An Adaptive Victim Cache Scheme

Osvaldo Navarro and Michael Hübner
Department of Electrical Engineering and Information Technology
Ruhr-Universität Bochum, Bochum, Germany Telephone: +49-234 32-25975 Fax: +49-234 32-14499
Email: {Osvaldo.NavarroGuzman, Michael.Huebner}@rub.de

***Abstract*—A victim cache is a small cache block usually located between two main cache levels, which main objective is to recover conflict cache misses. In the usual case, the victim cache is designed as an always enabled cache block with fixed size. However, different applications may have very different memory access requirements. In this article, we present an analysis of the relations between a victim cache, the cache's logical organization and the application's behaviour, and based on preliminary results, an victim cache scheme which is reconfigured at runtime to adapt to the current conditions of the system is proposed.**

I. Introduction

Cache memories are a crucial component of microprocessors, since they greatly decrease the accesses to main memory, diminishing the performance bottleneck in current microprocessors caused by memory latency. The cache system's performance varies with each application's memory access behavior and its power consumption can amount up to 50% of that of the overall system [1]. Thus, the design of an energy-efficient cache system is an important and non-trivial task.

To reduce the number of conflict cache misses, a small cache known as *victim cache* is placed between the first and second level of the cache system, and collects the evicted cache blocks from the first cache level. If an evicted block is requested and is found in the victim cache, it can be accessed without requesting it from main memory. The victim cache is designed fully associative in the usual case, so it provides the high associativity needed by the memory accesses which are likely to result in conflict misses.

Several approaches have shown that reconfiguring the cache at runtime to fit the application's memory access behaviour can yield and improvement in energy efficiency [2]. In this article, we realize an analysis of the relations between the cache and the victim cache for three benchmark applications, and propose as a continuation for this work, an adaptive victim cache scheme, which will reconfigure the victim cache at runtime such that it maximizes its energy efficiency given the runtime status of the cache and the application's behaviour.

II. Design Tradeoffs

Figure 1 shows a general scheme of a microprocessor with a cache system of two levels and a victim cache between the first and the second level of the cache. Using this model, we realized several experiments to observe the relation between the victim cache, the application's behavior and the cache's logical organization.

A. Experimental Setup

To perform the experiments we used the sim-alpha simulator [3], which models an Alpha 21264 microprocessor [4]. Sim-alpha contains a configurable L1 data cache similar to those of the 21264, which allows for experimentation with different parameters. The configuration chosen for the experiments is shown in Table I. For test applications, we chose three applications: *fft, jpeg* and *stringsearch*, from the MiBench benchmark [5].

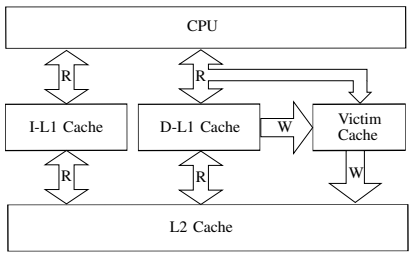

Fig. 1. General diagram of the approach. The figure shows the victim cache, which stores the evicted cache blocks from the D-L1 cache and is checked upon memory requests at the same time than the D-L1 cache.

TABLE I. **Simulation parameters**

Parameter	Value
L1 Instruction Cache	512 sets, 64B blocks, 2-way associative, 1 cycle latency
L1 Data Cache	512 sets, 64B blocks, 2-way associative, 3 cycle latency
L2 Unified Cache	32768 sets, 64B blocks, direct-mapped, 7 cycle latency
Instruction/DATA TLB	32-Entries Fully Associative
Victim Cache	64B blocks, fully associative, 1 cycle latency

Simulation parameters used for the experiments.

B. Preliminary Results and Discussion

Figure 2 shows the percentage of cache misses recovered by the victim cache, for different victim cache sizes. The figure shows that for the *fft* and *jpeg* applications, the number of cache misses recovered increases with the victim cache size. In the case of *fft*, the percentage of misses recovered goes up to 27.8% when having a victim cache of 20 entries, while in the case of *stringsearch*, it increases in a slower way, reaching 3% when having a victim cache of 8 entries and going up to 3.18% at the largest victim cache. In the case of *jpeg*, the victim cache only recovered one miss with the 3 largest victim caches and none for the rest.

