Universidade Federal do Paraná

Departamento de Informática

Renato Carmo, Giancarlo C Heck, Roberto A Hexsel

Unpopular Addresses Should Not Pollute the Cache

Relatório Técnico
RT_DINF 002/2011

Curitiba, PR
2011

Unpopular Addresses Should Not Pollute the Cache

Renato Carmo, Giancarlo C Heck, Roberto A Hexsel
Universidade Federal do Paraná
Departamento de Informática
CEP 81.531-990 – Curitiba, PR, Brasil
giancarloheck@gmail.com,{roberto,renato}@inf.ufpr.br

Abstract

The *popularity* of a memory word is the fraction of references to this word over the total of memory references in the execution of a program. In this paper we formally define the metric *popularity of reference* and explain the reference patterns of the CommBench programs in terms of this metric. We investigate memory systems that are suitable for embedded applications that often exhibit a bimodal distribution of popularity: a few thousand words are very popular while the rest of the data are referenced once or twice, thus polluting the cache. We introduce a new design for a pollution control cache named Pollution Control Victim Cache (PCVC) that is simpler and yields better performance than designs of comparable complexity. Detailed simulations with SimpleScalar were executed with the CommBench benchmarks. We compare performance$\times\{\text{area}, \text{energy}\}$ of four designs for the top level of a memory hierarchy: direct mapped primary cache (L1); 2-way associative L1; direct mapped L1 with an Pollution Control Cache; and direct mapped L1 with a PCVC. The area and energy models were simulated with CACTI. For capacities in the range 4-8 Kbytes the PCVC yields the best overall performance$\times\{\text{area}, \text{energy}\}$ of the designs investigated.

1 Introduction

The design of embedded systems is constrained by several parameters that include power, size, weight, reliability, time to market, performance, and cost. Cache memories provide performance gains but do have an impact on IC size, cost (area and design/test effort) and power. Ideally, one would like to use the smallest cache because of IC size, cost and reduced power, but needs the performance that would be achieved with the largest capacity.

In many embedded applications, a data stream is read from a device (a FIFO from another processor, memory, camera or analog to digital converter), undergoes some processing such as compression or filtering, and is written to another device (FIFO to another processor, memory or digital to analog converter). Each word on the data stream is operated upon a few times and never touched again. In these applications, keeping the live words in the cache might be difficult because the data streams continuously flush the cache contents. This behaviour is called *cache pollution* because data that is used only once or twice displaces from the cache data that is reused several times. Clearly, pollution degrades performance as data with little or no reuse displaces data that should be kept in the cache because of being reused often.

We examine the behaviour of eleven applications in networking/multimedia from CommBench [38], in terms of a new metric named *popularity of reference*. The popu-

larity of a memory word w is the number of references to w, divided by the number of references in the execution of the program. Popularity is a lower bound on temporal locality, and is less expensive to compute than other measures of locality such as stack distance [17], and is an exact measure computed over the entire lifetime of the program. Thus, the popularity of memory words might be used to allocate them at the "correct places" in the memory hierarchy to improve performance. The performance attained with the designs for the top level of the memory hierarchy is explained in terms of popularity of reference in Section 5.

Notice that the term "popularity of reference" as defined here has a different meaning from its common usage in web caches [22, 35], where the "popularity of a web object" is more closely related to a "beauty contest" among cached objects, rather than to an exact reference count by one definite program.

The majority of the programs from CommBench exhibits a bimodal distribution of popularity. The measurements reveal that for 6 out of 10 of those programs, under 5% of the addresses account for over 70% of all data references. For `jpeg`, as few as 1.2% of the addresses (about 15,000 words) account for 90% of all memory references.

Because the distribution of popularity is often severely skewed, or bimodal, we investigate the performance of data caches coupled to buffers that reduce cache pollution, looking for combinations of design parameters that produce the best results at the smallest capacities, area and energy. Through detailed simulations we assess the performance (hit rate and IPC), area and energy of the Pollution Control Victim Cache (PCVC, [13]). Initially we compare the PCVC to the Pollution Control Cache (PCC, [36]) and to a baseline direct mapped cache. The PCVC is a novel design based on the PCC, that is simpler than, and outperforms, the PCC. The cache simulators are extensions to the SimpleScalar suite [3] and we use detailed models for the memory hierarchy and cache-memory bus. The PCVC is then compared with respect to performance, area and energy, to the PCC, direct mapped and two-way set associative caches [32], making use of CACTI models [29].

Our simulations indicate that the PCVC yields higher performance than the other designs. For the capacities and workload simulated, the PCVC has the best overall cost/performance of the four designs studied – the global miss rates are always the lowest, the product area×miss rate is better or compares favourably to the best, and the product energy/reference×CPI is comparable (within 30%) to direct mapped and two-way set associative caches.

The text is organised as follows. Section 2 introduces the popularity of reference metric to relate the performance of the designs to the behaviour of the reference streams of the benchmarks. Section 3 presents the design of the Pollution Control Victim Cache. Section 4 describes the simulation environment, and Section 5 contains the simulation results: in Section 5.1 miss rates and performance are assessed; and in Section 5.2 the performance, area and energy comparisons are drawn. Section 6 contains a survey of related work, and a summary is given in Section 7.

2 Popularity of Reference

The often quoted *90/10 rule* states that "90% of the execution time is spent on 10% of the code" [15]. As is the case with other "$x/\overline{x}$ rules", which are to be found in several

contexts, this one informally encodes an assertion concerning the distribution of a certain random variable, namely, the value of the instruction pointer during the execution of a program. As we discuss below, in the applications we consider here, we find that memory references are highly concentrated on a relatively small set of addresses.

Consider the execution of a program on a given input data and let w be a particular memory word in its address space. We define the *popularity* of w as the *ratio between the number of references to w and the total number memory references made by the program.* Consider the execution of a program on given input and let a_i be the address of the i-th memory reference by the program. We will call the sequence $T = (a_1, a_2, \ldots, a_N)$ the *(memory) trace* of this program on the given input data, where N is the total number of memory references made by the program.

Let W denote the set of *distinct* words in the trace T, that is $W = \{a_1, a_2, \ldots, a_n\}$, and let n denote the number of words in W, that is, $n = |W|$ is the number of distinct words referenced by the program. The set W is usually referred to as the *address space* of the program. For each word $w \in W$ let $r(w)$ be the number of times address w is referenced by the program. In other words, $r(w)$ is the number of occurrences of address w in the trace T, *viz* $r(w) = |\{i\colon a_i = w\}|$. For each $w \in W$ we define the *popularity of w* as the ratio

$$\pi(w) = \frac{r(w)}{N}\,. \tag{1}$$

The set P of the *most popular words in W* is defined as the set of words in W with popularity greater than $1/n$, that is

$$P = \{w \in W\colon \pi(w) \geq 1/n\}\,, \tag{2}$$

where p denotes the number of words in P, $p = |P|$. In general, $N \gg n$.

From an operational point of view, the set P is obtained by counting the references to each word in the address trace T.

There are other definitions of P that are perfectly reasonable, and perhaps of a more practical character. For instance, rather than using $1/n$ in Equation 2, the popularity limit could be set by some design-specific parameter such as a performance or energy target. We chose $1/n$ because it is at the average point of the distribution of popularity curves – see Figure 1. For practical purposes, $1/n$ is the looser limit that makes sense – we expand this idea in what follows.

It is worth noting that popularity and temporal locality [2, 16, 6] are distinct, though related, metrics. More precisely, the popularity of a word sets a lower bound on the temporal locality of this word. On the other hand, knowing the temporal locality of a word is not enough to infer anything about its popularity. In this sense, *popularity is a stronger (more precise) measure than temporal locality.* This is so because *popularity is a global or absolute measure* which depends only on the application code and its input data, while *temporal locality is a local or relative measure* which also depends on a time frame – it is meaningless to say "this word has good/bad temporal locality" without at least implicit reference to some time frame. It should be noted that, from a computational point of view, measuring the temporal locality of a reference stream (stack distance) is more costly than determining the popularity of each of its memory words.

The values of n, N and p for the CommBench programs [38] are shown in Table 1. The last two columns show, respectively, the ratio p/n and the sum of the popularities of the most popular words $\sum_{w \in P} \pi(w)$.

The benchmarks contain message-header processing programs that read and possibly change the message headers, and message-body processing programs that read and possibly modify the message contents. The header processing programs are `rtr` (Radix-Tree Routing), `frag` (IP packets fragmentation), and `drr` (Deficit Round Robin scheduling). The body processing programs are `cast` (CAST-128 encryption), `zip` (Lempel-Ziv compression), `reed` (Reed-Solomon encoding) and `jpeg` (image compressing). Some of these applications perform an encoding and a decoding function, denoted by the suffixes `_enc` and `_dec`, respectively.

Table 1: Popularity metrics for CommBench

program	N [10^6]	n	p	p/n	$\sum \pi$
`rtr`	30.90	1,292,378	29,565	2.3	89
`drr`	742.81	5,868	300	5.1	95
`frag`	93.80	5,591	114	2.1	98
`cast_enc`	37.19	1,973	75	3.8	74
`cast_dec`	37.85	1,974	93	4.7	76
`zip_enc`	56.87	76,075	12,806	16.8	57
`zip_dec`	8.48	19,510	1,535	7.9	67
`reed_enc`	137.47	1,260	297	23.6	95
`reed_dec`	218.15	1,451	404	27.9	94
`jpeg_enc`	92.51	1,000,488	14,835	1.5	91
`jpeg_dec`	87.54	1,002,221	15,364	1.5	91

p/n and $\sum \pi$ shown as percentages.

