Perceptron-Based Prefetch Filtering in Gem5

CS752 Project

Eric Brandt
University of Wisconsin, Madison
Email: elbrandt@wisc.edu

Sangeetha Grama Srinivisan
University of Wisconsin, Madison
Email: sgsrinivasa2@wisc.edu

I. Introduction

Prefetching has been used as an effective strategy to improve processor performance. Many prefetching strategies are proposed based on spatial or temporal access patterns observed in the program. Machine learning algorithms ranging from simple perceptrons to complex LSTM (Long-Short-Term-Memory) models are leveraged to design and optimise prefetching strategies. One such design uses a perceptron-based prefetch filter [1] to enhance Signature Path Prefetching [2], a lookahead prefetcher. In this project we aim to implement this perceptron-based prefetch filter [1] in GEM5 [3] simulator and evaluate the performance implications of the filtered prefetches on a wide range of workloads.

II. Relevance

Memory accesses span over multiple cycles. The rate of improvement in memory access speeds compared to the rate of improvement in processor speeds indicates that there is a wide gap known as the Memory Wall [4]. This significant difference in speeds can be mitigated by prefetching memory into the cache. The prefetching strategies are continuously optimised to improve accuracy of prefetching while also maintaining a good coverage, i.e, the prefetcher should be able to eliminate cache misses and all the prefetched memory should be used by the processor. While the former requirement focuses on improving processor performance, the latter requirement helps conserve memory bandwidth and the space availability in the cache. Machine learning algorithms have been used to improve or enhance prefetching, like, offline parameter optimisation for baseline prefetchers or on-line parameter tuning to provide feedback to the prefetcher based on the result of the prefetch, such as the perceptron-based prefetch filter [1]. Implementing this filter in GEM5 will help in getting a detailed analysis of performance implications on a wide range of program workloads and architectures. Further, the prefetch filter can be used as a standalone unit with any baseline prefetcher, giving researchers the opportunity to explore the use of this filter with their designs.

III. Evaluation Methods

Implementation success will be measured both qualitatively and quantitatively. The *qualitative* evaluation will verify that the Perceptron Prefetcher algorithm is implemented *correctly* (e.g, behaves in a manner consistent with the proposed design, and does not contain bugs). This will be achieved through a combination of custom log files generated during runtime that are inspected post-mortem, real-time inspection within the debugger itself, and carefully crafted sample workloads designed to elicit certain behaviors of the prefetcher to exercise all possible code paths. The *quantitative* evaluation will involve benchmarks run on our prefetcher implementation as well as existing prefetcher strategies for comaprison purposes. For these benchmarks, we hope to use industry standard benchmarks such as SPEC CPU 2017 used by [1] and SPEC2006 suite as used by [5]. Assuming we are able to run these benchmarks in Gem5 and use the same warmup and measured code sections as those authors did, we may be able to very roughly compare performance numbers with their stated results.

IV. References

Pugsley et.al present a prefetching technique known as Sandbox Prefetching [5]. The design first uses global pattern confirmation of the cache accesses using a bloom filter and once the pattern is confirmed, can immediately start prefetching addresses using the confirmed pattern. The sandbox is used to prevent unwanted memory accesses and conserve memory bandwidth and pages available in the cache. The sandbox is used to match the program's cache access patterns with one of the candidate patterns (each with a different offset). The candidate pattern for which most of the memory accesses match is treated as the confirmed pattern and prefetches are issued using this pattern. While the design reduces the prefetch aggressiveness and uses spatial locality, it does not take into account the program order for prefetching memory.

Signature Path Prefetching [2] (SPP) is a type of a lookahead prefetcher where program order is used. SPP is designed to not only predict the memory location accessed but also the order of the access within a given page. The method predicts the future memory access patterns without requiring inputs from the PC or the branch predictor. The design has 3 components - Signature table, Pattern table and the Prefetch engine. The signature table stores the physical page numbers, the previously accessed block in that page along with a hash of the previous access patterns to that page, compressed as a signature. The pattern table, indexed using this signature, is used to store future stride access patterns and the confidence for each of the prospective prefetch patterns. The pattern table is indexed by the signature, unlike the signature table which is indexed using the physical page number since multiple pages

can have the same memory access pattern. The prefetch engine issues the prefetch pattern which has a confidence greater than a threshold (known as filtering) and also performs lookahead by using a lookahead signature. This lookahead signature is generated from the old signature and used to access the pattern table again to look further ahead and find more prefetch candidates, repeating the prefetching process. Though the design is simple and performs better than spatial prefetchers like AMPM (Access Map Pattern Matching prefetcher), the effectiveness of the design is dependent on the threshold used to determine if a prefetch pattern with a certain confidence value can be issued.

Eshan et. al propose a method of using a perceptron-based prefetching filter [1] to control the aggressiveness of prefetchers. The method is implemented using SPP [2] as the baseline prefetcher. Since in SPP, the threshold for confidence is used to determine whether a prefetch needs to be issued, using a perceptron to make this decision helps in 2 aspects. First, it provides a generalisation of the process used in SPP to decide to issue a prefetch and secondly, it helps to decide which level of memory to prefetch into based on the output of the perceptron. The SPP method in [2] reserved a part of the L2 cache for prefetching while the perceptron-based prefetching filter allows prefetching to be done either to L2 or the next cache level. The filter acts as a check to control the aggressiveness of the prefetcher by maintaining 3 tables - weight table, prefetch table and a reject table. The weight table maintains the weight for each feature (input to the perceptron) and is used to compute the weighted sum of the features - the confidence level of the prefetch, which is then compared against a threshold. The prefetches that have a confidence level higher than the threshold are stored in the prefetch table, while those that do not are stored in the reject table, both of which are used to tune the weights in the perceptron. Training is done when an address from the memory request is found in either of the tables, indicating a correct or a misprediction following which the weights are updated accordingly. This design uses the perceptron-based prefetch filter as a stand-alone module that can be used with any type of baseline prefetcher, and thus, can enhance any existing prefetcher.

