Pangloss: a novel Markov chain prefetcher

Philippos Papaphilippou, Paul H. J. Kelly, Wayne Luk
Department of Computing, Imperial College London, UK
{pp616, p.kelly, w.luk}@imperial.ac.uk

ABSTRACT

We present Pangloss, an efficient high-performance data prefetcher that approximates a Markov chain on delta transitions. With a limited information scope and space/logic complexity, it is able to reconstruct a variety of both simple and complex access patterns. This is achieved by a highly-efficient representation of the Markov chain to provide accurate values for transition probabilities. In addition, we have added a mechanism to reconstruct delta transitions originally obfuscated by the out-of-order execution or page transitions, such as when streaming data from multiple sources. Our single-level (L2) prefetcher achieves a geometric speedup of 1.7% and 3.2% over selected state-of-the-art baselines (KPCP and BOP). When combined with an equivalent for the L1 cache (L1 & L2), the speedups rise to 6.8% and 8.4%, and 40.4% over non-prefetch. In the multi-core evaluation, there seems to be a considerable performance improvement as well.

1. INTRODUCTION

Markov models have been used extensively in prior research for prefetching purposes, by estimating and utilising address transition probabilities for subsequent accesses. Distance prefetching is a generalisation of the common Markov model prefetchers [1], that uses deltas instead of addresses to build more general models (originally for TLBs [2]). In such cases, the acquired knowledge is applied to other addresses, including previously unseen. A faithful implementation of a Markov-chain for delta transitions would be a directed graph, with deltas as states/nodes and probabilities as weighted transitions/arcs.

A delta is the difference between two consecutive addresses. As we can see from the simplified example below, given an initial address and a stream of deltas, the address stream can be reconstructed.

```
Address:      1   4   2   7   8   9
Delta:          3  -2   5   1   1
```

In real systems, we have page limits, which constrain the reach of deltas. Both the virtual and physical memory space are divided into pages. For security and integrity reasons, the page allocation is usually not considered to be sequential. The page contents are indexed by the remaining least significant address bits and stay unaltered between translations. When prefetching, any predicted addresses that fall outside the page limits are discarded.

One challenge in distance prefetching is that many pages might be accessed in interleaving patterns and thus obfuscating the produced delta stream. The delta stream, that would otherwise be used in its entirety to update the Markov (or alternative) model, has invalidated deltas, from comparing addresses from different pages, such as when accessing data from many sources iteratively. Our general idea is to track *deltas per page instead of globally, but build an accurate Markov model for global decisions*.

The main contribution of this paper is the *introduction of an efficient, more-faithful representation of a Markov chain*, that provides a metric of delta-transition probability. This results in increased accuracy for exploiting more complex access patterns.

1.1 Motivation/ Preliminary Experiment

We overview the real-world complexity of such delta-transition Markov chains, to gain an insight into related challenges and optimisations. Using a simple experiment, we monitor all the delta transitions using the competition's evaluation framework (see 3.1). We implement a dummy cache prefetcher, where all occurrences of valid delta transitions (from addresses falling in the same page) are counted inside an adjacency matrix.

Figure 1 on the left, shows a visualisation of the frequencies for the (L2) delta transitions in a run of 607.cactuBSSN_s-3477B. On the right, we can see the produced Markov chain (LLC), with the width of the arrows representing the probability of transition. The sum of the width of all arcs going out of a node sum to 1 (some transitions with low probability are excluded).

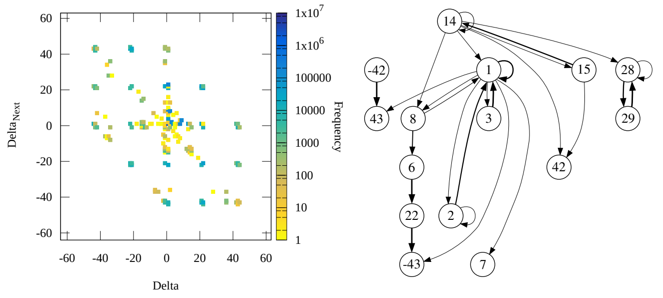

Figure 1: Two visualisations for cactuBSSN

Figure 2 shows the respective visualisation of the (L2) adjacency matrx for all benchmark traces. There are some interesting observations: 1) The matrices are sparse, but 2) not as sparse to justify only supporting regular strides (such as $(1, 1)$, i.e. the model of the next line/sequential prefetcher).

