Quantifying the Cache Filtering Effect for Multimedia Applications in Embedded Systems

David Fritz, Wira Mulia, Sohum Sohoni

Abstract - **This paper is part of an ongoing comprehensive study on the interaction between various cache memories employed in a typical memory hierarchy in embedded systems. The goal of the study is to quantify the effects of various cache organizations on the other caches within the hierarchy, and to develop an optimal architecture in which each cache is customized to perform a specific task. This paper looks at the basic effect of a level-one cache on the access patterns for the level-two cache. We limit the study to multimedia applications in order to obtain useful, meaningful, and understandable results at a per-application scale.**

Our results show that the L1 has a significant *filtering effect* on the L2 access pattern, making it highly streamlined. Based on the results, we propose small L2 caches with simple prefetch buffers for media-processing architectures.

Index Terms— **Cache memories, memory hierarchy, multimedia, prefetching, embedded systems.**

1. Introduction

The dramatic improvement in CPU performance in the last two decades has led to a further increase in the processor-memory performance gap. Several software and hardware enhancements have been proposed and implemented to bridge the gap, or at least to reduce its impact on overall performance. Despite these efforts, the gap has been increasing, and new solutions are being proposed. A simple and effective solution is to increase cache size. This however, increases total on-chip power dissipation and area, which is undesirable, particularly in embedded solutions. For example, the StrongARM SA110 uses 43% of its total power budget on the cache hierarchy [1]. We assert that the relatively specialized nature (as compared to general purpose CPUs) and unique thermal, power, and area needs of embedded processors makes them ideal candidates for novel, highly specialized cache structures. We believe we can achieve break-even performance or better with specialized cache structures that consume less area, and therefore less energy [2].

D. Fritz is with Oklahoma State University, 202 Engineering South, OSU, Stillwater, OK 74074 USA. (email: david.fritz@okstate.edu)
W. Mulia is with Oklahoma State University, 202 Engineering South, OSU, Stillwater, OK 74074 USA. (email: wira.mulia@okstate.edu)
S. Sohoni is with Oklahoma State University, 202 Engineering South, OSU, Stillwater, OK 74074 USA. (phone: 405-744-8040; email: sohum.sohoni@okstate.edu)

This paper takes a step towards understanding the nuances of memory hierarchy design in embedded systems for a specific subset of applications— multimedia codecs, by examining the effect the L1 cache has on the L2. *Our hypothesis is that the L1 cache creates significant streaming activity in the L2 cache by filtering temporal locality in the L1.* By quantifying this effect, the paper provides data that will aid system designers to consider alternative designs for the L2.

2. Background and Related Work

Most research on CPU caches has focused on improvements to the L1 cache. A number of performance evaluation projects have analyzed cache behavior, but no one has quantified the effect of the L1 cache on the access patterns of the L2. Several researchers have studied CPU cache performance for technical [3], [4], commercial [5], and multimedia workloads [6],[7].

Cantin et al. [4] present cache miss rates for SPEC2000 applications for a wide variety of cache configurations. They present per-application results as well as a number of averages. Although a comprehensive study that provides a large set of results, it is limited to the L1 cache.

Talla et al. [6] studied the cache performance of several multimedia applications and found that their cache miss rates were not higher than those of regular applications. They performed studies on the execution characteristics of multimedia applications [8] and suggested performance optimizations for these applications. Our previous work [7], found that multimedia cache performance is not worse than that of technical applications, but the study does not showcase the filtering effect of the L1.

With L1 cache miss rates less than 1% [4] for many applications, Amdahl's law suggests that we should look elsewhere for performance improvement. Some previous work on L2 caches shows that the L2 could very well be a performance bottleneck. Bhandarkar et al. [9] study the performance of the Pentium Pro processor for a number of applications. They stress that overall performance loss is largely due to L2 cache misses. Wong et al. [10] in their work on modifying the L2 replacement policy based on temporality, present a strong case for improving the hit rate of L2 caches. Lin et al. [11] found that half of their system's execution time was spent servicing L2 cache misses for SPEC2000 applications. Thus, there is strong motivation for analyzing and improving the performance of L2 caches.

