An Area Model for On-Chip Memories and its Application

Johannes M. Mulder, *Member, IEEE*, Nhon T. Quach, *Student Member, IEEE* and Michael J. Flynn, *Fellow, IEEE*

Abstract—**In the implementation of a processor, it is often necessary to abstract cost constraints into architecture measures for making trade-offs. An important cost measure for an on-chip memory is its occupied silicon area. Since the performance of an on-chip memory is characterized by size (storage capacity), a mapping from size to area is needed. Simple models have been proposed in the past for such a purpose. These models, however, are of unproven validity and only apply when comparing relatively large buffers (≥ 128 words for caches, ≥ 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 a memory cell and includes such buffer overhead as control logic, driver logic, and tag storage, thereby permitting comparison of data buffers of different structures and arbitrary sizes. The model gave less than 10% error when verified against real caches and register files. We then show that comparing cache performance as a function of area, rather than size, leads to a significantly different set of organizational trade-offs.

I. Introduction

PERFORMANCE requirements and costs constraints placed on an implementation directly influence processor and memory architecture design decisions. In the design of an architecture, it is necessary to abstract these cost constraints to architectural measures for making trade-offs. An important cost measure for an on-chip buffer is its occupied silicon area. Since the performance of a data buffer is characterized by its size (storage capacity), a mapping from size to area is needed.

Hill and Smith [1] and Alpert and Flynn [2] have used simple area models for such a purpose. These simple models account for tag and line-status bits in addition to the data bits. The difference in area between the content addressable memory (CAM) cells and the normal storage cells is also included [2]. The validity of these simple models, however, has thus far remained unproven. 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 versus register), the simple area models do not suffice. In small caches the area overhead dominates, but is not included in the simple models. When comparing buffers of different structures, it becomes important to consider the supplied bandwidth of the buffers 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 difference between the simple models and the present model is shown in Fig. 1 for a two-way set associative cache, a fully associative cache, and a register set. For small cache, the differences in area predicted by the models are significant.

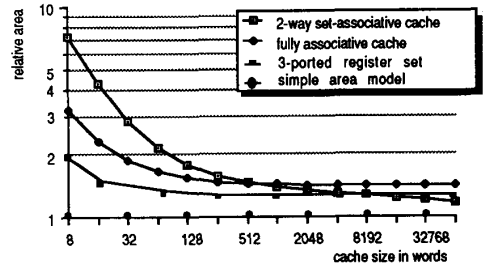


Fig. 1. Proposed area model relative to simple models.

The area model presented in this paper corrects these deficiencies by 1) including data bits, tag bits, and overhead logic (i.e., drivers and comparators) in the model, 2) considering the effects of bandwidth on individual memory cells, and 3) establishing the model validity by comparing the model prediction with real caches and register files. The area model is presented in Section II and verified in Section III. Section IV follows with an application of the area model to assess cache organization trade-offs.[1] Concluding remarks are given in Section V.

II. Area Model

In the present area model, the total amount of area occupied by a combination of buffers is simply the sum of the individual areas, as shown in Fig. 2. We ignore wiring overhead necessary to combine the buffers for modeling simplicity.

A. Area Unit

Although the most obvious unit for area is square micrometers, the unit for the present area model is a technology-

Manuscript received March 14, 1990; revised October 8, 1990. This work was supported by the NSF under Contract MIP88-22961 using facilities provided by NASA under Contract NAGW 419.

J. M. Mulder is with the Department of Electrical Engineering, Delft University of Technology, 2600 AG Delft, The Netherlands.

N. T. Quach and M. J. Flynn are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.

IEEE Log Number 9041648.

[1]Although the area model presented in this paper allows us to compare buffers of different structures (e.g., caches versus register files) as mentioned previously, doing so requires the introduction of a timing model for each type of buffer with different timing characteristics. Due to space limitation, only cache design trade-offs are considered in this paper. The reader is referred to [3] for a comparison of relative cycles of caches and register files.