In order to provide a broader perspective, the plots in Figure 1 show the values of $\sum \pi$ (displayed as percentages) for some of the programs in Table 1. Note that these are cumulative distribution plots: for each point (x, y) the value of y is given by $\sum_{w \in P(x)} \pi(w)$, where $P(x)$ is the set of the x/n most popular words in W. Figure 1 shows the extremes of popularity for these programs: the distribution of references for `jpeg_enc` has the highest concentration and `cast_enc` and `reed_dec` have the lowest. The vertical spikes represent all pairs (p/n, $\sum \pi$) in Table 1.

Notice that `jpeg_enc` has the highest concentration of popularity ($p/n = 1.5\%$) but the values for n and p are both large. `cast_enc` and `reed_dec` show less concentration but their n and p are small.

If all addresses were referenced exactly the same number of times, the cumulative distribution plot would be a straight line from $(0, 0)$ to $(100, 100)$, each address contributing exactly $1/n$ – shown as the faint diagonal line in Figure 1. Note also that, being a cumulative distribution, the popularity curve of *all* programs is confined to the top-left triangle of the diagram in Figure 1 and that addresses to the left of the popularity limit p contribute more than $1/n$ to the distribution.

If the popularity limit were, somewhat arbitrarily, re-defined so that some energy or performance level can be sustained by the system, for instance to 90% or 95% of all refer-

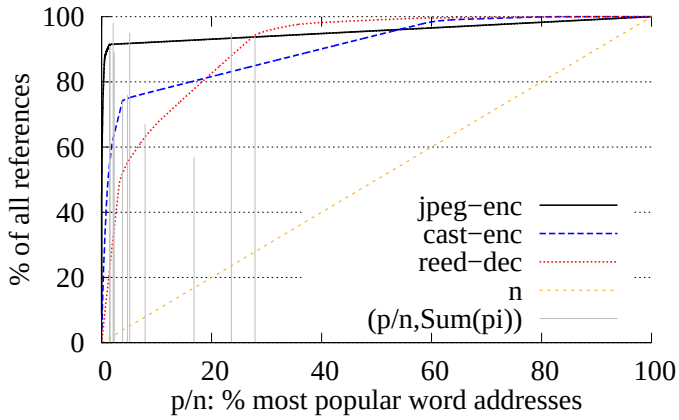

Figure 1: Popularity for `jpeg`, `cast` and `reed`.

ences, the sets of popular words according to the new limits could be quickly computed. Table 2 shows the sizes of the popularity sets needed to capture those references. The fraction of all address space comprised by the popularity sets are also shown. Column p from Table 1 is repeated for ease of comparison.

Table 2: Popularity for 90 and 95% of all references

references		90%		95%	
program	p	p_{90}	p_{90}/n	p_{95}	p_{95}/n
`rtr`	29,565	58,965	4.6	284,904	22.0
`drr`	300	135	2.3	315	5.4
`frag`	114	70	1.3	93	1.7
`cast_enc`	75	783	39.7	1,018	51.6
`cast_dec`	93	752	38.1	990	50.2
`zip_enc`	12,806	30,304	39.8	43,168	56.7
`zip_dec`	1,535	9,034	46.3	12,655	64.9
`reed_enc`	297	239	19.0	293	23.3
`reed_dec`	404	361	24.9	417	28.7
`jpeg_enc`	14,835	10,520	1.1	420,935	42.1
`jpeg_dec`	15,364	11,586	1.2	452,901	45.2

p_{90}/n and p_{95}/n shown as percentages.

Keeping popular addresses on fast memory improves the average memory access time because these references are frequent. Increasing the capacity of the fast memory to accommodate unpopular addresses may not be cost-effective because their contribution to average access time is (very) small. The unpopular addresses are best left out of the cache, or can be mapped into a region that is prefetched by a stream buffer or prefetch cache [19, 1] or a DMA engine. The "somewhat popular" addresses are those just beyond the popularity limit and might exhibit good temporal locality during certain program phases. Maintaining these addresses in fast memory can improve overall performance as long as they do not often evict the very popular addresses from the cache. In Section 3 we present a design that minimises these evictions.

2.1 Sensitivity to Input Data

In order to gauge the sensitivity of the popularity distribution to input data we run simulations of `jpeg` and `zip` with additional data sets. Table 3 contains the same statistics as in Table 1 for the new data sets – `jungle` and `mist` for `jpeg`; `hard` and `easy` for `zip`, which are described below.

Table 3: Sensitivity to various inputs: n, N, p

program	N [10^6]	n	p	p/n	$\sum \pi$
		`jpeg_enc`			
`orig`	92.51	1,000,488	14,835	1.5	91
`jungle`	59.78	601,059	12,743	2.1	92
`mist`	21.70	242,037	9,376	3.9	91
		`jpeg_dec`			
`orig`	87.54	1,002,221	15,364	1.5	91
`jungle`	59.83	598,106	8,722	1.5	92
`mist`	25.52	240,358	6,672	2.8	93
		`zip_enc`			
`orig`	56.87	76,075	12,806	16.8	74
`hard`	27.17	65,172	934	1.4	66
`easy`	15.05	54,103	8,480	15.7	64
		`zip_dec`			
`orig`	8.48	19,510	1,535	7.9	67
`hard`	4.70	18,989	1,281	6.7	65
`easy`	4.51	9,938	270	2.9	26

p/n and $\sum \pi$ shown as percentages.

jpeg We used two JPEG files in addition to the one provided with CommBench. The three are: `orig` 1280x1024 pixels, 167 Kbytes, distributed with CommBench; `jungle` 1024x763, 255 Kbytes; and `mist` 640x480, 161 Kbytes. JPEG files are the inputs to `jpeg_dec` and the corresponding BMP files are inputs to `jpeg_enc`. The BMP files were obtained from the JPEGs by processing them with display.

Figure 2 shows the popularity curves from the simulations with the three data files. Notice that the horizontal scale extends to just 5%, and the vertical from 30% to 100%. The concentration of reference is slightly less on the smaller files (`mist`) but the 3.4% most popular word addresses capture 90% of the references, or 8,332 out of 242,037 words, 21.7×10^6 references.

zip In addition to the HTML file (and its compressed version), we used two input files that are, to some extent, pathological inputs to `zip`. The three input files are: `orig`, HTML text, 1,022,976 bytes as distributed with CommBench, the compressed version has 547,522 bytes; `easy`, file with 1,022,976 `'a'`s, the compressed file is 1040 bytes long; and `hard`, gzipped version of the `orig`inal HTML text, the compressed file is 715 bytes larger than the input file.

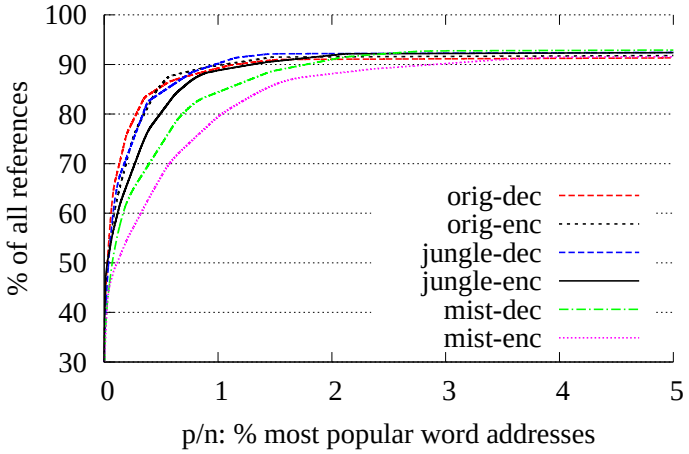

Figure 2: Popularity of references, `jpeg` with 3 data sets.

The popularity curves are shown in Figure 3. The bottommost curve is the decoding of the file with 1 million `'a'`s and it seems that the amount of processing per memory word is very small for this file. Compressing it is slightly more involved and the popularity curve is closer in shape to that of the other inputs. The best concentration is for the most difficult input, which is an already compressed file, as shows the plot for `hard_enc`.

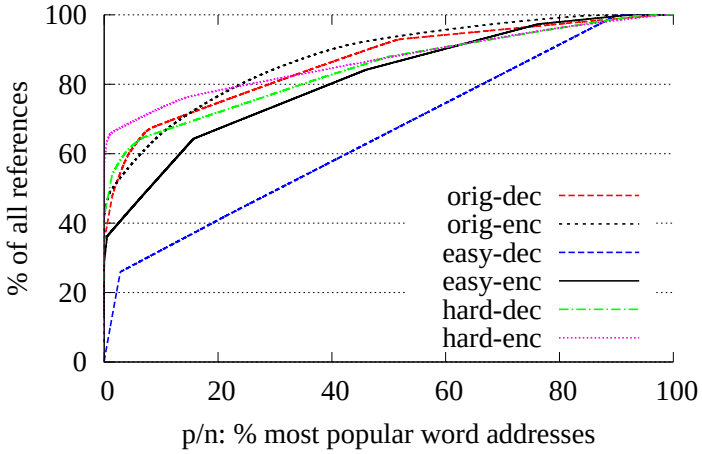

Figure 3: Popularity of references, `zip` with 3 data sets.

