Improving Data Cache Performance using Persistence Selective Caching

Sumeet S. Kumar, Rene van Leuken
Circuits and Systems Group, Faculty of EEMCS,
Delft University of Technology, The Netherlands
{s.s.kumar, t.g.r.m.vanleuken}@tudelft.nl

***Abstract*—This paper presents Persistence Selective Caching (PSC), a selective caching scheme that tracks the reusability of L1 data cache (L1D) lines at runtime, and moves lines with sufficient potential for reuse to a low-latency, low-energy assist cache from where subsequent references to them are serviced. The selectivity of PSC is configurable, and can be adjusted to suit the varying memory access characteristics of different applications, unlike existing schemes. By effectively identifying reusable cache lines and storing them in the assist, PSC reduces average memory access time by upto 59% as compared to competing schemes and conventional data caches. Furthermore, by ensuring that only reusable lines are cached by the assist, PSC reduces cache line movements, and thus decreases average energy per access by upto 75% over other assists.**

Keywords—cache memory, memory architecture, memory management, microprocessors

I. Introduction

The limited size of processor caches in comparison to the data set of modern applications leads to the emergence of expensive misses, necessitating high latency and energy consuming accesses to lower levels of the memory hierarchy. Although large set-associative caches appreciably reduce miss rates, their size causes them to have a higher hit-latency, and consume more energy per access than smaller direct-mapped caches.

This paper presents the *Persistence Selective Caching (PSC)* scheme which reduces *average memory access time (AMAT)* through the selective caching of reusable lines in a small, fully-associative assist cache. The reuse potential of a line is estimated at runtime based on its *access persistence*, i.e. the number of accesses to the line within a certain window of data references by the processor. Lines with sufficient access persistence are moved from the *L1 data cache (L1D)* into the assist cache from where subsequent references to them are serviced. Due to the assist's small size, these references only incur a short access latency, and consume considerably lesser energy than an L1D access. PSC's selectivity ensures that only the most reusable lines are moved to the assist, leading to a significant reduction in the number of cache line movements (*swaps*), and thus lower energy per access than competing schemes. The significant contributions of this paper are:

- A configurable scheme that tracks the access persistence of cache lines at runtime, and selectively caches those with sufficient persistence in a low-latency, low-energy assist cache. The selectivity of PSC is configurable, and allows the scheme to be adjusted to suit the varying memory access characteristics of different applications, unlike existing schemes.
- Illustration of the performance and energy benefits of selective assist caching. PSC reduced AMAT by upto 59% and average energy per access by upto 75% as compared to conventional data caches and competing assists [1][2].

This paper is organized as follows: In Section II, we review the state of the art in cache assists, and outline the motivation for PSC. In Section III, we describe the architecture and algorithms of PSC, and in Section IV, evaluate its effectiveness in reducing AMAT and energy per access.

II. Related Work

A number of studies have, in the past, used small memory buffers to augment the capacity of the main L1D, and thus improve performance and energy consumption. In this paper, such memory structures are referred to as assist caches. The *Filter cache* [3] for instance is an assist that reduces energy consumption for cache memory accesses by using a very small memory buffer in between the processor and L1D. However, these energy savings are obtained at the cost of increased access latencies, and thus higher *average memory access time (AMAT)*. The *Victim Cache (VC)* [4] on the other hand aims to decrease AMAT by reducing the cost of conflict misses. The VC stores victims of L1D evictions such that in the event of future references to them, the lines can be returned to the L1D in a single cycle rather than through a long latency, energy consuming cache miss. On a VC hit, the requested line is moved to the L1D, and the corresponding entry from the L1D evicted to the VC. This swap operation constitutes an energy overhead, and is a significant disadvantage of the victim cache. Stiliadis et al. overcame this disadvantage with their proposal, *Selective Victim Caching (SVC)* [1]. In SVC, the swap operation is prevented from occurring if the incumbent L1D cache line is found to be more reusable than the requested VC line. SVC considerably reduces the number of swaps as compared to a conventional victim cache with the same miss rate and latency improvements.

However, these proposals consider the L1D as the primary target for data references by the processor, and the assist as an auxiliary cache. A majority of references are thus serviced by the larger L1D cache, and consequently, the relatively shorter latency and energy per access of the assist cache remain

This research was supported in part by the CATRENE programme under the Computing Fabric for High Performance Applications (COBRA) project CA104.