0018-9200/91/0200-0098$01.00 ©1991 IEEE

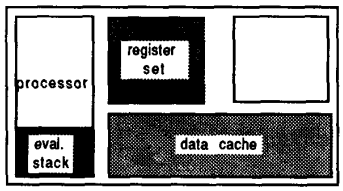


Fig. 2. Area of on-chip data memory as chip cost function. $A_{total} = A_{e\text{-}stack} + A_{r\text{-}set} + A_{d\text{-}cache}$.

independent notion of a register-bit equivalent or rbe. The advantage of this is the relatively straightforward relation between area and size, facilitating 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 crossing 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 present area model uses three types of storage cells with different bandwidths: a six-transistor static cell with high bandwidth, a six-transistor static cell with medium bandwidth, and a three-transistor dynamic ell with low bandwidth [4] (henceforth referred to, respectively, as register cell, static cell, and dynamic cell). The area unit, rbe, equals the area of the register cell.[2] 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.

B. Register Set and Memory Areas

Register buffers are generally an integral part of the data path. These buffers use high-bandwidth register cells, normally consisting of a read port and a port that can be used for reading and 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 "three-ported cells," though they actually have less hardware overhead than ones with two read ports and a separate write port.[3] 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 6 rbe. Fig. 3(a) shows the area model of a register buffer or on-chip memory. The total area in rbe for a single array is

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

[2]MIPS-X [5] is used as the basis for certain empirical parameterizations. This experimental microprocessor was implemented in CMOS technology with 2-μm minimum geometry. Its register cell was 37×55 μm and its cache storage cell (static) was 30×40 μm.

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

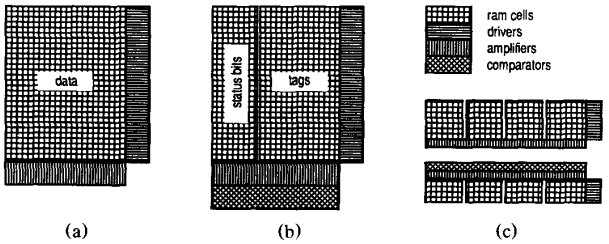


Fig. 3. Data- and tag-area model.

where $registers_w$ is the number of registers in words, L_{sense_amp} is the length of the bit-line sense amplifiers, $datawidth_b$ is the width of the data path in bits, and W_{driver} is the width of the drivers, all in units 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 4 bytes or 32 b.) From MIPS-X data, $L_{sense-amp}$ and W_{driver} are 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 b for all register buffers (or register files) unless otherwise stated.

Large on-chip buffers, other than register buffers, are generally associated with cache or a similar structure [6]. The bandwidth requirements of these buffers or memories are significantly 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,[4] we scale the equation for the register area model (i.e., equation (2)) by 0.6 for the 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 (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 [4]. The area of dynamic memory is approximated as

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

C. Cache Areas

The area occupied by caches is more complicated. Besides data bits, which we have modeled previously, a cache consists of area for address tags, dirty and valid bits, comparators,

[4]Here, it can be confusing. Cell area refers to the area of one cell, register or memory areas refer to the areas of the whole register buffers and the whole memory array, respectively.

and control logic. The control logic is usually implemented in a 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 as compared with registers.

1) Set-Associative Caches: The tag area for a set-associative cache (sac) 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 $300\times30\ \mu m^2$, mainly to allow 24 metal wires with 10-μm pitch to cross. Based on these figures the area model assumes a comparator area of 6×0.6 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 gigaword covered by the cache. The number of status bits per line depends on the transfer-unit[5] 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 PLA's 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 [7], a fairly low estimate.

Fig. 3(a) shows the layout of a cache array (data), and Fig. 3(b) shows the layout of a directory area. Fig. 3(c) 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 buses, the total area of a set-associative cache is

$$area_{sac} = pla + data + tags + status.$$

The area of the different items are a function of the storage capacity $size_b$, 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 $tags$, the total number of tag and status bits tsb_b are

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

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

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

where γ equals 2 for a write-back cache and 1 for a write-through cache. According to Fig. 3(c) the area of a set-associative cache using static cells is

$$area_{sac} = 130 + 0.6(line_b \cdot assoc + 6)\left(\frac{tags}{assoc} + 6\right) + 0.6(tsb_b \cdot assoc + 6)\left(\frac{tags}{assoc} + 6 + 6\right) \text{ rbe}$$

$$= 195 + 0.6 \cdot ovhd_1 \cdot size_b + 0.6 \cdot ovhd_2 \cdot tsbits_b \text{ rbe}$$

