Author Retrospective for
The Dual Data Cache

Antonio González
Department of Computer Architecture, UPC
Intel Barcelona Research Center, Intel Labs
antonio@ac.upc.edu

Carlos Aliagas
D. Enginyeria Informàtica I Matemàtiques
Universitat Rovira I Virgili
carles.aliagas@urv.net

ABSTRACT
In this paper we present a retrospective on our paper published in ICS 1995, which to best of our knowledge was the first paper that introduced the concept of a cache memory with multiple subcaches, each tuned for a different type of locality. In this retrospective, we summarize the main ideas of the original paper and outline some of the later work that exploited similar ideas and could have been influenced by our original paper, including two actual industrial microprocessors.

DOI: http://dx.doi.org/10.1145/224538.224622

Categories and Subject Descriptors
B.3.2 [**Memory Structures**]: Design Styles – *Caches memories*

1. INSPIRATION and MOTIVATION
Cache memories had received a lot of attention by the research community prior to our work. There were many research proposals on better prefetching schemes, replacement algorithms, indexing functions, just to name a few. However, we were intrigued by the little use of specialization in cache architectures, as opposed to what was happening in other parts of the microprocessors. There was a trend to include more specialized units for particular program behaviors, but this trend was not seen in caches. For instance, the datapath had different parts for integer, FP and vector instructions; the front-end had hybrid branch predictors to use the most adequate one depending on the type of code. On the other hand, cache memories kept being composed of a monolithic unit with a smart but single policy to deal with all memory references. These observations motivated us to investigate cache architectures that had more than one caching policy, with mechanisms that dynamically choose the most adequate one for each memory reference.

Cache memories are based on exploiting locality but there are different types of locality, so our plan was to investigate whether we could build a cache system that had multiple subcaches, each one specialized for a different type of locality.

2. MAIN CONTRIBUTIONS
Among the different types of locality, the two main categories are spatial and temporal. Some memory references exhibit both, whereas others have only one of them, and some have none. The simplest way to exploit spatial locality is by means of very large cache lines, whereas the most effective way to exploit references with only temporal locality is by means of one-word lines. For memory references without locality, the most effective way to handle them is to bring only the required data (as opposed to a whole cache line) and not store it in the cache, to avoid polluting it. All these were the main goals of the Dual Data Cache [1].

The Dual Data Cache is a novel design with two subcaches, the Temporal Cache and the Spatial Cache. The Temporal Cache has very short lines to better exploit data that only exhibits temporal locality, whereas the Spatial Cache has longer lines to better exploit data with spatial locality and perhaps temporal too.

A key part of the Dual Data Cache is the scheme that decides in which subcache each reference is placed. For this purpose, we proposed a structure called the Locality Prediction Table (LPT), which keeps track at runtime of the behavior of memory instructions and tries to estimate their locality properties. The LPT is based on capturing the stride exhibited by memory instructions. Instructions with strides smaller than the Spatial Cache line size are mapped into the Spatial Cache. Instructions with zero stride (e.g., scalar references) and instructions with strides larger than the Spatial Cache line size that do not interfere with themselves when placed in the Temporal Cache are stored in the Temporal Cache. Other references simply bypass the two subcaches because they have poor locality or they would interfere when placed in cache. Besides, the Spatial Cache uses a very simple next-block prefetching scheme, which is very effective for this type of references.

In the same paper [1] we also propose a Selective Cache, which is a simplified version of the Dual Data Cache, in which there is only one subcache and the LPT that decides which references are cached and which ones are not.

Some previous works related to our paper include:

- Block algorithms [2], [3], [4] deal with large working sets.
- Bypassing [5] deals with pollution due to non-unit strides.
- Victim Cache [6], Skewed-associative Cache [7] and Column-associative Cache [8] deal with interferences (conflict misses).
- Prefetching techniques [9], [10], [11] that are based on dynamically identifying the stride of memory references.

