Appears in the 2020 Workshop on Approximate Computing Across the Stack (WAX 2020)

Decoupling Approximation and Cache Compression

Amin Ghasemazar
University of British Columbia
aming@ece.ubc.ca

Mohammad Ewais
University of Toronto
mewais@ece.utoronto.ca

Mieszko Lis
University of British Columbia
mieszko@ece.ubc.ca

Abstract

In this paper, we make the case that approximation and compression should be decoupled: that is, approximation should be designed as an optional component in a standard cache compression pipeline.

As proof of concept, we combine an approximation module with several existing cache compression schemes; through simulation-based evaluation on a range of approximable workloads, we show that decoupled designs offer higher compression than bespoke approximate caches without sacrificing performance.

1 Introduction

Approximate caches [11, 12] can substantially increase effective cache capacity by taking advantage of *approximate value locality* — the observation that many values stored in a cache are so close that replacing one value by another makes little difference to the effectiveness of many applications.

This locality can be converted to increased cache capacity by detecting cachelines with approximately equal value sequences, storing only one of those lines in the cache, and returning an acceptable approximation when a line is retrieved. For example, Doppelgänger [12] treats the average of all values in the cacheline as a "signature" and stores only one representative cacheline for each signature.

Existing designs, however, share two significant limitations. First, because they are in effect lossy compression techniques, they can only compress data that identified as approximable (e.g., by the programmer), and are of little use for applications where approximation is not practical. Second, because approximation is an integral part of the lossy compression mechanism, they can only compress approximable data that fit a single redundancy pattern: for example, Doppelgänger captures value similarity *between* cachelines, but ignores value similarity *within* each cacheline [12], and Bunker Cache is only effective on image-like data [11]. Both of these are serious barriers to adoption in commercial CPUs.

In this paper, we make the case that approximation and compression are orthogonal, complementary techniques that should be *decoupled* in cache designs. Such a design, illustrated in Figure 1, would comprise two stages: (a) a lightweight approximation stage, applied only to data identified as approximable, and (b) a cache compression stage, applied to all data, that leverages one of the many existing cache compression proposals [1–3, 5, 6, 9, 10, 15]. To demonstrate the practicality of this approach, we combine approximation with three representative cache compression techniques [6, 10, 15]. The decoupled paradigm allows the designer to choose a compression algorithm that is suitable for the application rather than one that is dictated by the approximate cache design, which in turn can result in better space savings: simulations on a range of applications from AxBench [16], Parsec [4], and SPEC [7] show that, under the same quality-of-results criteria, decoupled designs achieve up to 63% more compression (7.4× geomean) than a bespoke design like Doppelgänger. Importantly, decoupled designs can also compress non-approximable data, potentially making approximate caches more attractive for commercial applications.

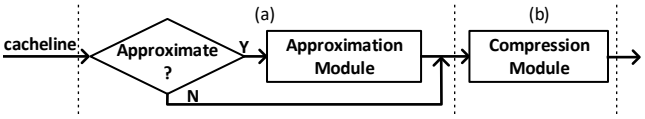

Figure 1. A decoupled approximate cache design that separates the approximation (a) and compression (b) aspects.

2 Experimental setup

Quality of results criteria. For all benchmarks, we determined quality cutoffs where degradation was (a) barely noticeable visually or (b) did not compromise the purpose of the benchmark. The quality metrics are shown in Table 1.

We focused on floating-point array structures, and applied the maximum single approximation level suitable for the entire workload; more approximation may be possible by using per-datastructure levels. Approximation and compression are applied only at the LLC level whenever a new cacheline is brought in from DRAM; decompression and deapproximation are applied whenever the line is read or written.

Bespoke approximate cache. We implemented Doppelgänger [12], the best-performing approximate cache proposal to date (the more recent Bunker Cache [11] has worse compression [11, fig. 22]) Value ranges for the approximation maps were taken from recorded runtime minimum and maximum values in the manually annotated approximable regions of each benchmark using the 14-bit map space.