where

$$ovhd_1 = 1 + \frac{6 \cdot assoc}{tags} + \frac{6}{line_b \cdot assoc}$$

and

$$ovhd_2 = 1 + \frac{12 \cdot assoc}{tags} + \frac{6}{tsb_b \cdot assoc}.$$

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

$$area_{sac} = 195 + 0.3 \cdot ovhd_3 \cdot size_b + 0.3 \cdot ovhd_4 \cdot tsbits_b \text{ rbe}$$

where

$$ovhd_3 = 1 + \frac{6 \cdot assoc}{tags} + \frac{12}{line_b \cdot assoc}$$

and

$$ovhd_4 = 1 + \frac{12 \cdot assoc}{tags} + \frac{12}{tsb_b \cdot assoc}.$$

Fig. 4(a) shows the effect of line sizes on a direct-mapped cache area relative to the storage capacity ($area_{rbe}/size_b$). The area reduction is rather small when moving from a two-word line to a 16-word line since the tag-area reduction is partially compensated by an increase in transfer-unit status bits. Fig. 4(b) 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. Fig. 4(c) shows the area of a direct-mapped cache and set-associative caches relative to the area of a three-ported register set. For the same storage capacity, caches generally occupy more area than registers for small sizes (the exact crossover point depends strongly on line size) because the cache overhead dominates the cache area at these sizes. For larger sizes, the smaller storage cells in the cache provide a total cache area smaller than the register set. A four-way set-associative cache of 1024-word size with two-word lines, for example, only takes 75% of the area of a register file of 1024 words.

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

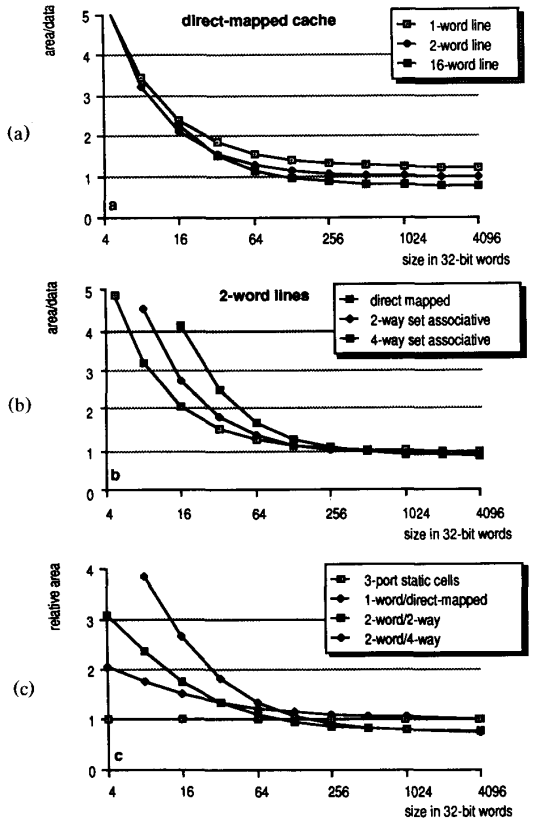


Fig. 4. Relative area for associative caches as a function of line size, associativity, and provided storage.

2) Fully Associative Caches: The tag area of a fully associative cache (fac) 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 [8] assumed CAM cells to be twice the size of a static cell (1.2 rbe), basing his assumption on data for the Z80,000. Our tag-area model assumes the same ratio. Fig. 5 shows the 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 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

$$area_{fac} = pla + data + status + CAM$$

$$= 130 + 0.6(tags + 6)\left(\beta \cdot line_{b\_data} + 6\right)$$

$$+ 0.6\left(\sqrt{2} \cdot tags + 6\right)\left(\sqrt{2} \cdot line_{b\_CAM} + 6\right) \text{ rbe} \quad (3)$$

where $\beta = 1 + \gamma / transfer_b$ and $line_{b\_CAM} = 30 - \log_2(line_w)$. The derivation of the equation follows the static memory one in the previous subsection. The CAM cells are assumed to have an aspect ratio of 1, so that the width and length are equal ($\sqrt{2}$ rbe). Defining $size_{b\_CAM} = tags \cdot line_{b\_CAM}$, expanding, and rearranging, we rewrite (3) as

$$area_{fac} = 175 + 0.6 \cdot \beta \cdot ovhd_5 \cdot size_{b\_data} + 1.2 \cdot ovhd_6 \cdot size_{b\_CAM} \text{ rbe}$$

where

$$ovhd_5 = 1 + \frac{6 \cdot \beta}{tags} + \frac{12}{\beta \cdot line_b}$$

and

$$ovhd_6 = 1 + \frac{8.5}{tags} + \frac{8.5}{line_b}.$$

The effect of organization on the area of fully associative caches is shown in Fig. 6(a). Increasing the line size has significantly more effect for fully associative caches than for