The simulation time in number of cycles for these experiments is shown in table II. The simulation time of the *fft* and *stringsearch* applications increases with the victim cache size, most likely due to the delay introduced with the search in a larger victim cache. However, with the larger victim caches, the simulation time decreases and when reaching 12 entries in the case of the *fft* application and 16 entries in the case of the *stringsearch* application, the simulation time gets lower even than the one registered when using the smallest victim

978-1-4799-5944-0/14/$31.00 ©2014 IEEE

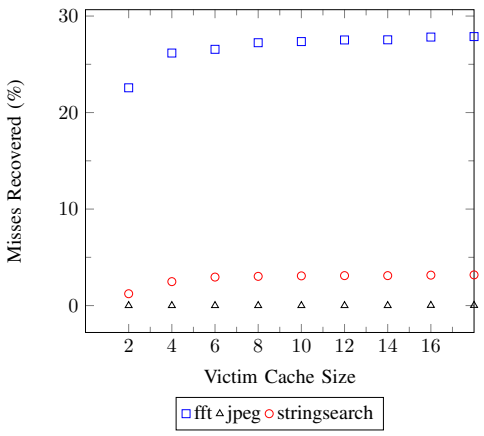

Fig. 2. Percentage of misses recovered for different victim cache sizes

cache. This likely occurs because with a higher percentage of recovered misses more accesses to main memory are avoided. Since the victim cache of the *jpeg* application did not recovered any cache misses, its simulation time did not decrease, and even increased when using the larger victim caches.

TABLE II. **Simulation time for different victim cache sizes**

Victim Cache Size	Simulation Time (cycles)		
	fft	jpeg	stringsearch
2	184328426	29118934	2938595
4	184350020	29118934	2938343
6	184350165	29118934	2938175
8	184343855	29118934	2938175
10	184350109	29118934	2938175
12	184312084	29118934	2938175
14	184312084	29125789	2841135
16	184312084	29125789	2938166
18	184312084	29125789	2938166
20	184312084	29125789	2938166

Total simulation time in number of cycles for different victim cache sizes.

Figure 3 shows the percentage of cache misses recovered by the victim cache when having different associativity. The victim cache was fixed at 20 entries for this set of experiments. The graph shows that at an associativity level of 16, none of the applications registered recovered misses. However, when the associativity decreases, more conflict misses occur, and then the victim cache begins to recover a percentage of them. At the lowest level of associativity, a direct-mapped cache, the percentage of recovered misses goes up to 86.7% in the case of the *fft* application. The victim cache of the *jpeg* application had the least recoverings of the three applications, registering a 0.02% at a 2-level associativity cache and up to 3.24% with the direct-mapped cache.

Table III shows the simulation time in number of cycles for these experiments. In these experiments, the simulation time increased as the associativity level decreased, even when the percentage of misses recovered victim cache increased. This is most likely because when having a very low associativity the number of conflict misses generated are much more than what the victim cache can recover.

Figure 4 shows the percentage of misses recovered by the victim cache for different cache sizes. The victim cache was fixed to 20 entries in these experiments and the rest of the parameters remained with default values. The figure

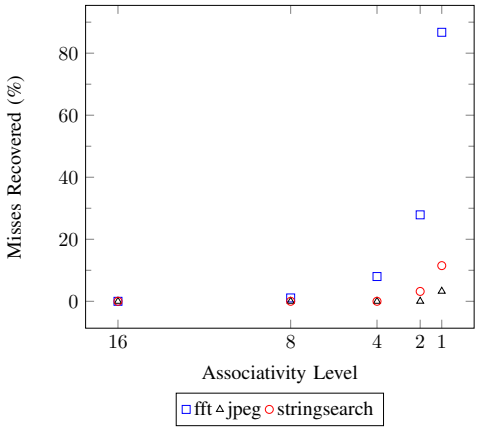

Fig. 3. Percentage of misses recovered by the victim cache for different associativity levels, and a victim cache of 20 entries.