Jouppi [12] proposes using a victim cache in which a simple fully associative cache is placed after the L1 cache. The victim cache swaps data with the L1 cache on references that hit in the victim cache but miss in the L1. Jouppi also suggests the inclusion of stream buffers that prefetch cache lines starting at a cache miss address. The prefetched lines are inserted into the buffers, instead of the regular cache, to avoid capacity and compulsory misses.

Gonzáles et al. [13] proposes a split temporal and spatial cache to store data with different locality of reference characteristics. A locality prediction mechanism determines into which cache data is fetched. Although this mechanism reduces negative interaction between temporal and spatial localities, it may introduce memory fragmentation if a program displays only one type of locality. This may waste unnecessary area and power in an embedded system, especially if both cache segments have to be scaled up in size to achieve break-even performance.

Embedded cache memories have received special attention, with increasing efforts to reduce overall cache area, and therefore power. The Semiconductor Industry Association (SIA) predicts by 2011, a typical SoC design will dedicate over 90% of the area to memory [14].

Panda et al. [15] suggest a scratch pad memory, which is a small on chip memory in a separate address space from the rest of the memory. The scratch pad memory bypasses any data cache, enabling the programmer to use the memory for critical data structures.

3. EXPERIMENTAL METHODOLOGY

In this section we provide details for the simulators used, the applications tested, and the experiments conducted.

3.1. Simulators and Benchmark Programs

We use the *DineroIV* [16] trace-driven cache simulator and Virtutech's *Simics* [17] full-system execution-driven simulator. We obtain traces for Dinero through instrumentation using *Pin* [18], and through Simics to simulate a real working environment with multiple processes running during the execution of a benchmark program.

Pin is a tool used for profiling applications by inserting instrumentation code into program binaries dynamically at runtime. Pin generates a trace file consisting of every read, write, and instruction fetch made by the application. All addresses are virtual. Pin is limited in that it cannot generate memory access traces of the entire system. Many embedded systems do not run with any supervisor code or operating system, so single application traces taken from Pin are still useful. To represent systems that do use operating systems, we use another simulator, described below.

Dinero is a trace driven cache simulator that provides a variety of cache statistics. It was originally developed at the University of Wisconsin as part of the Wisconsin Architectural Tool Set. Most cache parameters such as size, associativity, block size as well as the replacement policy can be varied to simulate different types of caches.

Virtutech's Simics is a full-system execution-driven simulator that can simulate various families of processors. We use Simics/x86 with the *enterprise* target provided with the simulator. Enterprise is an x86 architecture running Red Hat Linux 2.4. Of the various architectures that can be simulated for an enterprise machine we simulate a single-cpu Pentium 4 machine. Traces from Simics contain references not just from the benchmark application but all the other processes that might execute during the run-time of the application. This provides a significantly larger trace file compared to the Pin trace, but this trace more accurately portrays actual operating conditions. We believe that the effect of switching to other processes, and the resulting replacement of cache blocks should be considered while analyzing a memory hierarchy.

We study a set of multimedia applications represented by the *MediaBench* [19] benchmark suite. MediaBench consists of complete applications coded in high level languages. It includes core algorithms for most widely used multimedia applications, all of which are publicly available and commonly used on general purpose processors.

3.2. Cache Parameters

We simulate a 512 KB, 1 MB and a 2 MB L2 cache, each 8-way set-associative with the block size varied from 64 bytes to 256 bytes. The traces for the L2 are filtered through 32 KB, 64 KB and 128 KB, 8-way set-associative I and D caches with a 32 byte block size. With 9 different L2 configurations and 3 L1 configurations, most of our results are for a relevant subset of the 27 possible combinations.

3.3. Stream Detection

To measure the filtering effect of the L1, we measure L2 miss rates, as well as the percentage of total cache blocks accessed that are part of a stream. To measure the second metric, we designed a stream detector. The detector is situated between the L1 and L2 caches and monitors all L2 accesses. It detects both unit and strided streams. A unit stream is defined as having an access pattern L, L+1, L+2, L+3… whereas a strided stream contains some distance N between accesses such as L, L+N, L+2N, L+3N… Negative strides are also detected in the stream detector.