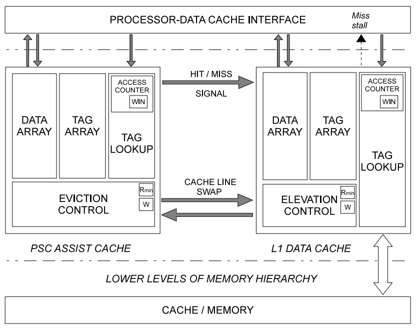

Fig. 1. Architecture of the PSC assist cache and its interface for cache line swaps with the L1D

unexploited. Duong's *L0* scheme (*I1P101* in particular) [2] and the *HitME* cache [5] are examples of cache organizations that service a majority of data references through an assist placed alongside the L1D. In these proposals, any cache line that is referenced atleast once in the L1D is immediately moved to the assist. Subsequent accesses to these lines, for as long as they remain in the assist, incur a relatively shorter access latency and energy. However, L1D cache lines are moved to the assist without determining how much potential they have for reuse. If the assist-cached lines are not sufficiently reused so as to amortize the energy consumed in moving them from the L1D, any latency benefits obtained will be at the cost of increased energy consumption. In addition, moving lines following just a single access results in the small cache quickly becoming inundated, further resulting in a large number of evictions back to the L1D. These cache line movements constitute an energy overhead, and thus it is imperative that the assist caching strategy implement some selectivity moving lines to the assist cache.

III. Persistence Selective Caching

PSC is a selective caching scheme that aims to reduce AMAT and energy per access by servicing a majority of data references through a small, fully-associative assist cache, while minimizing the number of cache lines moved to this cache from the L1D. It achieves this by limiting use of the assist cache to only those cache lines that are most frequently accessed. PSC is based on the premise that frequently accessed cache lines have potential for reuse, and are likely to be referenced again. The reuse potential of a line is determined based on its *access persistence*, i.e. the number of accesses to the cache line within a small window of references. Lines that exhibit persistence equal to or greater than a certain threshold level are regarded as reusable and subsequently *elevated* to the assist cache. Due to its small size, references to data stored in the assist incur a relatively shorter access latency, and consume a smaller amount of energy than the L1D.

The PSC assist cache is illustrated in Figure 1. Both the assist and L1D are probed in parallel on references by the processor. Since the two are mutually exclusive in their content, cache lines may reside in either cache but not both simultaneously. Consequently, a reference to a line may hit in either, or miss in both, with the latter case serviced through the L1D alone. To enable the swapping of cache lines between the two caches, the architecture includes an interface for the exchange of data, tags and hit/miss information as illustrated in Figure 1. PSC does not influence the caching of lines by the L1D from lower level caches and is thus non-filtering, unlike the Filter cache [3] and SVC [1]. This ensures that the full capacity of the L1D is always available irrespective of the amount of data reuse present in the application.

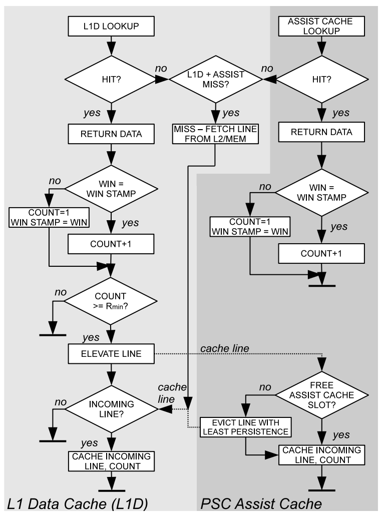

Fig. 2. Algorithms for the L1D and PSC assist cache

A. Selective Caching Criteria

The threshold level of access persistence that lines require in order to be elevated to the assist cache is defined in terms of two parameters:

- R_{min}**:** The number of references required to a line for it to be regarded as frequently used and hence, reusable.
- W**:** The number of references within which the R_{min} condition must be met. Each sequence of W references by the processor is considered as a window.

These two parameters essentially define PSC's selectivity in elevating L1D cache lines to the assist cache. Since access persistence is strongly dependent on memory access patterns, these parameters can be expected to have different optimal values for different applications.

The working of a data cache with a PSC assist is illustrated in Figure 2. Elevation decisions are made following L1D hits, after the requested data is returned to the processor. The decision to elevate a cache line to the assist is based on its access count. However, in order to account for changing memory access patterns of the workload and thus temporal variations in cache line usage, this count is considered valid only within the window of W references during which it is updated. The active window at any point in time is indicated by the *WIN* register, the value of which is incremented after every W references. Cache lines are stamped with this value to indicate the window active at the time of the reference. If during a subsequent access to the line, this stamp is found to differ from the value in the WIN register, the line's access count is considered invalid and is reset, and its WIN stamp updated.