TABLE III. **Simulation time for different associativity levels**

Associativity level	Simulation Time (cycles)		
	fft	jpeg	stringsearch
16	184164881	29090141	2937917
8	184203106	29090141	2937917
4	184224716	29090141	2937917
2	184312084	29125789	2938166
1	185309702	29151115	2956278

Total simulation time in number of cycles for different associativity level and a victim cache of 20 entries.

shows that for a large cache (e.g. 1024 sets), the victim cache does not recover any misses, even though the cache has low associativity (2-level associativity). However, as the cache size decreases, the victim buffer starts recovering more misses in all cases. This is more notorious in the case of the *fft* application, which recovering percentage gets up to 90.68% when having a cache size of 32 sets. Table IV shows the simulation time for these experiments. Here the table shows a general increase in simulation time as the cache size decreases, most likely because the increase in the number of cache misses when having smaller caches is too big to be handled by a victim cache of 20 entries.

Finding the most suitable victim cache size brings also a decrease in energy consumption. Following the Equation 1,

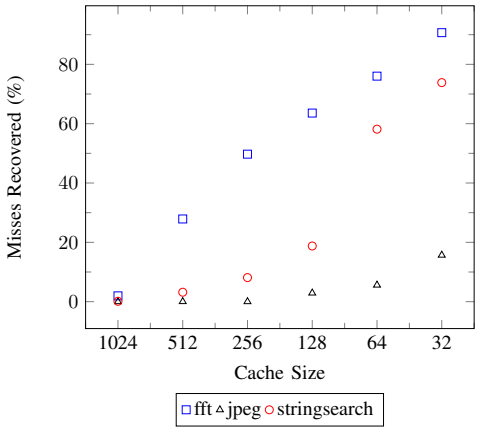

Fig. 4. Percentage of misses recovered by the victim cache for different cache sizes and a victim cache of 20 entries.

TABLE IV. **Simulation time for different cache sizes**

Cache Size	Simulation Time (cycles)		
	fft	jpeg	stringsearch
1024	184249934	29090141	2938129
512	184312084	29125789	2938166
256	184396594	29150011	2940752
128	184467836	29145802	2942104
64	184522937	29145127	2948794
32	185363622	29143687	2974222

Total simulation time in number of cycles for different cache sizes and a victim cache of 20 entries.

the energy reduction introduced by the victim cache can be calculated as the dynamic energy that would be spent by the cache misses recovered E_{miss_rec} minus the dynamic energy consumption of the victim cache E_{vic}. Then, E_{miss_rec} is calculated as the level 2 cache block's energy consumption per access E_{L2_acc} times the number of recovered cache misses N_{miss_rec}, and E_{vic} is calculated as the energy consumption per access of the victim cache E_{vic_acc} times the number of accesses to that block N_{vic_acc}.

$$E_{red} = E_{miss\_rec} - E_{vic} \quad (1)$$
$$= (E_{L2\_acc} * N_{miss\_rec}) - (E_{vic\_acc} * N_{vic\_acc}) \quad (2)$$

For instance, considering the experiments of Figure 2, using a victim cache with 20 entries, the *fft* application gets a decreasing in energy consumption of $287484nJ$, but in the case of the *stringsearch* application, the energy consumed by the victim cache exceeds the energy saved by avoiding accesses to the L2 cache by $239nJ$. The values for the equation were obtained with CACTI 2.0 [6].

The results obtained show that the percentage of the cache misses recovered by the victim cache vary in a great way in relation with the application's behavior and the logical organization of the cache. On the one hand, an application which has a considerable amount of data requests with high associativity will get a higher performance using a large victim cache. On the other hand, applications which do not need high associativity will barely make use of the victim cache, as is the case of the *jpeg* application (see Figure 3). Furthermore, the victim cache's effectiveness varied significantly when using different cache sizes and different levels of associativity. When constrained to a small cache and/or a low level of associativity, more conflict misses arise, which are then handled over to the victim cache. In this case a large victim cache will be more useful to decrease the accesses to main memory.