The detector itself is implemented using a simple FIFO miss history table. For every L2 access that is not already classified as part of a stream, an entry is placed in the miss history table. For successive accesses, memory references are checked against other elements in the miss history table to determine if they form a stream (unit or strided, positive or negative stride). If accesses do form a stream, and a minimum number of accesses, K, have occurred, the references are classified as a stream and removed from the miss history table. This stream is then stored in another table to be used for prefetching. Jouppi [12] provides a detailed discussion of unit and strided streams.

Two metrics determine the performance of this stream detector, the depth of the miss history table, and K, the minimum number of accesses classified as a stream. Ganusov et al. [20] showed that a miss history table depth of 16 is optimum for tracking streams. Sohoni [21] shows that a K value of 4 to 16 is sufficient for stream detection.

3.4. Block Access Frequencies

We also measure the streaming nature in the L2 cache as a result of the L1 by measuring the block access frequency in the L2. If a unit stream is present in the L2 cache, we expect to L2 block accesses equal to the ratio of L2 and L1 block size. We measure this by modifying Dinero to maintain a block access count for each unique block in the L2. When a block is evicted, the count is recorded and reset. We measure block access frequencies between 1 and 8 accesses.

4. RESULTS

Figures 1 and 2 show the percentage of streaming references in the L2 accesses. These are filtered through three different L1 caches, a 32 KB I and D cache, a 64 KB and a 128 KB. All the caches were 8-way set-associative with block sizes of 32 bytes.

The first observation from Figure 1 is the high percentage of streams shown by almost all applications, with the exception of *pegwit*. The average stream percentages are 31.96%, 35.03% and 44.47% when filtered by the 32 KB, 64 KB and 128 KB cache respectively.

The second observation from the figure is the increase in the percent streaming with increase in the size of the filtering cache. As the L1 captures more and more of the temporal accesses, the remaining accesses (misses from the L1 to the L2) tend to be more streaming in nature. Thus, with larger and more effective L1 caches, we can expect fewer accesses to the L2, and these will predominantly be of a streaming nature.

From the plot for the Simics trace (Figure 2), we see even higher percentages of streaming references. The average stream percentages are 46.72%, 50.94% and 55.42% when filtered by the 32 KB, 64 KB and 128 KB cache respectively. These results are particularly important because this is what happens on a real system, under actual

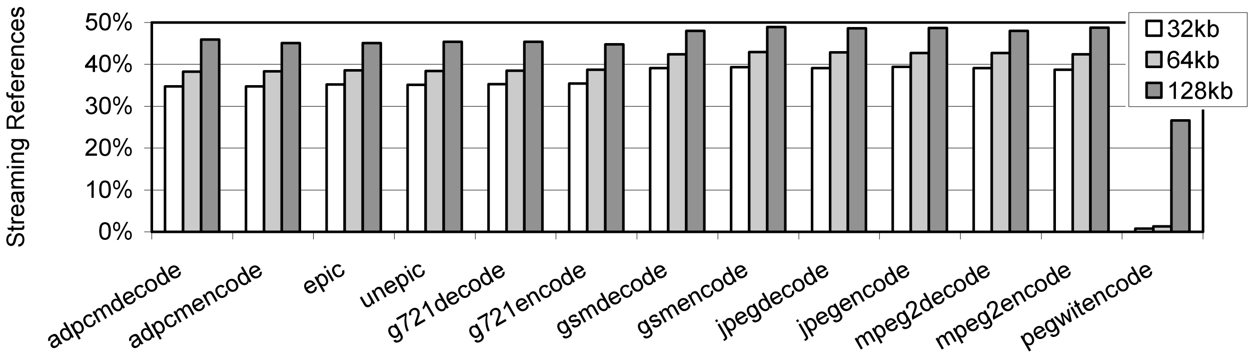

Figure 1. Streaming References to the L2: Pin Traces. L1 Cache Size Varied from 32 KB to 128 KB.

