# Using a Victim Buffer in an Application-Specific Memory Hierarchy

Chuanjun Zhang
Department of Electrical Engineering
University of California, Riverside
czhang@ee.ucr.edu

Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside
vahid@cs.ucr.edu
Also with the Center for Embedded Computer Systems at UC Irvine

***Abstract*** *– Customizing a memory hierarchy to a particular application or applications is becoming increasingly common in embedded system design, with one benefit being reduced energy. Adding a victim buffer to the memory hierarchy is known to reduce energy and improve performance on average, yet victim buffers are not typically found in commercial embedded processors. One problem with such buffers is, while they work well on average, they tend to hurt performance for many applications. We show that a victim buffer can be very effective if it is considered as a parameter in designing a memory hierarchy, like the traditional cache parameters of total size, associativity, and line size. We describe experiments on PowerStone and MediaBench benchmarks, showing that having the option of adding a victim buffer to a direct-mapped cache can reduce memory-access energy by a factor of 3 in some cases. Furthermore, even when other cache parameters are configurable, we show that a victim buffer can still reduce energy by 43%. By treating the victim buffer as a parameter, meaning the buffer can be included or excluded, we can avoid performance overhead of up to 4% on some examples. We discuss the victim buffer in the context of both core-based and pre-fabricated platform based design approaches.*

## 1 Introduction

Energy consumption is an important issue for battery powered embedded systems. On-chip cache consumes almost half of a microprocessor's total energy [11]. Energy efficient cache architecture design is thus a critical issue in the design of microprocessors that target embedded systems.

Direct-mapped (DM) caches are popular in embedded microprocessor architectures due to their simplicity and good hit rates for many applications. A DM cache requires less power per access than a set-associative cache that accesses multiple ways simultaneously. Furthermore, since a DM cache does not have to select the hit way among cache ways as in a set-associative cache, a DM cache may have faster access time. Furthermore, for many applications, a DM cache's hit rate is nearly as good as that of a set-associative cache. For example, in [13], 17 of 23 benchmarks consumed the least energy using a direct-mapped cache rather than a set-associative cache with any number of ways, even when the sizes of either cache could be varied.

However, for other applications, a DM cache has a poor hit rate, resulting in many accesses to the slower and power-costly next level memory (L2 cache or main memory), and hence poor performance and high energy consumption.

A victim buffer can improve the situation for such applications. A victim buffer is a small fully-associative cache, whose size is typically 4 to 16 cache lines, residing between a direct-mapped L1 cache and the next level of memory. The victim buffer holds lines discarded after an L1 cache miss. The victim buffer is checked whenever there is an L1 cache miss, before going to the next level memory. If the desired data is found in the victim buffer, the data in the victim buffer is swapped back to the L1 cache. Jouppi [7] reported that a four-entry victim buffer could reduce 20% to 95% of the conflict misses in a 4 Kbyte direct-mapped data cache. Albera and Bahar [1] evaluated the power and performance advantages of a victim buffer in a high performance superscalar, speculative, out-of-order processor. They showed that adding a victim buffer to an 8 Kbyte direct-mapped data cache results in 10% energy savings and 3.5% performance improvements on average for the Spec95 benchmark suite.

A victim buffer improves the performance and energy of a DM cache *on average*, but for some applications, a victim buffer actually degrades performance without much or any energy savings, as we will show later. Such degradation occurs when the victim buffer hit rate is low. Checking a victim buffer requires an extra cycle after an L1 miss. If the victim buffer hit rate is high, that extra cycle actually prevents dozens of cycles for accessing the next level memory. But if the buffer hit rate is low, that extra cycle doesn't save much and thus is wasteful. Whether a victim buffer's hit rate is high or low is dependent on what application is running. Such performance overhead may