3) Instead of only supporting a limited coverage of deltas [3], it seems worthwhile to be unbiased, including negative deltas [4] as well. 4) Matrices that are too sparse or empty (mcf_s1536B), indicate simple patterns or invalidated deltas (see 2.2).

Some additional observations: 1) The diagonal lines are most likely from cases of transitions from seemingly-random accesses inside a page, while a regular stride is performed. For example, in a streaming operation with a delta transition (δ, δ), any secondary accesses would yield transitions of the form $(\delta', -\delta' + \delta)$, where δ is the stride and δ' is a new temporary delta. 2) This also explains any vertical and horizontal lines near the axes, as such transitions are preceded and succeeded by the points (δ, δ') and $(-\delta' + \delta, \delta)$ respectively. 3) The reason that there seem to be 'inner' bounds that make the overall shape seem like a hexagon is because such outliers would imply two consecutive deltas pointing outside the page margins.

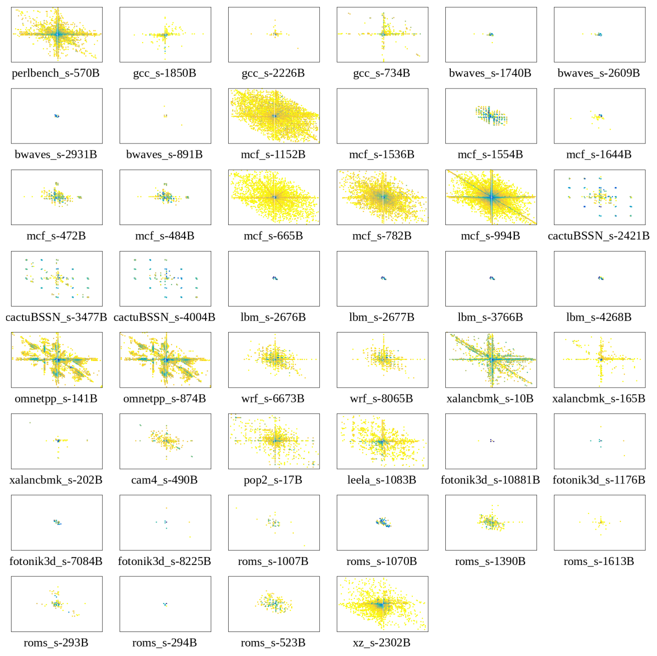

Figure 2: Adjacency matrix visualisations for delta-transition frequencies

2. PROPOSED SOLUTION

2.1 Delta cache: A novel Markov chain representation

The main structure is an efficient representation of Markov chain for distance prefetching. One challenge in implementing a Markov chain in hardware is that a naive accurate implementation would require N*N positions, where N is the number of states, for maintaining the transition probabilities in an adjacency matrix. For this reason and the fact that it usually has a high sparsity (see 1.1), existing implementations approximate it with associative structures. Those associative structures (such as a fully-associative or set-associative cache) usually employ a Least Recently Used (LRU) [2] (or approximations [5]), or a First-In First-Out (FIFO) replacement policy, which are both prone to losing track of important transitions due to thrashing. Moreover, with the information kept by LRU and FIFO, there is no real metric of frequency/probabilities, which is what Markov-chains are originally supposed to provide.

In figure 3, we present our Markov chain representation for the level 2 prefetcher. It is a set-associative cache, providing delta transitions based on the current delta. It is indexed by the current delta, and the blocks in each set represent the most frequent immediately-next deltas. Assuming that we observe line-addresses (L2 prefetcher in the framework), there are 64 possible positions (offsets) in a 4KB page. This totals in a delta size of 7 bits, representing values from -64 (excluding, since it points to a different page) to +63.

