The performance of pollution control victim cache for embedded systems

2 authors:

Giancarlo C. Heck
Instituto de Pesquisa para o Desenvolvimento

9 PUBLICATIONS **10** CITATIONS

SEE PROFILE

Roberto Hexsel
Universidade Federal do Paraná

30 PUBLICATIONS **35** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Sistema Cartão Nacional de Saúde View project

Project Multicores and embedded systems View project

The Performance of Pollution Control Victim Cache for Embedded Systems

Giancarlo C Heck[*]   &   Roberto A Hexsel[†]

Departamento de Informática
Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081, Curitiba, PR, Brasil
{giancarlo,roberto}@inf.ufpr.br

ABSTRACT

In this paper we explore the design space of data caches looking for the combinations of design parameters that produce the best results at the smallest sizes. We introduce a technique named Pollution Control Victim Cache (PCVC) which improves the Pollution Control Cache (PCC), is simpler and performs better. Our simulations were run on the SimpleScalar suite running the Commbench benchmarks. Our results indicate that Victim Caches and Stream Buffers do not perform well in the small systems we simulated. The PCC and our PCVC have shown promising results.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Memory Structures—*Cache memories*

General Terms

Design, Performance

Keywords

Pollution Control Victim Cache, Embedded Systems

1. INTRODUCTION

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

In this paper we explore the design space of data caches looking for the combinations of design parameters that produce the best results at the smallest cache sizes. We present

---

[*]LACTEC - `www.lactec.org.br`

[†]Núcleo de Redes Sem Fio e Redes Avançadas (LARSIS)

simulation results for a memory hierarchy consisting of a first level data cache (L1), one or more specialized buffers, and DRAM. We introduce an improvement over the Pollution Control Cache, named PCVC, that is simpler and performs better than the original PCC [15]. Our simulations compare four low cost techniques for improving the performance of the cache hierarchy with the same benchmarks and processor-memory design parameters. Thus, the comparisons are "on a level playing field" and the performance gains from each of the designs can be assessed more easily. The simulators are based on the SimpleScalar suite [2] and we use precise models for the memory hierarchy and for the memory bus, and four different processors with different levels of instruction level parallelism and complexity. The workload consists of six applications in networking/multimedia from Commbench [16].

There are several techniques to improve the performance of the memory hierarchy, such as temporality-based caches [10], non-critical loads [5, 14], non-vital loads [9], and slack [4]. These techniques can indeed improve performance, but their implementations appear to be too complex for embedded systems. The annex-cache is similar to the victim cache [6] but we chose the PCC because it compares more directly to the victim cache.

An evaluation that is somewhat similar to ours is presented in [13], where the effect of design parameters on cache performance is examined. The authors investigate the effects of organization, associativity, capacity and replacement policy on miss rate. The authors assessed only the miss ratio, used the most basic cache simulators from the SimpleScalar suite [2], and run simulations of the general purpose SPEC 2000 benchmarks.

In Section 2 we describe the simulation environment and methodology, while in Section 3 we present the designs for the various caches/buffers. Section 4 contains the simulation results and discussion, and our conclusions are presented in Section 5.

2. SIMULATION ENVIRONMENT

The simulations were run on modified versions of the *sim-outorder* simulator from the SimpleScalar suite [11, 2, 1]. Sim-outorder is an execution driven, cycle-accurate, out-of-order simulator. The simulators run the applications from the CommBench suite [16] since these are representative of the workloads found in networking systems. The reader should refer to [16] for details concerning the benchmark programs.

We modelled four processors, each defined by the triplet ⟨*scalarity, RUU size, LSQ size*⟩. By *scalarity* we mean the width of the fetch, decode, issue, commit stages, and the number of integer and floating point ALUs. In all models there is only one floating point multiplier/divider. The *RUU* is the Register Update Unit [12], similar to a reorder buffer, and the LSQ is a load-store queue that holds memory references that await completion at memory. The four models are ⟨1,8,4⟩, ⟨2,16,8⟩, ⟨4,32,16⟩, and ⟨8,64,32⟩. Instructions and data are 32 bits wide.

