Rollback-Free Value Prediction with Approximate Loads

Bradley Thwaites Gennady Pekhimenko[§] Amir Yazdanbakhsh Jongse Park Girish Mururu
Hadi Esmaeilzadeh Onur Mutlu[§] Todd Mowry[§]
Georgia Institute of Technology [§]Carnegie Melon University
{bthwaites,a.yazdanbakhsh,jspark,girishmururu}@gatech.edu gpekhime@cs.cmu.edu
hadi@cc.gatech.edu onur@cmu.edu tcm@cs.cmu.edu

ABSTRACT

This paper demonstrates how to utilize the inherent error resilience of a wide range of applications to mitigate the memory wall—the discrepancy between core and memory speed. We define a new microarchitecturally-triggered approximation technique called rollback-free value prediction. This technique predicts the value of safe-to-approximate loads when they miss in the cache without tracking mispredictions or requiring costly recovery from misspeculations. This technique mitigates the memory wall by allowing the core to continue computation without stalling for long-latency memory accesses. Our detailed study of the quality trade-offs shows that with a modern out-of-order processor, average 8% (up to 19%) performance improvement is possible with 0.8% (up to 1.8%) average quality loss on an approximable subset of SPEC CPU 2000/2006.

1 Introduction

Due to diminishing gains from CMOS scaling and the overwhelming growth of data, a growing body of recent work [6, 9, 11, 1, 10, 4, 8, 13, 3, 15] investigates approximation techniques in general-purpose computing that trade *Quality of Result* (QoR) for gains in performance and efficiency. These techniques exploit the inherent error resiliency of a wide range of applications including web search, data analytics, image processing, cyber-physical systems, recognition, and optimization to improve performance and efficiency through approximation. Instances of these approximation techniques include (i) voltage over-scaling [9, 4]; (ii) loop perforation [15]; (iii) loop early termination [3]; (iv) computation substitution [11, 8, 2, 1, 3]; (v) limited fault recovery [6]; and (vi) approximate storage design [10, 13]. We define a new technique, rollback-free value prediction, which operates at the fine granularity of a single load instruction. We exclusively focus on mitigating the effects of cache misses through exploiting application error-tolerance. A unique characteristic of our technique is its trigger. Microarchitectural events, namely cache misses, invoke the approximation rather than explicit software invocation as in other techniques. Despite these differences, our technique may be effectively combined with prior work on approximation due to its concentrated and limited use of approximation. Rollback-free value prediction speculates on the value of an *approximate load* that misses in the cache, allowing computation to continue with speculative values without rollback from mispredictions[1]. We define approximate loads as instructions that with low probability may return values other than what is stored in the memory. Thus, due to the approximate load semantics, tracking mispredictions and costly rollback from misspeculations is avoided. However, our design principle for rollback-free value prediction is to restrain the effects of approximation to low levels and only utilize approximation when it is most beneficial.

Following this design principle, we have identified the following three challenges that must be addressed in order to effectively realize the rollback-free value prediction approximation technique.

(i) Targeting performance-critical and safe loads. A framework needs to be developed that only employs rollback-free value prediction on performance-critical loads. To this end, we develop a profile-driven approach that identifies a limited set of performance-critical loads as candidates for rollback-free prediction. The profiling pass identifies which loads cause the majority of cache misses, and which ones may be approximated at low quality cost. We then utilize explicit programmer oversight to only approximate loads whose approximation will never lead to program crashes (Section 2).

(ii) Utilizing fast-learning load value predictors. Due to the single-load granularity of our technique, the prediction mechanisms should be light weight and fast learning. We investigate using two-delta [7] and stride [14] predictors and show that two-delta provides significant performance gains with very low quality degradation.

(iii) Integrating the rollback-free prediction in the architecture. Both low-overhead microarchitectural and architectural extensions are needed to intercept cache misses for approximate loads and continue execution. Section 3 provides details about our design.