3. IMPACT ON LATER WORKS
To the best of our knowledge, our paper was the first time a data cache with multiple caching strategies tuned to different types of locality was proposed. In the following years, there were multiple proposals that referenced our work and proposed alternative implementations that exploit this concept. In this section, we briefly summarized some of the main proposals in this area. We have classified them into three categories:

- Evolutionary: They normally propose alternative implementations that improve the behavior for some scenarios or combine this concept with others.
- Adapted to other scenarios.
  - Embedded systems.
  - Multicore systems.
- Implementations in commercial processors.
  - Samsung ClamRISC/32.
  - Intel StrongARM SA-1110.

3.1 Evolutionary

3.1.1 Hardware Oriented Proposals

A number of papers have studied the spatial/temporal locality characteristics of the memory references in different manners and proposed hardware mechanisms to exploit them.

Moshovos et al. [12] use data dependence information to decide to bypass a memory instruction, to store it in the Transient Value Cache or in the main cache. The TVC has some similarity to the Temporal cache of the Dual Data Cache.

To avoid some conflict misses in the main cache, the Annex cache[13] acts as an entry cache for compulsory misses and when some locality is detected the data is moved to a bigger cache that acts as a conventional cache. It resembles a victim cache but for incoming data as opposed to outgoing.

Johnson et al. focus on integer applications and propose a scheme that decides to do prefetching [14] or bypassing [15] of memory references via hardware monitoring of locality.

Rivers et al. [16] compare several mechanisms to obtain the locality of memory references in the L1 cache. Some uses some bits of the address and others some bits of the PC to access a table to catch the reusability of the memory access. They use this information to place the data in one of the two parts of the cache.

Kumar et al.[17] focus on spatial locality to emulate a large block cache via prefetching of the next predicted block.

Ju et al.[18] claim that it is more beneficial to cache those memory references that are in the critical path of a program instead of the ones that have more reusability.

Juurlink [19] uses a main cache with sub-blocking to emulate blocks of different sizes like those of the dual data cache. This scheme avoids to replicate data when a block has both types of locality.

Qureshi et al.[20] propose a special design for the L2 cache. Like the Dual Data Cache, it consists of two subcaches, a Line-Organized Cache and a Word-Organized Cache. When a line is evicted from the former, the used words of the lines are transferred to the later, and the unused words are discarded.

3.1.2 Software Oriented Techniques

These proposals do not require any hardware mechanism to track the behavior of memory references. Instead, this information is derived statically, during compilation, and is passed to the hardware through some ISA feature (e.g. a field in the memory instructions). The hardware uses this information to handle the memory instruction accordingly.

A follow-up work of the same group that proposed the Dual Data Cache [21] shows that a very small full-associative cache can store the data that shows only temporal locality next to the main cache.

Grun et al.[22] propose a software optimization via profiling to best map data in the address space of the application. This can be applied to any cache configuration including Dual Data Cache.

3.1.3 Survey Papers

There are a number of papers that survey the main proposals that deal with a cache organization divided in several parts to store different types of memory reference [23] [24] [25].

3.2 Adapted to Other Scenarios

3.2.1 Embedded Systems

For embedded systems, most of the works make a special emphasis in energy consumption. At the time when we proposed the Dual Data Cache, mid 90s, energy consumption was not yet a main concern. In our proposal, every memory request has to access both cache modules, then, it has to compute some data and store in a table, and in some cases, has to move or copy data form one subcache to the other. This obviously generates an increase in dynamic energy consumption.

Hsien-hsin et al. [26] propose a three module cache system managed by the type of memory references: stack data go to one module, global data is mapped to another, and the rest go to the main cache, but only one module is accessed for a given memory reference.

Petrov [27] and Kim [28] partition a cache memory in several subcaches and try to use only some of them for power reduction. The former uses static information to decide the partition assigned to each individual load/store instruction. The latter use dynamic information to map memory references to the various cache partitions. This dynamic partitioning can take into account spatial and temporal locality properties among others.