The cache models were simulated with capacities of 1, 2, 4, 8, and 16 KB, and with block sizes of 8, 16, 32 and 64 bytes. Unless stated otherwise, the data cache is direct-mapped. All CPU models have 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 KB so its performance does not interfere with that of the data cache.

Miss Status Holding Register.

In a *lockup-free cache* a reference that misses in the cache is recorded in a *Miss Status Holding Register* (MSHR) buffer while the missing datum is fetched from memory. During the fetch other memory references that hit can be serviced directly from the cache. An MSHR holds all the information regarding an outstanding miss: address of the missing block, functional unit to which the missing word must be delivered, and several status and control bits [8]. In a given memory system, the number of MSHRs determines how many outstanding misses can be sustained by the cache in any interval. If there are no free MSHRs, on a miss the processor must stall until one becomes free. Some improvements to the original lockup-free caches are presented in [3], along with some design variations that reduce the stalls caused by more than one outstanding miss to the same cache block.

Memory interface in SimpleScalar and MSHRs.

The model of the interface between the L1 caches and L2/memory in SimpleScalar assumes there is an unlimited number of MSHRs, 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 an 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 in [3].

A series of experiments were run to assess the impact of limiting the number of outstanding misses. Figure 1 shows the changes in the number of instructions completed per cycle (IPC) as capacity and block size are varied. We tested two extreme versions of the processor: one is a single-issue pipeline, and the other is an aggressive superscalar processor
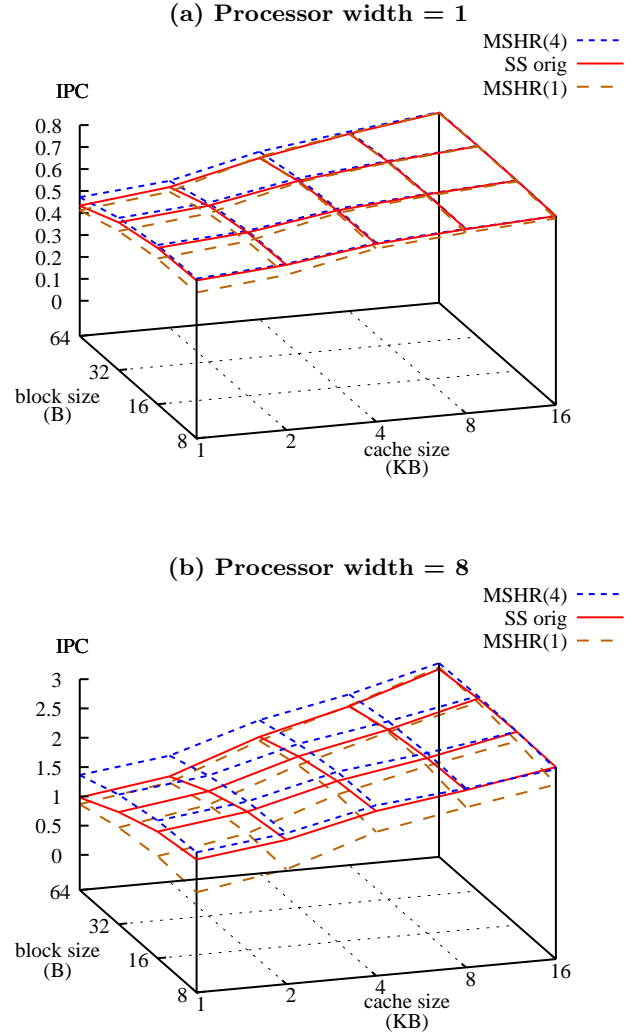

(a) Processor width = 1

(b) Processor width = 8

Figure 1: Original SS *versus* MSHR limited SS

that can issue eight instructions per cycle. The values shown are averaged over the six benchmarks.