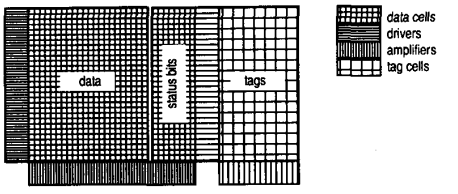


Fig. 5. Fully associative cache layout.

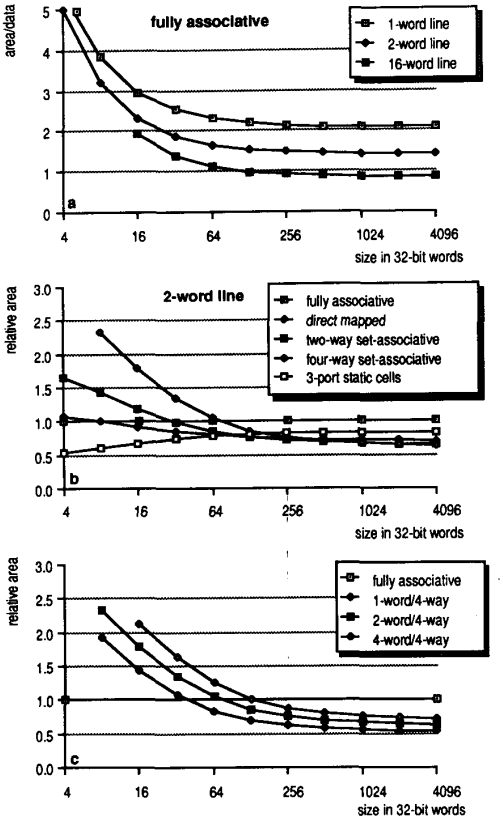


Fig. 6. Relative area of fully associative caches as a function of line size and provided storage.

direct-mapped ones (Fig. 4(a)). Moving from one-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. Fig. 6(b) shows the area of various cache and register configurations relative to the area occupied by a fully associative cache of indicated sizes in 32-b words. Generally, fully associative caches occupy the most area per bit for sizes in excess of 64 words and registers occupy the next most area per bit with direct-mapped and set-associative caches occupying the least area over the same range. Similarly from Fig. 6(c), fully associative caches occupy more area than four-way set-associative caches at large sizes with the crossover point depending on the line size.

D. 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 capacity. Second, the area only depends on the buffer organization and not on the layout specifics. Finally, the aspect ratio is not significant for modeling purposes. We consider each of these assumptions in more detail below.

1) 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 32×32 b for register buffers and 2 kilobytes for caches.

2) Influence of Layout on Area: In any implementation, the amount of wasted area depends on the actual 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.

3) Aspect Ratio: Fig. 7 illustrates the relation between 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 Fig. 3(c), the aspect ratios may become large. Fig. 7(a) 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 (Fig. 7(b)) and four times (Fig. 7(c)) improves the aspect ratio from 7 to 2 and to 0.6 but increases the area by 15% and by 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 factors. First, every fold requires its own drivers for both tag and data arrays and, second, every fold increases the area for both address and data buses supplying the cache. While the aspect ratio in cache design can be important [9], we chose to ignore it to simplify modeling. This necessarily limits the achievable accuracy of our model.