2 Profile-Directed Approximation

Providing safety guarantees. Any viable approximation technique needs to provide strict guarantees that the program will only experience graceful quality degradation without catastrophic failure. We define a safety violation as an execution which, due to approximation, leads to a catastrophic failure such as segmentation fault, invalid jump address, memory out of bounds error, or infinite loop. As other approximation techniques [9, 3, 13, 12], the programmer is responsible for identifying the safe-to-approximate loads through programming language constructs. However, to reduce programmer effort, our profiling pass presents an initial set of performance-critical loads to the programmer as described below.

Targeting performance-critical loads. To limit the undesirable effects of approximation to the lowest level, we develop an automated compile-time profiling pass that consists of two stages. The first stage determines the set of load instructions which account for the largest portion of cache misses. As prior work has shown [5] and our own experiments corroborate, a few load instructions tend to produce a large majority of cache misses. Since our prediction hardware has limited resources, we select as candidates only those loads which can provide significant benefit with approximation, typically 15-20 of them. The second profiling pass further prunes this subset by examining whether individually approximating these candidate loads will lead to significant quality degradation or increase in runtime. The profiler observes the resulting output quality and eliminates any candidate load which significantly degrades quality or causes an increase in runtime relative to a fully precise baseline.

[1]The work in [16] explores recovery-free value prediction for prefetch generation, but the speculative values never affect the microarchitectural state as in our technique.

3 Rollback Free Value Prediction

While profiling identifies approximate loads at compile time, the architecture support enables rollback-free prediction at runtime.

ISA support and semantics. To avoid exposing memory subsystem or value prediction details to the compiler, we define a probabilistic semantic for approximate loads and provide a simple hardware/software interface. For all load instructions in the ISA, we introduce a dual *approximate* variant. A bit in the opcode identifies whether the load is precise or approximate. Semantically, executing the load.approx Reg<id>, MEMORY<address> instruction will assign Reg<id> the exact value stored in MEMORY<address> with probability p, and an arbitrary value with probability $1 - p$. In practice, p is usually high with our approximation technique for two reasons. First, our technique is only triggered by cache misses. Loads which hit in the cache will always return the correct value, a common case for modern architectures with high cache hit rate. Second, even when a cache miss happens, the value predictor can predict the correct value or at least a reasonably close value. These effects inherently limit the undesirable effects of approximation.

Microarchitecture integration. When an approximate load misses in the cache, the value predictor intercepts the event and generates a prediction value. The predictor writes this value to the physical register, then sends a broadcast message to the reservation stations. The approximate load then commits normally. However, the data request is still sent to the memory.[2] When the data for an approximate load arrives, the core updates the prediction tables without checking the status of the approximate load that generated the request.

Predictor design. We examine both two-delta [7] and stride value predictors [14]. Two-delta is an extension of the stride predictor in which the stride value is only updated when the same stride is observed twice or more in a row. Although we used these predictors, our technique is independent of the specific prediction mechanism. Since our profiling pass produces a small number of load instructions for approximation, we use a prediction table with only 256 entries (6 KB overhead). The table index is a hash of the load PC.

4 Methodology and Results

We use an approximable subset of SPEC CPU 2000/2006 to evaluate our technique. Similar to [15], we define a quantitative metric to understand the quality degradation. For each SPEC benchmark, we compare the relevant numerical outputs and compute the normalized root mean square error. For profiling and quality evaluations, we use Valgrind with Cachegrind. For performance evaluations, we use the Marssx86 cycle-accurate simulator. The core is modeled after the Nehalem microarchitecture. The baseline memory system includes a 32 KB L1 cache and a 2 MB L2 with a 200-cycle memory latency. In modern processors, the LLC cache size is usually 2 MB $\times$ number of cores. Thus, a 2 MB LLC is used for our single core experiments. We sweep LLC size and issue width to investigate how our technique performs under increased memory pressure. We use Simpoint to identify the representative application phases.

Figure 1 shows how different value prediction mechanisms, in particular stride [14] and two-delta prediction, affect quality of results. With stride prediction, the average quality loss is 10.8%, but with two-delta prediction, the average loss is reduced to 0.8%. Figure 2 shows the application speedup for each configuration studied. We observe that rollback-free value prediction improves application performance with the default system (first bar) by as much as 19.2% (geometric mean: 8.1%). As expected, the observed benefits are inversely correlated with the degree of memory pressure (e.g., the smaller the cache and/or the larger the width, the higher the benefits). The highest average improvement is 15.2% (for 512KB and 4-wide configuration), up to a maximum of 40.1%.