The effects of a non-blocking cache on a narrow single-issue pipeline are minimal since at any one cycle there is at most one load or store at the memory interface. As expected, the plots for this processor show slight differences in performance between the original SimpleScalar and the model with MSHRs. For large blocks (64 bytes), the MSHRs add a limited degree of associativity to the direct mapped cache since secondary references to an outstanding miss are all recorded on the MSHR and serviced directly from it, thus incurring in smaller latencies. Furthermore, for the smaller caches, with as few as 16 blocks (1 KB), the MSHRs add capacity and thus hit ratios improve. The plots for 1 MSHR show a decrease in IPC, when compared with an unlimited number of MSHRs. This is expected because of the more precise accounting of latency and hit ratio.

The wide-issue processor imposes the highest demand on the memory system, since it is possible to execute up to eight concurrent memory references. As shown on Figure 1, for caches with many blocks (8 bytes/block) there is little difference between the original model and that with 4 MSHRs.

For small caches with few blocks (1KB=16x64 bytes/block) the number of secondary hits on the MSHRs are sufficient to improve the performance of our model by roughly one half of an IPC. This improvement comes from the combined effects of associativity, and the victim block being held in the cache until just before the new block is loaded, possibly affording more hits on the outgoing block. As expected for this wide-issue processor, a single MSHR performs the worst in all combinations of block and cache sizes.

3. CACHE MODELS

In all that follows the simulator is run with four MSHRs, results are shown only for processors capable of issuing two and four instructions per cycle, and block sizes are either 16 or 32 bytes.

Victim Cache.

A *victim cache* (VC) is a small fully-associative buffer that holds blocks which were evicted from the L1 [7]. The VC adds some associativity to a direct mapped cache, thus eliminating costly conflict misses, without increasing significantly its size or complexity. Figure 2 shows a block diagram of a VC attached to a direct mapped cache.
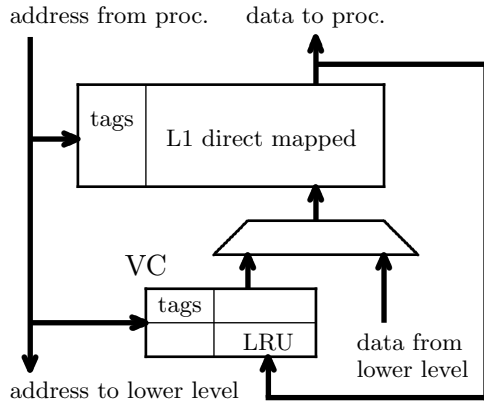
Figure 2: Victim cache

Stream Buffer.

A *stream buffer* (SB) is a small associative buffer that prefetches cache blocks with addresses just beyond that of a missing block [7]. If the processor references words in some linear sequence, it is likely that the stream buffer can fetch blocks ahead of the misses. When the actual references do occur, they are serviced from the buffer with low latency. The block diagram of a stream buffer is very similar to that in Figure 2, except that the SB operates as a queue rather than as a cache. An SB can be very efficient in removing capacity and compulsory misses in L1, and is also efficient in reducing conflict misses on the instruction cache. In both data and instruction fetches, an SB can lead to substantial improvement in the latency of the references that miss. However, bus traffic can be dramatically increased by the eager fetches that are not always useful.

Pollution Control 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. On a miss, this pollution tends to cause two misses, one to load the unpopular block and another to reload the evicted popular block. The *Pollution Control Cache* (PCC) is a small fully associative cache that operates in parallel with the L1 [15]. When a miss occurs, the block is loaded on the PCC; if this block is referenced a second time, then it is moved to the L1, possibly avoiding the eviction of a popular block.

4. RESULTS

Victim Cache.

The simulation results for the victim cache are shown in Figure 3, with a processor of width two and block size of 16 bytes, and compared to the *base model* that is equipped with 4 MSHRs ⟨MSHR(4)⟩. On the top are shown the averages of the miss rates for the six programs. For the smallest cache (1 KB) the addition of a victim cache with 16 blocks ⟨VC(16)⟩ improves the miss rate by 11%, while with a VC with just one block ⟨VC(1)⟩ the improvement is 6%. The VC adds both capacity (more blocks) and associativity to the cache, and this is more evident for the smaller sizes. For the larger caches, the gains are inversely proportional to cache size. The Figure 3 (bottom), shows for a VC with 4 blocks, the miss rates for the individual programs. The benefits are obvious for the smaller caches, especially for the applications JPEG-ENC, CAST-ENC and DRR, that have large data sets and good temporal locality.
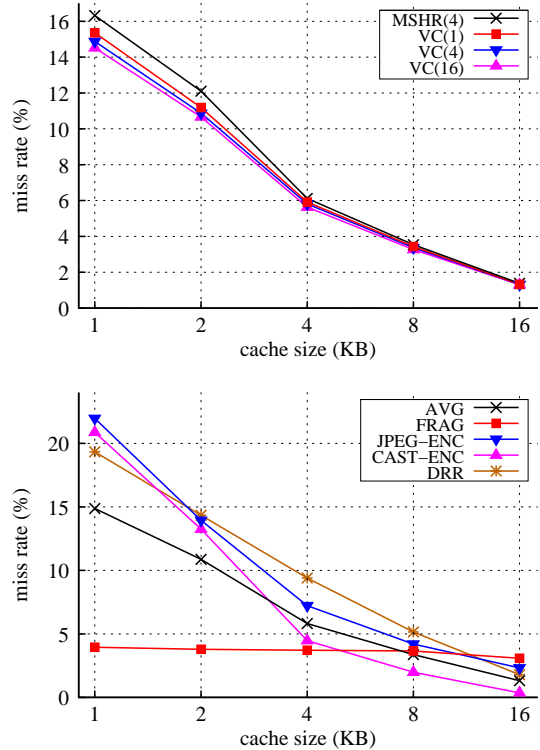
Figure 3: MSHR and VC, width=2, block_size=16

Stream Buffer.

The tags in the simulated stream buffers (SB) are checked on all blocks in all ways because this improves the latency for reference streams of arbitrary stride. On a hit on the SB, the requested word is forwarded to the processor and L1. In the plots shown below, the organization of the buffers is given as $\langle \mathrm{SB}(depth, associativity)\rangle$. Figure 4 shows bus occupation for processors of width (w) one and four, and block size of 32 bytes (b_s=32)). This block size was chosen since it causes the most traffic on the bus. For SBs with a single block $\langle \mathrm{SB}(1, \mathrm{x})\rangle$, bus occupation is roughly 30% higher than in the base model. SBs with more than four blocks have a much higher occupation, but associativity alleviates this, as with $\langle \mathrm{SB}(4, 1)\rangle$ and $\langle \mathrm{SB}(4, 4)\rangle$, for example.
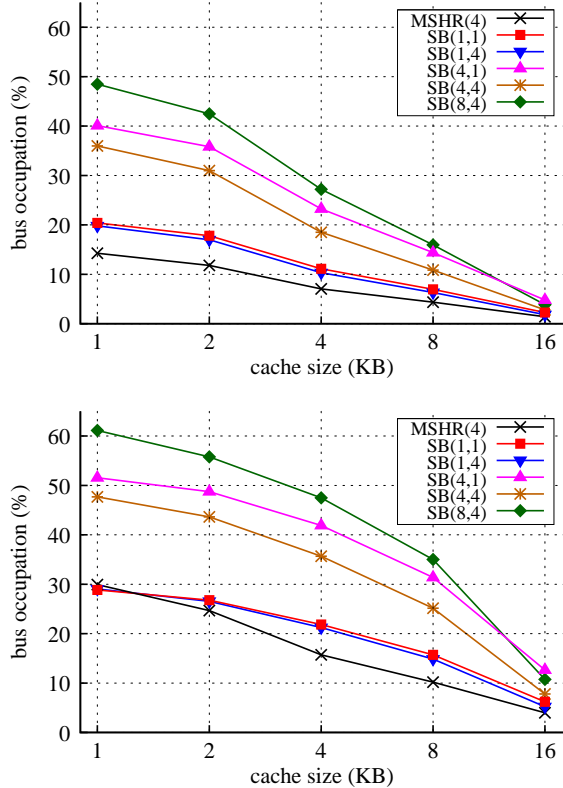

Figure 4: SB: bus occupation for processors of w=1 (top) and w=4 (bottom), b_s=32

In [7], the author investigates the effects of the SB on miss rates and ignores bus occupation and IPC. Figure 5 shows plots of miss rate (top) and IPC (bottom) for a processor of width two (w=2) and block sizes of 32 bytes. The smaller SBs, $\langle \mathrm{SB}(1, 1)\rangle$ and $\langle \mathrm{SB}(1, 4)\rangle$, cause the lowest bus occupation and the higher miss ratios while yielding the best performance, or perhaps, not the worst performance. Except for the largest cache sizes, the SBs cause a decrease in performance. This can be explained by a *very* rough estimate: there is a memory reference every 4 instructions (2 cycles for width=2), with more than one miss every 10 references. There is a miss every 20 cycles and each miss is serviced in 20 cycles. If the SB is causing additional fetches from memory (some of which useless) then bus occupation and cache refill latency both increase and slow down the processor.
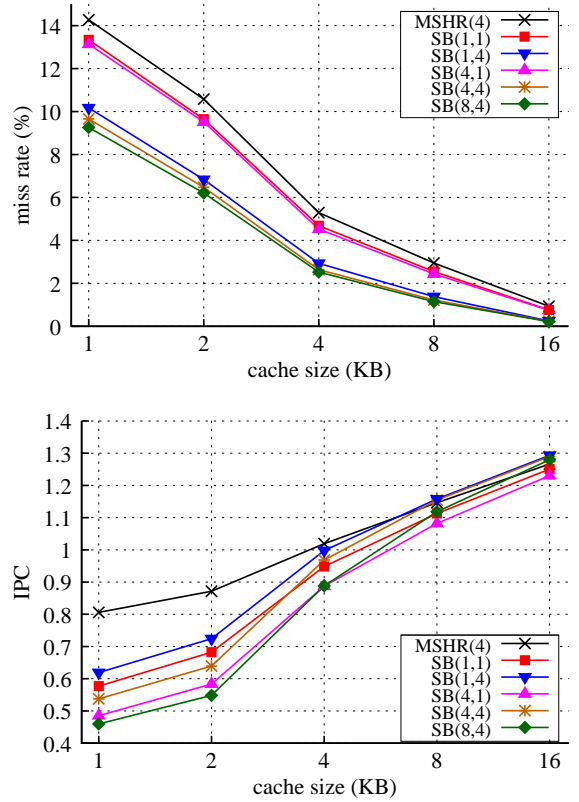

Figure 5: SB: miss rate (top) *versus* IPC (bottom), w=2, b_s=32

Pollution Control Cache.

We simulated two different models of the pollution control cache, namely *PCC1* and *PCVC*, with the second differing from the proposal in [15]. Figure 6 shows the two models. In the *PCC1* there is an 4 block VC attached to the L1 so that blocks evicted from L1 are stored in the VC. The three buffers L1, VC and PCC are searched in parallel on a reference. In our new model, named *Pollution Control Victim Cache (PCVC)*, the PCC acts as a victim cache for L1 so that blocks evicted from L1 are stored back in the PCC. In the plots shown below, the organization of the buffers is given as $\langle \mathrm{PCC1}(PCC\ blocks, VC\ blocks)\rangle$.

PCC1.

On a miss in L1, the block α is loaded onto the PCC. On a second reference to block α, 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 for the 1 KB 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 the IPC, as can be seen on the bottom of Figure 7. For the larger PCC1s, the gains in IPC range from 13% (8 blocks) to 15% (32 blocks).

PCVC.

