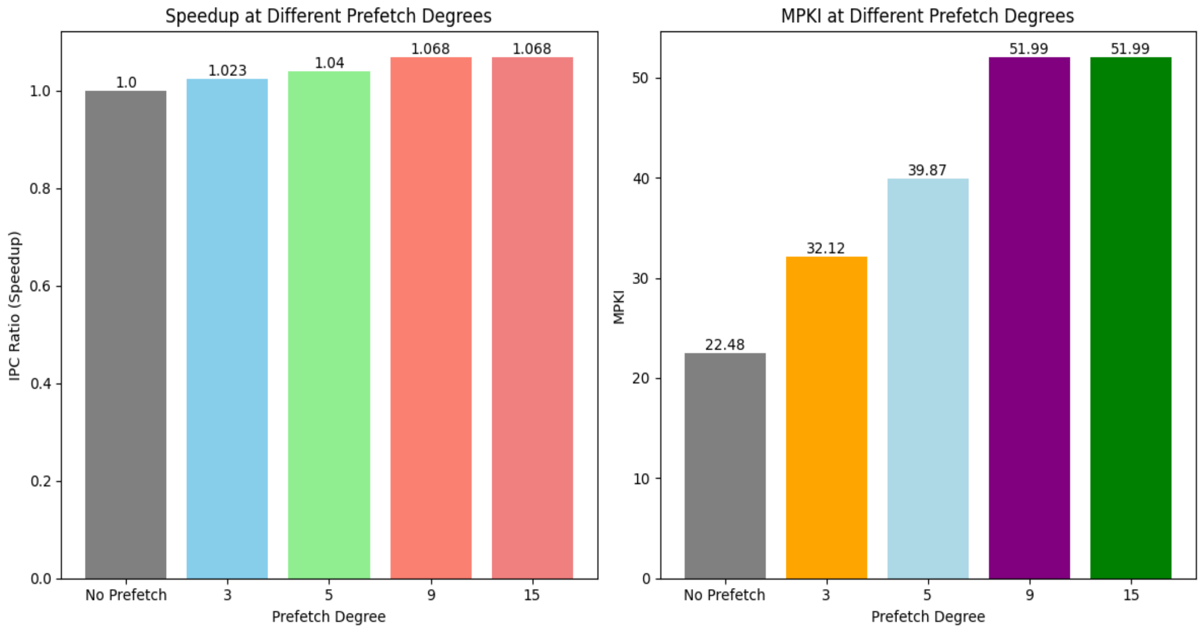
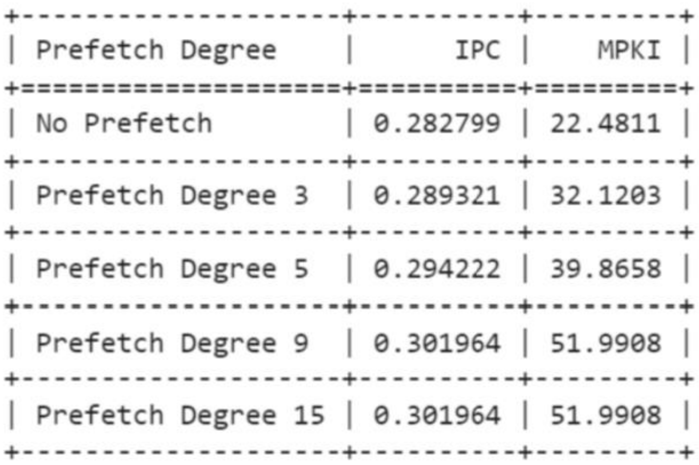
Assignment 2

