Implementation of GHB PC/DC prefetcher

CS5222 Advanced Computer Architecture

Simone Mezzaro

Contents

1 Overview

This report describes the implementation of the GHB PC/DC prefetcher presented in [2] and explained also in [1].

In [2] Kyle Nesbit and James Smith illustrate a new technique that can be used to improve the prediction accuracy of any already existing prefetcher which makes use of a hash table to store the history of memory accesses. This technique is based on a two-level structure composed by an index table (IT) and a global history buffer (GHB). In particular the paper also shows how to adapt a distance prefecther using these structures to obtain the GHB PC/DC prefetcher.

1.1 Description of GHB PC/DC prefetcher

The GHB PC/DC prefetcher has an IT indexed by the program counter of a memory access instruction. Each entry of the IT contains a pointer to an element of the GHB.

The GHB is a circular FIFO buffer. Each of its elements corresponds to a previous memory access from an instruction and contains the accessed address and a pointer to another element of the GHB which represents the preceding memory access from the same instruction.

Elements in the GHB are therefore organized in linked list. Each list corresponds to an instruction and contains memory addresses previously accessed by that instruction, from the most recent to the oldest. The head of the list is pointed by the entry of the IT indexed by the instruction's PC.

When a cache miss occurs d memory blocks, where d is a parameter called degree, are prefetched following these steps:

- The PC of the instruction causing the miss is used to index the IT and the pointer to the head of the linked list for that PC is obtained.
- A pair of strides (called deltas) is computed as differences between the two most recent miss addresses for that PC and between the most recent miss address and the address of the triggering miss. The two most recent misses are the first two elements of the list. The pair here computed is called pattern.
- The entire list is traversed to find another pair of consecutive deltas matching the pattern. Once it has been found, the list is traversed backward from the match to compute the d deltas chronologically following the match.
- The d deltas are summed to the triggering miss address to obtain d memory addresses to prefetch.

Once the prefetching is completed the triggering address is pushed on the GHB with a pointer to the element that previously was the head of the list and the PC entry in the IT is updated to contain a pointer to the newly inserted element.

1.2 Advantages of GHB PC/DC prefetcher

GHB PC/DC and more in general any prefetcher adapted using the technique presented in [2] show many advantages with respect to the traditional hash table prefetchers.

First of all, the FIFO circular buffer gives the possibility to eliminate stale data because oldest elements in the GHB are overwritten as newer addresses are stored. Having only recent data in the GHB allows to make more accurate predictions increasing performances.

Moreover, in GHB PC/DC the number of elements stored for each entry of the IT is no more fixed since it corresponds to the length of its linked list, which can vary depending on how frequently that entry is accessed. This wasn't possible with the previous prefetchers that saved the history related to an entry in the entry itself, which had limited and fixed space.

Finally in GHB PC/DC each component of the two-level structure can be sized independently, allowing to save space by tuning these dimensions correctly.

2 Implementation

The GHB PC/DC prefetcher has been implemented using [3]. The provided implementation strictly follows the description given in section 1.1.

There are three interesting aspects of the implementation to underline:

- The IT is implemented as an hash table. The hashing function used is very simple and consists in computing the value of program counter modulo size of the IT and using this value to index the table.
- In the provided implementation, the GHB is implemented as a fixed length array with an head index. In this implementation therefore each pointer becomes an index for the GHB array.
- As described in [2] (section 3.3) a mechanism to detect pointers to invalid values is required. This implementation solves the issue as suggested by the paper. Each index (corresponding to a pointer) is divided into two parts: the four most significant bits are used for the detect mechanism and the others are used to index the GHB array. An index is invalid if the difference between it and the head index is greater than the GHB size.

However there are also two differences to point out between this implementation and the one presented in [2] and [1]:

- As stated also by Nesbit and Smith, the mechanism for detecting invalid indexes is not perfect. This may cause loops in the linked lists in some rare cases when there is an element with an invalid index pointing to a previous element of the list. A traversed nodes counter has been added to prevent the prefetcher from getting stuck in this loop while traversing the list. When the counter reaches the length of the GHB the traversing is stopped.

- The paper (section 4.1) requires a one bit prefetch flag with each cache line, which is set on prefetch and cleared when that line is accessed. The flag is used to preserve the same cache miss address stream of an execution without any prefetcher. However, this mechanism is difficult to implement using the interface provided in [3] because there is no possibility to directly change cache behavior.