[2] Our current scheme merely targets reducing effective memory latency. However, we are exploring designs that drop a certain fraction of the requests to reduce memory bandwidth requirements.

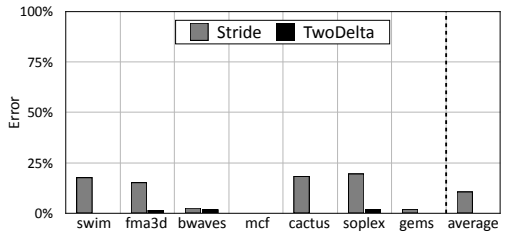

Figure 1: Error with stride and two-delta predictors.

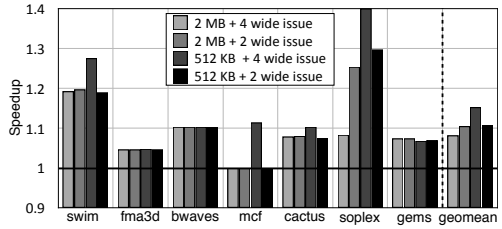

Figure 2: Speedup for different configurations with two-delta prediction (baseline: no value prediction).

5 Conclusions

In the emerging landscape of computing in which mobile and cloud services aim to provide a more personalized experience for users, applications with error resiliency are becoming dominant. Leveraging this error resiliency, we introduce a new rollback-free value prediction technique to mitigate the memory wall. Our technique represents a new class of approximation techniques that are triggered by microarchitectural events, e.g., cache misses. The microarchitectural trigger and our profiling technique limit the undesirable effects of approximation to small levels, while delivering significant gains in performance. Our sensitivity studies confirmed that when the memory subsystem is under pressure, even greater benefits are possible. These results suggest that our technique will be even more effective as we enter the era of processing overwhelming amounts of data that put more pressure on the memory subsystem.

References

[1] C. Alvarez *et al.*, "Fuzzy memoization for floating-point multimedia applications," *IEEE Trans. Comput.*, 2005.

[2] R. S. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014.

[3] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *PLDI*, 2010.

[4] L. N. Chakrapani *et al.*, "Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology," in *DATE*, 2006.

[5] J. D. Collins *et al.*, "Speculative precomputation: Long-range prefetching of delinquent loads," in *ISCA*, 2001.

[6] M. de Kruijf *et al.*, "Relax: An architectural framework for software recovery of hardware faults," in *ISCA*, 2010.

[7] R. J. Eickemeyer and S. Vassiliadis, "A load-instruction unit for pipelined processors," *IBM JRD*, 1993.

[8] H. Esmaeilzadeh *et al.*, "Neural acceleration for general-purpose approximate programs," in *MICRO*, 2012.

[9] H. Esmaeilzadeh *et al.*, "Architecture support for disciplined approximate programming," in *ASPLOS*, 2012.

[10] S. Liu *et al.*, "Flikker: Saving refresh-power in mobile devices through critical data partitioning," in *ASPLOS*, 2011.

[11] M. Samadi *et al.*, "Sage: self-tuning approximation for graphics engines," in *MICRO*, 2013.

[12] A. Sampson *et al.*, "EnerJ: Approximate data types for safe and general low-power computation," in *PLDI*, 2011.

[13] A. Sampson *et al.*, "Approximate storage in solid-state memories," in *MICRO*, 2013.

[14] Y. Sazeides and J. E. Smith, "The predictability of data values," in *MICRO*, 1997.

[15] S. Sidiroglou-Douskos *et al.*, "Managing performance vs. accuracy trade-offs with loop perforation," in *FSE*, 2011.

[16] H. Zhou and T. M. Conte, "Enhancing memory level parallelism via recovery-free value prediction," in *ICS*, 2003.

This work was supported in part by a gift from Google.