CS6600: Computer Architecture

Assignment 4: Design your own Data Prefetcher

Bala Dhinesh - EE19B011

Vishnu Varma V - EE19B059

Introduction:

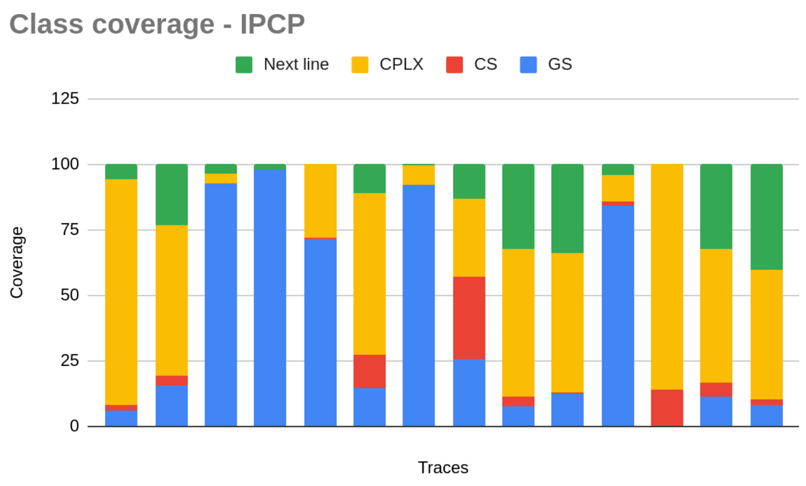
This assignment aims to design our own custom Data Prefetcher using Champsim microarchitectural simulator and benchmark the results with several built-in prefetchers.

Our Prefetcher - MYPREF-IPCP:

Our custom prefetcher (MYPREF-IPCP) modifies the current state-of-the-art IPCP prefetcher. Here, in IPCP, the prefetcher training will happen in the L1D cache. So our custom prefetcher also modifies IPCP in the L1D prefetcher instead of LLC. The final analysis of our prefetcher will be made using the IPC, L1D, L2C, and LLC hit rate values.

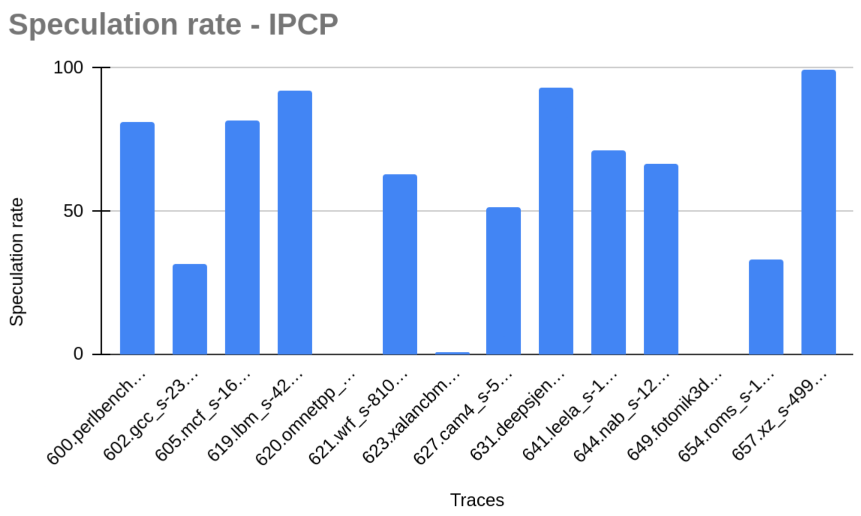
Motivation for our Prefetcher:

1. We observed that memory access from the given traces only occasionally increases. When the IPCP Prefectcher takes the worst-case scenario - the Next Line (NL) Prefetcher, we found that many negative stride access pattern is observed along with positive stride access pattern on the same page.



Plot 1: IPCP Class Coverage

2. We plotted the class coverage of different prefetchers in the IPCP as below and observed a considerable amount of next-line prefetcher class is being used (Plot-1). We also plotted the confidence level of Speculative next-line access with all the given traces (Plot-2). We can observe that 50% percent of traces have a confidence level of less than 50%. This shows room for improvement in the IPCP prefetcher by modifying the next-line prefetcher.