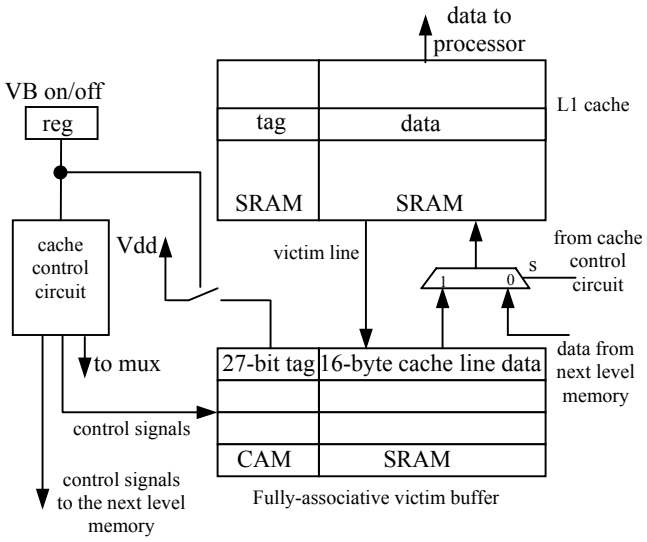


Figure 1: Cache architecture with a configurable victim buffer that can be turned on or off. VB stands for victim buffer. MUX control: s = 0 when VB is off; s = 0 when VB is on and a victim buffer miss; s = 1 when VB is on and a victim buffer hit.

be one reason that victim buffers aren't always included in embedded processor cache architectures.

An embedded system typically runs one application (or a small number of applications) for the system's lifetime. Thus, while an architectural feature that works on average is nice, what really matters to an embedded system designer is whether the feature works well on *the one application* that the designer is implementing. Thus, there is a strong benefit to creating embedded system architectures that are configurable, and can thus be tuned to a particular application.

In this paper, we show that treating the victim buffer as a configurable memory parameter to a DM cache – i.e., the buffer can be included or excluded in a synthesis methodology, or turned on or off in a configurable hardware architecture) – is superior to either using a DM cache without a victim buffer, or using a DM cache with an always-on victim buffer.

Furthermore, we show that a victim buffer parameter is even useful with a cache that itself is highly parameterized. Cache parameters are standard in microprocessors sold as cores [3][4][10][12]. Furthermore, several parameterized cache architectures have recently been introduced such that total size, associativity, and line size can even be configured after chip fabrication [2][9][13]. One might think that having a parameterized cache obviates the need for a victim buffer. Instead, we show that a victim buffer enables further improvement, usually by enabling the parameterized cache to be configured as direct-mapped for applications where otherwise the cache would have been set-associative.

## 2 Victim Buffer as a Cache Parameter

We consider adding a victim buffer to both core-based and pre-fabricated platform based design situations.

### 2.1 Core-Based Design Approach

A core-based approach involves incorporating a processor (core) into a chip before the chip has been fabricated, either using a synthesizable core (soft core) or a layout (hard core). In either case, most core vendors allow a designer to configure the level 1 cache's total size (typical sizes range from no cache to 64 Kbyte), associativity (ranging from direct mapped to 4 or 8 ways), and sometimes line size (ranging from 16 bytes to 64 bytes). Other parameters include use of write through, write back, and write allocate policies for writing to a cache, as well as the size of a write buffer.

Adding a victim buffer to a core-based approach is straightforward, involving simply including or not including a buffer into the design.

### 2.2 Pre-Fabricated Platform Approach

A pre-fabricated platform is a chip that has already been designed, but is intended for use in a variety of possible applications. To perform efficiently for the largest variety of applications, recent platforms come with parameterized architectures that a designer can configure for his/her particular set of applications. Recent architectures include cache parameters [9][13][2] that can be configured by setting a few configuration register bits. We therefore developed a configurable victim buffer that could be turned on or off by setting bits in a configuration register.

### 2.3 Configurable Victim Buffer Architecture

Our configurable victim buffer is shown in Figure 1. The first level cache is backed up with an eight-entry fully-associative victim buffer (a four-entry buffer is drawn). A content addressable memory (CAM) holds the tags of the eight cache lines in the victim buffer. A mux located between the victim buffer and the L1 cache selects the data to the L1 cache either from the victim buffer or the next level memory.

