CS5222 Advanced Computer Architecture
Project 3: Prefetcher Implementation

Sharadh Rajaraman
A0189906L

Sunday 24th April 2022

1 Introduction

1.1 Data Prefetching

Prefetching is to data, what branch prediction is to text: both aim to alleviate stalled cycles as a result of fetching text or data lines from main memory, by guessing future accesses given a past history. In this project, the global history buffer with program counter (PC) localisation and delta correlation (GHB PC/DC) prefetcher has been implemented.

1.2 GHB PC/DC Overview

Nesbit and Smith provide a detailed overview of GHB PC/DC prefetching[1]; a very quick summary is detailed as follows.

The prefetcher consists of two data structures: an Index Table (IT), which is a hash-table mapping some localisation key to pointers into a Global History Buffer (GHB), which in turn is a fixed-size first-in, first-out (FIFO) queue (also called a circular or ring buffer), containing pairs of miss addresses, and pointers to other elements in the GHB. For GHB PC/DC, the indexing key is the PC.

1.3 GHB PC/DC Operation

The GHB PC/DC prefetcher's algorithm is detailed (albeit rather sparsely) in *A Primer on Hardware Prefetching*[2]. A detailed algorithm is listed below, including edge cases handled by neither Nesbit and Smith nor Falsafi and Wenisch.

1. Upon a cache *miss*, the index table is indexed with the PC.
   (a) If there is a hit, then go to 2.
   (b) If there is no hit, or the location dereferenced by the corresponding pointer has been evicted from the GHB, an entry is created, and no prefetching is performed.
2. The miss address is saved onto the stack, and the corresponding pointer is inserted into the head of the GHB.
3. Given the previous pointer from the IT hit, the 'next' pointer of the above new miss address is set to the previous pointer, thus building a linked list.
4. The pointer is dereferenced to the GHB, and the linked list is followed, adding every miss address found, to an array of addresses accessed by the same PC.
5. Given the nature of the linked list, this array is in reverse-chronological order, with the latest miss address at the beginning of the array, and the earliest miss address at the end. Therefore, this array is reversed.
6. Using this reversed array, the *last three* elements in said array are used to build the *latest* delta pair, i.e. the two differences between said last three elements.
7. A delta stream is built given the differences between each element in the array, too.
8. The delta stream is searched to find the *last* occurrence of the latest delta pair (excluding that final occurrence itself).
9. From *this* occurrence of the delta pair, up to the defined prefetch degree number of deltas are collected, searching *forwards* in the delta stream.
10. Finally, the latest miss address is added to each delta, and these are the addresses to be prefetched.

2 Implementation

The prefetcher was implemented in C++20, and the standard library was put to great use while building the delta stream.

2.1 Data Structures

To begin with, to simulate hardware limitations, the IT was built using an implementation of a least recently used (LRU) hash table, with a fixed size. The GHB was implemented using `boost::circular_buffer`, in the Boost library. Both data structures have fixed, equal capacities.

2.2 Pointer and Memory Management

The existence of pointers and the dynamic destruction of data by the GHB poses a potential memory management issue: multiple pointers to the same data would mean that as data is evicted by the GHB, the pointers in the IT and *within* the GHB are not invalidated correctly.

To solve this, the GHB is set to *own* the pointers. As such, it contains `std::shared_ptr<Node>`, with the following definition of `Node`:

```
struct Node {
        // ... constructor ...
        AddressType miss_address;
        std::weak_ptr<Node> next;
}
```

The *entries* in the IT are of type `std::weak_ptr<Node>`, too. This allows the graceful destruction of the `Node`s as they are evicted from the GHB. Any `std::weak_ptr`s that dereference to said `Node` may be queried for validity by way of `std::weak_ptr<T>::expired()`.

It is worth noting that `std::adjacent_difference` and `std::find_end` were critical methods from the C++ standard library to build and verify the delta streams mentioned in § 1.3.

2.3 Implementation Discussion

The implementation was relatively straightforward, and completed with `CLion 2022.1`, with `clang++` version 13 and `gdb` version 11.2. Initially, a templated implementation was sought; however, to ease debugging and development, this was quickly abandoned in favour of hardcoding types in the IT and GHB implementations.