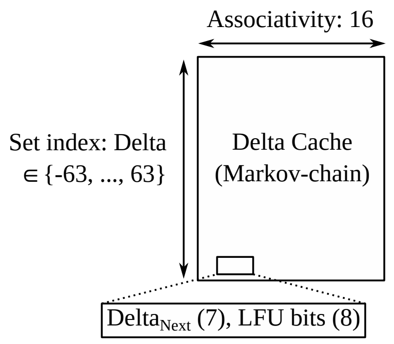

Figure 3: Delta Cache (in L2 prefetcher)

With respect to the replacement, we use an approach similar to the Least Frequently Used (LFU) replacement policy, with the goal to keep the correct transition probabilities, but also give the opportunity of phased-out prominent deltas to be evicted quickly (slight resemblance in [6]). Each block in a set contains the next delta alongside a counter (LFU bits). This counter is incremented each time there is a hit. When there is an overflow, all blocks in the respective set have their counter values halved. In this way, we retain almost the same count proportions, with the reduced accuracy favouring higher values. In order to find the transition probability, we divide the value with the sum of all values in the set, which can also be calculated progressively. Keeping the proportions is important in both replacement and prioritising prefetches.

2.2 Page cache: Reconstructing obfuscated delta transitions

In this subsection we describe a mechanism designed to help reconstruct delta transitions obfuscated by 'unexpected', sometimes temporary, page transitions. There are similar approaches in related work [5], [7]. This does not modify the decision part of Pangloss, as it only helps to increase the number of valid observations for updating the Delta cache.

In a naive Markov-chain distance prefetcher, the deltas would be calculated by comparing with the latest address. Instead, we modify the algorithm to keep the last address/delta per page, thus maximising the probability of yielding a valid transition (i.e. inside the page limits).

In figure 4, we can see an overview of the page cache. The page cache is set-associative, indexed by the page address. Each block has 4 fields:

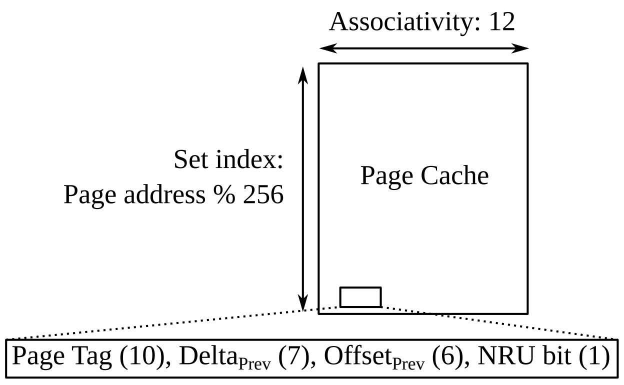
Associativity: 12
Set index:
Page address % 256
Page Cache
Page Tag (10), $Delta_{Prev}$ (7), $Offset_{Prev}$ (6), NRU bit (1)

Figure 4: Page Cache (in L2 prefetcher)

- Page tag: to identify the page and distinguish from others in the set. We found that restricting it to only 10 bits had a marginal impact on performance, despite the small probability of false positives.
- $\text{Delta}_{\text{Prev}}$: the previous delta, with which the transition is found. In the L2 prefetcher (cache line address granularity), the deltas are 7 bits long. On insertion, the value -64 is used as an initial value, to indicate that there was no previous delta, since it always points to a different page.
- $\text{Offset}_{\text{Prev}}$: the previous address offset. This is used to calculate the current delta based on the new address. It consumes 6 bits (values from 0 to 63).
- NRU bit: This bit is used for approximating the LRU replacement policy with 1 bit [5], by always evicting the Not-Recently Used (NRU) block.

2.3 Markov-chain traversal

Given a prefetch degree and the current delta, the prefetcher must decide how to traverse the approximated Markov chain, to provide the most profitable next deltas. Since the degree can allow paths of length > 1, accurately evaluating the probabilities of all possible paths in the graph becomes expensive. This is because it would require $degree - 1$ matrix multiplications involving the adjacency matrix.

We propose a simple heuristic to predict the most likely next deltas: recursively, prefetch the addresses occurring from the child deltas with probability $> 1/3$ and proceed with the highest probable delta for the next iteration, until we count as many prefetches as the prefetch degree.

Note that if a resulting prefetch address falls out of the current page, it is discarded, but the path remains valid. This is done to preserve subsequent accesses to the same page, even if the same pattern started from other offsets during training.