3 The Pollution Control Victim Cache

Cache pollution occurs when blocks that are referenced often get replaced by blocks that are scarcely referenced – thus 'popular' blocks are evicted from the cache by 'unpopular' blocks, leaving the cache polluted. When referenced, the polluting blocks tend to cause two misses, one to load the unpopular block and another to reload the evicted popular block [36]. The individual components of the memory systems are described below.

In a lockup-free cache, a reference that misses in the cache is recorded in a *Miss Status Holding Register* (MSHR) while the missing datum/block is fetched from memory [20]. In a given memory system, the number of MSHRs determines how many outstanding misses can be sustained by the cache. All our models employ four MSHRs.

A *victim cache* (VC) is a small fully-associative buffer that holds blocks which were evicted from the L1 [19]. On a miss, the missing block is loaded onto the L1 and the evicted block is loaded onto the VC's LRU block. The VC adds some associativity to a direct mapped cache, thus eliminating costly conflict misses, without increasing significantly its size or complexity. Figure 4 shows a block diagram of a VC attached to a direct mapped cache. In the experiments we performed, as few as four blocks can eliminate a large fraction of conflict misses [13].

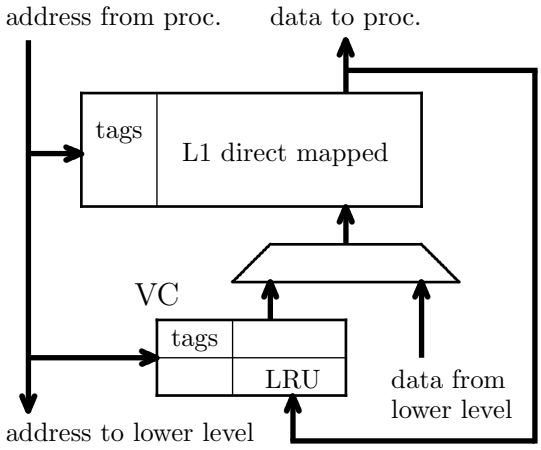

Figure 4: Victim cache

The *Pollution Control Cache* (PCC) is a small fully associative cache that operates in parallel with the L1 [36]. On a miss, the missing block is loaded on the PCC; if this block is referenced a second time, then it is promoted to the L1, possibly avoiding the eviction of a popular block. A hit on the PCC costs 2 cycles: one to detect the miss on the L1 and the second to access the PCC. When a block is promoted, the latency incurred in writing the evicted block onto the PCC can often be hidden as these events are not frequent. The annex-cache [18] is similar to the PCC.

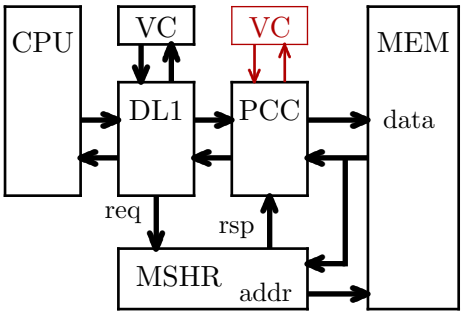

Figure 5: Pollution Control Cache

The design as described in [36] connects a one-block VC, called a *victim buffer*, to the L1 and another VC to the PCC, shown in darker and lighter colour, respectively, in the diagram in Figure 5. The design in our simulations differs from the original in that there is only one VC attached to the L1. In the simulated PCC model there is a 4 block VC attached to the L1, so that blocks evicted from L1 are stored back in the VC. The three buffers L1, VC and PCC are searched in parallel on a reference.

It is possible to achieve better performance from the memory hierarchy with a simpler design than the PCC. A *Pollution Control Victim Cache* (PCVC) is a fully associative cache interposed between the L1 and memory that operates as a combined PCC and VC [13]. Blocks referenced for the first time are loaded into the PCVC, and on the second reference are promoted to the L1; the evicted block is exchanged with that from the PCVC. A hit on the PCVC costs 2 cycles to deliver a word to the processor, plus 1 cycle (often hidden) to perform the exchange of blocks. Figure 6 shows a block diagram of the PCVC. In the PCVC model, the entire pollution control buffer acts as a victim cache for the L1 so that blocks evicted from L1 are stored back in that buffer.

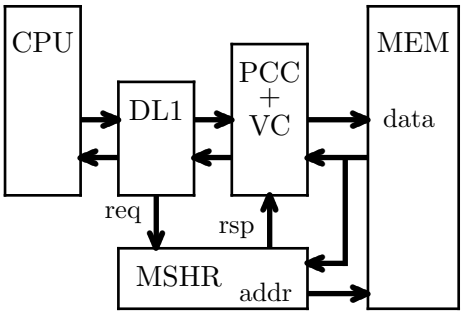

Figure 6: Pollution Control Victim Cache

4 Simulation Environment

Our focus is on small first-level data caches that might be suitable for embedded systems. In order to constrain the search space, we investigate only the first level of the memory hierarchy. The design parameters used in the simulations for the second level are representative of an aggressive design for the main memory, or of a system with a second level cache.

The simulations were run on modified versions of the *sim-outorder* simulator from SimpleScalar [30, 3]. Sim-outorder is an execution driven, cycle-accurate, out-of-order simulator. The simulators run the applications from the CommBench suite, as these are representative of the workloads found in networking systems. Table 1 (Section 2) shows the dynamic instruction count and the fraction of all instruction that are memory references. The reader should refer to [38] for details concerning the benchmark programs.

We run a series of preliminary experiments to determine the processor model to employ in the simulations [14]. For these applications we found that the most cost-effective organisation is a processor that issues two instructions per cycle – the stages fetch, decode, issue, execute, and commit have a width of two, the Register Update Unit (RUU, [31]) has 16 slots, and the load-store queue (LSQ) has 8 elements. The RUU is similar to a reorder buffer, and the load-store queue holds memory references that await completion at memory.

The cache models were simulated with capacities of 1, 2, 4, 8, and 16 Kbytes, and with block sizes of 32 bytes. Unless stated otherwise, the data cache is direct-mapped. The CPU has two ports into the data cache to support up to one read and one write per cycle. The bus from the data cache to memory is 8 byte wide (two words), and the DRAM

access latency is 18 cycles for the first reference to a block, with subsequent pairs of words being accessed every 2 cycles. In all simulations the instruction cache has a relatively large capacity of 32 Kbytes so its performance does not interfere with that of the data cache. Instructions and data are 32 bits wide.

To model the interface between the L1 caches and L2/memory, sim-outorder assumes there is an unlimited number of miss status holding registers, and thus the cache interface can handle an unlimited number of concurrent misses. This is a reasonable assumption when considering the design of very aggressive superscalar processors, which was one of the intended applications of SimpleScalar. For less ambitious processors, like most of those in embedded applications, a somewhat more realistic model for the memory interface is desirable.

We adapted the simulator to use a limited number of MSHRs and to keep a record of all activities concerning outstanding misses. In our model, a second reference to a block previously recorded in a MSHR is not counted as an additional miss. However, the latency of that reference is computed as the time elapsed until the block is brought in from memory to satisfy the primary miss to that block, as suggested by [9].

A series of experiments were run to assess the impact of limiting the number of outstanding misses. We found that the best cost-performance point is four MSHRs [14], hence all simulations described here were run with four MSHRs.

5 Simulation Results

5.1 Performance – Miss Rate and IPC

We now present the results of experiments that compare the performance, measured as global miss rate and IPC, of three designs for the first level (L1) of the memory hierarchy. The designs are: a *baseline* first level direct mapped ($L1_{DM}$); a direct mapped L1 plus a pollution control cache with 4, 8, 16, or 32 blocks – PCC(4) to PCC(32); a direct mapped L1 and pollution control victim cache with 4, 8, 16, or 32 blocks – PCVC(4) to PCVC(32). In the plots shown below the baseline is shown as DM L1.

5.1.1 PCC

On an L1 miss, the missing block β is loaded onto the PCC. On a second reference to block β, it is moved to L1 and the block evicted from the L1 is moved to the VC. If the PCC is small (1-4 blocks), either the miss rate increases or the gain is small, as shown in Figure 7(a) for the 1 Kbytes cache. With the larger PCCs (8-32 blocks) there is a reduction of roughly 50% in the miss rate. As for the overall performance, the poor miss ratios of the small PCCs are reflected on IPC, as can be seen on Figure 7(b). For the larger PCCs, the gains in IPC range from 13% to 15%, for PCC(8) and PCC(32), respectively.