A one-bit software programmable register controls the on/off state of the victim buffer. When the victim buffer is configured as on, it is accessed when there is an L1 cache miss. If the desired content is found in the victim buffer, then the hit cache line in the victim buffer is swapped with the L1 cache to replace the missed cache line. If the victim buffer is configured as off, then

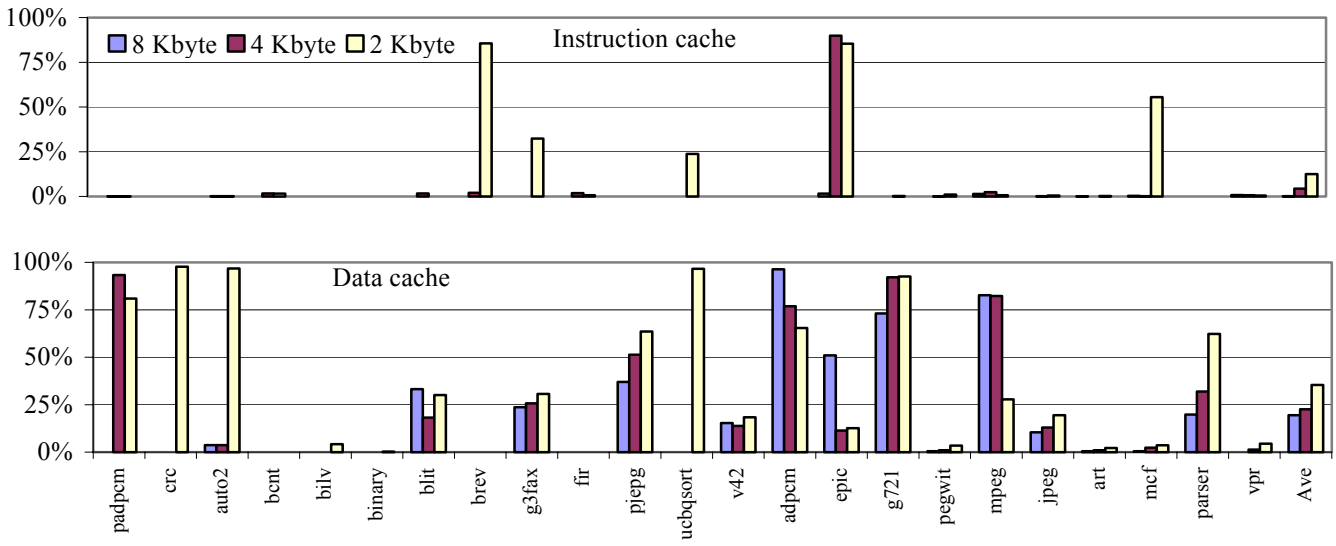


Figure 2: Hit rate of a victim buffer when added to an 8 Kbyte, 4 Kbyte, or 2 Kbyte direct-mapped cache.

the next level memory is accessed when there is a L1 cache misses.

We assume separate L1 caches for instruction and data, and thus add a distinct victim buffer to each L1 cache.

## 2.4 Configurable Buffer Overhead Analysis

Because the victim buffer resides after the L1 cache, the buffer causes no critical path overhead and hence no degradation of clock rate. While CAM tags used in the victim buffer for full-associativity can be energy expensive compared with conventional SRAM based tags, the victim buffer is small, and is only accessed when there is an L1 cache miss, making the energy overhead of the victim buffer very small.

The configuration circuit, a 1-bit register and a switch to turn off the victim buffer, accounts for trivial area overhead.

