Madison Bradley and Sarah Thomson
Department of Computing Science
University of Alberta
Edmonton, Alberta
{mlbradle, smt}@ualberta.ca

I. Overview

Processors require data stored in memory while executing tasks. While the processor waits for data to be retrieved from main memory for a computation, valuable computation time is lost. Alternatively, if the required data is stored in an on-chip cache, the data is orders of magnitude faster to retrieve. This concept motivates the use of a data *prefetcher*, a method of transferring data from main memory to the cache system in advance to hide memory access latency for a future memory operation.

Prefetcher design is complex; it can be difficult to predict where the required data should come from since the size of the address space is massive. Some memory accesses occur in obvious patterns, such as iterating over every element in array. Other accesses, however, are more irregular and difficult to predict. The performance of a prefetching mechanism is measured by three main metrics: **accuracy**, **coverage**, and **timeliness**, where accuracy and coverage are often at odds with each other.

II. Relevant work and Motivation

The architecture of the proposed project is built upon the perceptron-based prefetch filter [1], intending to include various insights obtained from static analysis.

Perceptrons [2] are a very simple neural network implementation connecting several input units (features) by weighted edges to an output unit. The perceptron allows for multiple features and their weights to be considered in a binary prediction while remaining small enough to be implemented in hardware.

A perceptron-based prefetch filter [1] uses perceptrons as a predictive filter for existing hardware prefetchers. The filter is trained to predict whether a prefetch is going to be useful or not, to determine whether a given prefetch should actually be fetched. This method increases both coverage and accuracy of the prefetcher, as the actual prefetching mechanism does not have to balance coverage against accuracy - the filter manages accuracy entirely on it's own, while the prefetching mechanism can focus on maximizing coverage.

The use of a variety of features surrounding the prefetch context such as speculation depth and page address can assist in predicting whether or not a given prefetch would be valuable. It is important to note that the perceptron-based filter in [1] is not a prefetching mechanism on it's own: it filters on top of a state-of-the-art prefetching mechanism. For example, [1] uses the Signature Path Prefetcher (SPP) [3], which tracks patterns across page boundaries using path confidence-based prefetching.

The model's features are often obtained in the hardware - no semantic information is preserved for the prefetch filter to use. However, information obtained from static compiler pass can be used to gain valuable semantic information. For example, in the context of prefetching, a static pass could be able to find loop information such as trip count. Typical implementations of prefetching done at compiler time with statically obtained information usually focus on inserting software prefetch instructions (such as shown in [4]). However, software prefetching suffers from timeliness issues and additional overhead compared to hardware prefetching.

Thus, communicating static analysis information to hardware prefetching mechanisms may allow for semantic information to be preserved while avoiding the pitfalls of software prefetching. One method of communicating static analysis results to the hardware can be found in [5], which introduces a new set of load opcodes to the AArch64 ISA to communicate statically-obtained memory dependence information. However, ISA modifications are difficult to justify, and often cause portability issues between architectures. An alternate solution may be found by storing the information in free registers, such as architecture-specific special-purpose registers, or general-purpose registers that have been reserved by the compiler for use prior to the register-allocation stage.

III. Objectives and Methodology

We propose an implementation of perceptron-based prefetch filtering that considers static analysis data, obtained at compile-time. Including static analysis results may prove useful in training the perceptron predictors, possibly allowing for higher accuracy using semantic information if the information is useful to determining prefetch outcome. We propose separating this idea into four components:

1) An implementation of the Signature Path Prefetcher (SPP) [3] in Gem5
2) An implementation of the perceptron-based prefetch filter [1] on top of the SPP in Gem5
3) A LLVM transformation pass that collects relevant (existing) static analysis results and inserts instructions to

store parts of these results to architecture-specific special purpose registers or general-purpose registers that have been reserved in the compiler pass

4) A modification of Gem5 to interpret the results of the LLVM transformation pass as saved to the registers, providing the information as features to the perceptron prefetch filter

After implementing these four components, we will use the evaluation mechanism as outlined in section 5.5 of [1] to determine which features included from the LLVM static analyses are most correlated with the prefetching outcome. This method investigates the perceptron weights to determine these correlations. Using this evaluation mechanism, we will be able to compare the usefulness of statically analyzed features with existing hardware-obtained features as found in [1].

References

[1] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-Based Prefetch Filtering," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2019, pp. 1–13. Accessed: Oct. 04, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/8980306

[2] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, Jan. 2001, pp. 197–206. doi: 10.1109/HPCA.2001.903263.

[3] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12. doi: 10.1109/MICRO.2016.7783763.

[4] C.-K. Luk and T. C. Mowry, "Compiler-based prefetching for recursive data structures," in *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, in ASPLOS VII. New York, NY, USA: Association for Computing Machinery, Sep. 1996, pp. 222–233. doi: 10.1145/237090.237190.

[5] L. Panayi, R. Gandhi, J. Whittaker, V. Chouliaras, M. Berger, and P. Kelly, "Improving Memory Dependence Prediction with Static Analysis." Accessed: Sep. 26, 2024. [Online]. Available: http://arxiv.org/abs/2403.08056