5.1.2 PCVC

On a miss in L1, the block β is loaded onto the PCVC. On a second reference to block β, it is swapped with the block evicted from the L1. The behaviour of the smaller PCVCs is slightly better than that of the smaller PCCs, as shown in Figure 8. The gains in miss rate of the larger PCVCs decrease slightly with each doubling in size: for the 8, 16, and

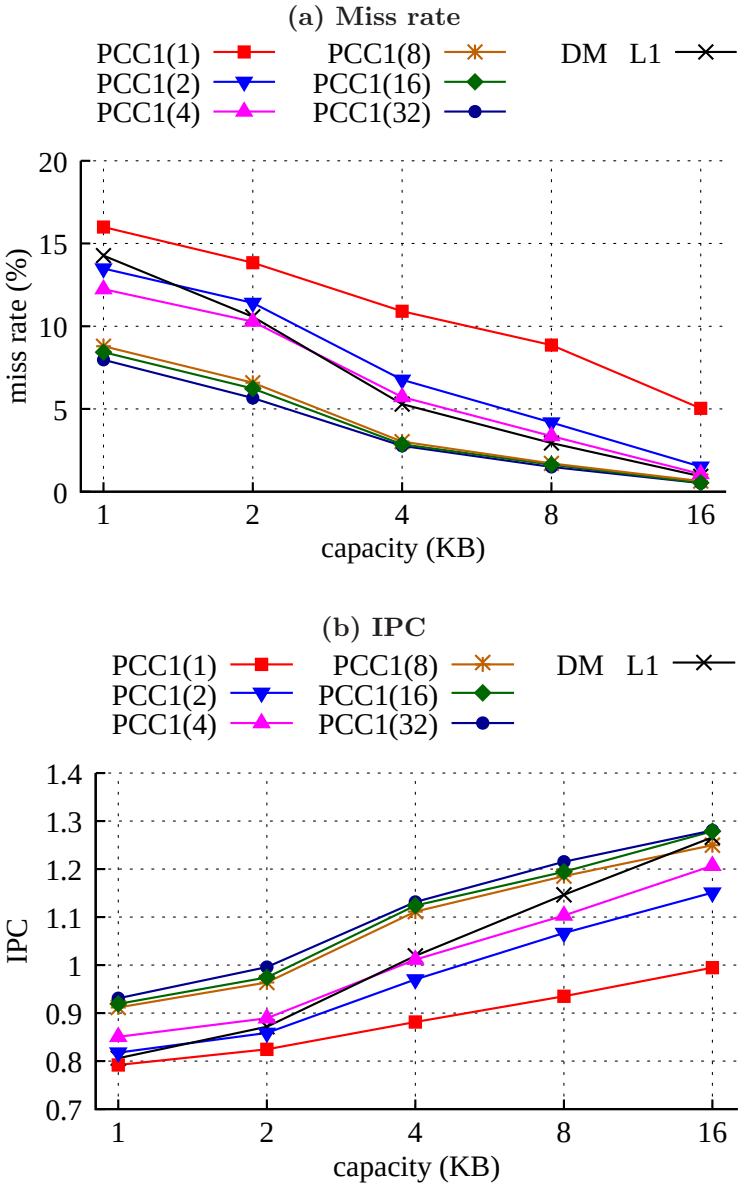

Figure 7: PCC: miss rate and IPC

32 block PCVCs the gains are 64%, 88%, and 119%, respectively. Since the PCVC also acts as a victim cache, its overall performance improves with size, as shown on the bottom of Figure 8. For the smaller 1 Kbytes cache, the gains are 13% and 19% for PCVC(8) and PCVC(32), respectively.

5.1.3 IPC Gains Over Baseline L1

Our simulations indicate that a L1+PCVC is a more effective design than a L1+PCC, because it is simpler – comparisons on two sets of tags rather than three – and smaller – only n blocks in the PCVC(n) rather than the $n + m$ blocks in the PCC(n)+VC(m).

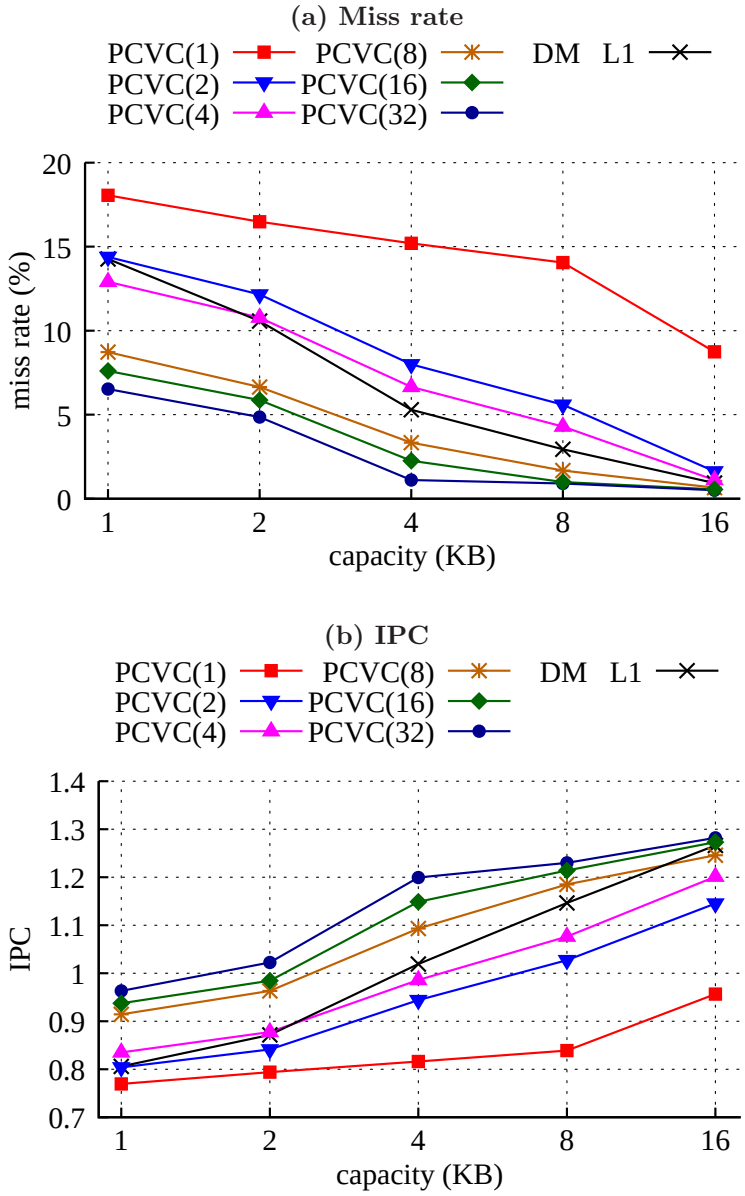

Figure 8: PCVC: miss rate and IPC

Table 4 shows a comparison of the results in Figures 7 and 8. It shows the performance gains achieved with the PCC and the PCVC when compared to a base model which is an L1 cache with 4 MSHRs. The first row, labelled MSHR(4) shows the IPC for the base model. The next four rows show the IPC ratio for two of the configurations that provide gains, buffers with 4 and 32 blocks. As mentioned before, smallest caches benefit the most from the buffers. The next two rows compare a given combination of cache+buffer with a simple L1 twice as large: a $\mathrm{L1}_{\mathrm{DM},1KB}$+PCVC(32) is compared to a 2 KB L1, a $\mathrm{L1}_{\mathrm{DM},2KB}$+PCVC(32) to a 4 KB L1, and so on. The last three rows compare an $\mathrm{L1}_{\mathrm{DM},C\,KB}$+PCVC(32) to caches with capacity $4C$, $8C$ and $16C$.

For the smallest 1 Kbytes caches, the L1+PCVC with at least 4 blocks has a better performance than a simple L1 twice as large, while for the PCC, 8 blocks are needed. The results for the 32 block PCVC show a 10% improvement in IPC over the simple 2 Kbytes L1, and just a 4% *decrease* when compared to a simple L1 four times larger

Table 4: IPC gains over a simple L1 cache

L1 capacity	1K	2K	4K	8K	16K
IPC base MSHR(4)	0.81	0.87	1.02	1.15	1.27
PCC(4,4)/base	1.06	1.02	0.99	0.96	0.95
PCVC(4)/base	1.04	1.01	0.97	0.94	0.95
PCC(32,4)/base	1.15	1.14	1.11	1.06	1.01
PCVC(32)/base	1.20	1.17	1.18	1.07	1.01
PCC(32,4)/base·2	—	0.96	0.86	0.86	0.85
PCVC(32)/base·2	—	1.10	1.00	1.05	0.97
PCVC(32)/base·4	—	—	0.95	0.89	0.95
PCVC(32)/base·8	—	—	—	0.84	0.81
PCVC(32)/base·16	—	—	—	—	0.78

(4 Kbytes) and with twice the capacity of the combined L1+PCVC. For the 8 block PCC and PCVC, the gains in performance over a simple 2 Kbytes L1 are 4.6% and 4.8%, respectively (not on the table). The last rows show that the PCVC(32) performs well indeed when compared to much larger caches. The performance of a $L1_{DM,1KB}$+PCVC(32) is 0.78 of that of a 16 KB L1.

5.1.4 Popularity of Reference in CommBench

We measured the popularity of data references for the CommBench programs. For 6 of the 11 programs, less than 5% of the addresses satisfy more than 70% of the references. In `jpeg`, for instance, less than 1.2% of all addresses (approx. 15,000 words) satisfy 90% of all data references. A different behaviour is displayed by `zip_enc`, for which the 13,000 most popular words (17% of the addresses) satisfy less than 60% of the references.

The plot for `zip_dec`, in Figure 9, displays two inflexion points: the first at 67% of all references, and the second at 93%. The plot for `zip_enc` has no obvious inflexions; the popularity limit is at 57% of all references. The addresses referenced by the two versions of `zip` can be split into three ranges: *popular*, *somewhat-popular*, and *unpopular*. To achieve the best performance from a given memory hierarchy, the popular addresses should be kept in the L1, the somewhat-popular addresses at one of the buffers (PCC or PCVC), and the unpopular would be better allocated to non-cacheable addresses – at least, not cacheable at the L1.

