Computer Architecture - CS 301

Rishit Saiya - 180010027, Assignment - 12

November 28, 2020

1

This is a direct application to find the CPI as mentioned in the video lecture. Figure 1 shows the formula we will be using which is given in lecture.

In here, 1 is the $L1_{HitTime}$. So, formula might change accordingly.
To get an overview, we will try to list down all the given values to us. They are as follows:
Baseline IPC = 0.8
Fraction of Memory Operations = 30 %
L1 Cache Hit Rate = 100 %
L1 Cache Hit Time = 1 Cycle
L1 Data Cache Miss Rate = 5 %
L2 Cache Miss Rate = 50 %
L2 Cache Hit Time = 10 cycles
L2 Miss Penalty = 100 cycles

$$CPI_{ideal} = \frac{1}{IPC_{baseline}}$$

$$CPI_{ideal} = \frac{1}{0.8} = 10/8 = 1.25$$

We are given the f_{mem} as follows:

$$f_{mem} = 30\% = 0.3$$

$$CPI = CPI_{ideal} + stall_{rate} * stall_{cycles}$$
$$= CPI_{ideal} + f_{mem} * (AMAT - 1)$$

Figure 1: CPI Calculation

$$AMAT = L1_{hit\ time} + L1_{miss\ rate} * L1_{miss\ penalty}$$
$$= L1_{hit\ time} + L1_{miss\ rate} * (L2_{hit\ time} + L2_{miss\ rate} * L2_{miss\ penalty})$$

Figure 2: AMAT Calculation

So, we calculate the AMAT variable value. We are given in the lecture that AMAT value is as given in Figure 2. With that reference, we will calculate AMAT here.

$$AMAT = L1_{Hit\,Time} + L1_{Miss\,Rate} \times (L2_{Hit\,Time} + (L2_{Miss\,Rate} \times L2_{Miss\,Penalty}))$$

$$AMAT = 1 + 0.05 \times (10 + (0.5 \times 100)) = 4$$

So, finally we calculate CPI.

$$CPI = 1.25 + 0.3 \times (4 - 1) = 2.15$$

So, final IPC will be as follows:

$$IPC = \frac{1}{CPI}$$

$$IPC = \frac{1}{2.15} = 0.4651162790$$

2

Before we attempt to understand the motivation of Prefetching, we need to understand the aspect and environment where it will prove to be effective. As mentioned in the lecture, we have following types of misses:

- **Compulsory Miss:** These misses occur when a data stream is read first time. Since it is read first time, it compulsory has to miss and hence the name. The following are some of the suggested mitigation techniques to decrease number of Compulsory Misses:
  1. As mentioned in the lecture, we can increase the Block Size, in order to store more data blocks at single read and further more this will lead to usage of spatial locality. With this the number of Compulsory Misses will reduce.
  2. Another proposed mitigation is prefetching the memory locations that are to be used in the near future. This is done by generating an algorithm which can wisely and effectively guess upcoming cache locations based on usage history.
- **Conflict Miss:** These misses occur when there is limited amount of associativity in a direct mapped cache or set associative cache. The following are some of the suggested mitigation techniques to decrease number of Conflict Misses:

1. A simple way to mitigate is to write algorithms/codes which have cache involvement. Basically, trying to put together code which is cache independent to maximum possible extent.
2. Another way is to use a smaller fully associative cache which is meant to store the line which is thrown out of main cache so as to use immediately next time instead of fetching. This is generally called as Victim Cache.
3. We can also increase the associativity of the cache. As mentioned in lecture, this comes with a trade off cost of high consumption of power and latency also increases resulting in slower computation time.

- **Capacity Miss:** These misses occur due to limited size of Cache blocks. The following are some of the suggested mitigation techniques to decrease number of Capacity Misses:

  1. A simple fix would be straight away increase the size of Cache blocks.
  2. Another mitigation technique would be produce a novel and wise technique for prefetching in order to accommodate more such misses.