BM	fp64/fp32	Quality	BM	fp64/fp32	Quality
swaptions	45/0	MSE<1	fft	0/17	MSE<82.5
streamcluster	0/11	AE=0%	sobel	0/18	PSNR>60
calculix	47/0	MSE=0	vips	43/17	PSNR>60
inversek2j	0/17	MSE<5e-3	milc	32/0	MSE<5e-8
blackscholes	0/18	MSE<5e-3	ferret	0/5	AE=0%
kmeans	0/6	PSNR>50	namd	32/0	AE<1e3
cactusADM	38/0	MSE<2e-7	jmeint	0/17	RE<5e-3

Table 1. Quality cut-off levels for fp64/fp32 and QoR criteria.

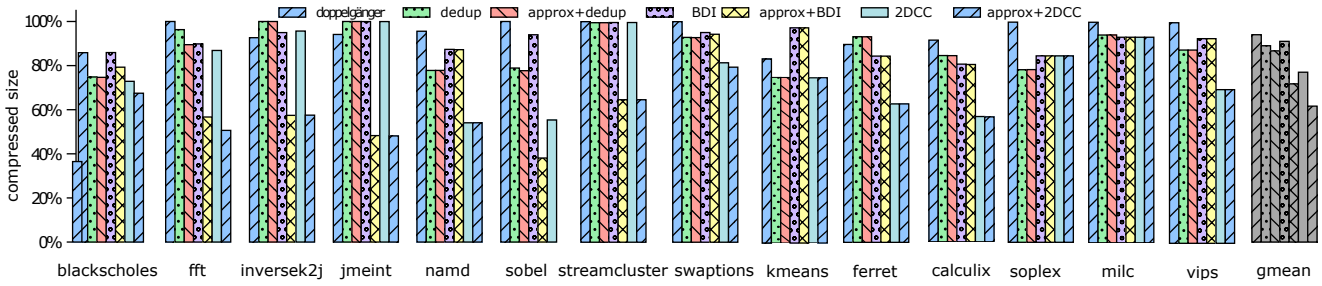

Figure 2. Compressed working sizes for Doppelgänger and several decoupled approximation+compression combinations.

Decoupled approximation scheme. To approximate floating-point numbers, we zero the *s* rightmost bits of the mantissa, where *s* is chosen per application according to the QoR metrics above; we then right-shift the entire value by *s* bits. When an approximate value is fetched, we left-shift the stored value by *s* bits and use that to service the request. This mechanism, illustrated in Figure 3(a), naturally lends itself to a simple, fast hardware implementation.

Like prior work [11–13], we assume that approximate data structures are annotated by the programmer. In our design, the approximation level *s* is included in every cache request, as is a single bit identifying the datatype (fp32 or fp64); this can be implemented either by extending the ISA with approximate variants of load/store instructions or by adding a hardware region lookup table that can be filled by the application and consulted during load/store execution.

To fairly compare with Doppelgänger, we only approximate to cachelines where all elements have the same type.

Decoupled compression schemes. We combined this approximation scheme with three representative cache compression techniques: exact deduplication (Dedup) [15], base-delta-immediate compression (BDI) [10], and a proposal that combines both (2DCC) [6]. Briefly, BDI compresses a line containing values within a close range by storing one uncompressed "base" (e.g., 64 bits) followed by shorter offsets (e.g., 16 bits) that can be added to the base to recreate the original value. Dedup, on the other hand, compresses the cache by detecting exact duplicate cachelines and storing only one copy in the data array. 2DCC captures both intra-line and inter-line redundancy, effectively combining BDI and Dedup.

Compression is applied after approximation and before deapproximation, as shown in Figure 3(b).

Simulation environment. We implemented all schemes in ZSim [14]; both functionality (computing with approximate values) and all the timing events on and off the critical path were modelled. The simulated system was configured as shown in Table 2. We simulated a range of floating-point-focused benchmarks from AxBench [16], SPEC2006 [7], and Parsec 3.0 [4]. Input sizes were selected to fill a 2MB cache whenever possible. For each benchmark, we simulated different approximation levels and chose the highest approximation level consistent with acceptable QoR (vide supra).

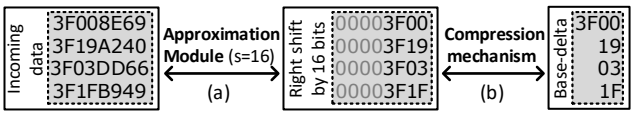

Figure 3. Approximating and compressing a cacheline from *jmeint* [8, 16] using a base-delta representation [10].