3.2.2 Multicores

Cache architectures for multiprocessors have been a hot topic in recent years due to the popularity of multithreaded processors, multicores and network on chips. Some proposals have leveraged the concept of different types of locality to optimize the design.

Sahuquillo et al. [29] study a multiprocessor with a L1 cache organization very similar to the Dual Data Cache, which is composed of two modules, one for temporal locality and another for spatial locality.

Memik et al. [30] investigate the Dual Data Cache in depth and optimize it for a multicore embedded system. They claim that the Dual Data Cache was not optimized for energy consumption, but due to its performance benefits, there is a reduction of the bus activity and the overall energy consumption is reduced.

NUCA caches have been extensively studied for multicore systems. In general, these caches try to move data next to the core for which the data has higher locality.

Kim et al. [31] explain that a D-NUCA is a cache organization that inherently adapts to the data locality. Chishti et al. [32] propose a new NUCA organization that has a flexible placement of blocks to try to keep them near the core that uses them. In both works data with poor locality is placed in farther modules, which has some similarity with our work on selective caching, in which we avoid storing in L1 blocks with poor locality.

3.3 Real implementations

3.3.1 Samsung

The CalmRISC [33] uses a “cooperative cache” that resembles the structure of the Dual Data Cache. The TOC (Temporal Cache) is a direct-mapped cache with a block size of 8 bytes and the SOC (Spatial Cache) is a four-way associative cache with a block size of 32 bytes. Both have a capacity of 8Kbytes. They claim that performance increases with respect to a conventional cache while maintaining similar values on energy consumption.

3.3.2 Intel

Intel StrongARM SA-1110 [34] uses a design similar to the Dual Data Cache. They use a ‘main cache’ of 8 Kbytes with a block size of 32 bytes and 32-way associativity and a ‘minicache’ of 512 bytes with write-back policy, 2-way set-associative and a block size of 32 bytes. The placement is decided by two bits contained in the memory instruction. The C bit indicates to cache or bypass and the B bit decides to store the data in the main cache (Temporal and

Spatial locality) or in the minicache (Spatial locality only). So the placement of the data is decided at compile time. This scheme is similar to a follow up work of our group published in 1997 [21].

4. CONCLUDING REMARKS

This paper has presented a retrospective on our ICS 1995 work on a novel cache architecture with multiple caching strategies to exploit different types of locality. This was the first paper that presented this concept to the best of our knowledge. Afterwards, a large number of papers have appeared which exploit this concept in a different manner and we have reviewed some of them in this retrospective. We have also briefly outlined two implementations in commercial microprocessors that have exploited this idea.

In the era of abundant and transistors that is characterizing our current and future technology, there will be even more opportunities for specialization, and we believe we will see this concept more often applied to all components, including the memory hierarchy.

5. ACKNOWLEDGMENTS

This work is partially supported by the Generalitat of Catalunya under grant 2009SGR-125 and the Spanish Ministry of Economy and Competitiveness under grant TIN 2010-18368.

6. REFERENCES

[1] A. González, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *9th ICS*, 1995, pp. 338–47.

[2] M. Wolfe, "More iteration space tiling," in *3rd ICS*, 1989.

[3] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," in *ASPLOS-IV*, 1991, pp. 63–74.

[4] S. Carr and K. Kennedy, "Compiler blockability of numerical algorithms," in *6th ICS*, 1992, no. April, pp. 114–24.

[5] S. McFarling, "Cache replacement with dynamic exclusion," *ACM SIGARCH News*, vol. 20, no. 2, pp. 191–200, 1992.

[6] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH News*, vol. 18, 1990.

[7] A. Seznec, "A case for two-way skewed-associative caches," *ACM SIGARCH News*, vol. 21, no. 2, pp. 169–78, 1993.

[8] A. Agarwal and S. D. Pudar, "Column-associative caches," *ACM SIGARCH News*, vol. 21, no. 2, pp. 179–90, 1993.

[9] J.-L. Baer and T.-F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," *5th ICS*, p. 176, 1991.