Plot 2: IPCP Speculation rate

Modifications to IPCP:

Our Prefetcher replaces only the next-line prefetcher in the IPCP with our custom prefetcher. The remaining prefetcher classes, such as Global Stream, Constant Stride, and Complex Stride, are left untouched since each has its own advantage. Our custom prefetcher, where, instead of prefetching (X+1) all the time, we also support prefetching (X-1), depending on the direction of the demand requests. The code for the above logic is as follows:

```
// MYPREF modification - replacement of next-line class in IPCP

/* History-Buffer Module for the prefetcher */
HistoryBuffer hist_buff[NUM_CPUS];

uint32_t CACHE::prefetcher_cache_operate(uint64_t addr, uint64_t ip, uint8_t
cache_hit, uint8_t type, uint32_t metadata_in) {
   uint64_t page_id = EXTRACT_PAGE_ID(addr);
   uint32_t block_id = EXTRACT_BLOCK_ID(addr);
/* Temporarily store the block ID that we'll increment/decrement later on */
   int32_t prefetch_block_id = (int32_t) block_id;
```

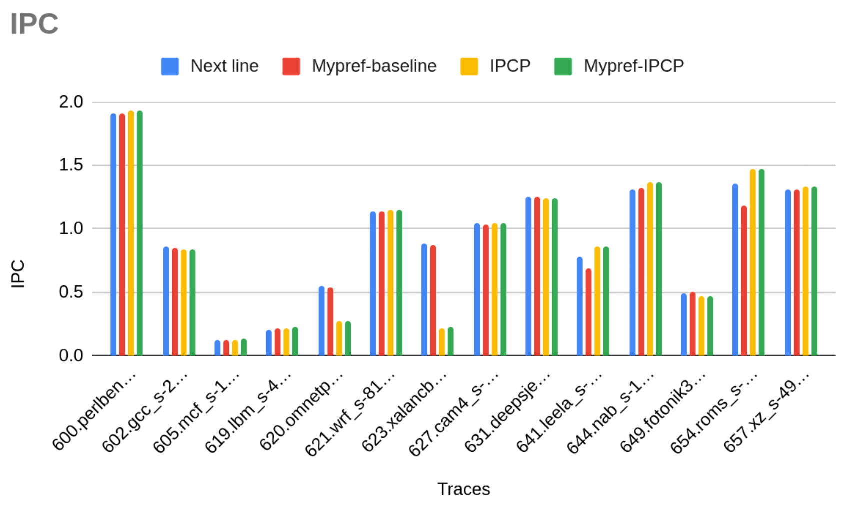
```
    uint64_t prefetch_addr;
    /* Check if the access is on the same page as previous page */
    /* If no, then reset the buffer and return */
    if(!is_on_same_page(cpu, page_id)) {
        hist_buff[cpu].reset();
        return  metadata_in;
    }
    /* Append the latest block ID to the buffer */
    hist_buff[cpu].append(block_id);
    /* Increment/decrement prefetch counter based on confidence value */
    if(hist_buff[cpu].has_forward_movement())
        prefetch_block_id++;
    else
        prefetch_block_id--;

    /* Check if prefetch block falls in the same page */
    if(prefetch_block_id >= BLOCK_ID_MIN && prefetch_block_id <= BLOCK_ID_MAX) {
        prefetch_addr = prepare_prefetch_address(page_id, prefetch_block_id);
/       * Prepare the address to prefetch */
        prefetch_line(ip, addr, prefetch_addr, FILL_L1, 0);
    }
    return metadata_in;
}
```

We used a ***deque*** data structure to implement our history buffer. We set the history buffer size to five. Whenever there is a demand access, we will first check if the demand access address is on the same page as the previous demand access. If both are the same, we append the data to the history table. If they are on different pages, we need to create a new history table for that particular page. For simplicity, we reset the history buffer if such a case occurs. If the value of ***prefetch_block_id*** is greater than or equal to (5-1)/2, then prefetch the next line; else, prefetch the previous line.