III. Proposed Approach

Since the victim cache's effectiveness depends on the cache's logical organization and the application's data, a victim cache that increases its size when having numerous requests for data which requires high associativity and decreases its size or shuts down completely when is not required, would provide a significant improvement in energy efficiency. Thus, as a continuation of this work, an adaptive victim cache will be designed and developed. This scheme will reconfigure the victim cache at running time, such that it tunes to the conditions provided by the application's behavior and the rest of the cache system. As a platform for this scheme, we will use the heuristic and tune module proposed in [7]. This scheme enables the reconfiguration of the cache's size, line size and associativity at runtime, thus providing a good baseline for our proposed scheme. For the resizing mechanism, we plan to adapt the *way concatenation* technique [1] to the victim cache module.

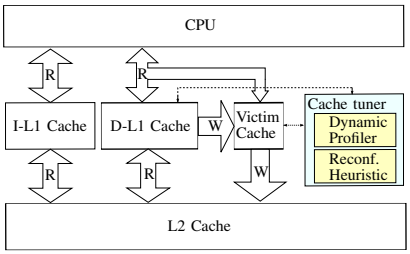

Fig. 5. General diagram of our proposed approach. The figure shows a cache tuner module alongside the cache system of Figure 1, which will reconfigure the victim cache at runtime.

Figure 5 shows a general diagram of our proposal. This approach comprises a dynamic profiling scheme which calculates the runtime cache energy efficiency, by monitoring parameters such as the number of cache hits and misses, and the current cache's dimensions, *i.e.*, cache size, block size, and associativity level. Moreover, a reconfiguration heuristic will determine the best victim cache configuration based on the measures of the dynamic profiler. Then, the victim cache will be modified to the determined size or the module will be turned on/off according to the new configuration. This process can be triggered at the start of each application or at the start of a different application's phase. Finally, an energy model will be used along testing with benchmark applications to evaluate the energy efficiency of the cache system and validate that the overhead introduced by the adaptive scheme remains in an acceptable level. The tuner chosen as a platform has only 3% area overhead [7], and the resizing logic is negligible (four OR and four AND gates) [1]. Since the victim cache is much smaller than the cache and has less configurable features (resize or disable), the overhead for our approach is expected to be significantly lower.

Our approach will aim to provide an improvement in the energy efficiency of the system, since the victim cache will only be enabled and with the required size by the current conditions imposed by the cache and the running application, thus decreasing the energy consumption of the main memory accesses. With the formula shown in the Equation 3, we can estimate the rate of energy reduction R_{red} provided by our approach, and also estimate the maximum energy E_{tuner} that the tuning module can consume, such that the resultant energy reduction remains beneficial. In this equation, E_{L1} means the energy consumed by the data level 1 cache and N_{L1_misses} means the number of misses in the data level 1 cache.

$$R_{red} = \frac{E_{red} - E_{tuner}}{E_{L1} + (E_{L2\_acc} * N_{L1\_misses})} \quad (3)$$

A. Application Domain

The flexibility provided in our approach make it specially suitable for systems which have a strong requirement for

energy efficiency. This include systems with high resource constraints, such portable devices, which are required to provide high performance within a small area and with a limited power-source, and large systems where energy savings is a challenging issue, such as large data centers [8]. Moreover, our approach could also benefit cyber-physical systems, such as wireless sensor/actuator networks, which use a quality of service (QoS) management, since there is usually a tradeoff between guaranteeing reliability while minimizing energy consumption [9].