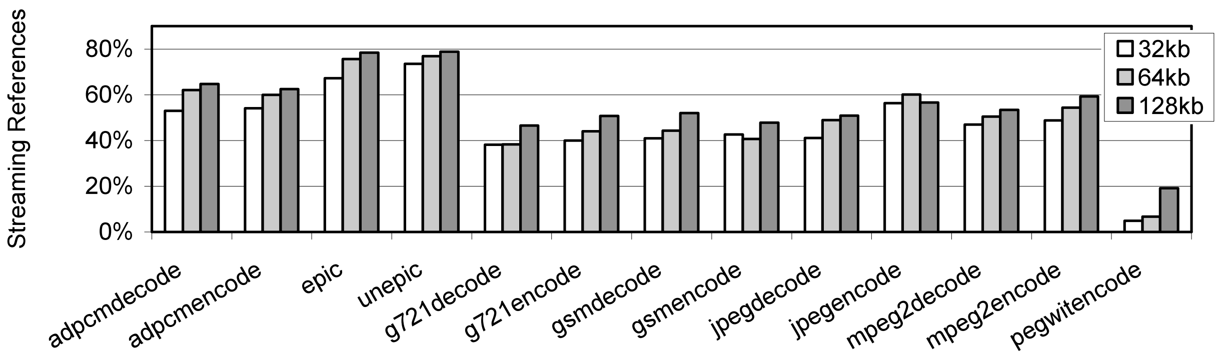

Figure 2. Streaming References to the L2: Simics Traces. L1 Cache Size Varied from 32 KB to 128 KB.

operating conditions. Multimedia applications tend to have streaming patterns, and we expected that the addition of references from non-streaming processes and idle processes would lower the overall streaming percentage for the Simics traces compared to the Pin traces. However, because other processes may have evicted data from the L1, some data will be brought back in, creating additional streaming accesses at the L2. *These results further support our previous observation that the L2 sees a high number of streaming references due to the filtering effect of the L1.*

Figures 3 and 4 show the percentage miss rates for local L2 misses for 9 different configurations. It is clear that the block size has a significant impact on cache performance for the mediabench applications. Due to their small memory footprint, the miss rates are almost identical in the Pin trace. However, for the Simics trace, we see the effect of other processes and a noticeable increase in the miss rate with decrease in cache size.

Based on our prior observations on the streaming nature of L2 accesses, we expect to see a decrease in miss rates with an increase in block size. This is because a larger block size effectively prefetches streaming data for unit stride and small stride streams. This leads us to believe that a smaller L2 cache with a large block size or a simple stream-based prefetch mechanism will perform well for media-processing architectures. For example, for the Simics trace that includes references to other processes the miss rate for the 512 KB cache with a 256 byte block size is 8.15% compared to 13.83% for the 2 MB cache with a 64 byte block size.