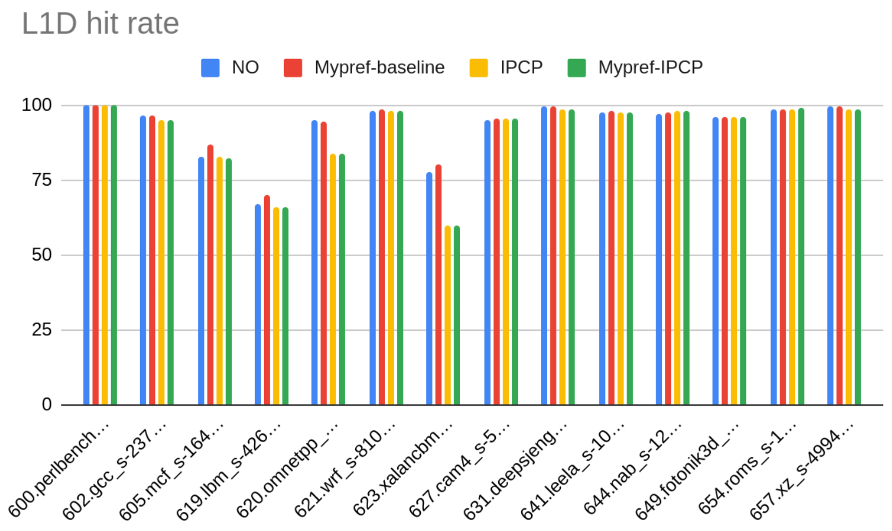
```
bool has_forward_movement(){
        int forward = 0;
        int backward = 0;
        for(int i=0; i<Deque.size(); i++){
            if(Deque[i] < Deque[i+1]){
                forward += 1;
            }
            else{
                backward += 1;
            }
        }
        return forward >= ((BUFFER_LENGTH-1)/2 + 1);
    }
```

Comparison of MYPREF-baseline:

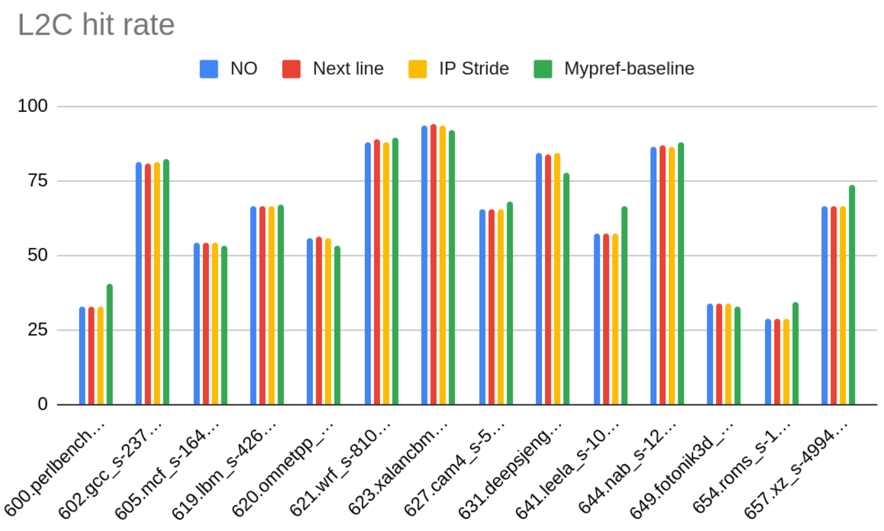
Before incorporating the above modification to the IPCP Prefetcher, we first analyze the above modification as a separate prefetcher (MYPREF-Baseline) with the inbuilt prefetchers.