III. Verification of Area Model

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

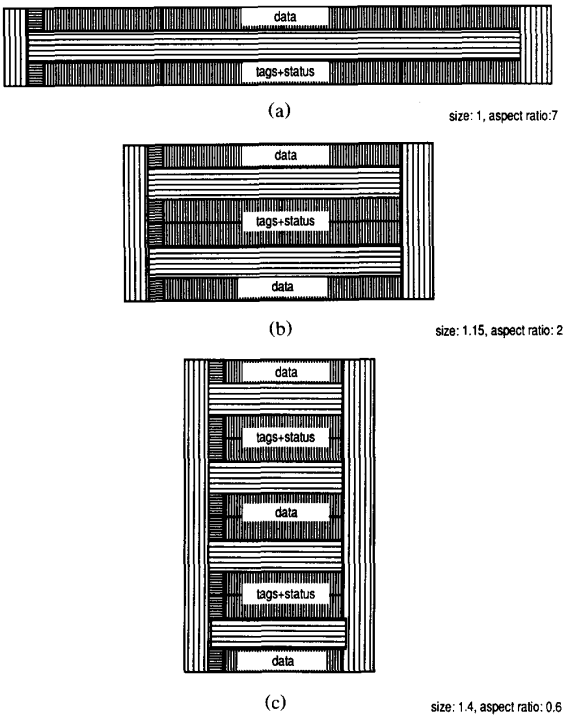


Fig. 7. Aspect ratio and area change as a function of layout.

MIPS-X data, which is built with a 2-μm technology. The use of TF permits comparison of caches and register files across generations of technologies (e.g., 1 versus 2 μm). Since TF is an area scale factor, it can be obtained simply as

$$\mathrm{TF} = \left(\frac{\textit{minimum geometry in } \mu m}{2} \right)^2.$$

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 (1) as

$$area = \left(registers_w + L_{sense\_amp}\right)\left(datawidth_b + W_{dec} \cdot N_{dec}\right) \cdot PF \text{ rbe} \quad (4)$$

where W_{dec} is the width and N_{dec} is the total number of the decoders,[6] and PF is an empirical factor accounting for the number of register ports in the register file. W_{dec} and PF are modeled as

$$W_{dec} = \alpha \cdot datawidth_b \quad (5)$$

and

$$PF = \left[1 + 0.25\left(N_{bit\_lines} - 2\right)\right]. \quad (6)$$

[6]In a register file, the word-line drivers are the decoders. We used decoders in (4) but drivers in (1).

TABLE I
COMPARISON OF ACTUAL AND PREDICTED CACHE AREAS

μP	TECH. (μm)	TYPE[♭]	SIZE (Bytes)	AREA[‡] (Kμm²)	MODEL (Kμm²)	ERROR (%)[†]	REF.
M68020	2.0	I,1w	246	4449	4048	−9.0	[13]
M68030	1.2	I,1w	256	2445	2184	−10.7	[13]
	1.2	D,1w	256	2345	2184	−6.9	
HP RISC	1.6	I,1w	256	2775	3134	12.9	[14]
NS32532	1.25	I,2w	512	3776	3246	−14.0	[15]
	1.25	D,1w	1K	7699	6153	−20.1	
Matsushita2	1.2	I,1w	1K	9448	8596	−9.0	[16]
DEC1 (μVAX)	2.0	I/D,2w	1K	8750	8705	0.5	[17]
DEC2	1.5	I,S	1K	9448	9858	4.3	[18], [10]
	1.5	D,1w	2K	20125	16935	−15.9	
DEC3	1.5	I,1w	2K	18463	15773	−14.6	[19]
MIPS-X	2.0	I,S	2K	27517	27545	0.1	[20]
Matsushita1	1.0	I,2w	2K	11188	10448	−6.6	[21]
i860	1.0	I,2w	4K	13347	12805	−4.1	[22]
	1.0	D,2w	8K	26977	23904	−11.4	
i486	1.0	I/D,4w	8K	26000	26500	1.9	[23]

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; 1w—Direct-mapped; 2w—two-way set-associative; 4w—four way set-associative; S—Sector cache.
[‡]Measured or reported areas.
[†]Percent error is calculated as:

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

TABLE II
COMPARISON OF ACTUAL AND PREDICTED REGISTER-FILE AREAS

μP	TECH. (μm)	PORTS R/W/(R/W)	$N_{bit\_lines}^{\sharp}$	TYPE[♭]	SIZE (bits)	AREA[‡] (Kμm²)	MODEL (Kμm²)	ERROR[†] (%)	REF.
MIPS-X	2.0	2/0/1*	2	I	32×32	3330	3217	−3.4	[20]
DEC3	1.5	1/0/1	4	I	48×32	3534	3523	−0.3	[19]
HP1	1.5	2/2/0	4	I	31×32	3450	3737	8.0	[24]
GE1	1.2	2/1/1	4	FP	8×64	4760	4558	−4.2	[25]
GE2	1.2	2/1/1	4	FP	21×32	3734	4396	17.7	[26]
i860	1.0	3/2/0	5	FP	8×128	2581	2343	−9.2	[27]

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:

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

*MIPS-X's register file has three sets of decoders but has only two bit lines (see text).