3. EVALUATION

3.1 Framework and Baseline configurations

We are using the ChampSim micro-architectural simulator for the competition's baseline configurations for 1-core and 4-core simulations. The warmup phase takes 50M instructions and the simulation runs for another 200M. We are using the provided selection of SPEC CPU2017 benchmark traces (with over 1 Misses per K instructions (MPKI)). All runs use the same branch predictor (hashed perceptron) and cache replacement algorithm (LRU).

We compare the performance of our prefetcher to two state-of-the art prefetchers, the Best-Offset Prefetcher [3] (BOP) and the prefetcher from KPC [7] (KPCP). The first was the winner of the previous Data Prefetching Championship (DPC2) and was ported to work as an L2 prefetcher in the current version of ChampSim. The latter is already included in the ChampSim repository and represents the prefetcher part of KPC.

Our final multi-level prefetcher, *'Proposal L1&L2'*, includes two versions of the same design, one for L1 and one for L2. In order to be fair with the related work, we also report results for the single-level prefetcher, *'Proposal L2'*, which is the L2 part in standalone.

The L1 part had some additional changes to benefit from the fact that the framework allows byte-address granularity for L1. We observed a 64-bit alignment in L1, which resulted in 512 possible offsets in a 4KB page. This increases the number of sets in delta cache to 1024, the offset size to 9 bits and the delta size to 10 bits. The LFU bits are reduced to 7.

3.2 Single-program

Figure 5 illustrates the single-program evaluation. Our solution achieves a geometric speedup of 6.8% and 8.4% over KPCP and BOP respectively. The geometric speedup over non-prefetch is 40.4%. The respective geometric speedups for the single-level version are 1.7%, 3.2% and 33.5%.

3.3 Multi-program

In order to produce representative program mixes, we divide the 46 traces in two groups, the 'low' and 'high', for those yielded a speedup of 1.3 and below (last 21 from fig. 5) and those above 1.3 respectively, using 'Proposal L1 & L2'. Then, for each of the 5 group combinations (l-l-l-l, l-l-l-h, ...) we produce 8 random mixes, totalling 40 mixes.

Figure 6 shows the weighted IPC speedups of 'Proposal L1 & L2' and KPCP over the single-core runs with non-prefetch (i.e. $\sum(IPC_i/IPC_{alone\_i})$ [8]). It is clear that the multi-level prefetcher performs generally better than KPCP, while the single-level version (not shown) is roughly in-between.

3.4 Resources

Space budget.

In table 1, we present the space requirements of our multi-level prefetcher. The total number of bits (59.4 KB) is below the space budget of the competition (64 KB) for the single-core configuration. Since we have not included an LLC prefetcher, the space requirements for the multi-core configuration is multiplied by 4, which is also under the competition's space budget (4×64 KB). In the single-level case, the L2 prefetcher only consumes 13.1 KB.

Logic complexity.

Pangloss is H/W-friendly. The low associativity in the Page Cache and Delta Cache ensures that there will be few