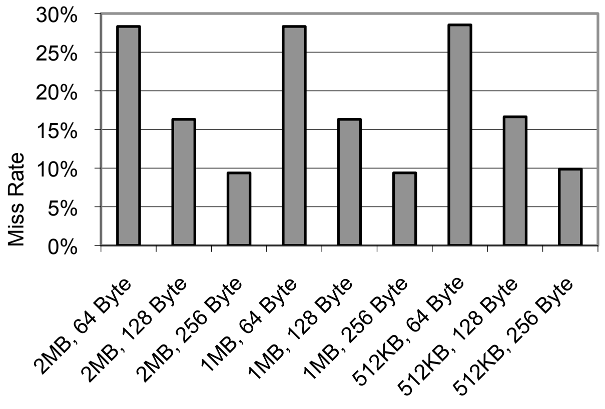

Figure 3. Miss Rates for the L2: Pin Traces

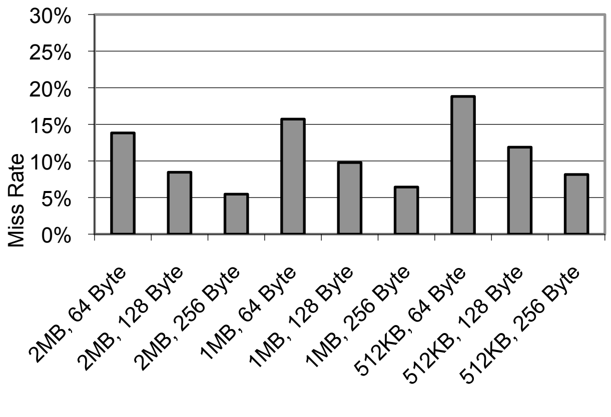

Figure 4. Miss Rates for the L2: Simics Traces

The most convincing evidence of the streaming nature in the L2 cache as a result of the L1 is the access frequency of blocks in the L2 cache. For example, suppose we have a L1 cache with a 32 byte block size and a L2 cache with a 64 byte block size. If the L1 creates a stream in the L2, either by filtering out temporal locality, or by the inherent streaming nature of the process, we expect to see only 2 accesses to a given block in the L2 cache, since each L2 block consists of 2 L1 blocks. This assumes a unit stride in the stream. Figures 5 and 6 show the block access frequencies from 1 to 8 in the L2 cache for *adpcmencode*, a voice encoding application, for Pin and Simics traces, respectively. The L1 cache has a block size of 32 bytes, and the L2 cache has a block size varied from 64 to 256 bytes. In each block size configuration, the highest block access frequency occurs at the ratio of L2 to L1 block size. This indicates that a significant amount of unit streaming is occurring in the L2 cache as a result of filtering in the L1 cache. For example, over 40% of all L2 accesses with a 64 byte block size in the Simics traces access a block only twice. This observation is similar in each of the applications used in this experiment.

This observation is important in determining the importance of the L2 cache. If most accesses belong to streams, and are only accessed by those streams, which is indicated here

5. CONCLUSIONS

This paper provides quantification for the filtering effect of L1 caches on the L2 access patterns. Through an actual count of the number of references detected as part of a stream and the variation of these counts with the size of the L1 cache through which they are filtered, we show that the L1 has a significant effect on L2 access patterns. This is illustrated through the direct measurement of unit and strided streams in the L2 cache and through block access frequency statistics that indicate the degree of streaming activity in the L2 cache caused by the L1 cache. We show that in some cases, nearly 80% of all L2 cache accesses belong to an easily identifiable stream. Furthermore, references in the L2 generally access a block equal to the ratio of L2 and L1 block size. This is indicative of the L1 cache creating streaming activity in the L2 by filtering out temporal activity.

We believe that alternate models for the memory hierarchy should be considered based on the quantification of the filtering effect. One option is to reduce the L2 cache size and increase its block size. Another option would be to assign filtered prefetch buffers and completely removing the L2 cache as suggested by Palacharla and Kessler [22]. While removing the L2 cache is not practical for general

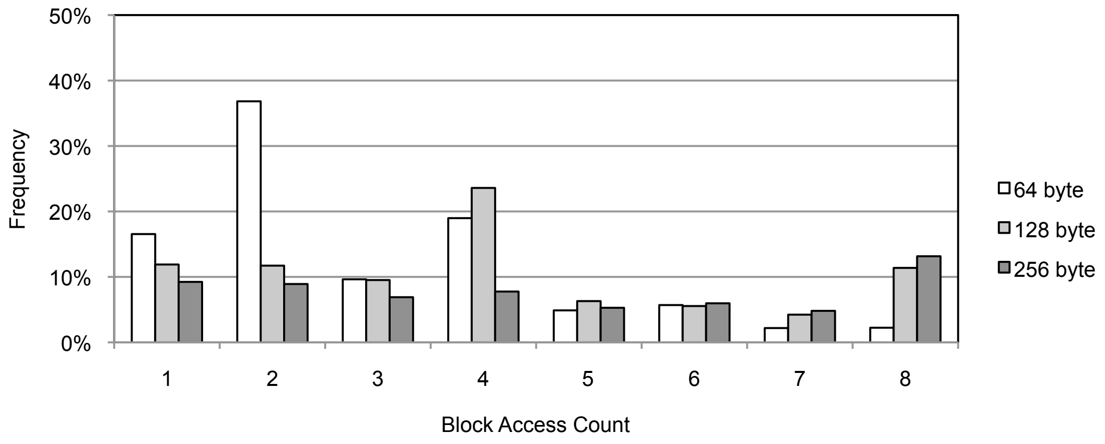

Figure 5. Block access frequencies in the L2: Pin Traces. The greatest block access frequencies occur when the block access count equals the ratio of L2 block size and L1 block size.