# 3 Experiments

We simulated, using SimpleScalar [5], a variety of benchmarks for a variety of cache configurations. The benchmarks include programs from Motorola's Powerstone suite [9] (*padpcm, crc, auto2, bcnt, bilv, binary, blit, brev, g3fax, fir, pjpeg, ucbqsort, v42*), MediaBench [8] (*adpcm, epic, jpeg, mpeg2, pegwit, g721*) and some programs from Spec 2000 [6] (*art*, *mcf, parser, vpr)*. We chose programs that simulated without problems and ran in a tolerable amount of time, and show results for *all* programs that we ran. We use the test vectors that came with each benchmark as program stimuli.

$$E_{total} = E_{cache} + E_{memory}$$
$$E_{cache} = E_{dynamic} + E_{static}$$
$$E_{dynamic} = Cache_{total} * E_{hit} + Cache_{Misses} * E_{miss}$$
$$E_{static} = Cycles_{total} * E_{static\_per\_cycle}$$
$$E_{miss} = E_{uP\_stall} + E_{cache\_block\_fill}$$
$$E_{memory} = Cache_{misses} * E_{offchip\_access}$$

Equation 1**:** Equations for calculating memory-access energy.

## 3.1 Energy Evaluation

Power dissipation in CMOS circuits is comprised of two main components, static power dissipation due to leakage current, and dynamic power dissipation due to logic switching current and the charging and discharging of the load capacitance. Dynamic energy consumption contributes to most of the total energy dissipation in micrometer-scale technologies, but static energy dissipation will contribute an increasingly larger portion of total energy consumption in nanometer-scale technologies. Therefore, we consider both types of energies. We also consider energy consumption of both on-chip cache and off-chip memory. The energy consumption of on-chip cache includes both dynamic energy and static energy. We calculate energy using Equation 1. The cache access energies, $E_{hit}$ and $E_{cache\_block\_fill}$, come from our own layout of the cache; the energy dissipation of off-chip memory and static energy are estimated as in [13]. $Cache_{total}$,

$Cache_{Misses}$ , and $Cycle_{total}$ are obtained through SimpleScalar.

## 3.2 Results

### *3.2.1 Victim buffer with a direct mapped cache*

Most benchmarks run well using a direct-mapped cache. Figure 2 shows the hit rate of the victim buffer when the L1 cache is an 8 Kbyte, 4 Kbyte, or 2 Kbyte direct-mapped cache. The higher the victim buffer hit rate, the more performance improvement and energy reductions we can achieve, because the victim buffer would reduce the visits to off chip memory, which is both time consuming and energy expensive.

Figure 3 shows the performance and energy improvements when adding an always-on victim buffer to a direct-mapped cache. Performance is the program execution time. Energy is calculated from Equation 1. 0% represents the performance and energy consumption of an 8 Kbyte direct-mapped cache. From Figure 3, we see that a victim buffer improves both performance and energy for some benchmarks, like *mpeg, epic*, and *adpcm*. In a core-based approach, a victim buffer should be included for these benchmarks. In a pre-fabricated platform approach, a configurable victim buffer should be turned on for these benchmarks. For other benchmarks, energy is not improved but performance is degraded, as for *vpr, fir,* and *padpcm*. A victim buffer should be excluded or turned off for these benchmarks. Some benchmarks, like *jpeg, parser*, and *auto2*, yield some energy savings at the expense of some performance degradation using a victim buffer – a designer might choose whether to include/exclude or turn on/off the buffer in these cases depending on whether energy or performance is more important.

From the above analysis, we can see that adding a victim buffer as a cache design parameter is imperative for embedded system designers to fully take advantage of the victim buffer based on an application's specific requirements.