On the other hand, if the values are found to be matching, the access count is considered valid and is compared with R_{min} to evaluate whether the line can be elevated to the assist.

TABLE I. System Configuration

Processor	
Architecture	32b SimpleScalar/ARM
Issue Width	1 instruction/cycle
Memory Hierarchy	
L1 Data Cache	1KB, 8KB, 32KB, 64KB / 4-way
Hit Latency	1, 2, 3, 3 cycles resp.
Miss Penalty	64 cycles
Assist Cache (PSC/SVC/L0)	512B / Fully Associative
Hit Latency	1 cycle
L2 Data Cache	1MB / 8-way
Line Size/Write Policy	64B / Write-back

Therefore, if a line is accessed atleast R_{min} times within the current reference window, it is moved across the swap interface and placed within an available slot in the fully-associative assist cache. In the event that no free slot exists, the assist cache line with the least persistence is evicted and moved back to the L1D, while the newly elevated line is stored in the resulting free slot. The line evicted from the assist may however map to a different set inside the L1D and subsequently displace other useful data in that set. This is referred to as an *induced replacement* and can ironically increase contention in the L1D. In general, the occurrence of induced replacements decreases as PSC's selectivity is increased.

IV. Evaluation

We evaluate *Persistence Selective Caching (PSC)* using trace driven simulations. Data memory traces are generated using *SimpleScalar/ARM* for a single issue processor, and subsequently passed into a trace-driven cache simulator. The simulator models a conventional data cache together with the PSC assist, *Selective Victim Cache (SVC)* and write-back *L0 cache* according to the configurations listed in Table I. Seven workloads from the *MiBench* embedded benchmark suite [6] are used in the evaluation. For each simulated cache, the corresponding latency, area and power data are gathered using *CACTI* [7] for the $32nm$ technology node. The access latencies of the caches are also listed in Table I. In this paper, the persistence threshold for each workload is determined through an exploratory simulation. The R_{min} and W values thus obtained are listed in Table II, and reflect the access persistence of the most frequently used data set for the tested workloads. These thresholds can also be determined using compiler-based design space exploration frameworks [8], and memory access pattern analysis [9][10]. We are currently working on integrating an access persistence analyzer based on these schemes into our software toolchain, and also a dynamic control loop to continuously adapt the persistence threshold during execution. We intend to cover these topics in a future paper.

TABLE II. Access Persistence Thresholds

	blowfish	*dijkstra*	*ghostscript*	*gsm*	*ispell*	*mad*	*sha*
R_{min}	70	30	70	70	100	90	130
W	2048	512	1024	2048	2048	1024	1024

A. AMAT and Energy

Figure 3(a) illustrates the percentage of memory references that hit in the L1D, hit in the assist cache, and miss in the L1D. Since PSC applies access persistence as the condition for elevations, lines are not moved into the assist until they demonstrate sufficient potential for reuse. While this drastically reduces the number of swaps as compared to the L0, the reusability of the assist-cached data results in a large portion of references being serviced by the assist. For instance, in the case of $blowfish$ with a PSC assisted $8KB$ data cache, a 45% improvement in AMAT is obtained from only 18 cache line elevations to the assist. Effectively, 80% of all data references in the workload hit this small set of cache lines in the assist as seen from Figure 3(a). This observation highlights the reusability of cache lines that exhibit access persistence, and illustrates the significant performance gains that can be obtained from the selective caching of reusable lines in low-latency assists. With the $dijkstra$ workload, an L0 assisted $32KB$ data cache results in over $20E6$ swaps, while with PSC, this is reduced to $8E3$ with 50% lower AMAT and energy consumption. Figure 3(b) and (c) illustrate how the servicing of references from the assist and selectivity in elevations translate into AMAT improvements and energy savings. PSC yields consistently lower AMATs and energy per access as compared to conventional caches, and the L0 and SVC assists, across all tested workloads and cache configurations. The most significant improvements are observed when PSC is used with larger set-associative caches with access latencies and energy greater than that of the assist. PSC is also observed to be equally effective in reducing miss rates as the SVC.