3 Results and discussion

Figure 2 shows the savings from compression and approximation normalized to the uncompressed footprint for all caches, measured as a running average over each workload's region of interest. The decoupled approximate caches reduce workload footprints to as little as 36.5% (sobel) and 60% on average (gmean); all outperform Doppelgänger, which only manages to reduce the footprint to 95% (gmean).

This is largely due to two factors. One is that the decoupled approximate caches capture the intra-line redundancy created by the approximation stage (e.g., fft, inversek2j, jmeint, sobel, streamcluster); because Doppelgänger effectively implements near-deduplication, it cannot capture intra-line effects. The other is that the decoupled designs can also compress non-approximable data (e.g., ferret).

All of the approximate caches perform on par with the 2MB uncompressed baseline (±1–3% (gmean), not shown).

Overall, these results make a case for decoupling approximation and cache compression — general-purpose designs where approximation is implemented as a separate, optional stage in the cache compression pipeline. These achieve better compression than bespoke approximate caches, and offer a practical strategy for introducing approximation in commercial cache hierarchies.

component	configuration
CPU	x86-64 1 GHz, 4-wide OoO, 80-entry ROB
private L1$	16KB, 4-way, 1-cycle latency, 64B blocks, LRU
private L2$	128KB, 8-way, 3-cycle latency, LRU
shared LLC	2MB (baseline) / 1MB (compressed variants), single bank, 16-way, inclusive, 6-cycle latency,
main memory	1GB, 160-cycle latency

Table 2. Simulated system configuration.

References

[1] Alaa R. Alameldeen and David A. Wood. 2004. *Frequent pattern compression: A significance-based compression scheme for L2 caches.* Technical Report 1500. University of Wisconsin-Madison.

[2] Angelos Arelakis, Fredrik Dahlgren, and Per Stenström. 2015. HyComp: A Hybrid Cache Compression Method for Selection of Data-type-specific Compression Methods. In *MICRO*.

[3] Angelos Arelakis and Per Stenström. 2014. SC^2: A Statistical Compression Cache Scheme. In *ISCA*.

[4] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors.* Ph.D. Dissertation. Princeton University.

[5] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas. 2010. C-Pack: A High-Performance Microprocessor Cache Compression Algorithm. *IEEE Transactions on Very Large Scale Integration Systems* 18, 8 (Aug 2010), 1196–1208.

[6] Amin Ghasemazar, Mohammad Ewais, Prashant Nair, and Mieszko Lis. 2020. 2DCC: Cache Compression in Two Dimension. In *DATE*.

[7] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17. https://doi.org/10.1145/1186736.1186737

[8] Tomas Möller. 1997. A Fast Triangle-Triangle Intersection Test. *Journal of Graphics Tools* 2, 2 (1997), 25–30. https://doi.org/10.1080/10867651.1997.10487472 arXiv:http://dx.doi.org/10.1080/10867651.1997.10487472

[9] B. Panda and A. Seznec. 2016. Dictionary sharing: An efficient cache compression scheme for compressed caches. In *MICRO*.

[10] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2012. Base-delta-immediate Compression: Practical Data Compression for On-chip Caches. In *PACT*.

[11] Joshua San Miguel, J. Albericio, Natalie Enright Jerger, and A. Jaleel. 2016. The Bunker Cache for spatio-value approximation. In *MICRO*. 1–12.

[12] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. 2015. Doppelgänger: A Cache for Approximate Computing. In *MICRO*.

[13] Joshua San Miguel, M. Badr, and Natalie Enright Jerger. 2014. Load Value Approximation. In *MICRO*.

[14] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *ISCA*.

[15] Yingying Tian, Samira M. Khan, Daniel A. Jiménez, and Gabriel H. Loh. 2014. Last-level Cache Deduplication. In *ICS*.

[16] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. 2017. AxBench: A Multiplatform Benchmark Suite for Approximate Computing. *IEEE Design Test* 34, 2 (April 2017), 60–68. https://doi.org/10.1109/MDAT.2016.2630270