For the majority of the CommBench programs, $p < 512$ words, and that is the total capacity of the design PCVC(32)+$L1_{DM,1K}$, *viz* 32 blocks at the PCVC and 32 blocks at the $L1_{DM}$, with 8 words per block.

jpeg Figure 12 shows the global hit rate at the top level of the hierarchy. The hit rates for `jpeg` are (very) close to the average and that program's good performance stems from its tight concentration of popular words. Even with $p \approx 15.000$, the specialised buffers stop the unpopular words from polluting the $L1_{DM}$, as can be seen from the plots for the global hit rates in Figure 10, for `jpeg_enc`. As discussed in Section 5.2.2, the active working set of `jpeg` is held by a 4 Kbytes L1. For an L1 with capacity greater or equal to 4 Kbytes, `jpeg_enc` enjoys a hit rate close to the maximum for that program. If a

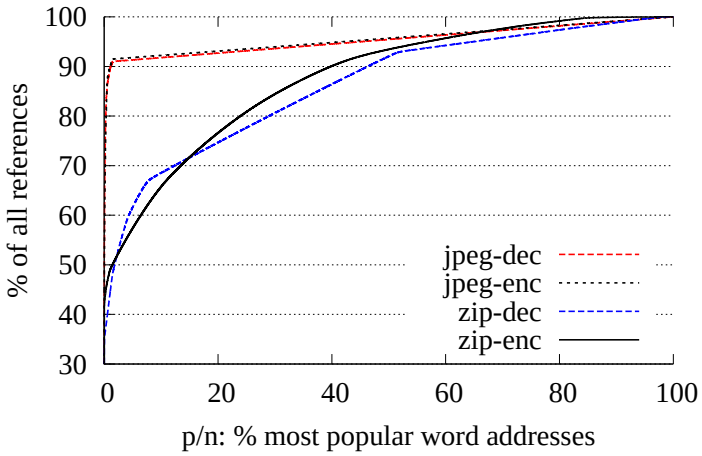

Figure 9: Popularity of `jpeg` and `zip`.

PCVC(32) is added to a $L1_{DM,1K}$, the CPI improves by 31%; if the PCVC(32) is added to a $L1_{DM,4K}$, the CPI improves by just 10% – these figures are similar to the averages in Figure 11.

zip_enc For `zip_enc`, for all simulated L1 capacities, $p \approx 13.000 > |L1_{DM}|$ and the reuse of words loaded into the cache is small, hence the global hit ratio does not saturate for $L1_{DM}$'s up to 32 Kbytes, as shown in Figure 10. The 32 block buffers improve markedly the hit ratio, protecting the L1 from pollution by unpopular addresses. The performance gain of a $L1_{DM,1K}$+PCVC(32) w.r.t $L1_{DM,1K}$ is 25%, whereas in relation to a 4 Kbytes $L1_{DM}$, the gain is 10%. When added to the largest L1 ($L1_{DM,32K}$), a PCVC(32) still improves CPI by 16%.

5.1.5 Performance *vs* Capacity Revisited

In what follows the performance of the four designs is given in cycles per instruction (CPI) because, in general, CPI is inversely proportional to cache capacity and this characteristic is used in the analyses that follow. The values for CPI and miss rate are the average for the 11 CommBench programs.

The performance of the four designs, given as CPI × L1 capacity is shown in Figure 11. As would be expected, the simplest design ($L1_{DM}$) displays the worst behaviour for all cache sizes. The associative cache ($L1_{2w}$) performs better than the direct mapped and worse than the other designs. The PCC(4) and PCC(32) perform similarly to the 4 blocks PCVC(4). The PCVC(32) displays the best performance of all, and that is uniform for all L1 capacities.

Figure 12 shows the hit rate at the memory port of the processor, served by the L1+buffers, which is the effective hit rate as seen by the processor. For the programs and data sets in CommBench, PCVC(32) has mostly compulsory misses because it holds the data sets – a 32 Kbytes L1 is large enough to accommodate almost completely the working sets of the programs, as the data for $L1_{DM}$ indicates.

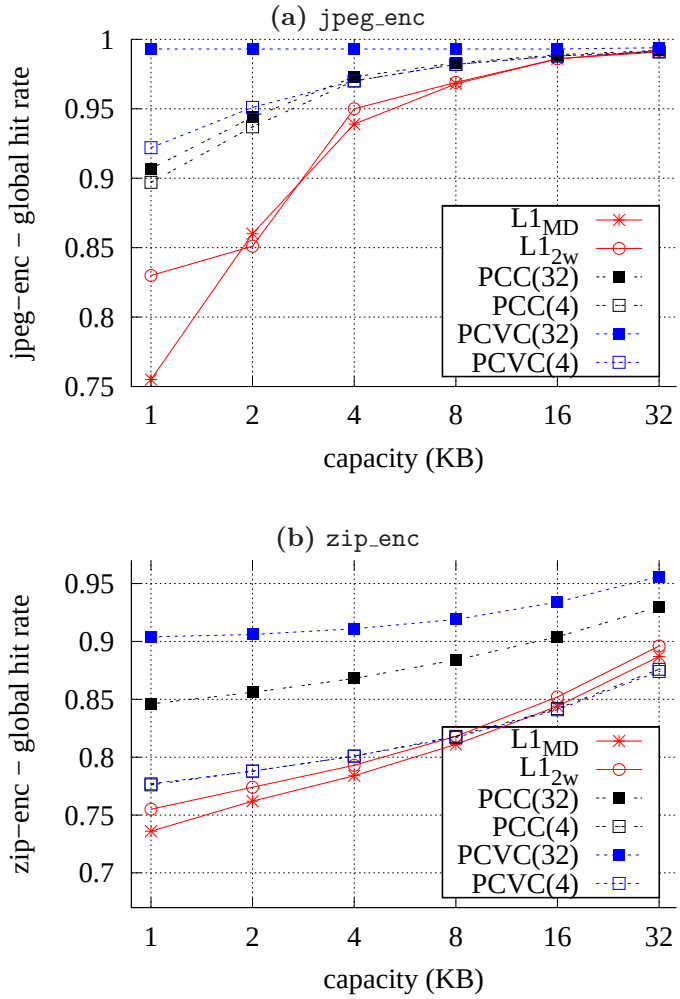

Figure 10: Global hit rates `jpeg_enc` and `zip_enc`

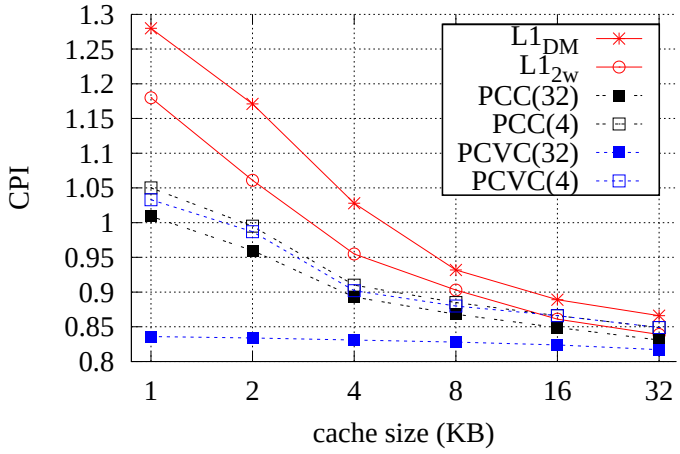

Figure 11: CPI × L1 capacity

5.2 Area and Energy

We now compare the cost×performance of the two pollution control caches to that of simpler designs, a direct mapped L1 and a two-way set associative L1. The four designs

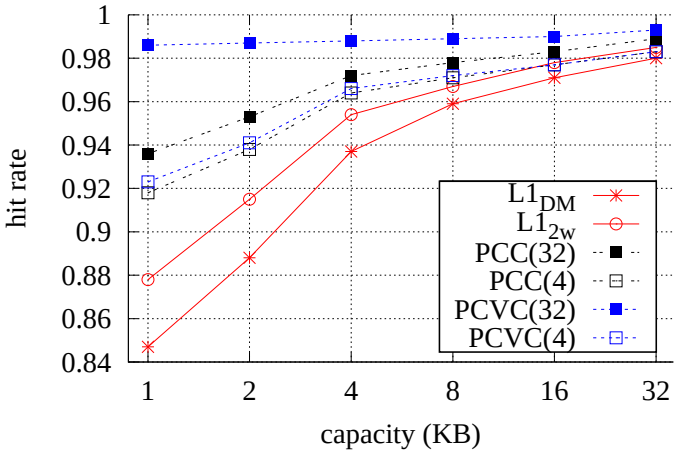

Figure 12: Global hit ratio × L1 capacity

investigated are: first level direct mapped ($L1_{DM}$); 2-way set associative L1 ($L1_{2w}$); direct mapped L1 plus a pollution control cache with 4, 8, 16, or 32 blocks – PCC(4) to PCC(32); direct mapped L1 and pollution control victim cache with 4, 8, 16, or 32 blocks – PCVC(4) to PCVC(32). To extend the limits of comparison, in the experiments that follow L1 capacity was simulated up to 32 Kbytes.

5.2.1 CACTI Models

To perform the area, energy and performance comparisons of the four designs, the caches were modelled with CACTI-3.2 [37, 29]. CACTI-3.2 models 180nm processes, and this is the newest version that can simulate caches as small as those studied here. Therefore, the area and energy comparisons that follow ought to be taken as qualitative measures.

