Best-Offset Hardware Prefetching & Temporal Prefetching Without the Off-Chip Metadata

Yash Shah (ys562)

October 2024

1 Background and related work

Prefetchers are critical for reducing the cache miss rate by speculatively fetching data into cache lines in a timely manner and rely on spatial and temporal correlation to form their predictions.

1.1 Spatial correlation

Spatial correlation allows prefetchers to predict regular memory access patterns (like strided array access) effectively. Next-line prefetchers are the simplest spatial example, with strided prefetchers learning the optimal stride from recent memory accesses. PC-localization is often used to isolate interleaved streams based on the Program Counter (PC) address of the instruction issuing the load/store. Stream prefetching [1] [2] prefetches into a separate stream buffer to improve timeliness and reduce cache pollution.

The Sandbox Prefetcher (SBP) [3] is an offset prefetcher and avoids streams. It prefetches $X + D$, when X is requested, where D is dynamically adjusted. Although aggressive prefetching can be beneficial, it may also waste memory bandwidth and scarce cache capacity. SBP thus allows "fake" prefetch addresses to be tried within a Bloom filter (which answers whether an element is possibly or definitely not in a set). On subsequent cache access, the presence of a corresponding prefetch in the Bloom filter increases the score for the particular D value leading to that prediction. This approach can be extended to evaluate different aggressive prefetching schemes safely.

1.2 Temporal correlation

Irregular memory access patterns, e.g. resultant of pointer chasing, cannot be effectively captured by spatial methods. Address correlation aims to learn pairs of correlated memory addresses from temporal streams. Unfortunately, this leads an enormous amount of metadata, which is often stored on off-chip DRAM, and needs to be carefully managed to minimize memory traffic and latency.

Markov prefetchers [4] store a table with multiple possible successors (with probabilities) for each address. The latency of accessing this large off-chip table is amortized by storing chains of successors, but introduces redundancy and complicates support for variable length patterns. PC-localization eliminates the need to track multiple successors in each table entry, reducing table size considerably.

Global History Buffer (GHB)-based approaches [5] like STMS [6] aim to support variable-length streams by a global FIFO circular buffer logging memory accesses, along with an index table to point to references to a particular memory location in the buffer. The FIFO buffer allows contiguous fetching of temporally-related data, but does not lend well to caching. It is infeasible to combine the GHB's address correlation with PC-localization. A naïve approach requires storing a separate GHB for each PC value. Alternatively we make approximations like PC+Spatial Region Offset.

The Irregular Stream Buffer (ISB) [7] maps correlated addresses to consecutive addresses in a structural address space, supporting variable-length streams, PC-localization, and cachability by caching the Physical-to-Structural (PS) and Structural-to-Physical (SP) translation tables. Synchronization of PS and SP with the TLB helps mask off-chip metadata access latency during TLB misses, but leads to large traffic overhead as large amounts of useless metadata ($> 90\%$ never used) is fetched.

These shortfalls are addressed by Efficient Metadata Management for Irregular Data Prefetching (MISB) [8] which decouples the ISB from the TLB, manages the PS and SP caches at a finer granularity, and introduces metadata prefetching for latency reduction.

2 BO prefetcher

BO [9], like SBP, is an offset prefetcher, but aims to also address prefetch timeliness. A Recent Requests (RR) table records the base (triggering) address of completed prefetch requests (e.g. for $X + D$, the base address is X). Unlike SBP, the RR table consists of real prefetches.

For optimal timing, we require the time between accesses to lines $X - d$ and X be (slightly) greater than the latency for completing a prefetch request. So, if a prefetch request for $X - d + D$ was recently issued and completed, and $X - d$ is in the RR table, a prefetch with offset d instead of D would have been timely.

An offset list stores possible values of d, and a score table associates a score with every d_i in this list. On every miss or prefetched hit for an L2 read, a round is triggered. Every round, the next d_i is evaluated by increasing its score if $X - d_i$ is the RR table. Rounds are iterated until we either reach ROUNDMAX, or one of the score reaches SCOREMAX, where the highest score is chosen as the optimal value for prefetch offset D. This also concludes the end of a learning phase, after which all scores are zeroed and learning restarts.