Roll No :- 21D070048,210110014

Q1 STLB Prefetcher





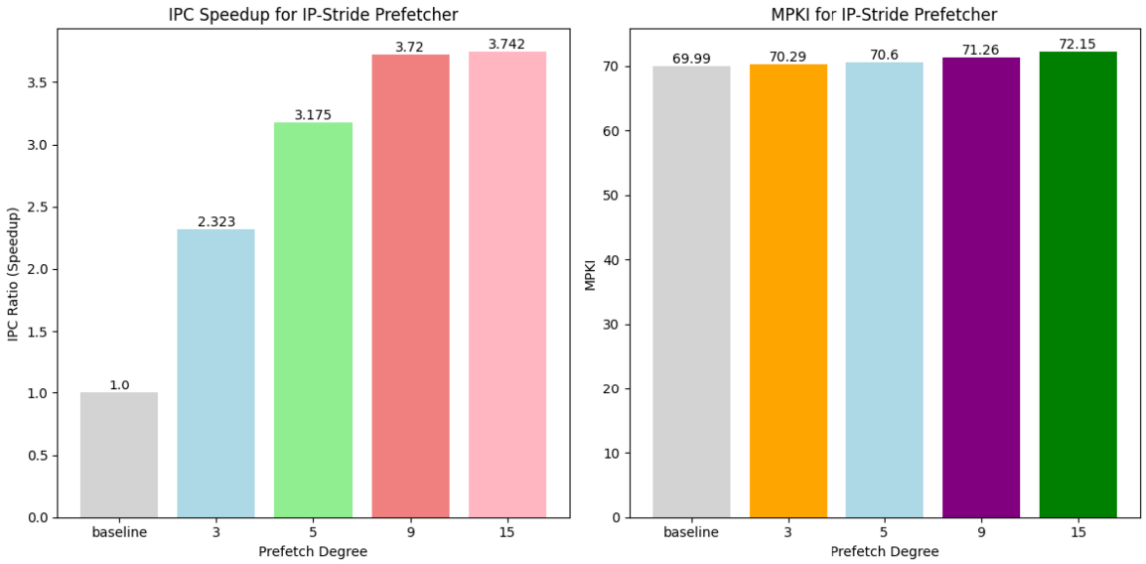
```
+--------------------+----------+----------+
| Prefetch Degree    |      IPC |     MPKI |
+====================+==========+==========+
| No Prefetch        | 0.282799 | 22.4811  |
+--------------------+----------+----------+
| Prefetch Degree 3  | 0.289321 | 32.1203  |
+--------------------+----------+----------+
| Prefetch Degree 5  | 0.294222 | 39.8658  |
+--------------------+----------+----------+
| Prefetch Degree 9  | 0.301964 | 51.9908  |
+--------------------+----------+----------+
| Prefetch Degree 15 | 0.301964 | 51.9908  |
+--------------------+----------+----------+
```

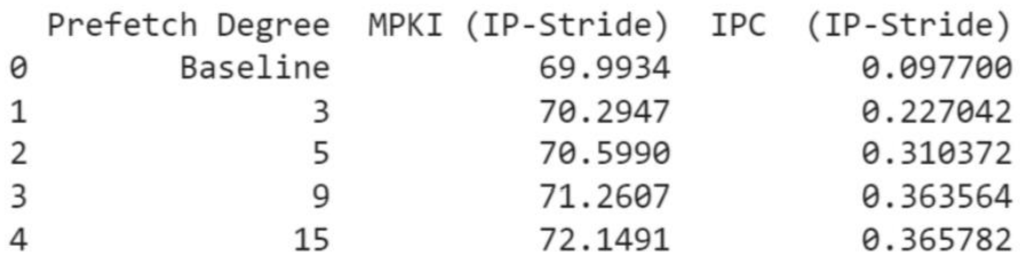
The prefetcher works by tracking the stride, which is the difference between consecutive memory addresses accessed by each instruction pointer (IP). When it identifies a consistent stride pattern with enough confidence, it proactively issues prefetch requests for future addresses based on that stride. The prefetch degree (PREFETCH_DEGREE) determines how many lines ahead it will prefetch.