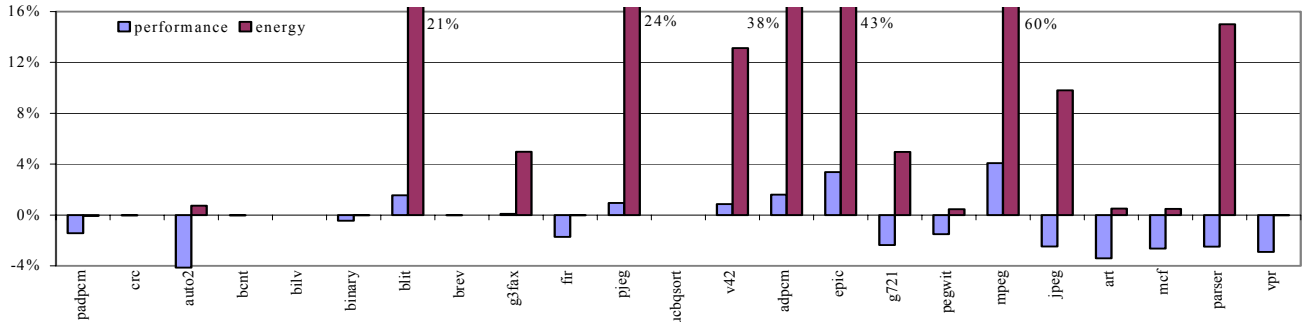


Figure 3: Performance and energy improvements when adding a victim buffer to an 8 Kbyte direct-mapped cache. Positive values mean the victim buffer improved performance or energy, with 0% representing an 8 Kbyte direct-mapped cache without a victim buffer. Benchmarks with both bars positive should turn on the victim buffer, while those with negative performance improvement and little or no energy improvement should turn off the victim buffer.

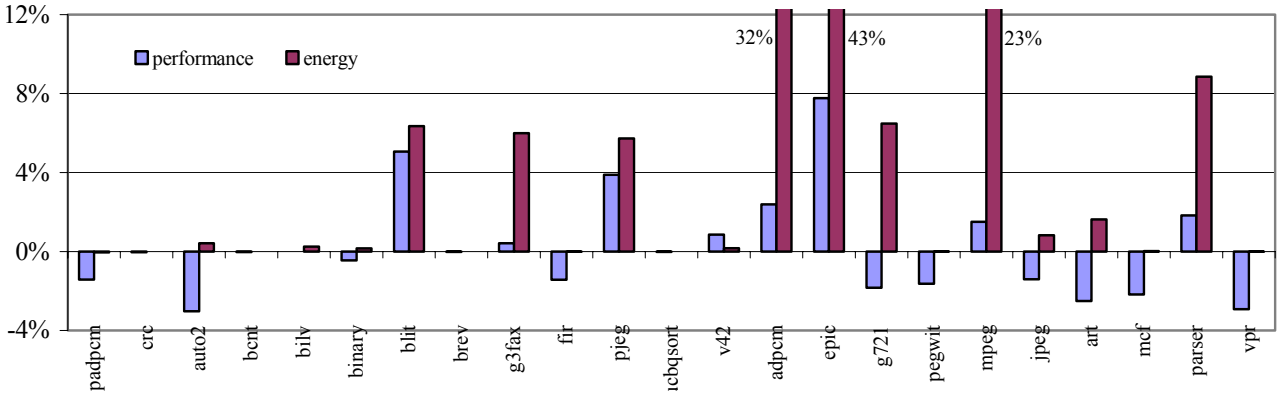


Figure 4: Performance and energy improvements when adding a victim buffer to an 8 Kbyte configurable cache. 0% represents a configurable cache without a victim buffer, tuned optimally to the particular benchmark.

### *3.2.2 Victim buffer with a parameterized cache*

Previous work has shown that tuning cache parameters to a particular application's requirements can dramatically improve energy efficiency [2][8]. Zhang [13] also showed that a pre-fabricated 8 Kbyte 4-way set-associative cache with configurable associativity and size, tuned to a particular application, reduces memory-access-related energy by 40% on average compared to a 4-way set-associative cache, and by 20% compared to a direct-mapped cache (and as high as 185% on particular benchmarks). That cache could be configured as 4-way, 2-way or 1-way (direct mapped), and as 8 Kbyte, 4 Kbyte or 2 Kbyte.

One might think that adding a victim buffer as a cache parameter might not be useful with a parameterized cache, since size and ways can already be tuned to an application. Instead, we show here that a victim buffer is indeed a useful parameter to add to even a parameterized cache.