The offset list is a selection of offsets rather a contiguous range from 1 to MAXOFFSET to prevent requiring a large score table and long learning phase. Offsets with small prime factors (≤ 5) are chosen. This assumes smaller offsets are more likely to be useful, and the property that if two offsets are in the list, so is their Least Common Multiple—useful for selecting a single offset when prefetching data for two interleaved strided streams. A BADSCORE threshold is the minimum score required for prefetching.

3 Triage prefetcher

Triage [10] is a temporal prefetcher aiming to keep all metadata on-chip, unlike STMS and (M)ISB. This greatly reduces memory traffic and latency as well as hardware complexity and power as there is no need to implement complex metadata caching and prefetching.

Triage relies on a dynamically-adjustable partition of the LLC to store its metadata through changing which ways of the set-associative cache are assigned to data and metadata (according to OPTgen's estimated optimal hit rate). The metadata is a simple PC-localized table holding address-correlated pairs. Since the table is on-chip, chaining is not required, saving it from redundancy and variable chain-length issues.

Triage relies on two key observations. Firstly, most metadata reuse is due to a few metadata entries. Secondly, even among frequently used metadata entries, many prefetch requests are redundant as they hit the cache. Thus, the Hawkeye replacement policy used captures long-term reuse and is trained positively only when the metadata yields a prefetch missing the cache, in-line with both of these considerations.

4 Discussion

4.1 Benchmark performance

BO and Triage are designed for regular and irregular memory access patterns respectively and perform well on benchmarks reflecting those characteristics, but both behave differently when this is not the case. When BO is tested on the irregular subset of the SPEC2006 benchmark, there is no performance regression compared to the baseline—in fact, there is an uplift in excess of 10% on sphinx3. However, when Triage is tested on the regular subset, there are numerous performance regressions from the baseline. The speedups from the prefetches issued by Triage were not enough to offset the loss in LLC space as prior temporal patterns are a poor, and importantly, space-inefficient proxy for predicting spatial memory access characteristics like in strided memory access.

4.2 LLC partitioning

The decision for Triage to share LLC space is peculiar. The Triage paper notes that metadata partition updates are not very common and partitioning the ways of the LLC does not provide particularly granular partitioning. There are diminishing speedups as metadata store size approaches 1MB, which is not too large. Moreover, the metadata in the LLC is not the only state stored by Triage. Additional metadata replacement state must also be stored to recompute the optimal metadata partition size. This means that the dynamic, shared nature of the cache provides little benefit over a well-sized, separate static cache.

Paritioning also leads Triage to store 16 metadata entries within each cache line. Thus, the tag for the lookup and prefetch target addresses are compressed to 10 bits with a lookup table to translate these to full addresses. The remaining 11 bits in the 32-bit metadata entry are used for the set identifier for the prefetch target. This design forces a specific cache configuration designed for a 2 MiB and 16-way set associativity cache with 64-byte cache lines. If we had an L3 cache with different associativity or line size, this would force the metadata entries to become misaligned with the cache [11] increasing access latency, or force us to use fewer bits for the tag leading to aliasing issues.

The performance implications regarding regular memory access patterns, Triage already storing additional state, and cache design flexibility issues outweigh the little benefit provided by the dynamic partitioning scheme. Triage may benefit from a separate metadata cache for practical implementation.

4.3 BO+TriageDynamic hybrid scheme

The Triage paper states that a BO+Triage hybrid scheme performs well, citing a 1–5% uplift on the irregular subset over using TriageDynamic alone. Surprisingly, the hybrid scheme performs well on regular programs too, with performance matching BO-only. This makes a BO+TriageDynamic scheme compelling, but a future comparison with BO+MISB would be necessary, especially in higher core-count scenarios, where Triage provides a 1.9% speedup over MISB due to MISB's larger metadata traffic overhead. Additionally, both BO and Triage offer low off-chip traffic overhead, in theory making a BO+TriageDynamic scheme more attractive than BO+MISB.