Looking at the results, we see that as the prefetch degree increases from 3 to 15, the system is able to issue more prefetches. This leads to improved IPC (instructions per cycle) because more data is readily available when needed. However, there's a downside: the MPKI (misses per thousand instructions) also rises, particularly at higher degrees like 9 and 15. This increase in MPKI occurs because too many unnecessary prefetches can clutter the cache, causing useful data to be evicted. Essentially, there’s a trade-off: while we gain better data availability and higher IPC with prefetching, we also risk greater cache pollution, which can hurt performance

Q2 L1D Prefetcher

IP Stride



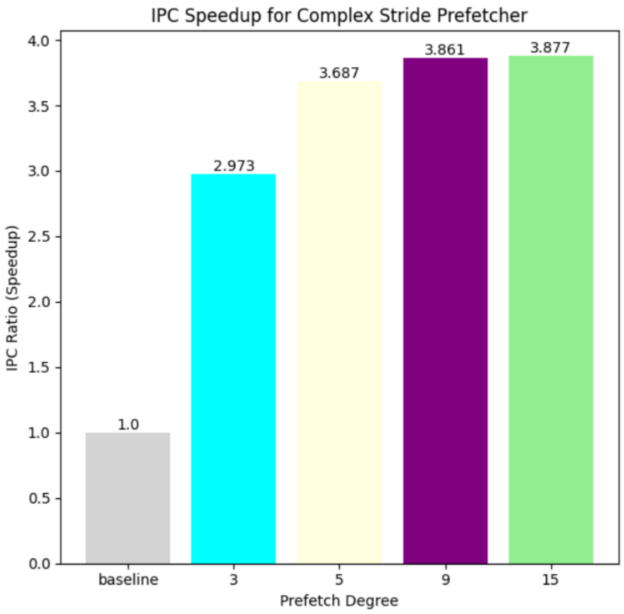


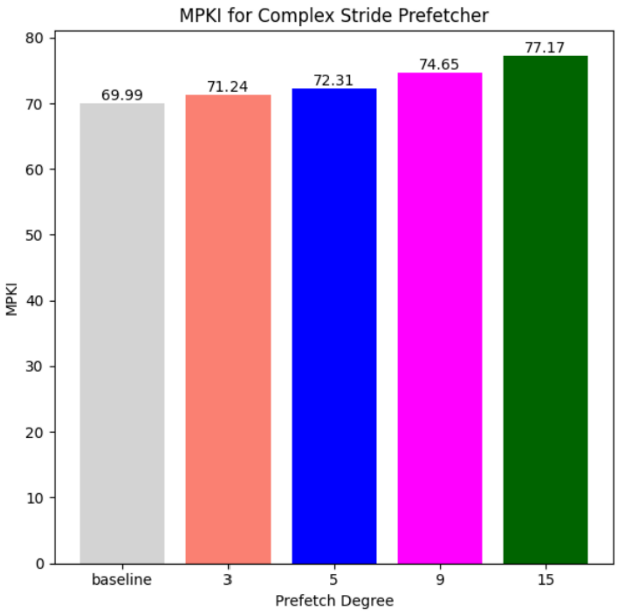
```
  Prefetch Degree  MPKI (IP-Stride)  IPC  (IP-Stride)
0        Baseline           69.9934          0.097700
1               3           70.2947          0.227042
2               5           70.5990          0.310372
3               9           71.2607          0.363564
4              15           72.1491          0.365782
```

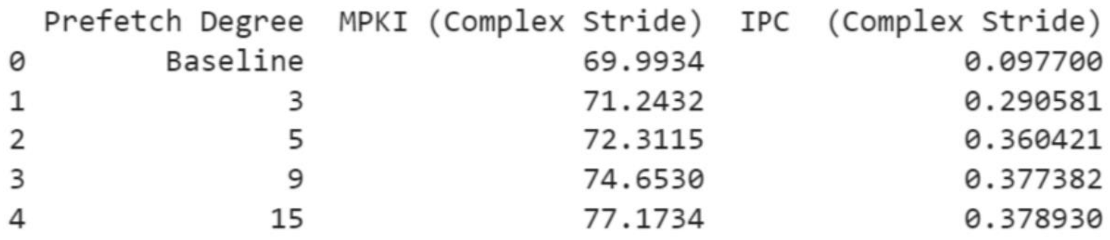
The IP_STRIDE PREFETCHER keeps an eye on memory accesses by tracking the stride patterns related to each instruction pointer (IP). When it identifies a consistent stride, it proactively requests data prefetches based on that stride, allowing the CPU to load data into the cache before it's actually needed. The function also fine-tunes its confidence in stride predictions based on how accurate those predictions have been in the past.

Looking at the results, we see that as the prefetch degree increases from 3 to 15, the MPKI (misses per thousand instructions) rises slightly. This suggests that while the prefetcher is loading more data into the cache, some of it may not be utilized effectively. On the other hand, IPC (instructions per cycle) shows a notable improvement, especially at lower degrees, indicating that the prefetcher does a good job of making data available when needed. This points to a trade-off: although higher prefetch degrees can lead to better performance, they can also cause cache inefficiency, resulting in higher MPKI and diminishing returns on IPC gains.

Complex Stride







```
  Prefetch Degree  MPKI (Complex Stride)  IPC  (Complex Stride)
0        Baseline                69.9934               0.097700
1               3                71.2432               0.290581
2               5                72.3115               0.360421
3               9                74.6530               0.377382
4              15                77.1734               0.378930
```

The complex stride (CPLX) prefetcher plays a key role in improving memory access efficiency by keeping track of how data is accessed through each instruction pointer (IP). It generates an n-bit signature that captures the most recent strides, which helps it predict future memory accesses. This signature is then used to look up the complex stride prediction table (CSPT). When the prefetcher recognizes a stride pattern, it updates a confidence counter that guides whether to prefetch future data based on the predicted strides.

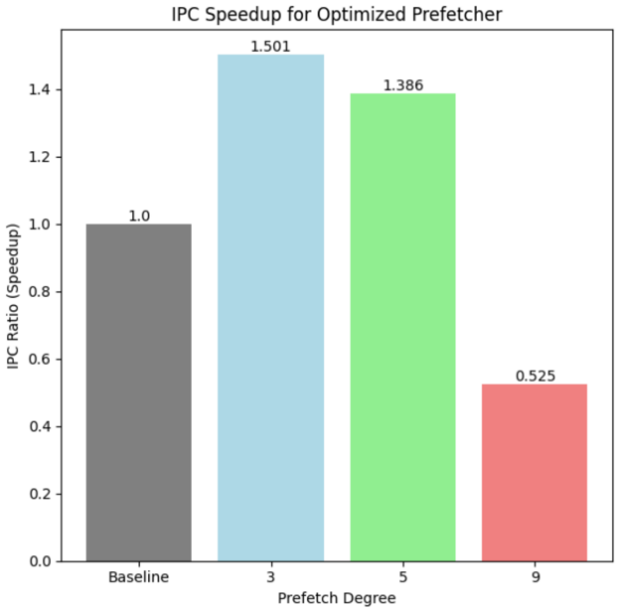
However, as the prefetch degree increases, the MPKI (misses per thousand instructions) also tends to go up, which indicates that while the system is bringing in more data ahead of time, not all of it is being used effectively. Although the IPC (instructions per cycle) improves at lower prefetch degrees, this improvement starts to level off or even decline at higher degrees. This highlights a balance: while prefetching can enhance data availability, it can also lead to inefficiencies in cache usage if not managed carefully.

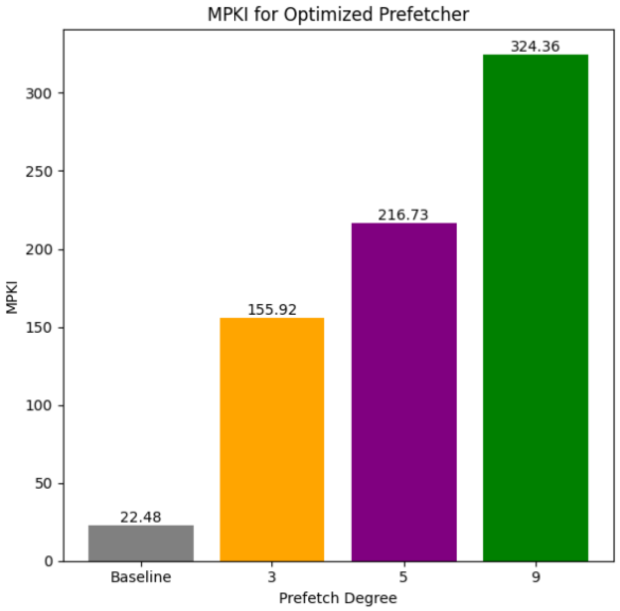
Q3 Optimized Prefetcher

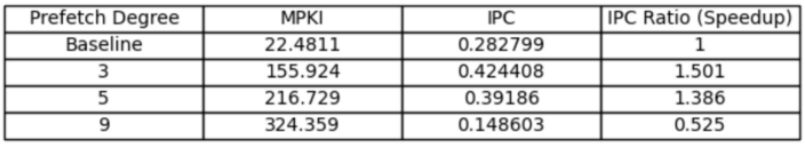
The Optimized Prefetcher dynamically adjusts the prefetching strategy based on observed memory access patterns. It begins by tracking the number of prefetch requests issued and switches between different prefetcher types (IP-Stride, Complex Stride, and Next-Line) in phases.

During the operation, if a stride pattern is detected and the confidence level is sufficient, it issues prefetch requests to bring in future cache lines. The function also maintains counters to evaluate the effectiveness of the different prefetchers, allowing it to adapt its approach based on which method yields the best performance during the training phase. This flexibility helps optimize cache efficiency and improves overall system performance.