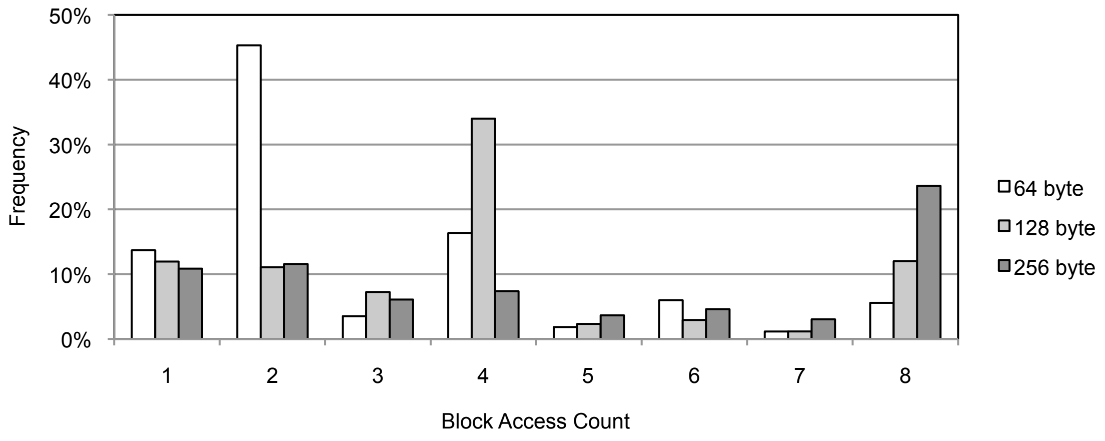

Figure 6. Block access frequencies in the L2: Simics Traces.

purpose systems, it may indeed be a good solution for an embedded systems domain.

With novel prefetching schemes for streams [23],[24], media-processing embedded cores will have excellent memory performance even with much smaller L2 caches than those of current commercial general-purpose cores. Smaller L2 caches will have lower latency, further improving overall memory system performance.

6. Future Work

This paper is the first step in a wider study of novel cache architectures in embedded systems. We plan to continue the experiment detailed here on a larger number of application domains for embedded systems, including office productivity, networking, security, and telecommunication applications. Benchmark suites already exist to facilitate in this study [25]. We expect to see less overall streaming activity in other application domains, which may justify a hybrid cache architecture to accommodate the widest array of application behavior.

We also plan to explore additional cache architectures, including those designed to exploit the interactions between an embedded operating system and user applications. As embedded devices become more complex, developers are turning to more general purpose solutions such as embedded Linux devices. This trend creates new avenues of embedded memory research. One possible solution is to include a supervisor cache lateral to the L2 cache. When the processor is in supervisor mode, all cache writes occur in the supervisor cache, to avoid interference with the user data stream. This mechanism could help prevent any interference in critical user process locality, while still providing the benefits of caching to the operating system.

7. References

[1] S. Santhanam, “StrongARM SA110 – A 160MHz 32b 0.5W CMOS ARM Architecture,” *Hot Chips 8: A Symposium on High-Performance Chips,* Aug. 1996.

[2] T. Ishihara and H. Yasuura, “A power reduction technique with object code merging for application specific embedded processors,” *Proc. Of Design, Automation and Test in Europe Conference 2000*, pp. 617-623, Mar. 2000.

[3] S. Sair and M. Charney, “Memory Behavior of the spec2000 Benchmark Suite,” technical report, IBM T.J. Watson Research Center, October 2000.

[4] J. Cantin and M. Hill, “Cache Performance for Selected spec CPU2000 Benchmarks,” *SIGARCH Computer Architecture News*, ACM, pp. 13-18, September 2001.