B. Implementation

To make the most of the flexibility of the proposed victim cache, its implementation should seek to reconfigure the disabled victim cache, for instance, to perform specific computations and obtain a benefit in performance. This can be achieved using dynamic resource configuration with an FPGA, as proposed in [10]. Although the experiments of this work were realized in a single processor system, it can be easily applied in a multicore scenario as well, in the case of having a victim cache per core. In the case of having a shared victim cache, the profiler and heuristic should be extended to consider the accesses to each cache block connected to the victim cache, to ensure that the victim cache is large enough to recover the conflict misses of each core, only when the improvement in energy efficiency is significant.

IV. Related Work

With the introduction of victim caches, [11] already reported percentages of cache misses avoidance up to 95%, and also getting 25% of avoided misses with a victim cache of only 2 entries. Since then, more sophisticated victim cache schemes have been proposed. [12] describes an approach for saving power consumption, by analyzing the application source code, and then inserting instructions which control a victim cache module. These instructions can turn on/off the victim cache and decide which data is sent to the victim cache and which is discarded after a cache miss, according to the application's cache requirements. Moreover [13] proposed using the cache blocks predicted not to be used any more before their eviction, which are called *dead blocks*, as a virtual victim cache, yielding a significant improvement in cache efficiency. Furthermore, several approaches have shown that adapting the cache to the running application improves the energy efficiency of the overall system. In this regard,[2] describes several approaches where features such as cache size, associativity level, and cache block size are reconfigured online. These approaches monitor the behavior of the cache during runtime and adjust the mentioned features accordingly, such that the resultant cache yields a better energy efficiency. So far, an online profiling and adaptive scheme for the victim cache has been unexplored, to the best of our knowledge.

V. Conclusions and Future Work

In this article we present an analysis of the relation of the victim cache with the cache's logical organization and the behaviour of the application, and an adaptive victim cache scheme is proposed as a continuation of this work. Preliminary results show that the effectiveness of the victim cache depends significantly on the cache size, associativity level and also the the application's data requests. Thus, a victim cache which can be reconfigured at runtime to tune to these conditions, such that it would only be enabled and with just the required size when it can yield a significant improvement in performance. Therefore, as future work, an adaptive victim cache scheme will be designed and developed. The scheme will aim to provide the best victim cache reconfiguration according to the current cache logical organization and the application's behaviour, which will yield an improvement in energy efficiency and flexibility of the overall system.

Acknowledgment

This work was done under partial support of CONACyT (Grant 359472)

References

[1] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Proceedings of the Int. Symp. on Computer Architecture*, no. June, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1206995

[2] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, pp. 1–11, Nov. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S2210537913000516

[3] R. D. Doug, D. Burger, S. W. Keckler, and T. Austin, "Sim-alpha: a validated, execution-driven alpha 21264 simulator," 2001.

[4] R. E. Kessler, "The alpha 21264 microprocessor," *Micro, IEEE*, vol. 19, no. 2, pp. 24–36, 1999.

[5] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[6] G. Reinman and N. P. Jouppi, "Cacti 2.0: An integrated cache timing and power model," *Western Research Lab Research Report*, vol. 7, 2000.

[7] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 407–425, 2004.

[8] L. Parolini, N. Tolia, B. Sinopoli, and B. H. Krogh, "A cyber-physical systems approach to energy management in data centers," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*. ACM, 2010, pp. 168–177.

[9] F. Xia, L. Ma, J. Dong, and Y. Sun, "Network qos management in cyber-physical systems," in *Embedded Software and Systems Symposia, 2008. ICESS Symposia'08. International Conference on*. IEEE, 2008, pp. 302–307.

[10] H. Kim, A. K. Somani, and A. Tyagi, "A reconfigurable multifunction computing cache architecture," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 4, pp. 509–523, 2001.

[11] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *Computer Architecture, 1990. Proceedings., 17th ...*, pp. 364–373, 1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=134547

[12] C.-Y. Lee, J.-C. Chang, and R.-G. Chang, "Compiler Optimization to Reduce Cache Power with Victim Cache," *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pp. 841–844, Sep. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6331990

[13] S. Khan, D. Jiménez, D. Burger, and B. Falsafi, "Using dead blocks as a virtual victim cache," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, no. September, 2010, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1854333