3 Results

The given implementation has been simulated on the benchmarks already provided in [3]. The benchmarks are listed in the following table together with the IPC obtained by GHB PC/DC in each of them.

gcc	GemsFDTD	lbm	leslie3d
0.305041	3.448045	1.293536	1.221748
libquantum	mcf	milc	omnetpp
3.276553	0.348966	1.099301	2.11549

The parameters used for the simulation are those provided by [2]. In particular:

- IT size: 256 entries
- GHB size: 256 elements
- Degree: 4

Results obtained by GHB PC/DC have been compared with results of other prefetchers. Figure 1 shows the percentage of IPC improvement of each prefetcher with respect to the IPC of an execution without any prefetcher.

GHB PC/DC has a positive improvement on all benchmarks except for *omnetpp*. GHB PC/DC generally performs better than stream, but is beaten by the other three prefetchers on most of the benchmarks. This can be also seen in figure 2 where the average IPC improvement achieved by GHB PC/DC is 6.52%.

4 Conclusion

The provided implementation of the GHB PC/DC prefetcher conforms with the description provided in [2] and [1] and obtains a positive IPC increase over the baseline with no prefetcher.

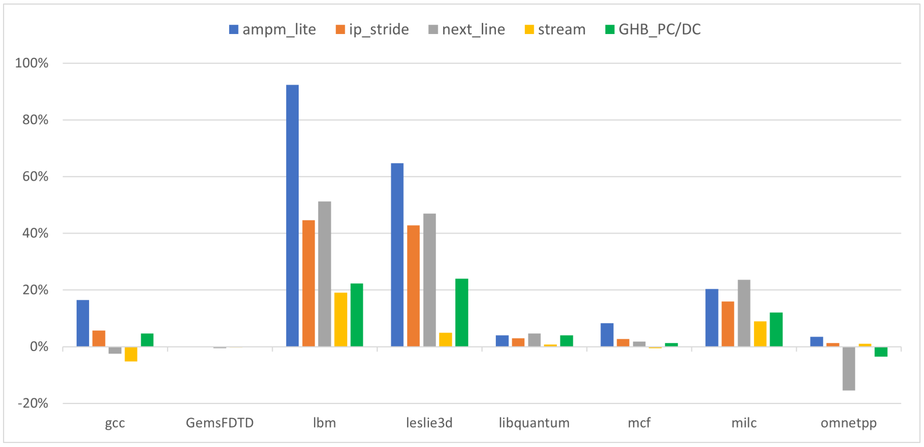

Figure 1: IPC improvement over a baseline with no prefetching

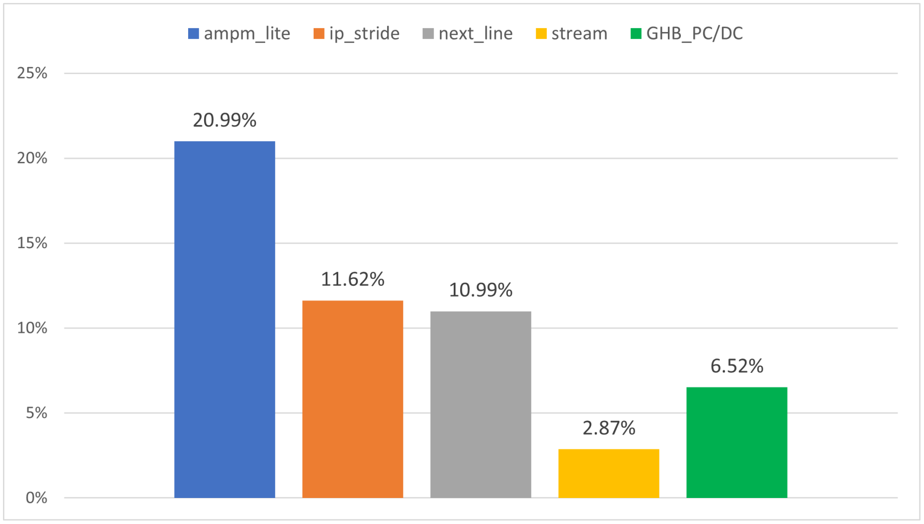

Figure 2: Average IPC improvement over a baseline with no prefetching

However, as the result section shows, there is still room for improvement. One way to achieve a better performance can be to implement the cache flag mechanism illustrated in section 2. Moreover an accurate tuning of the parameters could also allow to further increase the IPC, since the parameters provided by the original paper were prepared for a different set of benchmarks.

References

[1] Babak Falsafi and Thomas F. Wenisch. *A Primer on Hardware Prefetching*. 2014.

[2] 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`.

[3] *Repository based on the second Data Prefetching Championship*. `https://github.com/nus-comparch/cs5222-lab-prefetcher.git`.