[5] L. Barroso, K. Gharachorloo, and E. Bugnion, “Memory System Characterization of Commercial Workloads,” *Proceedings of the 25th International Conference on Very Large Databases*, IEEE Computer Society, (Barcelona, Spain), pp. 3-14, 1998

[6] D. Talla and L. K. John, “Execution Characteristics of Multimedia Applications on a Pentium II Processor,” *Proceedings of the 1999 IEEE International Performance, Computing, and Communications Conference*, (Phoenix, Arizona, USA), pp. 516-524, February 2000.

[7] Z. Xu, S. Sohoni, R. Min, and Y. Hu, “An Analysis of the Cache Performance of Multimedia Applications,” *IEEE Transactions on Computers*, IEEE Computer Society, vol. 53, no. 1, pp. 20-38, January 2004.

[8] D. Talla, L. John, and D. Burger, “Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Enhancements," *IEEE Transactions on Computers*, IEEE Computer Society, vol. 52, no. 8, pp. 1015-1031, August 2003.

[9] D. Bhandarkar and J. Ding, “Performance Characterization of the Pentium Pro Processor,” *Proceedings of the 3rd International Symposium on High Performance Computer Architecture*, IEEE Computer Society, (San Antonio, Texas), pp. 288-297, February 1997.

[10] W.Wong and J.-L. Baer, “Modified LRU Policies for Improving Second-Level Cache Behavior,” *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, (Toulouse, France), pp. 49-60, January 2000.

[11] W. Lin, S. Reinhardt, and D. Burger, “Designing a Modern Memory Hierarchy with Hardware Prefetching,” *IEEE Transactions on Computers Special Issue on Memory Systems*, IEEE Computer Society, vol. 50, no. 11, pp. 1202-1218, November 2001.

[12] N. P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,” *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ACM, (Seattle, Washington, USA), pp 364-373, 1990.

[13] A. González, C. Aliagas, M. Valero, “A data-cache with multiple caching strategies tuned to different types of locality,” *International Conference on Supercomputing*, July, 1995.

[14] Semiconductor Industry Association, “International Technology Roadmap for Semiconductors: Design,” 2000 Update.

[15] P. Panda, N. Dutt, A. Nicolau, “On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems,” *ACM Trans. on Design Automation of Electronic Systems,* July, 2000.

[16] J. Edler and M. Hill, “DineroIV Trace-Driven Uniprocessor Cache Simulator.”

[17] Virtutech. https://www.simics.net/.

[18] C.-K Luk, R.Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation,” *Programming Language Design and Implementation*, ACM, (Chicago, Illinois, USA), pp. 190-200, June 2005.

[19] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems,” *Proceedings of the 30th Annual International Symposium on Microarchitecture*, IEEE Computer Society, (Research Triangle Park, North Carolina, USA), pp. 330-335, December 1-3, 2007.

[20] I. Ganusov and M. Burtscher, “On the Importance of Optimizing and Configuration of Stream Prefetchers,” *3rd Annual ACM SIGPLAN Workshop on Memory Systems Performance*, ACM, (Chicago, Illinois, USA), pp. 54-61, June 2005.

[21] S. Sohoni, “Improving L2 Cache Performance through Stream-Directed Optimizations,” Tech. Report, University of Cincinnati, June 2003.

[22] S. Palacharla and R. E. Kessler, “Evaluating Stream Buffers as a Secondary Cache Replacement,” *Proceedings of the 21st Annual International Symposium on Computer Architecture*, IEEE Computer Society Press, (Chicago, Illinois, USA), pp. 24-33, 1994.

[23] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. G. Abraham, “Effective Stream-Based and Execution-Based Data Prefetching,” *Proceedings of the 18th Annual International Conference on Supercomputing*, ACM, (Malo, France), pp. 1-11, 2004.

[24] C. Zhang and S. A. McKee, “Hardware-Only Stream Prefetching and Dynamic Access Ordering,” *Proceedings of the 14th International Conference on Supercomputing*, ACM, (Santa Fe, New Mexico, USA), pp. 167-175, ACM, 2000.

[25] M.R. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” *Proc. 4th IEEE Workshop on Workload Characterization,* pp. 3-14, Dec. 2001.