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

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

where N_{bit_ines} 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, (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 α 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 two bit lines (specifically, over the MIPS-X register file).

Table I compares the actual cache sizes with the present area model prediction. The cache areas in the "AREA" column are in thousands of square micrometers, 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 modeled here using the read equation for dynamic memory. The DEC2 processor uses four-transistor

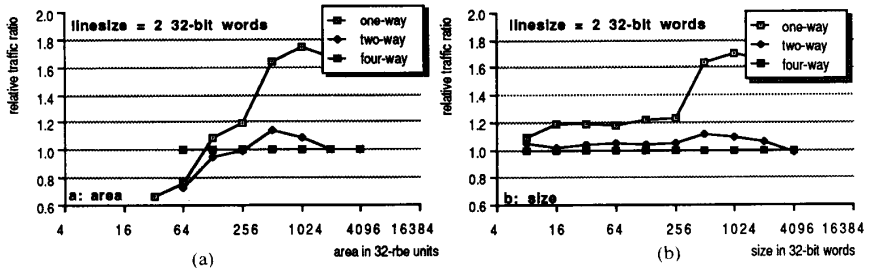


Fig. 8. Performance as a function of set associativity, area, and size.

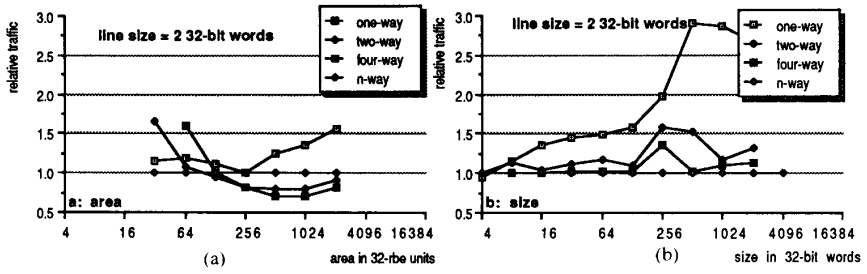


Fig. 9. Full versus set associativity.

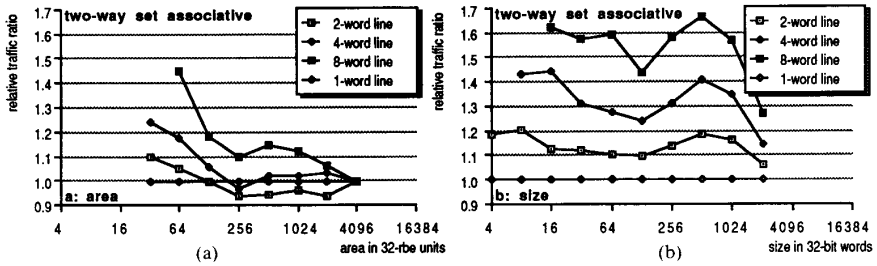


Fig. 10. Performance as a function of line size, area, and size.

cells in the cache, which are about 10% smaller than the six-transistor static cells assumed in the present study [10]. The data and error given in Table I 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 II 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 [11]. 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 37×100 μm^2 in a 1.2-μm technology. We accounted for this by using this given size as the area unit (instead of rbe). The data presented in Table II include all these adjustments.

IV. Cache Organization Trade-Offs as a Function of Area

To assess trade-offs in cache design, we consider the area and size effects with different line size and associativity on traffic ratio. Traffic ratio is defined here as the ratio of the total number of words transferred between the cache and the memory to the total number of cache accesses. In essence, traffic ratio measures the cache effectiveness in reducing memory traffic. Only write-back caches are investigated in this study and all caches use a cell size of 0.6 rbe. The benchmarks used consist of five medium-sized programs (dynamic size of 2.5 to 35 million bytes) generally representative of a workstation environment (nonscientific). The reader is referred to [12] for additional information.

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.

A. Associativity

The traffic ratio of caches with different set associativity (Fig. 8(b)) 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 reduces to zero. Cache traffic as a function of area (Fig. 8(a)) 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 a function of area than as a function of size.