MSHR The design of a MSHRs makes use of more logic than state, hence these are not modelled with CACTI. In all simulations there are 4 MSHRs with the same block size attached to the processors and these structures may be considered as part of the CPU rather than the memory hierarchy.

Level 1 Cache The primary data caches were modelled differently in the four designs: ⟨i⟩ direct mapped (DM) with two pairs of read-write (RW) ports, one pair connected to the processor/MSHRs, and the other connected to memory; ⟨ii⟩ two-way set associative (2w) and two pairs of RW ports, connected as the DM model; ⟨iii⟩ the L1 attached to a PCC is direct mapped and is connected to the VC, to the processor, and to memory – the L1 has three pairs of RW ports, as shown in Fig. 5; ⟨iv⟩ the L1 attached to a PCVC is direct mapped with two pairs of RW ports, one attached to the processor and the other to the PCVC, as shown in Fig. 6.

Victim Cache The pollution control cache has a VC connected to the L1. The VC was modelled, and simulated, with 4 blocks of the same size as L1 and fully associative. This device has one RW port.

Pollution Control Cache The PCC employs a fully associative memory with blocks that have the same width as those in L1. The associative memory was modelled with two pairs of RW ports, one attached to the L1 and the other to memory. The VC could be connected to the L1 and PCC through a 3×3 crossbar, rather than to the third port of the L1. As we do not have implementations for these two designs, we cannot compare their areas and energy. Our conjecture is that for the small capacity L1s (up to 4 Kbytes), the third L1 port occupies roughly the same area as the crossbar; for the larger L1s, the third port is larger than the crossbar.

Recall that in our model for the PCC there is no VC attached to the associative buffer, as described in [36]. In the original design, the associative buffer has three RW ports, being therefore larger and more power hungry than our model.

Pollution Control Victim Cache The PCVC has the same configuration as the PCC: fully associative memory with blocks as wide as those in the L1. The PCVC was modelled with two pairs of RW ports, one connected to the L1 and the other to memory. The cache controller for the PCVC is more complex than that of the PCC because the "victim cache" functionality is embedded in the PCVC. This is not captured by the CACTI models.

Interconnect The design models, and the estimates obtained from them, do not account for the interconnections between the several components. The interconnect between processor/MSHRs and memory are the same in all designs. The interconnects between L1 and the specialised buffers, in the designs with PCCs and PCVCs correspond to a fixed amount, be it area or energy/reference.

5.2.2 Area

We compared the area of the complete designs for several capacities, L1 associativity of 1 and 2 ways, and buffer type (PCC and PCVC) and capacity. The total area of the PCC is given by the sum of the areas of the 3-ported direct mapped L1, a four entry VC, and an associative buffer with b entries (see Section 5.2.1 and Figure 5). The total area of the PCVC is the sum of the area of a 2-ported direct mapped L1 and an associative buffer with b entries – see Figure 6.

The center columns of Table 5 show the area for associative buffers in the PCCs and PCVCs with 4, 8, 16 and 32 blocks. Table 6 shows the area for the complete designs; the areas for PCC and PCVC are shown only for buffers with 4 and 32 blocks. Figure 13 contain plots for the areas of an direct mapped and 2-way L1, PCC and PCVC with 4 and 32 blocks. L1 capacity ranges from 1 Kbytes to 32 Kbytes.

Table 5: Area and energy **vs** capacity [blocks] for associative buffers

	area [mm^2]		energy/ref [nJ]	
blocks	VC	assoc. buffer	VC	assoc. buffer
4	0.22	0.70	0.21	0.45
8	–	0.76	–	0.47
16	–	0.89	–	0.50
32	–	1.12	–	0.58

Table 6: Total area [mm^2] **vs** L1 capacity [Kbytes]

			PCC		PCVC	
capac	$L1_{DM}$	$L1_{2w}$	4	32	4	32
1	0.80	1.18	2.99	3.42	1.45	1.86
2	1.07	1.38	3.93	4.36	1.77	2.19
4	1.58	1.86	4.92	5.35	2.28	2.69
8	2.67	2.87	7.66	8.09	3.37	3.78
16	5.02	5.02	13.81	14.23	5.72	6.13
32	8.93	9.26	25.02	25.44	9.63	10.04

PCC: 3-ported L1, VC(4), assoc. buffer with b blocks.

PCVC: 2-ported L1, assoc. buffer with b blocks.

If the direct mapped L1 ($L1_{DM}$) is taken as the basis for comparison, the 2-way L1 ($L1_{2w,1K}$) is nearly 50% larger, but the difference is inversely proportional to capacity as the $L1_{2w,32K}$ is only 3% larger than the $L1_{DM,32K}$. For a given combination of capacity, block size and associativity, CACTI picks the best geometry for that set of parameters. Thus, the area difference does not drop as the square of L1 capacity.

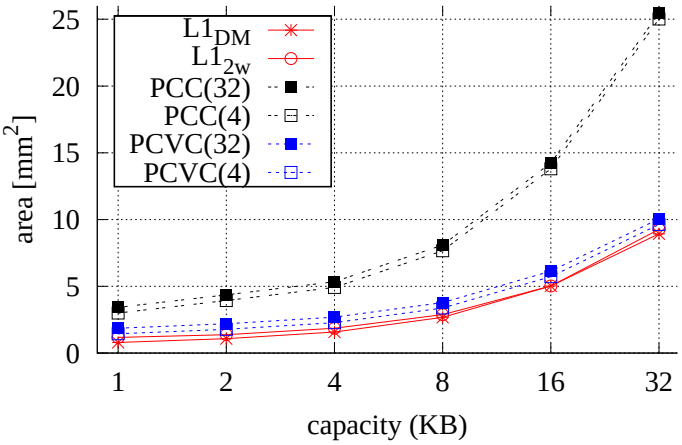

Figure 13: Area **vs** L1 capacity (from Table 6)

In comparison to a $L1_{DM}$, the $PCC(4)_{1K}$ ($PCC(32)_{1K}$) is 3.8 (4.3) times larger than $L1_{DM,1K}$. The difference reduces to 2.8 (2.9) times for the largest L1. The contribution of the VC to the total area of the PCC is proportionally smaller as the PCC's capacity increases. The most significant contribution is the third L1 port, needed to connect to the PCC.

For all simulated capacities, the PCVC is larger than the $L1_{DM}$, by a factor of 1.8 (2.3) times for a PCVC(4) (PCVC(32)) w.r.t the 1 Kbytes $L1_{DM}$, decreasing to 1.08 (1.12) times for 32 Kbytes. For a given PCVC capacity c, this buffer increases the area of the design $\langle$PCVC(c)+L1$\rangle$ by a fixed amount, that is inversely proportional to L1 capacity.

Performance *vs* **Area** The area used up by of the top level of the memory hierarchy grows with capacity at that level whereas the miss rate decreases. The smallest product *area*×*miss rate* is an 'optimal' design for which the 'derivative' of the product, with respect

to capacity, is zero. By 'derivative' we mean an approximation in which the tendency lines behave as if capacity was a continuous variable. Figure 14 shows the area×miss rate plots for the designs (smaller is better).

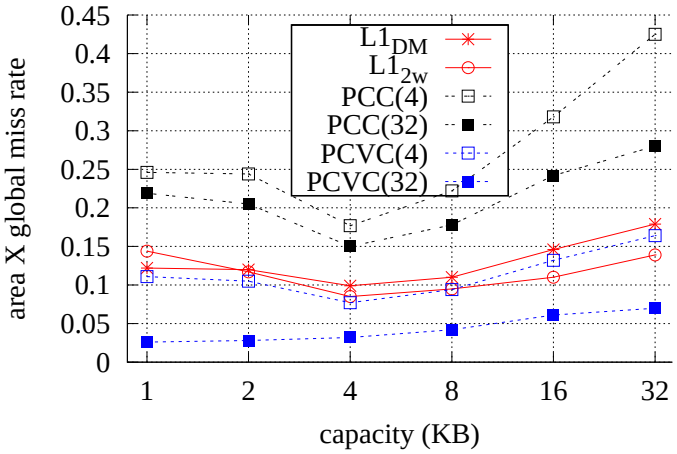

Figure 14: Area × global miss rate

Figure 11, in Sec. 5.1.5, shows that for all designs except PCVC(32), the performance gains are relatively small as L1 capacity grows beyond 4-8 Kbytes. This is also borne out by the product *area*×*miss rate*, as shown in Figure 14. Except for the PCVC(32), the best product is at 4 Kbytes, and w.r.t. *area*×*miss rate* the best designs are $L1_{DM,4K}$, $L1_{2w,4K}$ and PCVC(4,4K). The PCVC(32) is always better than the others, for all $L1_{DM}$ capacities, because the active subset of the data is always kept near the CPU, as mentioned in Section 2.

5.2.3 Energy

We assessed the energy per reference spent in each of the four designs. The right hand side of Table 5 (Sec. 5.2.2) shows the energy per reference for each of the buffers. Table 7 shows the power for the complete designs and Figure 15 displays the energy/reference plots, in the same conditions as in Section 5.2.2.