It was very pleasant to see the linked lists being built in the GHB, as verification that the implementation was indeed correct.

2.4 Parameters

The parameters provided to the GHB PC/DC prefetcher are the same as the literature: 256 entries in the IT and GHB each, with a maximum prefetch degree of 4.

2.5 Result Collection

The `stdout` from each program run was written to a correspondingly-named file in the `output/` directory. Then, the `make_csv.sh` script compiled the final IPC output from each simulator run into a `.csv` file for further mangling and data processing in a spreadsheet/chart generation program.

3 Results and Discussion

The results of the implementation are compared with the baseline no-prefetcher simulator i.e. `skeleton.cc`. The percentage improvements of each implementation is detailed in **Figure 1**.

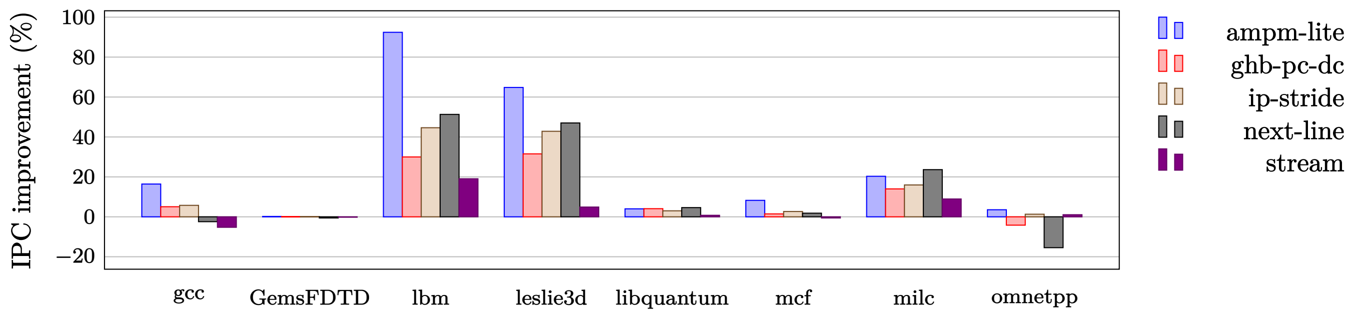

Figure 1: Prefetcher performance

3.1 Comparisons

It was not surprising to see the AMPM prefetcher outperforming GHB PC/DC: this is a *very* recent prefetcher and in benchmarks by the authors (Ishii, Inaba and Hiraki), performs up to twice as well as GHB PC/DC[3]

What *is* surprising, however, is the comparatively poor performance of GHB PC/DC versus IP Stride and even Next Line. While debugging, it was noted that GHB PC/DC was not performing as well as it *should* have been. A large proportion of access deltas were found to be rather *constant* for many benchmarks, meaning that only *two* addresses were prefetched, rather than the maximum specified by the prefetch degree.

It is possible that additional heuristics, such as finding the second-latest occurrence of the delta pair, searching for *multiple* delta pairs in the stream and somehow concatenating/averaging these results, or simply excluding some arbitrary initial portion of the linked-list, could lead to improved performance for the GHB PC/DC prefetcher.

4 Conclusion

In conclusion, a global history buffer with PC localisation and delta correlation (GHB PC/DC) prefetcher was built, tested, and run using the Data Prefetching Championship framework, and these results were compared with example prefetchers provided in the implementation.

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)*. 10th International Symposium on High Performance Computer Architecture (HPCA'04). Feb. 2004, pp. 96–96. DOI: 10.1109/HPCA.2004.10030 (page 1).

[2] Babak Falsafi and Thomas F. Wenisch. *A Primer on Hardware Prefetching*. Morgan & Claypool Publishers, 1st May 2014. 69 pp. ISBN: 978-1-60845-953-7. Google Books: CWm7AwAAQBAJ (page 1).

[3] Y. Ishii, M. Inaba and K. Hiraki. 'Access Map Pattern Matching for High Performance Data Cache Prefetch'. In: *J. Instr. Level Parallelism* (2011) (page 2).