Figure 4 shows the performance and energy improvement of adding a victim buffer to a parameterized cache having the same configurability described in [13]. 0% represents the performance and energy of the original configurable cache when tuned optimally to a particular application. The bars represent the performance and energy of the configurable cache when optimally tuned to an application assuming a victim buffer exists and is always on. The optimal cache configurations for a given benchmark are usually different for each of the two cases (no victim buffer versus always-on victim buffer).

We see that, even though the configurable cache already represents significant energy savings compared to either a 4-way or direct-mapped cache [13], a victim buffer extends the savings of a configurable cache by a large amount for many examples. For example, a victim buffer yields an additional 32%, 43%, and 23% energy savings for benchmarks *adpcm, epic,* and *mpeg2*. The savings of *adpcm* and *epic* come primarily from the victim buffer that reduces the visits to off-chip memory. The saving of *epic* comes primarily from the victim buffer enabling us to configure the configurable cache to use less associativity without increasing accesses to the next memory level. Yet, for other benchmarks, like *adpcm*, *auto2* and *vpr*, the victim buffer yields performance overhead with no energy savings and thus should be turned off.

It may be noticed that the energy reductions of a victim buffer for a direct-mapped cache are larger than for the configurable cache. This is because the configurable cache consumes less energy than a direct-mapped cache, so the original configurable cache leaves less room for the victim buffer to reduce energy.

Table 1: Optimal cache configuration when cache associativity, cache size and victim buffer are all configurable. *I* and *D* stands for instruction cache and data cache, respectively. *V* stands for the victim buffer is on. *nK* stands for the cache size is *n* Kbyte. The associativity is represented by the last four characters, such as benchmark *vpr, I2D1* stands for two-way instruction cache and direct-mapped data cache.

| Example | Best | Example | Best |
|---|---|---|---|
| padpcm | I8KD4KI1D2 | ucbqsort | I4KD**V**4KI1D1 |
| crc | I2KD**V**4KI1D1 | v42 | I8KD8KI1D1 |
| auto2 | I4KD2KI1D1 | adpcm | I2KD**V**2KI1D1 |
| bcnt | I2KD2KI1D1 | epic | I**V**4KD**V**8KI1D1 |
| bilv | I4KD2KI1D1 | jpeg | I8KD2KI4D1 |
| binary | I4KD2KI1D1 | mpeg2 | I4KD**V**4KI1D1 |
| blit | I2KD**V**2KI1D1 | g721 | I8KD**V**2KI2D1 |
| brev | I4KD2KI1D1 | art | I4KD**V**2KI1D1 |
| g3fax | I4KD**V**2KI1D1 | mcf | I4KD4KI1D1 |
| fir | I4KD2KI1D1 | parser | I8KD**V**4KI4D1 |
| pjepg | I4KD**V**2KI1D1 | vpr | I8KD2KI2D1 |
| pegwit | I4KD4KI1D1 | | |

Table 1 shows the best cache configurations for all benchmarks we simulated when cache associativity, cache size, and the victim buffer are configurable. There are 11 out of 23 benchmarks for data cache, and one benchmark for instruction cache, that consume less energy when the configurable victim buffer is turned on.

## 3.3 Influence of victim buffer size

We also did experiments with a 16-line victim buffer, which is 512 bytes (the data reported in the previous sections used an 8-line buffer). The performance and energy data when a 16-line victim buffer is incorporated to an 8 Kbyte configurable cache are shown in Figure 5. Comparing that data with Figure 4, we see only very small improvements on a few examples. Thus, an 8-line buffer seems sufficient for these benchmarks.