Table 7: Energy/ref [nJ] **vs** L1 capacity [Kbytes]

			PCC		PCVC	
capac	$L1_{DM}$	$L1_{2w}$	4	32	4	32
1	0.61	0.95	1.69	1.82	1.05	1.19
2	0.64	0.98	1.78	1.91	1.09	1.22
4	0.72	1.04	2.01	2.14	1.17	1.30
8	0.88	1.20	2.44	2.57	1.32	1.45
16	1.09	1.42	3.03	3.09	1.54	1.67
32	1.48	1.80	4.17	4.23	1.92	2.05

PCC: 3-ported L1, VC(4), assoc. buffer with b blocks.

PCVC: 2-ported L1, assoc. buffer with b blocks.

The energy/reference for the $L1_{2w,1K}$ is 1.56 times larger than that of a $L1_{DM,1K}$, and 1.22 times for the 32 Kbytes. For the Pollution Control Cache, for all simulated L1

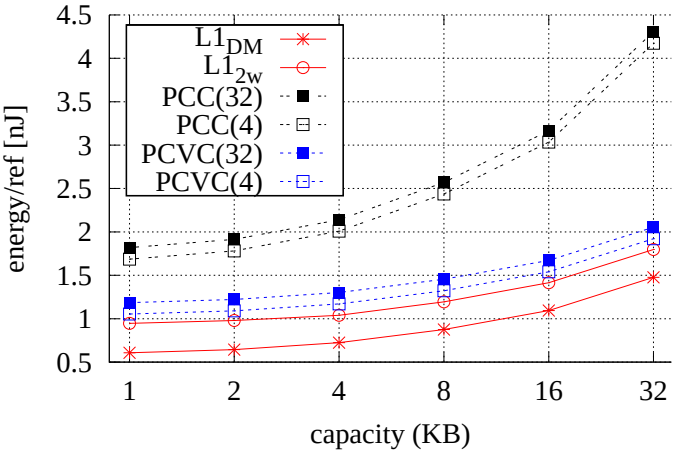

Figure 15: Energy/reference **vs** L1 capacity (Table 7)

capacities, a PCC(4) uses 2.77 to 2.83 times more energy than the $L1_{DM}$. For a PCC(32), the ratios are in the range of 2.89 to 2.99 times. For the PCVC, the ratios inversely decrease with L1 capacity: a PCVC(4) consumes 1.7 (1.3) times more than the $L1_{DM,1K}$ (32 K), whereas for the PCVC(32) the rates are 1.95 and 1.4 times for the $L1_{DM,1K}$ and $L1_{DM,32K}$, respectively.

Performance *vs* **Energy** The product *miss rate* × *energy/reference* is shown in Figure 16. The tendency curves display a marked inflexion as L1 capacity reaches 4 Kbytes. These points are not the 'minima', yet clearly indicate a point of diminishing returns. The products for the PCCs are worse than for the simpler designs. The products for the $L1_{DM}$s are closer to those of the PCVC(4) than to the $L1_{2w}$ as the PCVC(4)'s circuits are smaller, hence use less power than the $L1_{2w}$. For capacities larger than 4 Kbytes, there is little difference between the $L1_{DM}$, $L1_{2w}$, and PCVC(4). The PCVC(32) displays the best results, by a wide margin.

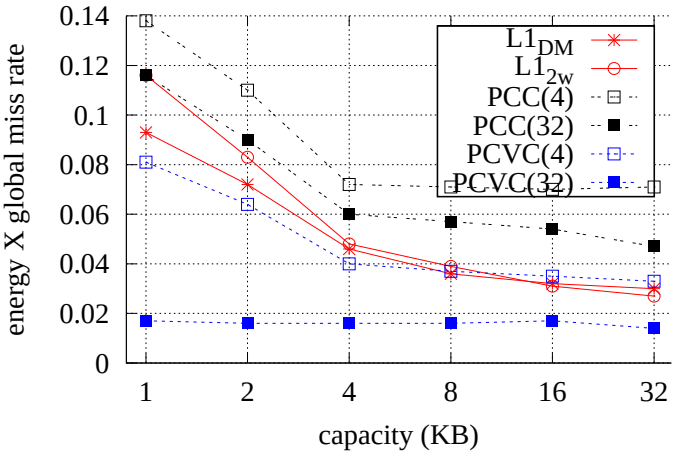

Figure 16: Energy/reference × global miss rate

Figure 17 shows the product *CPI* × *energy/reference* (smaller is better). With this metric, the numbers for $L1_{2w}$, PCVC(4) and PCVC(32) are similar. The inherently higher miss rate on the $L1_{DM}$ translates into a higher CPI – in spite of the reduced power this

design yields the worse performance, as shown in Fig. 11 (Sec. 5.1.5). The PCCs display better figures than the simpler designs yet the energy expenditure is such that the product CPI×energy is roughly twice that of the other designs.

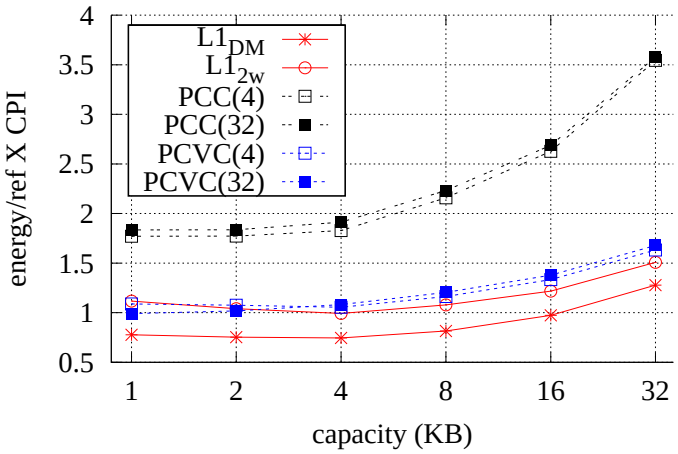

Figure 17: Energy/reference × CPI.

5.2.4 Putting It All Together

When considering programs with reference patterns similar to that of `zip`, the addition of a PCVC may be profitable because the performance gain (16%) is of the same order as the increase in area (12%) for a combination $L1_{DM,32K}$+PCVC(32). Regarding energy, the trade-off is not straightforward: 16% gain in performance and 38% increase in energy, and the decision might be constrained by other factors. For programs with a high popularity concentration, 4 or 16 blocks PCVC may be advantageous since the increase in area and energy are obviously less than for a 32 block PCVC, while performance does not decrease at the same rate as the number of blocks. [13] present a detailed performance evaluation of PCVCs, with systems from 1 to 32 blocks.

6 Related Work

Work related to ours falls into two categories: (i) models for locality of reference with their applications; and (ii) specialised buffers or caches. These are discussed in what follows.

Several analytical models for locality have been proposed. [34] model the memory reference pattern by a stack whose depth varies according to a pseudo-random sequence of numbers, derived from the program trace. [2] derive a model of program behaviour by measuring from a program trace the compulsory, capacity and conflict misses. [6] define the average stack distance and indicate how it can be measured from a program trace – the smaller the average stack distance, the better the program locality. These models represent program behaviour as the average over the program execution. Popularity of reference is based on the actual reference counts over the whole of the execution and thus can expose idiosyncratic behaviours that might be hidden by the averaging. [8] present reference distributions that are similar to the popularity reference distribution, only they measure the references to cache blocks rather than to individual words. The width of the

"reference unit" is a parameter of our popularity measurements, and we chose 4 bytes (one word) because this width is more general than 8 or 16 words.

By measuring the popularity on a word boundary, this information can be used to direct the compiler or linker to re-allocate that word onto an address that is 'near' other popular words, thus improving the hit rate of popular cache blocks. Another use might be to allocate the most popular words onto scratchpad memories, therefore improving execution time and reducing power consumption [26, 4, 23, 17]. Those methods reduce the allocation problem to an integer linear program, formulated as a knapsack problem, whereas popularity provides an exact reference count.

[24] present a performance evaluation of first-level caches for embedded systems, employing the MiBench suite. The authors investigate the effects of organisation, associativity, capacity and replacement policy on miss rate. Their results show that higher associativity improves performance because of the reduction in conflict misses, as should be expected from applications with streaming behaviour.

There are several techniques to improve the performance at the top level of the memory hierarchy, by attempting to predict what addresses should be kept at the top of the hierarchy, or which references will cause the worst loss of performance. The first set of techniques includes non-critical loads [11, 33], non-vital loads [27], and slack [10]. These techniques can indeed improve performance, but their implementations appear to be too complex for application in embedded systems.

The second set of techniques include the pollution control cache [36], the annex cache [18], and the usage filter [8]. As discussed earlier, these employ an associative buffer to filter out the references with poor temporal locality, while maintaining in the main cache the popular words. The temporality-based caches [28] maintain a *used* bit in the tag of the associative buffer; a block that is referenced with the used bit on is promoted to the main cache – a block has to be referenced twice before it is promoted, and the used bit is updated on every reference to the block. If the block replaced in the associative buffer is chosen by an LRU approximation or randomly, blocks referenced only once will not necessarily evict a block that is still active.

[8] present a design that is similar to the PCVC. Rather than loading a missing blocks onto the associative buffer, a Bernoulli trial is used to load the incoming block either directly on the L1 or onto the buffer. To decrease access time and power, a direct mapped tag memory is used to map blocks onto the set that contains the data. The simulation results employed the SPEC2000 benchmarks and 16 and 32 Kbytes caches, a 2 Kbytes associative buffer (filter), and cache blocks of 64 bytes. The PCVC was conceived and designed for embedded applications and was assessed for smaller capacities and with applications from the embedded domain.

A dual- or multi-buffer design can be employed to capture references to different memory regions, as proposed by [21]: two separate caches are used to hold stack and global data, and a third cache holds references to the other regions. The partitioning can be defined along temporal and spatial locality, with a buffer allocated to each class of reference, and is designed with different block sizes [25]. These designs can be easily adapted to embedded applications [5]. Because of the high predictability of the reference streams in embedded applications [7], data can be statically allocated to a "temporal" cache or to a "spacial" cache [12], thus reducing the power needed by the memory system.

7 Summary

The designers of embedded systems have to design systems that yield the appropriate level of performance, in systems that are constrained by several design parameters. To achieve the design goals, a good model of the behaviour of the applications can simplify the design task by limiting the search space.

The *popularity* of a memory word is the fraction of references to this word over the total of memory references in the execution of a program. In this paper we define the popularity of memory addresses and sketch a method for designing a memory hierarchy based on the popularity metric. The method is illustrated by the measures of popularity for applications from the CommBench suite, and by memory hierarchy simulations.

We investigate a new technique for improving the performance of the top level of memory hierarchies for embedded systems. We introduce the Pollution Control Victim Cache (PCVC), that is simpler, smaller, uses less power and performs better than the Pollution Control Cache (PCC). Four designs were compared with respect to area and energy: a direct mapped L1 ($L1_{DM}$); a two-way set associative L1; a combination of $L1_{DM}$ and PCC; and a combination of $L1_{DM}$ and PCVC. Most of the working sets of the CommBench programs display a high concentration of reference on popular addresses (good locality), and the sizes of the most popular sets of words are of the order of 4-8 Kbytes. Caches with those capacities yield performance gains that are near to the maximum that might be achieved with a reasonable amount of resources. For these capacities (4-8 Kbytes), the Pollution Control Victim Cache performs well, with increases in area and energy that are roughly twice the gains in CPI, when compared to a simple $L1_{DM}$. For larger caches, the performance gains are of the same order as the increases in area and energy.

The experiments described here are simulations of programs that execute in isolation. In real systems, the processor runs more than a single application and the cache is shared by the application programs, a more or less sophisticated operating system, interrupt handlers and device drivers. These last two execute briefly and reference a few data words yet their references pollute the cache and thus decrease the performance of the applications. We intend to assess the effectiveness of the PCVC in keeping those references out of the main L1 cache.

Acknowledgements We thank Anca M Molnoş for many helpful suggestions. Renato Carmo was partially supported by grant 308692/2008-0, from CNPq. Roberto Hexsel was partially supported by grant BEX 1656-09-0, from CAPES.

References

[1] S G Abraham, R A Sugumar, D Windheiser, B R Rau, and R Gupta. Predictability of load/store instruction latencies. In *MICRO'93: 26th Intl Symp on Microarchitecture*, pages 139–152, 1993.

[2] A Agarwal, J Hennessy, and M Horowitz. An analytical cache model. *ACM Trans on Computer Systems*, 7(2):184–215, 1989.

[3] T Austin, E Larson, and D Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[4] O Avissar, R Barua, and D Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Trans on Embedded Computing Systems*, 1(1):6–26, 2002.

[5] L Benini, A Macii, and M Poncino. Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. *ACM Trans on Embeded Computer Systems*, 2:5–32, Feb 2003.

[6] Mark Brehob and Richard Enbody. An analytical model of locality and caching. Tech Report MSU-CSE-99-31, Michigan State University, Dept of Computer Science and Engineering, Aug 1999.

[7] R Cucchiara, M Piccardi, and A Prati. Exploiting cache in multimedia. In *IEEE Intl Conf on Multimedia Computing and Systems*, pages 345–350, Jul 1999.

[8] Y Etsion and D G Feitelson. L1 cache filtering through random selection of memory references. In *PACT'07: 16th Intl Conf on Parallel Architecture and Compilation Techniques*, pages 235–244, 2007.

[9] K I Farkas and N P Jouppi. Complexity/performance tradeoffs with non-blocking loads. In *ISCA'94: 21st Intl Symp on Computer Architecture*, pages 211–222, 1994.

[10] B Fields, R Bodík, and M D Hill. Slack: maximizing performance under technological constraints. In *ISCA'02: 29th Intl Symp on Computer Architecture*, pages 47–58, 2002.

[11] Brian R Fisk and R Iris Bahar. The non-critical buffer: Using load latency tolerance to improve data cache efficiency. In *ICCD'99: 1999 IEEE Intl Conf on Computer Design*, pages 538–545, 1999.

[12] P Grun, N Dutt, and A Nicolau. Access pattern based local memory customization for low power embedded systems. In *DATE'01: Conf on Design, Automation and Test in Europe*, pages 778–784, 2001.

[13] Giancarlo C Heck and Roberto A Hexsel. The performance of Pollution Control Victim Cache for embedded systems. In *SBCCI'07: 20th Conf on Integrated Circuits and Systems Design*, pages 46–51, 2008.

[14] Giancarlo Covolo Heck. Investigação de técnicas de projeto de cache de dados para sistemas embarcados. Dissertação de mestrado, Departamento de Informática, UFPR, Aug 2008. `http://www.inf.ufpr.br/roberto/dissGiancarlo.pdf`.

[15] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1st edition, 1990. ISBN 1-55860-069-8.

[16] B L Jacob, P M Chen, S R Silverman, and T N Mudge. An analytical model for designing memory hierarchies. *IEEE Trans on Computers*, 45(10):1180–1194, 1996.

[17] B L Jacob, S W Ng, and D T Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008. ISBN 0-12-379751-3.

[18] L K John and A Subramanian. Design and performance evaluation of a cache assist to implement selective caching. In *ICCD'97: 1997 Intl Conf on Computer Design*, pages 510–518, 1997.

[19] Norman P Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *ISCA'90: 17th Intl Symp on Computer Arch*, pages 364–373, 1990.

[20] David Kroft. Lock-up free instruction fetch/prefetch cache organization. In *ISCA'81: 8th Intl Symp on Computer Arch*, pages 81–85, 1981.

[21] H-H S Lee and G S Tyson. Region-based caching: an energy-delay efficient memory architecture for embedded processors. In *CASES'00: Intl Conf on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 120–127, 2000.

[22] S H Lee, J S Hang, S H Lee, J S Hong, and L Kerschberg. A popularity-driven caching scheme for meta-search engines: An empirical study+. *Database and Expert Systems Applications*, 2113:877–886, 2001.

[23] P Marwedel, L Wehmeyer, M Verma, S Steinke, and U Helmig. Fast, predictable and low energy memory references through architecture-aware compilation. In *ASP-DAC'04: Conf Asia South Pacific Design Automation*, pages 4–11, 2004.

[24] A Milenkovic, M Milenkovic, and N Barnes. A performance evaluation of memory hierarchy in embedded systems. In *35th Southeastern Symp on System Theory*, pages 427–431, Mar 2003.

[25] V Milutinovic, B Markovic, M Tomasevic, and M Tremblay. The split temporal/spatial cache: A complexity analysis. In *SCIzzL-6 Workshop*, pages 89–96, 1996.

[26] P R Panda, N D Dutt, and A Nicolau. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Trans Des Autom of Electronic Systems*, 5(3):682–704, 2000.

[27] R Rakvic, B Black, D Limaye, and J P Shen. Non-vital loads. In *HPCA '02: Proc 8th Intl Symp on High-Performance Computer Architecture*, pages 165–174, 2002.

[28] Jude A Rivers and Edward S Davidson. Reducing conflicts in direct-mapped caches with a temporality-based design. In *ICPP'96: Intl Conf on Parallel Processing*, volume 1, pages 154–163, 1996.

[29] P Shivakumar and N P Jouppi. CACTI 3.0: An integrated cache timing, power, and area model. Technical Report 2001/2, DEC-WRL, 2001.

[30] SimpleScalar LLC, Mar 2007. `http://www.simplescalar.com/`.

[31] Gurindar S Sohi. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *IEEE Trans on Computers*, 39(3):349–359, 1990.

[32] R R Souza, G C Heck, R Carmo, and R A Hexsel. Avaliação de desempenho, área e energia de caches com controle de poluição. In *WSCAD-SSC'09: X Workshop em Sistemas Computacionais de Alto Desempenho*, pages 1–8, 2009.

[33] S Srinivasan, R Ju, A Lebeck, and C Wilkerson. Locality vs. criticality. In *ISCA'01: 28th Intl Symp on Computer Architecture*, pages 132–143, 2001.

[34] Dominique Thiébaut and Harold S Stone. Footprints in the cache. *ACM Trans on Computer Systems*, 5(4):305–329, Nov 1987.

[35] S Vanichpun and A M Makowski. Comparing strength of locality of reference – popularity, majorization, and some folk theorems. In *INFOCOM'04: 23rd Annual Joint Conf of the IEEE Computer and Communications Societies*, pages 838–849, 2004.

[36] S J Walsh and J A Board. Pollution control caching. In *ICCD'95: Intl Conf on Computer Design*, page 300, 1995.

[37] S J E Wilton and N P Jouppi. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[38] Tilman Wolf and Mark A Franklin. CommBench – a telecommunications benchmark for network processors. In *ISPASS'00: IEEE Intl Symp on Performance Analysis of Systems and Software*, pages 154–162, 2000.