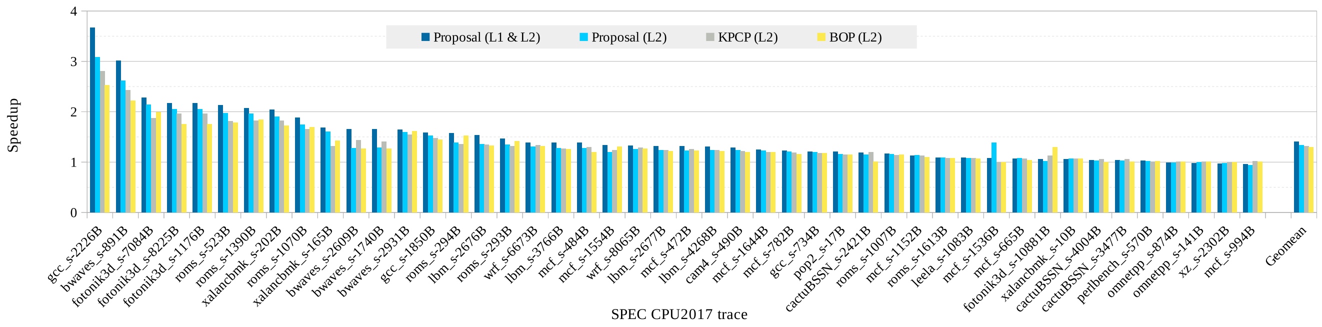

Figure 5: Single-program evaluation: speedups over non-prefetch

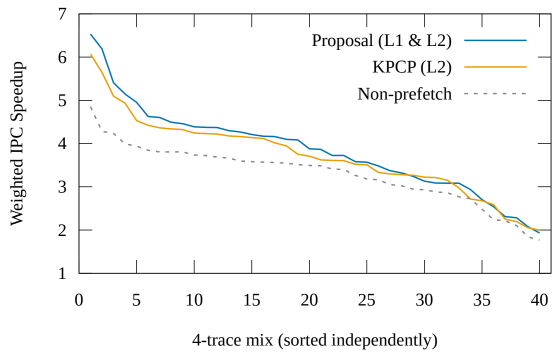

Figure 6: Multi-program evaluation

	Description (bits)	(KB)
L1D:		
Delta cache	1024 *sets* $\times$ 16 *ways* $\times$ $(10+7)$	34.8
Page cache	256 *sets* $\times$ 12 *ways* $\times$ $(10+10+9+1)$	11.5
L2:		
Delta cache	128 *sets* $\times$ 16 *ways* $\times$ $(7+8)$	3.8
Page cache	256 *sets* $\times$ 12 *ways* $\times$ $(10+7+6+1)$	9.2
LLC:	*None*	0.0
Total		59.4

Table 1: Single-core configuration budget

simultaneous comparisons of few bits. This allows keeping more information in a concise space. The Markov traversal heuristic that selects probabilities above 1/3, implies that up to 2 child deltas will be selected. Thus, one extra comparison is enough to point to the next layer. For a medium prefetch degree, the recursive lookup [5] remains relatively efficient, although allowing a delay could also prove beneficial for timeliness [3].

According to the use case, many parameters that impact the space/logic complexity, can be explored further.

4. FUTURE WORK AND CONCLUSION

All prediction mechanisms have some weaknesses. When observing short repeating delta patterns, such as 1, 1, 2, 1, 3, 1, 1, 2, 1, 3, ..., the transitions (1, 1), (1, 2) and (1, 3) would yield an equal probability. This in combination with other factors, like a low prefetch degree, could have a performance overhead. This does not happen with multiple-delta histories [5]. However, multiple-delta history matching could be negatively affected by some memory hierarchy effects that reorder or even hide deltas. Systematically evaluating the probability and overhead of pattern conflicts in Pangloss would be desirable. Alternatively, we could evaluate the presence of multiple-delta states in the Markov chain. One mechanism that differs from random walks on a Markov-chain is the traversal heuristic, which can be explored further.

In this paper, we introduce a H/W-friendly prefetcher with a more-faithful representation of a Markov chain, resulting in a higher accuracy and performance.

Acknowledgement

This research was sponsored by dunnhumby. The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1) is gratefully acknowledged.

References

[1] 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)*, IEEE, 2004, pp. 96–96.

[2] G. B. Kandiraju and A. Sivasubramaniam, "Going the distance for TLB prefetching: an application-driven study," in *Proceedings 29th Annual International Symposium on Computer Architecture*, IEEE, 2002, pp. 195–206.

[3] P. Michaud, "A best-offset prefetcher," in *2nd Data Prefetching Championship*, 2015.

[4] M. Grannaes, M. Jahre, and L. Natvig, "Storage efficient hardware prefetching using delta-correlating prediction tables," *Journal of Instruction-Level Parallelism*, vol. 13, pp. 1–16, 2011.

[5] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2015, pp. 141–152.

[6] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proceedings 28th annual international symposium on computer architecture*, IEEE, 2001, pp. 240–251.

[7] J. Kim, E. Teran, P. V. Gratz, D. A. Jiménez, S. H. Pugsley, and C. Wilkerson, "Kill the program counter: Reconstructing program behavior in the processor cache hierarchy," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 737–749, 2017.

[8] A. Jaleel, H. H. Najaf-Abadi, S. Subramaniam, S. C. Steely, and J. Emer, "CRUISE: cache replacement and utility-aware scheduling," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 40, 2012, pp. 249–260.