On a miss in L1, the block α is loaded onto the PCC. On a second reference to block α, it is swapped with the block
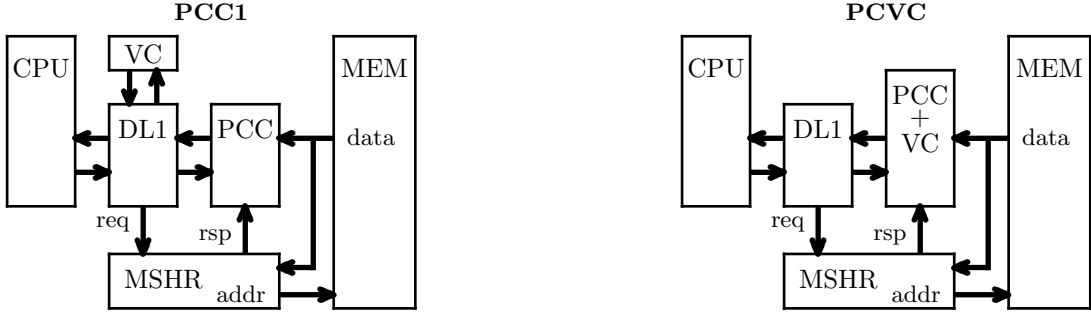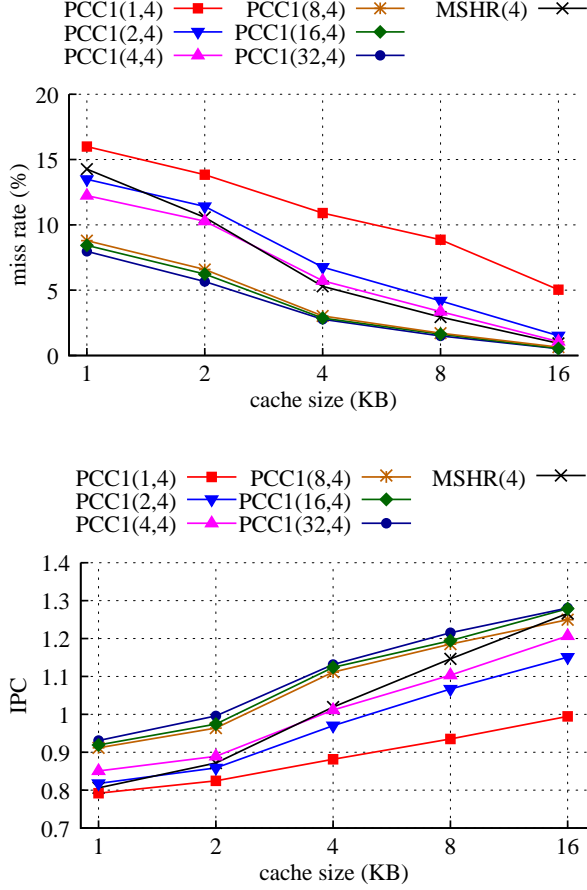
Figure 6: PCC1 and PCVC

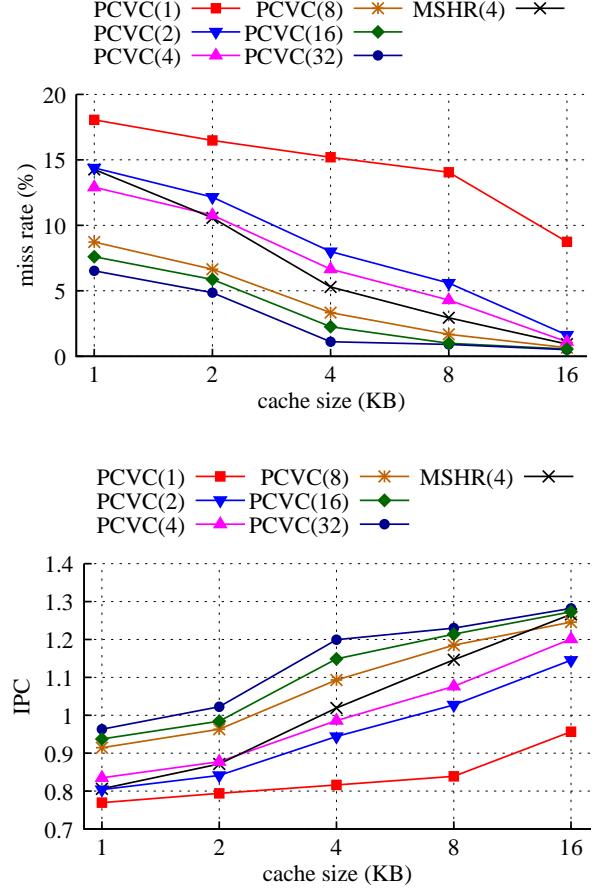**Figure 7: PCC1: miss rate (top) and IPC (bottom), w=2, b_s=32**

Figure 8: PCVC: miss rate (top) and IPC (bottom), w=2, b_s=32

evicted from the L1. The behavior of the smaller PCVCs is slightly better than that of the smaller PCC1s, 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 32 block PCVC the gains are 64%, 88%, and 119%, respectively. Since the PCC 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 the PCC with 8 and 32 block, respectively.