4.4 Triage and MISB energy-efficiency

One must be dubious regarding the claim that Triage is more energy-efficient than MISB. The energy-efficiency analysis only takes into account the energy required for the metadata access, and not the fact the MISB provides a significantly larger speedup ($\sim 8\%$ on average) (it is unclear if this is due to better accuracy or coverage), so less energy is spent servicing data misses. Additionally, the energy efficiency estimates have uncertainty $\times 4$ the mean value, which does not lend well to credibility.

4.5 BO timeliness-coverage trade-off

BO aims to maximize prefetch timeliness but, Benchmark 462 shows that this may not always be ideal. We see that smaller offsets lead to the best performance here as although the prefetches may be late, here they offer a better trade-off between reduced timeliness and greater prefetch coverage. This could possibly be addressed by the addition of a penalty term to the scores of offsets bigger than some fraction of the current offset, scaled proportionally with the miss rate.

5 Conclusions

BO performs well with good accuracy and coverage for regular workloads, with minimal off-chip traffic. In contrast, Triage needs work regarding flexibility for different cache configurations, and it's power-efficiency claims need to be verified. BO+TriageDynamic work well as a hybrid scheme with complementary characteristics, while minimizing off-chip traffic, but a comparison with BO+MISB would be beneficial. Triage seems well-positioned for low-power and embedded systems where an off-chip scheme like MISB is infeasible due to energy-efficiency and latency concerns.

References

[1] N.P. Jouppi. "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers". In: *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture.* 1990, pp. 364–373. DOI: `10.1109/ISCA.1990.134547`.

[2] Erik Hagersten. "Toward scalable cache only memory architectures". PhD thesis. Royal Institute of Technology, 1992.

[3] Seth H Pugsley et al. "Sandbox Prefetching: Safe run-time evaluation of aggressive prefetchers". In: *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA).* 2014, pp. 626–637. DOI: `10.1109/HPCA.2014.6835971`.

[4] Doug Joseph and Dirk Grunwald. "Prefetching using Markov predictors". In: *SIGARCH Comput. Archit. News* 25.2 (May 1997), pp. 252–263. ISSN: 0163-5964. DOI: `10.1145/384286.264207`. URL: `https://doi.org/10.1145/384286.264207`.

[5] K.J. Nesbit and J.E. Smith. "Data Cache Prefetching Using a Global History Buffer". In: *10th International Symposium on High Performance Computer Architecture (HPCA'04).* 2004, pp. 96–96. DOI: `10.1109/HPCA.2004.10030`.

[6] Thomas F. Wenisch et al. "Practical off-chip meta-data for temporal memory streaming". In: *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 2009, pp. 79–90. DOI: 10.1109/HPCA.2009.4798239.

[7] Akanksha Jain and Calvin Lin. "Linearizing irregular memory accesses for improved correlated prefetching". In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-46. Davis, California: Association for Computing Machinery, 2013, pp. 247–259. ISBN: 9781450326384. DOI: 10.1145/2540708.2540730. URL: https://doi.org/10.1145/2540708.2540730.

[8] Hao Wu et al. "Efficient Metadata Management for Irregular Data Prefetching". In: *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 2019, pp. 1–13.

[9] Pierre Michaud. "Best-offset hardware prefetching". In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2016, pp. 469–480. DOI: 10.1109/HPCA.2016.7446087.

[10] Hao Wu et al. "Temporal Prefetching Without the Off-Chip Metadata". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019. ISBN: 9781450369381. DOI: 10.1145/3352460.3358300. URL: https://doi.org/10.1145/3352460.3358300.

[11] Sam Ainsworth and Lev Mukhanov. "Triangel: A High-Performance, Accurate, Timely On-Chip Temporal Prefetcher". In: *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. Vol. 19. IEEE, June 2024, pp. 1202–1216. DOI: 10.1109/isca59077.2024.00090. URL: http://dx.doi.org/10.1109/ISCA59077.2024.00090.