[10] J. W. C. Fu, J. H. Patel, and B. L. Janssens, "Stride directed prefetching in scalar processors," *ACM SIGMICRO News*, vol. 23, no. 1–2, pp. 102–10, 1992.

[11] Y. Jégou and O. Temam, "Speculative prefetching," in *7th ICS*, 1993, pp. 57–66.

[12] C. F. Chen, B. Falsafi, and A. Moshovos, "Accurate and Complexity-Effective Spatial Pattern Prediction," in *IEEE 10th HPCA*, 2004, pp. 276–276.

[13] L. John and A. Subramanian, "Design and performance evaluation of a cache assist to implement selective caching," in *ICCD VLSI*, 1997, pp. 510–8.

[14] T. L. Johnson, M. C. Merten, and W. W. Hwu, "Run-time spatial locality detection and optimization," *IEEE/ACM Micro-30*, 1997.

[15] T. Johnson et al., "Run-time cache bypassing," *IEEE Tr. Computers*, vol. 48, no. 12, pp. 1338–54, 1999.

[16] J. A. Rivers et al., "Utilizing reuse information in data cache management," in *12th ICS*, 1998, pp. 449–56.

[17] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," *ACM SIGARCH News*, vol. 26, no. 3, pp. 357–68, 1998.

[18] R. D. Ju, A. R. Lebeck, and C. Wilkerson, "Locality vs. criticality," *ACM SIGARCH News*, vol. 29, no. 2, 2001.

[19] B. Juurlink, "Unified dual data caches," in *Euromicro DSD*, 2003, pp. 33–40.

[20] M. K. Qureshi, M. A. Suleman, and Y. N. Patt, "Line Distillation: Increasing Cache Capacity by Filtering Unused Words in Cache Lines," *IEEE 13th HPCA*, pp. 250–9, 2007.

[21] F. J. Sanchez, A. Gonzalez, and M. Valero, "Static locality analysis for cache management," in *6th. PACT*, 1997.

[22] P. Grun, N. Dutt, and A. Nicolau, "Access pattern based local memory customization for low power embedded systems," in *DATE*, 2001, pp. 778–84.

[23] J. Sahuquillo and A. Pont, "Splitting the data cache: a survey," *IEEE Concurrency*, vol. 8, no. 3, pp. 30–5, 2000.

[24] J. Sahuquillo, A. Pont, and V. Milutinovic, "The filter data cache: A tour management comparison with related split data cache schemes sensitive to data localities," *HPC*, 2000.

[25] Z. Sustran et al., "A survey of dual data cache systems," in *IEEE ICIT*, 2012, pp. 450–6.

[26] S. L. Hsien-hsin and G. Tyson, "Region-Based Caching: An Energy-Delay Efficient Memory Architecture for Embedded Processors," *Conf. Compilers Archit.*, 2000.

[27] P. Petrov and A. Orailoglu, "Performance and power effectiveness in embedded processors - customizable partitioned caches," *IEEE Tr. CDICS*, vol. 20, no. 11, 2001.

[28] S. Kim et al., "Power-aware partitioned cache architectures," *ISLPED*, pp. 64–7, 2001.

[29] J. Sahuquillo and A. Pont, "The split data cache in multiprocessor systems: an initial hit ratio analysis," in *7th Euromicro PDP*, 1999, pp. 27–34.

[30] G. Memik and W. H. Mangione-Smith, "Increasing power efficiency of multi-core network processors through data filtering," *CASES*, p. 108, 2002.

[31] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *ACM SIGARCH News*, vol. 30, no. 5, p. 211, 2002.

[32] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures," *Digital Avionics Sys-22*, 2003.

[33] K. Lee et al., "The cache memory system for CalmRISC32," in *2nd IEEE AP-ASICs*, 2000, pp. 323–6.

[34] Intel, "Intel StrongARM SA-1110 Microprocessor. Developer's Manual," no. June, 2000.