As clearly mentioned above, we see that two of the misses' mitigation involves Prefetching and making a novel and wise algorithm to predict the memory locations that will be used in the near future would help us to mitigate Compulsory and Capacity Misses. This would increase the overall efficiency output of the Cache and lead to increase in more hits than before. This constitutes the motivation to integrate ***Prefetching*** in modern caches.

In the following sections, we will explain different types of Prefetching and its processes.

- **Hardware Prefetching:** A hardware prefetcher is a dedicated hardware unit that predicts the memory accesses in the near future, and fetches them from the lower levels of the memory system.

  In the Figure 3, we can see the basic look of Hardware Prefetcher as mentioned in the video. In here, the prefetcher checks the hits/misses at the L1 cache. Based on results from L1 cache, now it decides whether the lists of addresses are likely to be accessed in the near future or not. It then sends the data to L2 cache to get in those lines to L1 cache in a such a way that when there is a call for those data, there is a hit in L1 cache and it can easily fetch there onwards.

- **Software Prefetching:** A software prefetcher uses the compiler and predicts future cache misses and inserts a prefetch instruction based on the miss penalty and execution time of the instructions.
  Below we will consider 2 different code snippets taken from the lecture to explain software Prefetching. Let us consider following Code_Snippet-1:

```
int addAll(int data[], int vals[]) {
```

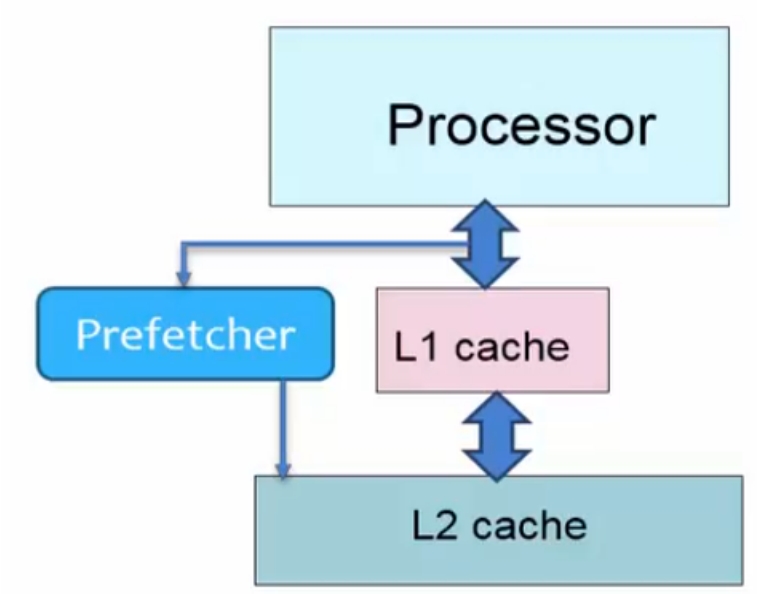

Figure 3: Hardware Prefetcher

```
      int i, sum = 0;
      for (i = 0; i < N; i++) {
         sum += data[vals[i]];
      }
      return sum;
}
```

Let us consider following Code_Snippet-2:

```
int addAllP(int data[], int vals[]) {
      int i, sum = 0;
      for (i = 0; i < N; i++) {
         __builtin_prefetch(& data[vals[i + 100]]);
         sum += data[vals[i]];
      }
    return sum;
}
```

Since *vals* array can have discrete values and can be stored in the sequence, *vals* array possess spatial locality whereas *data* array doesn't has any spatial locality. In such case, we integrate the above function in Code_Snippet-1 with *__builtin_prefetch* function which helps us to prefetch the address of data which are to be accessed after such 100 iterations. This helps us to increase the Memory Fetch efficiency of cache and also to reduce the MA time.

As much as Prefetching techniques are helping us to mitigate compulsory and capacity misses, there also exists some disadvantages of using it. Some of which are as follows:

- Addition of Extra Complexity to the Code Execution
- Addition of Risk of displacing useful/frequently used data from the caches.

Source used: Reference Book, Video Lecture, Wiki Page