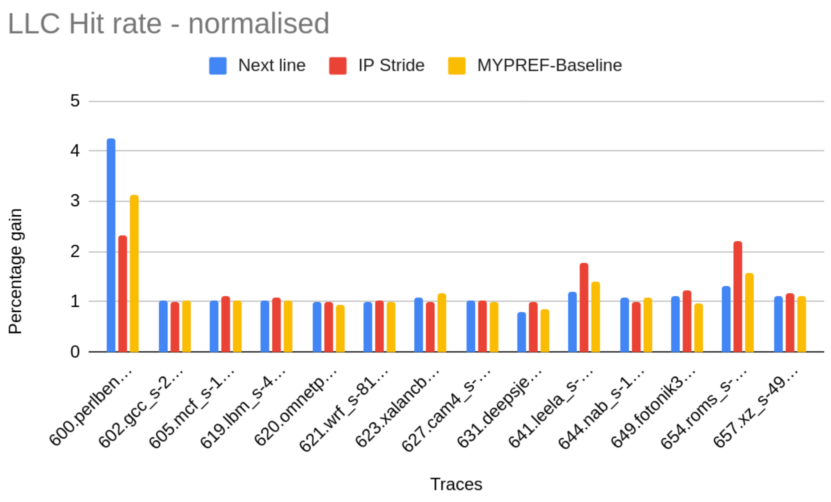
Plot 3: IPC values



Plot 4: L1D hit rate



Plot 5: L2C hit rate

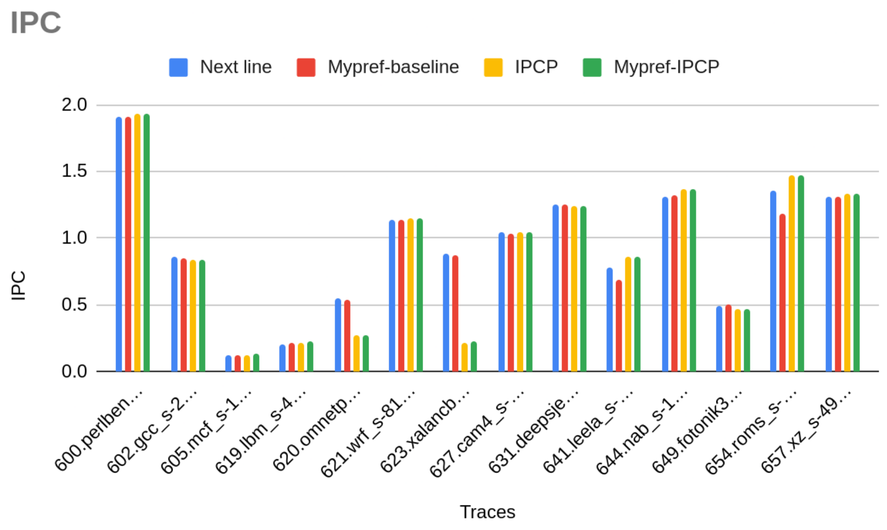


Plot 6: LLC Hit Rate

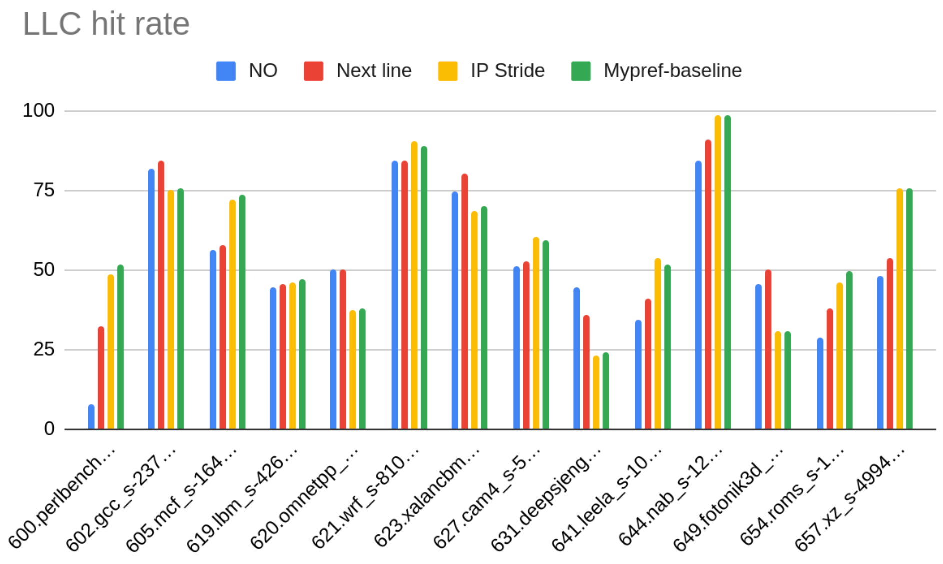
From the graph above (Plots-3,4,5,6), we can infer that the performance of MYPREF-Baseline is almost similar to other inbuilt prefetchers(next line and IP Stride), with a minute drop in some cases. The reason for similar performance is that most consecutive demand access from traces are from a different page. Our history table will predict from previous accesses only if they are from the same page. This results in the history table being set to reset, and no prefetching occurs most of the time.

Comparison of MYPREF-IPCP over IPCP:

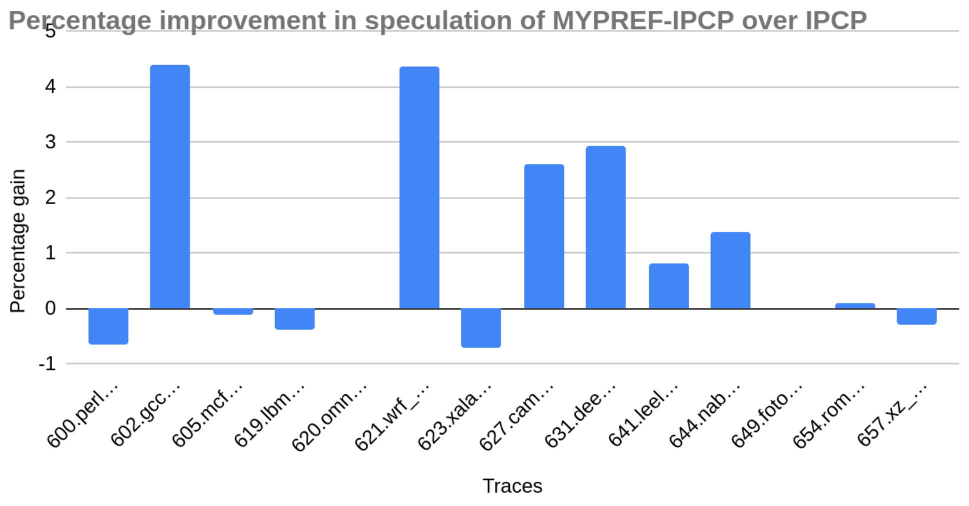
We modify the next-line prefetcher present in the IPCP prefetcher with our custom MYPREF baseline. We also note the Prefetcher Speculation values based on Misses Per Kilo-Cycles(MPKC) - The higher the MPKC value, the lower the speculation value. A lower speculation value implies the prefetching will be less likely to occur. The below graph (Plot-7) represents the percentage improvement in speculation of MYPREF-IPCP over IPCP. We can observe from the above plot that we achieve a significant improvement in the Prefetcher Speculation value for our custom prefetcher.



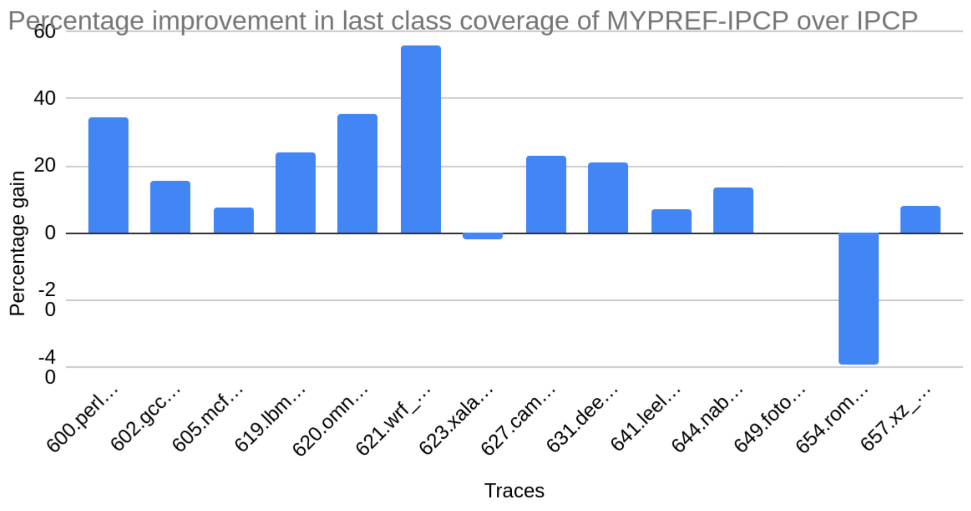
Plot 7: IPC comparison



Plot 8: LLC hit rate comparison



Plot 8: % gain in speculation of MYPREF-IPCP over IPCP



Plot 9: % gain in next-line coverage of MYPREF-IPCP over IPCP

The graph above (Plot-8) represents the percentage gain by replacing Next-line prefetcher in IPCP with our custom prefetcher. This clearly denotes the modified IPCP Prefetcher(MYPREF-IPCP) supports prefetching with our custom prefetcher. Thus our custom prefetcher helps in reducing the MPKC values.

Advantages:

- Supports both Positive and Negative Stride of 1 instead of default next-line prefetcher.
- Slight improvement to IPC values in most of the given traces.

- Up to a five-percent increase in speculatively prefetch according to our custom prefetcher that results in lesser MPKC(Misses Per-Kilo Cycle).
- Up to sixty-percent increase favoring our custom prefetcher class compared to Next-line prefetcher.

Limitations:

- Since most consecutive demand access from traces are from a different page, our custom prefetcher history table resets its history values more often, thereby not working at its fullest. This results in only minor improvement in performance.
- The size of our custom prefetcher will be greater than the next-line prefetcher, so there might be some difficulty in incorporating it into the L1D cache.

Google sheet link for the detailed report:
https://docs.google.com/spreadsheets/d/1n88iEJKFZ9E5uQeFiPq_g6-gmnmkmF96rNzyzjXBFu4/edit?usp=sharing