authors wish to thank them all.

References

[1] M. D. Hill and A. J. Smith, "Experimental evaluation of on-chip microprocessor cache memories," in *The 11th Annual Symposium on Computer Architecture,* June 1984.

[2] D. Alpert and M. J. Flynn, "Performance tradeoffs for microprocessor caches," *IEEE Micro,* pp. 00–00, 1987.

[3] M. G. H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI.* PhD thesis, Computer Science Division (EECS), University of California Berkeley, Oct. 1983.

[4] J. M. Mulder, "Tradeoffs in Processor-Architecture and Data-Buffer Design," Tech. Rep. CSL-TR-87-345, Stanford University, Dec. 1987.

[5] M. J. Flynn, C. Mitchell, and H. M. Mulder, "And Now a Case for More Complex Instruction Sets," *IEEE Computer,* pp. 71-83, Sept. 20, 1987.

[6] J. Newkirk and R. Mathews, *The VLSI Designer's Library. The VLSI Systems Series,* Addison-Wesley, 1983.

[7] P. Chow, *The MIPS-X RISC Microprocessor.* Boston: Kluwer Academic Publishers, 1989.

[8] *Reference Manual and Product Data.* INMOS Ltd., 1985.

[9] F. F. Lee. Stanford University, ERL-452 Department of Electrical Engineering. Stanford, Ca 94305. Personal Communications.

[10] D. Alpert, "Memory Hierarchies for Directly Executed Language Microprocessors," Tech. Rep. 84-260, Computer Systems Laboratory, Stanford University, Stanford, Ca 94305, Jun. 1984.

[11] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz, and J. Hennessy, "On-chip Instruction Caches for High Performance Processors," in *Advanced Research in VLSI,* Stanford University, Mar. 1987.

[12] R. Heye and S. Morris. Digital Equipment Corporation. Private communications.

[13] J. Yetter. Hewlett-Packard. Private communications.

[14] T. L. Harman, *The Motorola 68020 and 68030 Microprocessors.* Prentice Hall, 1989.

[15] A. Marston, G. Burroughs, K. Chen, A. Desroches, G. Emerson, J. Hsu, R. Lee, F. Najami, A. Peebles, K. Peterson, B. Saperstein, J. Wangunhardjo, A. Wiemann, and R. Wu, "A 32b CMOS Single-chip RISC Type Processor," in *IEEE International Solid-State Circuit Conference,* pp. 28-29, Feb. 1987.

[16] J. Levy. National Semiconductor Corporation. Private communications.

[17] K. Kaneko, T. Okamoto, M. Nakajima, Y. Nakakura, S. Gokita, J. Nishikawa, Y. Tanikawa, and H. Kadota, "A 64b RISC Microprocessor for Parallel Computer System," in *IEEE International Solid-State Circuit Conference,* pp. 78-79, 1989.

[18] D. Archner, D. Deverell, F. Fox, F. Gronowski, A. Jain, M. Leary, A. Olesin, S. Persels, P. Rubinfeld, D. Schumacher, B Supnik, and T. Thrush, "A 32b CMOS Microprocessor with On-chip Instruction and Data Caching and Memory Management," in *IEEE* International *Solid-State Circuit Conference,* pp. 32–33,329–330, Feb. 1987.

[19] R. Conrad, R. Delvin, D. Dobberpuhl, B. Gieseke, R. Heye, G. Hoeppner, J. Kowaleski, M Ladd, J. Montana, S. Morris, R. Stamm, H. Tumblin, and R. Witek, "A 50 MIPS (Peak) 32/64b Microprocessor," in *IEEE International Solid-State Circuit Conference,* pp. 76-77 1989.

[20] N. P. Jouppi, J. Y. F. Tang, and J. Dion, "A 20 MIPS Sustained 32b Microprocessor with 64b Data Bus," in *IEEE International Solid-State Circuit Conference,* pp. 84-85, 1989.

[21] M. Horowitz, J. Hennessy, P. Chow, P. Gulak, J. Acken, A. Agarwal, C.Y. Chu, S. McFarling, S. Prsybylski, S. Richardson, A. Salz, R. Simoni, D. Stark, P. steenkiste, S. Tjinag, and M. Wing, "A 32b Microprocessor with On-chip 2k Byte Instruction Cache," in *IEEE Internationul Solid-State Circuit Conference,* pp. 30–31,328, Feb. 1987.

[22] H. Kadota, J. Miyake, I. Okabayashi, T. Maeda, T. Okamoto, Y. Takagi, K. Kagawa, and E. Ichinohe, "A CMOS 32b Microprocessor with On-chip Cache and Transmission Lookahead

Buffer," in *IEEE International Solid-State Circuit Conference,* pp. 36–37,332–333, Feb. 1987.

[23] T. S. Perry, "Intel Secret Is Out," *IEEE Spectrum,* pp. 22-28, Apr. 1989.

[24] D. Alpert. Intel Corporation. Private communications.

[25] J. Yetter, M. Forsyth, W. Jaffe, D. Tanksalvala, and J. Wheeler, "A 15 MIPS 32b CMOS Microprocessor ," in *IEEE International Solid-State Circuit Conference,* pp. 26-27, 1987.

[26] K. Molner, C.-Y. Ho, D. Staver, B. Davis, and R. Jerdonek, "A 40MHZ 64-bit Floating Point Processor," in *IEEE International Solid-State Circuit Conference,* pp. 48-49, 1989.

[27] D. K. Lewis, T, J. Wyman, M. J. French, and F. S. Boericke II, "A 40MHZ 32b Microprocessor with Instruction Cache," in *IEEE International Solid-State Circuit Conference,* pp. 30-31, 1988.

[28] L. Kohn. Intel Corporation. Private communications.

[29] *280,000 CPU Preliminary Technical Manual.* Zilog Inc., Sep. 1984.

[30] *I80486 Reference Manual.* Intel Corp.

[31] *M68020 32-bit Microprocessor User's Manual.* Motorola, second ed., 1985.

[32] A. D. Berenbaum and D. R. Ditzel, "Architectural Innovations in the CRISP Microprocessor," in *Proceeding of the IEEE Spring COMPCON*, pp. 91–105, Feb. 23-27, 1987.

[33] *M68030 Advanced Information.* Motorola Inc., 1987.

[34] *M68040 Advanced Information.* Motorola Inc., 1989.

[35] *I80860 Reference Manual.* Intel Corp.

[36] D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 Microprocessor," in *IEEE International Conference on Computer Design,* Oct. 1987.

[37] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein, "Register allocation via coloring," *Computer Languages,* pp. 47-57, 1981.

[38] F. C. Chow, *A Portable Machine-Independent Global Optimizer - Design and Measurements.* PhD thesis, Computer Systems Laboratory, Department of Electrical Engineering and Computer Science, Stanford University, Dec. 1983.

[39] J. M. Mulder and M. J. Flynn, "Tradeoffs in Processor-Architecture and Memory Design,". Submitted for review.