PCC1 versus PCVC.

Our simulations indicate that a PCVC is a more effective design than a PCC1 because it is simpler —comparisons on two sets of tags rather than three— and smaller —only n blocks in the PCVC rather than PCC(n)+VC(m) blocks.

Table 1 compares the results in Figures 7 and 8. It shows the performance gains achieved with the PCC1 and the PCVC when compared to the 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 1 KB+PCVC(32) is compared to a 2 KB L1, a 2 KB+PCVC(32) to a 4 KB L1, and so on. The last three rows compare an N KB L1+PCVC(32) to caches with capacity $4N$, $8N$ and $16N$.

cache size	1K	2K	4K	8K	16K
IPC base MSHR(4)	0.81	0.87	1.02	1.15	1.27
PCC1(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
PCC1(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
PCC1(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

Table 1: IPC gains of PCC1 and PCVC

For the smallest 1 KB caches, the L1+PCVC with at least 4 blocks has a better performance than a simple L1 twice as large, while for the PCC1 8 blocks are needed. The results for the 32 block PCVC show a 10% improvement in IPC over the simple 2 KB L1, and just a 4% *decrease* when compared to a simple L1 four times larger (4 KB) and with twice the capacity of the combined L1+PCVC. For the 8 block PCC1 and PCVC, the gains in performance over a simple 2 KB 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 1 KB+PCVC(32) is 0.78 of that of a 16KB L1.

5. CONCLUSION

We investigated four techniques for improving the performance of memory hierarchies for embedded systems. The simulation results indicate that a Victim Cache might be useful for applications that have large data sets and good temporal locality. For applications without good locality, the Victim Cache did not perform well for the sizes, programs and data sets we employed. For the class of systems we tested, the Stream Buffer performed rather poorly. It may show better performance on a more complex bus, or if placed between the L2 and DRAM where the bus traffic is less intense than between the L1 and processor. The two versions of the Pollution Control Cache we tested have shown promising results, with our design, the PCVC, outperforming the more complex PCC1. We intend to compare the size, complexity and power consumption of the PCC1 and PCVC.

6. REFERENCES

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

[2] D Burger and T M Austin. The SimpleScalar Tool Set, Version 2.0. Technical report, University of Wisconsin-Madison and SimpleScalar LLC, 1997.

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

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

[5] B. R. Fisk and R. I. Bahar. The non-critical buffer: Using load latency tolerance to improve data cache efficiency. In *ICCD '99: Proc 1999 IEEE Intnl Conf on Computer Design*, pages 538–545, Oct. 1999.

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

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

[8] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA '81: Proc 8th Annual Symp on Computer Architecture*, pages 81–87, 1981.

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

[10] J. A. Rivers and E. S. Davidson. Reducing conflicts in direct-mapped caches with a temporality-based design. In *ICPP '96: Proc 1996 Intntl Conf on Parallel Processing, Vol. 1*, pages 154–163, 1996.

[11] SimpleScalar LLC, mar 2007. http://www.simplescalar.com/.

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

[13] M. Soryani, M. Sharifi, and M. H. Rezvani. Performance evaluation of cache memory organizations in embedded systems. In *ITNG '07: 4th Intnl Conf on Information Technology 2007*, pages 1045–1050, Apr. 2007.

[14] S. Srinivasan, R. Ju, A. Lebeck, and C. Wilkerson. Locality vs. criticality. In *ISCA '01: Proc 28th Intntl Symp on Computer Architecture*, 2001.

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

[16] T. Wolf and M. A. Franklin. CommBench - a telecommunications benchmark for network processors. In *Proc IEEE Intntl Symp on Performance Analysis of Systems and Software (ISPASS)*, pages 154–162, Apr. 2000.