Although unselectively moving cache lines to the assist and servicing references to them with a short 1 cycle latency can sometimes yield AMAT benefits, the frequent occurrence of swaps always results in higher energy consumption. An example of this is the $blowfish$ workload with the $32KB$ data cache where the PSC and L0 are observed to offer similar AMAT improvements, but while the L0 incurs a large number of swaps in doing so, the PSC yields the same improvements with only 29 swaps. Using access persistence as the selectivity criterion therefore results in a large reduction in the number of cache line movements as compared to the L0 assist. Unselective movement of data in the case of the L0 also results in aggravated miss rates due to induced replacements, and higher energy consumption. In the case of the SVC, despite its miss rate reduction, since a negligible fraction of references are serviced through the small assist, the AMAT and energy improvements obtained are small. For most cases in Figure 3(b) and (c), PSC results in AMAT upto 59% shorter, and energy consumption upto 75% lower than conventional data caches, as well as competing schemes. Note that in computing the energy per access in the case of PSC, we took into account the overheads of selective caching and maintaining of access counts and window timestamps.

B. Overheads and Implementation Cost

The swap operation for the PSC is identical to that of the SVC except for the fact that lines evicted from the PSC assist may map to a different set in the L1D than from which the newly elevated line originated. This necessitates an additional L1D lookup and in case the set is fully occupied, an extra cycle for the corresponding eviction (an induced replacement). However, the extremely low number of swaps that occur with PSC offsets any disadvantages that such overheads may pose. Adopting a very pessimistic estimate for time per PSC swap as twice that of SVC swap time (6 cycles and 3 cycles [1] respectively), the total time spent in cache line swaps is still upto 90% lower for PSC as compared to SVC, and over 99% lower than the unselective L0 assist. Table III lists the area overhead of the PSC scheme as compared to a conventional data cache. For the PSC, a $11b$ access count (corresponding to