# 4 Conclusion

While the average performance and energy improvements obtainable using a victim buffer with a direct-mapped cache are well known, we have shown that making that victim buffer an additional cache parameter is important in embedded systems, so that the buffer can be excluded or turned off if the particular application being executed yields a low buffer hit rate and hence performance overhead. Furthermore, we have shown that adding a victim buffer as a cache parameter expands the usefulness of a cache with

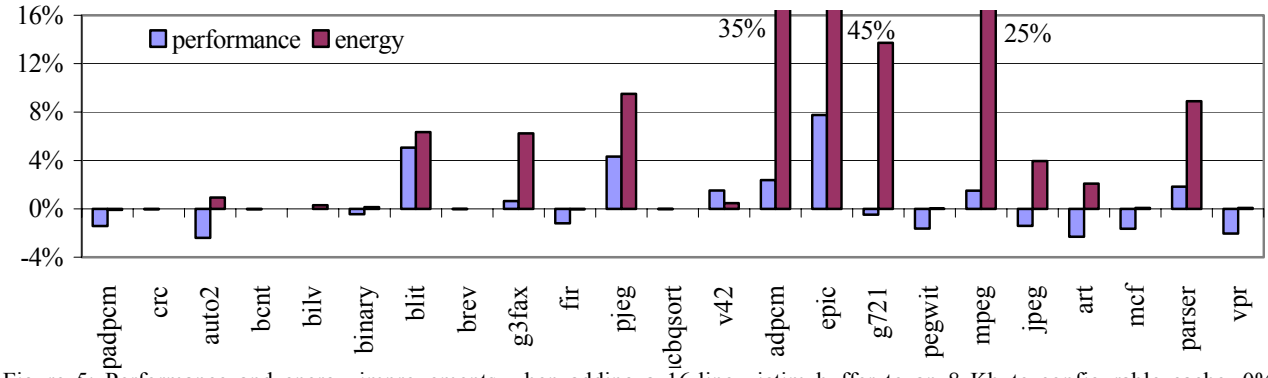


Figure 5: Performance and energy improvements when adding a 16-line victim buffer to an 8 Kbyte configurable cache. 0% represents a configurable cache without a victim buffer, tuned optimally to the particular benchmark

configurable ways and size, by further increasing the energy savings obtained from tuning the cache to a particular application (this conclusion is non-obvious – it could have happened that a cache's standard parameters would have subsumed the victim buffer's benefits). We conclude that adding a victim buffer as a cache parameter should be done in embedded system architectures utilizing either a direct-mapped or configurable cache.

The configurable victim buffer we introduced could be used in a static approach in which the buffer is configured once during microprocessor reset, or even in a dynamic approach in which the buffer is turned on and off dynamically depending on the presently-running application. Future work includes dynamically detecting when to turn a configurable victim buffer on or off, depending on the present application or application phase, and exploring the impact of different victim buffer sizes as another possible parameter.

## Acknowledgments

This work was supported by the National Science Foundation (CCR-0203829, CCR-9876006) and by the Semiconductor Research Corporation (CSR 2002-RJ-1046G).

## References

[1] G. Albera and R. Bahar, "Power/performance Advantages of Victim Buffer in High-Performance Processors," IEEE Alessandro Volta Memorial Workshop on Low-Power Design,1999.

[2] D.H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation," Journal of Instruction Level Parallelism, May 2000.

[3] ARC International, http://www.arccores.com.

[4] ARM Ltd., http://www.arm.com.

[5] D. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0. Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report #1342, 1997.

[6] http://www.specbench.org/osg/cpu2000

[7] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in the Proceedings of International Symposium on Computer Architecture, 1990.

[8] C. Lee, M. Potkonjak and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," in the Proceedings of International Symposium on Microarchitecture, 1997.

[9] A. Malik, B. Moyer, and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," in the Proceedings of International Symposiums on Low Power Electronics and Design, 2000.

[10] MIPS Technologies, Inc, http://www.mips.com.

[11] S. Segars. Low power design techniques for microprocessors. In IEEE Int. Solid-State Circuits Conference Tutorial, 2001.

[12] Tensilica Inc, http://www.tensilica.com.

[13] C. Zhang, F. Vahid, and W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems," in the Proceedings of International Symposium on Computer Architecture, 2003.