V. Project Plan

Our plan, broadly, is to implement and evaluate a Perceptron-based Cache Prefetch strategy based on the design of [1] within the Gem5 simulator. More specifically, we can break this plan into a series of tasks that will lead us to our goal. Because we cannot predict all of the obstacles we may encounter in the implementation, we will consider each successfully completed task as a worthwhile effort towards the full plan. To this end, we have developed the following list of tasks, and planned their completion around the deadlines under which we are working. We divide our tasks into milestones:

- **Milestone 1**: Tasks that we plan to complete before the progress report deadline:
  1) Survey the prefetch mechanisms that exist within Gem5 today. We will gain understanding of the implementation of cache prefetch mechanisms by studying of the files in `src/mem/cache/prefetch`, with careful attention paid to `signature_path_v2.cc`, and the inheritance hierarchy of classes within that folder.
  2) Continue a more comprehensive literature review of current cache prefetch strategies to better understand the lineage of the Preceptron Prefetch design. Particular attention will be given to studying the design of Signature Path Prefetching [2], as this is the groundwork on which Perceptron-Based Prefetch Filtering is built.
  3) Develop/curate a set of benchmarks and statistics on which to evaluate various prefetch strategies. This includes development of scripts to run the benchmarks in Gem5 as well as analyze the generated statistics.
  4) Create a new trivial-capability C++ prefetch object within Gem5, likely derived from `Prefetcher::Base` or `Prefetcher::Queued`, with associated Python wrappers, exposing settings that allow the prefetcher to be tuned via Gem5 Python system definition scripts.
  5) Test the framework of the new trivial prefetcher by ensuring our benchmarks can be executed on simulation systems defined to use this prefetcher within the memory heirarchy.
  6) Prepare and submit a mid-project progress report by the November 13 deadline.
- **Milestone 2**: Tasks that we plan to complete before the Lightning Talk deadline
  1) Implement the Perceptron Prefetcher described in [1] by modifying the trivial prefetcher created in Milestone 1.
  2) Develop test programs for execution in Gem5 that are both simple, and specifically engineered to test the correctness of the implementation of the Perceptron Prefetcher.
  3) Benchmark our implementation to test both for correctness of implementation, and for performance using the scripts developed in Milestone 1.
  4) Compare performance of key statistics with other prefetch designs that already exist in Gem5.
  5) Prepare a 'Lightning-talk' to evangelize the work and results involved in this project by December 2.
  6) Write a 'research paper'-quality report of our work and the results we measure by December 15.
- **Stretch Milestones**: Tasks that we hope to complete, but may be too large to fully complete within the time constraints of the project.
  1) Increase the number of 'tunable' settings exposed by our Perceptron Prefetcher, to allow for more experimentation.

2) Perform a more exhaustive evaluation of the effect of the available settings of our prefetcher with various different cache configurations in simulated systems.
3) Test different workloads to try to discern the particular conditions and code characteristics that favor the Perceptron Prefetcher as compared to other strategies. It may be interesting to run different prefetch strategies on physics-based-animation/simulation workloads, to compare prefetch performance in generalized benchmarks (e.g. SPEC CPU) with the numerical, highly-regular access patterns of this type of domain-specific code.
4) Evaluate our implemented prefetch filter on prefetch strategies other than Signature Prefetch, to ascertain the transferability of a prefetch filter to other prefetch strategies.
5) Propose ways to further improve the design of the Perceptron Prefetcher, or at least define roadmaps for future research.

REFERENCES

[1] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13. [Online]. Available: https://doi.org/10.1145/3307650.3322207

[2] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[3] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020.

[4] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, Mar. 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[5] S. H. Pugsley, Z. Chishti, C. Wilkerson, P. Chuang, R. L. Scott, A. Jaleel, S. Lu, K. Chow, and R. Balasubramonian, "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.

CS/ECE 752 Project Checklists

Project Proposal

Please use this checklist to confirm the completeness of your proposal. This checklist should be submitted with your proposal.

	Proposal Component	Feedback
✔	**Project Team List** (1 point). Proposal includes names and email addresses of each of the project team members.	
✔	**Topic Description** (2 points). Proposal includes at least a paragraph explanation of the area of focus and the problem of interest.	
✔	**Topic Relevance** (2 points). Proposal includes at least a paragraph explanation of the significance of the topic for CS 758.	
✔	**Evaluation Methods** (1 point). Proposal includes at least a paragraph description/list of methods that will be used to evaluate solutions to the problem described above.	
✔	**References** (3 points). Proposal includes citations in IEEE or ACM style for at least three relevant papers that you have already read, along with a description of how they are relevant and a plan for addressing additional related work.	
✔	**Project plan** (1 point). Proposal includes a list of tasks that the team needs to complete in order to complete the project. For each task, the proposal offers a brief description of how and when the team plans to complete the tasks. While the plan is an estimate, it will use it to inform the Project Progress Report that is due later.	

Comments: