RETROSPECTIVE: Dead-Block Prediction and Dead-Block Correlating Prefetchers

Babak Falsafi, EPFL

I. Context

CPU and memory performance have been diverging since the inception of microprocessors with CPU design and fabrication primarily optimized for speed and DRAM for density. Cache hierarchies exploiting locality in memory references have been instrumental in narrowing this performance gap. Around the time of this publication, (single-core) CPUs were being shipped with on-chip hierarchies of 10s of KB of L1 split instruction and data and 100s of KB of unified L2 caches.

While basic cache design was quite mature, CPU microarchitecture continued to advance with growing on-chip transistor budgets requiring commensurate advancements in designing cache hierarchies to bridge the CPU memory performance gap. At the same time, many were also designing and building scalable NUMA multiprocessors where accessing remote memory was easily several times longer than local memory and coherence often involved traversing multiple machine nodes. Efficient cache hierarchy management and tolerating or hiding memory access latency in such machines would have a dramatic impact on performance.

II. Livetime vs. Deadtime

Block livetime and deadtime analysis in caches dates back to work on analytic models to predict cache miss rate [10], [15]. In this paper, we define livetime to be the number of CPU cycles from the time a block is brought into the cache until the last reference hitting on the block. The deadtime is from this last reference until the block is replaced (due to a miss to another block). Wood et al. [15] showed that cache block frames follow an alternating renewal process with deadtimes being on average an order of magnitude larger than livetimes resulting in the majority ($> 80\%$) of a cache being dead at any given time. The reason blocks are dead for most of their cache residency while allowing for high ($> 90\%$) hit rates is because of locality of reference in memory accesses clustering accesses in time to a small fraction of the blocks.

Figure 1 plots the cumulative distribution of deadtimes in a 32KB L1 data cache for a simulated 2GHz 8-way out-of-order CPU with a 128-entry instruction window. The graph clearly shows that the majority of block deadtimes are orders of magnitude longer than even memory access latency. Predicting the last "touch" (i.e., a hit) to a block accurately would allow evicting the block early and reclaiming the available space.

III. Dead-Block Prediction

Dead-block prediction was based on the simple observation that control flow in programs is repetitive and therefore its impact on the movement of cache blocks in the hierarchy should also be repetitive and predictable. This observation was first [9] applied to self-invalidation in shared-memory multiprocessors to reduce coherence overhead by sending blocks back to their home node when no longer needed by predicting the last touch prior to eviction. A last-touch predictor would encode a trace of all program counters touching the block from the time it is brought into the cache until it is evicted.

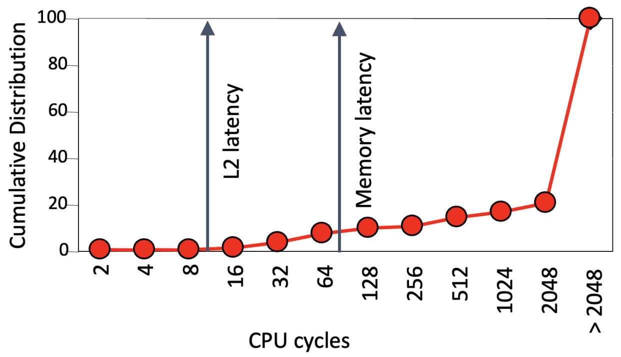

Fig. 1. Cache block deadtime in CPU cycles.

Figure 2 [9] shows examples of instruction traces accessing a cache block from the time the block is brought in until the last touch prior to eviction in various control flow scenarios. A predictor capturing the program counter trace in a signature and using it to predict a last touch with a confidence counter (e.g., a 2-bit counter) can self-invalidate the block at the earliest possible point in time to help reduce coherence overhead. The signature was formed with an xor operation on all PCs in the trace. The predictor showed an average accuracy of 80% when correlated with the cache block address.

IV. Dead-Block Correlating Prefetchers

Address correlating prefetchers (e.g., Markov [4]) emerged in the 90s as a solution to improve coverage over stream buffers and stride predictors. These prefetchers use a history of one or more missed cache block addresses to predict a future address. Unfortunately, the prefetchers fell short of being effective for three key reasons: (1) they used cache misses as triggers for prefetching which led to minimal prefetch lookahead because misses were often clustered, (2) they suffered from high misprediction, overfetching (useless) blocks wasting memory bandwidth, and (3) because miss addresses were used as index to look up predictions, the history state space grew with the memory working set size.

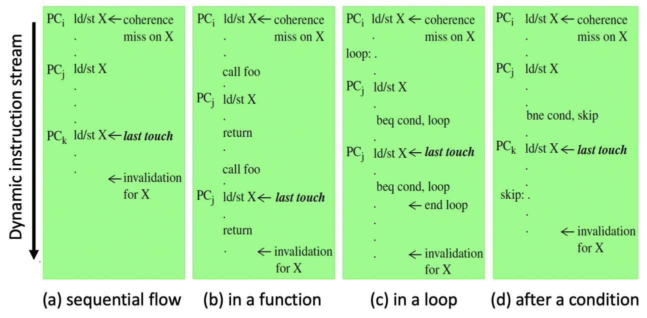

Fig. 2. Examples of last touch in shared-memory programs.

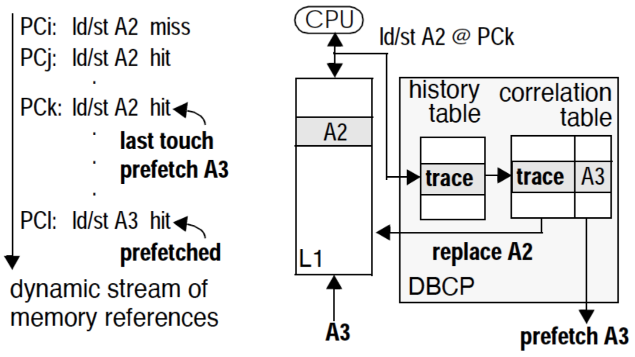

Fig. 3. A dead-block correlating prefetcher.

Dead-block correlating prefetchers' key contributions were [8]:

- Predicting a last touch allowed orders of magnitude lookahead (from deadtime) for prefetching (Figure 1).
- Correlating a block address with a last-touch trace allowed a correlation between a data structure being accessed and a program phase reducing misprediction (and overfetch) to a small percentage point.
- Proposed and evaluated both an on-chip 2MB version for smaller working sets and an off-chip 8MB version trading off memory bandwidth for a larger correlation table.

Figure 3 depicts an example of an instruction stream leading to a prefetch and the anatomy of the prefetcher. In this example, PC$_i$, PC$_j$ and PC$_k$ form a 12-bit signature xor'ed with address A2 and used as an index for the correlation table. The paper showed that correlating a last-touch trace with a block address improves coverage in address correlating prefetchers to 82% with only 4% overfetch. In contrast to prior work [4], the paper also showed that correlating bits from prior addresses improves coverage and accuracy.

V. The Paper's Legacy

This work inspired many in the community to work on dead-block prediction to better manage cache hierarchy capacity beyond L1 data caches with higher accuracy or lower silicon overhead. Timer-based (cycle or reference count) solutions [2] ended up not being accurate because of the high variability in timer thresholds both within and across blocks. With leakage power projecting the end of Dennard Scaling, these techniques were borrowed to control when to turn cache blocks on and off [5] with voltage supply gating [11].

While dead-block optimization opportunity was even higher with deeper hierarchies in the years that followed, dead-block prediction in lower cache levels became even a bigger challenge as memory accesses are filtered by higher cache levels and instruction traces in the pipeline being farther away from lower cache levels. Khan et al., extended this work with dead-block prediction for victim buffering [6], and set-sampling [7] to reduce predictor state in LLCs. LLC management in the multicore era eventually benefited mostly from work on cache block placement/replacement policies [12].

The lasting impact of this work has been temporal streaming [14] to prefetch entire streams of correlated addresses (e.g., in pointer chasing), with subsequent work on practical on-chip/off-chip designs for recording, looking up and fetching streams [1], [3], [13]. Examples of temporal streaming have appeared in IBM BlueGene/Q and ARM Neoverse N2.

References

[1] M. Ferdman and B. Falsafi, "Last-Touch Correlated Data Streaming," in *Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems & Software, 2007.*

[2] Z. Hu, M. Martonosi, and S. Kaxiras, "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior," in *Proceedings of the 29th International Symposium on Computer Architecture, 2002.*

[3] A. Jain and C. Lin, "Linearizing Irregular Memory Accesses for Improved Correlated Prefetching," in *Proceedings of the 46th IEEE/ACM International Symposium on Microarchitecture, 2013.*

[4] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors," in *Proceedings of the 24th International Symposium on Computer Architecture, 1997.*

[5] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *Proceedings of the 28th International Symposium on Computer Architecture, 2001.*

[6] S. M. Khan, D. A. Jiménez, D. Burger, and B. Falsafi, "Using Dead Blocks as a Virtual Victim Cache," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2010.*

[7] S. M. Khan, Y. Tian, and D. A. Jiménez, "Sampling Dead Block Prediction for Last-Level Caches," in *Proceedings of the 43rd IEEE/ACM International Symposium on Microarchitecture, 2010.*

[8] A. Lai and B. Falsafi, "Dead-block prediction and dead-block correlating prefetchers," in *Proceedings of the 28th International Symposium on Computer Architecture, 2001.*

[9] A. Lai and B. Falsafi, "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction," in *Proceedings of the 27th International Symposium on Computer Architecture, 2000.*

[10] A. Mendelson, D. Thiébaut, and D. K. Pradhan, "Modeling Live and Dead Lines in Cache Memory Systems," *IEEE Trans. Computers*, vol. 42, no. 1, pp. 1–14, 1993.

[11] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-V$_{dd}$: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *Proceedings of the International Symposium on Low Power Electronics and Design, 2000.*

[12] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive Insertion Policies for High Performance Caching," in *Proceedings of the 34th International Symposium on Computer Architecture, 2007.*

[13] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical Off-Chip Meta-Data for Temporal Memory Streaming," in *Proceedings of the 15th International Symposium on High Performance Computer Architecture, 2009.*

[14] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi, "Temporal Streaming of Shared Memory," in *Proceedings of the 32st International Symposium on Computer Architecture, 2005.*

[15] D. A. Wood, M. D. Hill, and R. E. Kessler, "A Model for Estimating Trace-Sample Miss Ratios," in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1991.*