Fig. 9(a) and (b) also shows 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 (> 128 rbe) the set-associative caches outperform fully associative caches of the same area. At this line size, a direct-mapped cache always produces equal or more traffic than a 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% to $+50\%$ versus $+40\%$ to $+200\%$).

B. Line Size

Fig. 10(a) and (b) shows 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 Fig. 10(b)), but become noticeably smaller when compared by area, especially for medium-size caches ($256 \leqslant \text{size} < 4096$ rbe). Fig. 10(a) also shows a different performance order from Fig. 10(b).

V. Conclusion

In this paper, we have presented an area model suitable for comparing data buffers of different organizations (e.g., caches versus register files) and arbitrary sizes. The model incorporates such overhead area as drivers, sense amplifiers, tags, and control logic. Data cells are distinguished according to their delivered bandwidth in the model. The model gave less than 10% error when verified against real caches and register files.

Comparing caches and register files in terms of area reveals that for the same storage capacity, caches generally occupy more area per bit than register files for small caches because the overhead dominates the cache area at these sizes. For larger caches, the smaller storage cells in the cache provide a smaller total cache area per bit than the register set. The exact crossover point depends strongly on the line size (Fig. 4).

Studying cache performance (traffic ratio) as a function of area with the present area model, we found: 1) for small caches (less than the area occupied by 256 register bits—rbe —or 32 bytes), direct-mapped caches perform significantly better relative to four-way set-associative caches (Fig. 9); and 2) for caches of medium areas (between 256 rbe and 4096 rbe), both direct-mapped and set-associative caches perform better relative to fully associative caches with set-associative caches actually outperforming fully associative caches (Fig. 8). Furthermore, for set-associative caches of these medium areas, line size has far smaller effects on traffic ratio for caches of the same area (Fig. 10(c)).

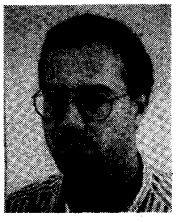
Acknowledgment

D. Alpert of Intel Corporation kindly provided information regarding the i486 cache. J. Levy of National Semiconductor Corporation, R. Heye, N. Jouppi, and S. Morris of Digital Equipment Corporation, L. Kohn of Intel, K. Molnar and D. Lewis of General Electric, and J. Yetter of Hewlett-Packard have been helpful in clarifying some of the data in their papers. The authors wish to thank them all. The authors wish to also thank the referees for their valuable comments on the paper.

References

[1] M. D. Hill and A. J. Smith, "Experimental evaluation of on-chip microprocessor cache memories," presented at the 11th Annual Symp. Computer Architecture, June 1984.
[2] D. Alpert and M. J. Flynn, "Performance tradeoffs for microprocessor caches memories," *IEEE Micro*, pp. 44–54, Aug. 1988.
[3] J. M. Mulder, N. T. Quach, and M. J. Flynn, "An area-utility model for on-chip memories and its application," Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-90-413, Feb. 1990.
[4] J. Newkirk and R. Mathews, *The VLSI Designer's Library* (The VLSI Systems Series). Reading, MA: Addison-Wesley, 1983.
[5] P. Chow, *The MIPS-X RISC Microprocessor*. Boston: Kluwer, 1989.
[6] INMOS Ltd., *Reference Manual and Product Data*, Bristol, England, 1985.
[7] F. F. Lee, Dept. Electrical Engineering, Stanford Univ., Stanford, CA, private communication, 1989.
[8] D. Alpert, "Memory hierarchies for directly executed language microprocessors," Computer Systems Lab., Stanford Univ., Stanford, CA, Tech. Rep. 84-260, June 1984.
[9] A. Agarwal *et al.*, "On-chip instruction caches for high performance processors," in *Advanced Research in VLSI*, Stanford Univ., Stanford, CA, Mar. 1987.
[10] R. Heye and S. Morris, Digital Equipment Corporation, Hudson, MA, private communication, 1989.
[11] J. Yetter, Hewlett-Packard, private communication, 1989.
[12] M. J. Flynn, C. Mitchell, and J. M. Mulder, "And now a case for more complex instruction sets," *IEEE Computer*, pp. 71–83, Sept. 20, 1987.
[13] T. L. Harman, *The Motorola 68020 and 68030 Microprocessors*. Englewood Cliffs, NJ: Prentice Hall, 1989.
[14] A. Marston *et al.*, "A 32b CMOS single-chip RISC type processor," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 28–29.
[15] J. Levy, National Semiconductor Corporation, private communication, 1989.
[16] K. Kaneko *et al.*, "A 64b RISC microprocessor for parallel computer system," in *ISSCC Dig. Tech. Papers*, 1989, pp. 78–79.
[17] D. Archner *et al.*, "A 32b CMOS microprocessor with on-chip instruction and data caching and memory management," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 32–33, 329–330.
[18] R. Conrad *et al.*, "A 50 MIPS (peak) 32/64b microprocessor," in *ISSCC Dig. Tech. Papers*, 1989, pp. 76–77.
[19] N. P. Jouppi, J. Y. F. Tang, and J. Dion, "A 20 MIPS sustained 32b microprocessor with 64b data bus," in *ISSCC Dig. Tech. Papers*, 1989, pp. 84–85.
[20] M. Horowitz *et al.*, "A 32b microprocessor with on-chip 2k byte instruction cache," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 30–31, 328.