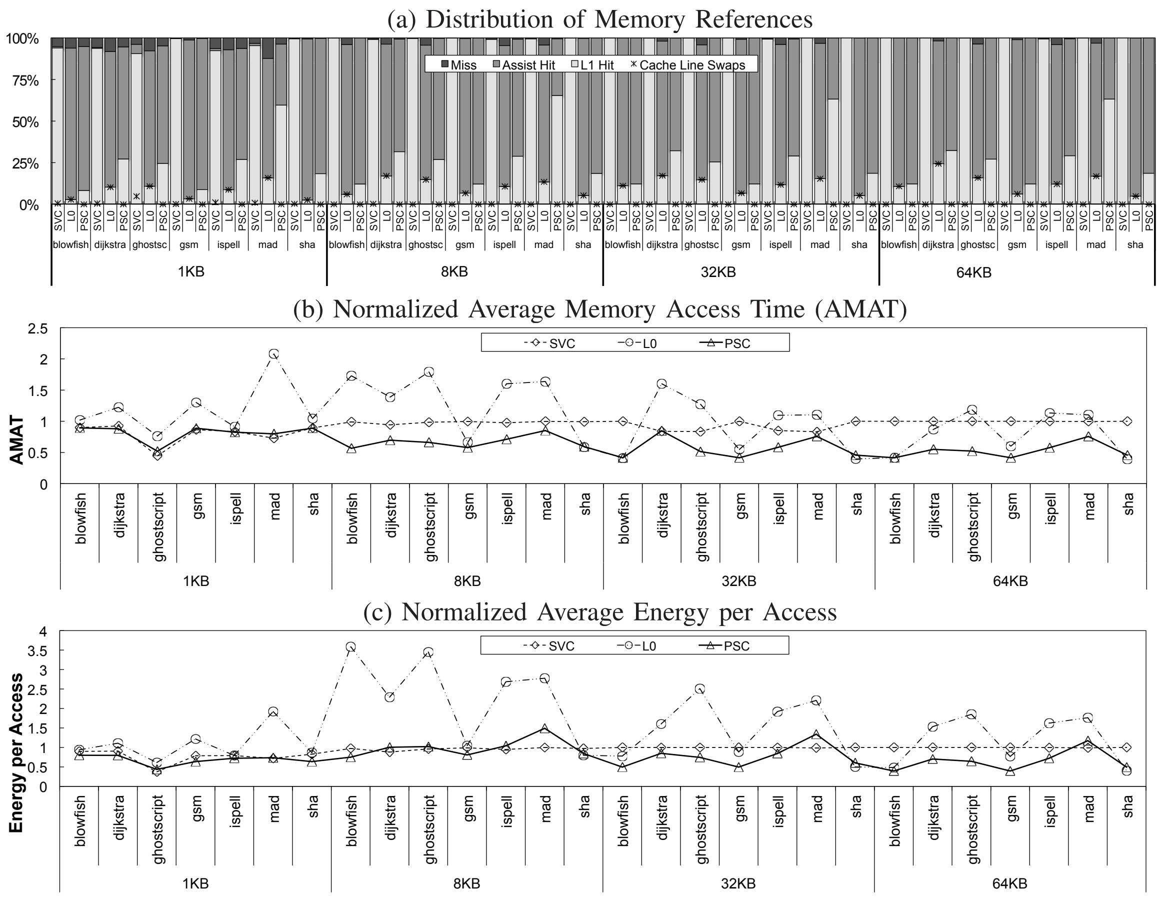

Fig. 3. (a) Distribution indicating the fraction of memory references that result in an L1D hit, miss and assist cache hit for each workload with the *Selective Victim Cache (SVC)*, *L0* and *Persistence Selective Caching (PSC)* assist. The number of cache line swaps normalized to the total number of references is also indicated. (b) Average Memory Access Time (AMAT) normalized to a conventional 4-way L1D of size 1KB, 8KB, 32KB and 64KB. (c) Corresponding average energy per access normalized to the same conventional L1D.

a window size of 2048 references), and a $21b$ window stamp were considered, translating to $4B$ of extra tags per line in both the L1D and the assist cache. The implementation overheads for the selective caching logic itself are minimal, comprising primarily of a small number of comparators and adders. Also listed in the same table are the overheads of the SVC and L0 (all assists in the table are of size $512B$).

V. Conclusions

Persistence Selective Caching (PSC) improves AMAT and energy by caching reusable lines in a low-latency, low-energy assist cache. PSC identifies reusable cache lines at runtime based on their access persistence, and by reducing the hit latency for these lines, improves AMAT by upto 59% as compared to conventional data caches. Furthermore, allowing only reusable lines into the assist, PSC reduces the number of cache line swaps between the L1D and assist cache, and thus decreases average energy per access by upto 75% over competing assist caches. This paper illustrated the benefits of selective caching using low-latency assists, and highlighted the significant amount of data reuse present in some applications.

TABLE III. Area overhead relative to a conventional L1D

	SVC	L0	PSC
1KB	46%	46%	52%
8KB	1.2%	1.2%	7.3%
32KB	1%	1%	7%
64KB	0.8%	0.8%	7%

References

[1] D. Stiliadis and A. Varma, “Selective victim caching: A method to improve the performance of direct-mapped caches,” *IEEE Transactions on Computers*, vol. 46, no. 5, pp. 603–610, May 1997.

[2] N. Duong et al., “Revisiting level-0 caches in embedded processors,” in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2012, pp. 171–180.

[3] J. Kin, M. Gupta, and W. H. Mangione-Smith, “The filter cache: an energy efficient memory structure,” in *Proceedings of the International Symposium on Microarchitecture*, 1997, pp. 184–193.

[4] N. P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers,” *SIGARCH Computer Architecture News*, vol. 18, no. 3a, pp. 364–373, May 1990.

[5] A. Janapsatya, S. Parameswaran, and A. Ignjatovic, “Hitme: Low power hit memory buffer for embedded systems,” in *Proceedings of ASP-DAC*, 2009, pp. 335–340.

[6] M. R. Guthaus et al., “Mibench: A free, commercially representative embedded benchmark suite,” in *Proceedings of the International Workshop on Workload Characterization*, 2001, pp. 3–14.

[7] N. Muralimanohar et al., “Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0,” in *Proceedings of the International Symposium on Microarchitecture*, 2007, pp. 3–14.

[8] A. Shrivastava, I. Issenin, and N. Dutt, “A compiler-in-the-loop framework to explore horizontally partitioned cache architectures.” in *Proceedings of ASP-DAC*, 2008, pp. 328–333.

[9] K. Beyls and E. H. Dholl, “Reuse distance as a metric for cache behavior,” in *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, 2001, pp. 617–662.

[10] F. J. Sanchez, A. Gonzalez, and M. Valero, “Software management of selective and dual data caches,” in *IEEE TCCA Newsletter, Special Issue on DSM and related issues*, 1997, pp. 3–10.