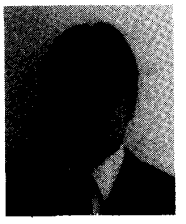
[21] H. Kadota *et al.*, "A CMOS 32b microprocessor with on-chip cache and transmission lookahead buffer," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 36–37, 332–333.
[22] T. S. Perry, "Intel secret is out," *IEEE Spectrum*, pp. 22–28, Apr. 1989.
[23] D. Alpert, Intel Corporation, private communication, 1989.
[24] J. Yetter, M. Forsyth, W. Jaffe, D. Tanksalvala, and J. Wheeler, "A 15 MIPS 32b CMOS Microprocessor," in *ISSCC Dig. Tech. Papers*, 1987, pp. 26–27.
[25] K. Molner, C.-Y. Ho, D. Staver, B. Davis, and R. Jerdonek, "A 40 MHz 64-bit floating point processor," in *ISSCC Dig. Tech. Papers*, 1989, pp. 48–49.
[26] D. K. Lewis, T. J. Wyman, M. J. French, and F. S. Boericke II, "A 40 MHz 32b microprocessor with instruction cache," in *ISSCC Dig. Tech. Papers*, 1988, pp. 30–31.
[27] L. Kohn, Intel Corporation, private communication, 1989.



Johannes M. Mulder (S'82–M'87) received the M.S. degree from Delft University of Technology, Delft, The Netherlands, and the Ph.D. degree from Stanford University, Stanford, CA.

He is an Assistant Professor in the Department of Electrical Engineering, Delft University of Technology. His main research interests are computer architecture, compilers and VLSI design for high-speed computing, and computer-aided architecture and system design. He is the principal investigator of the SCARCE project, which concerns the design of application-specific processors for high-speed embedded controllers.

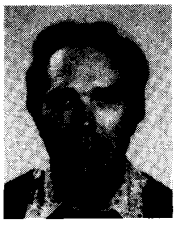
Dr. Mulder is a member of the IEEE Computer Society and the ACM.



Nhon T. Quach (S'87) received the B.S. degree from the University of Texas at Austin in 1982 and the M.S. degree from the Massachusetts Institute of Technology, Cambridge, in 1984. He is currently a Ph.D. candidate at Stanford University, Stanford, CA, where he researches in the area of high-speed computer arithmetic.

From 1984 to 1987 he was one of the principal developers of a 1-μm CMOS process at the Fairchild Advanced Research Laboratory. His other research interests include computer architecture, compilers, and VLSI circuits and systems design.

Mr. Quach is a member of the IEEE Computer Society and the ACM.



Michael J. Flynn (M'56–SM'79–F'80) is a Professor of Electrical Engineering at Stanford University, Stanford, CA. His experience includes ten years at IBM Corporation working in computer organization and design. He was also a faculty member at Northwestern University and Johns Hopkins University, and the Director of Stanford's Computer Systems Laboratory from 1977 to 1983.

Mr. Flynn has served as vice president of the IEEE Computer Society and was founding chairman of CS's Technical Committee on Computer Architecture, as well as ACM's Special Interest Group on Computer Architecture.