For Trace 1

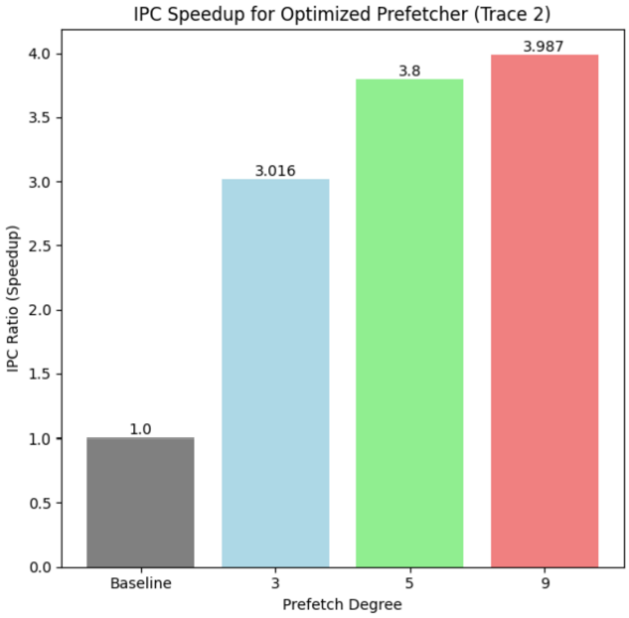


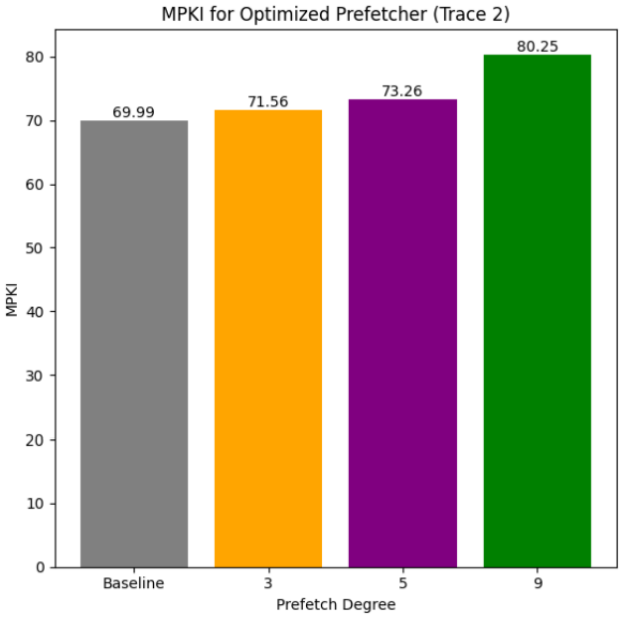


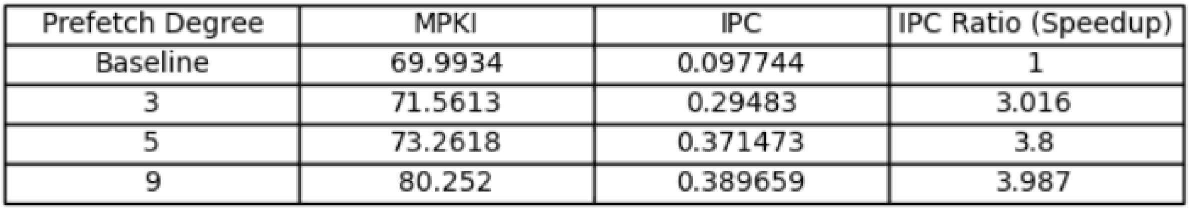


Prefetch Degree	MPKI	IPC	IPC Ratio (Speedup)
Baseline	22.4811	0.282799	1
3	155.924	0.424408	1.501
5	216.729	0.39186	1.386
9	324.359	0.148603	0.525

For Trace 2

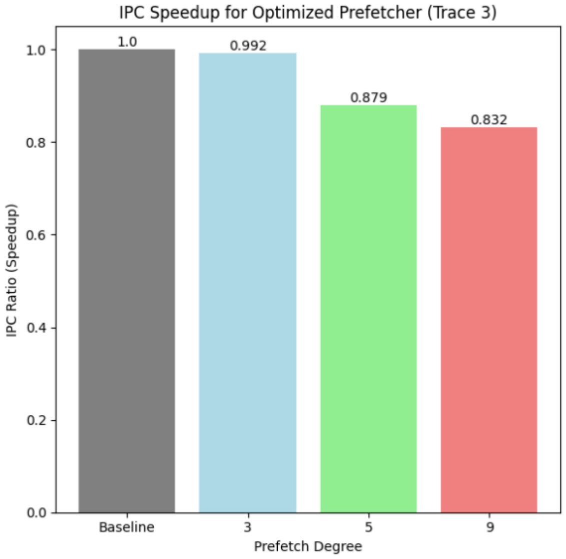


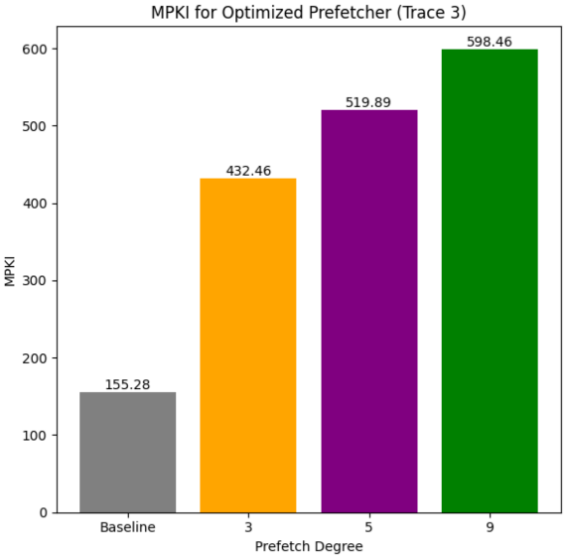


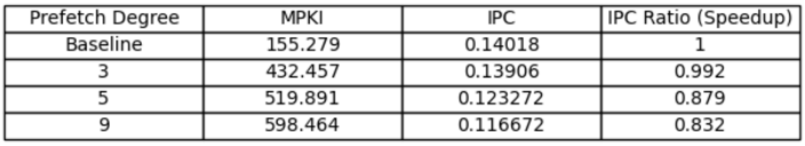


Prefetch Degree	MPKI	IPC	IPC Ratio (Speedup)
Baseline	69.9934	0.097744	1
3	71.5613	0.29483	3.016
5	73.2618	0.371473	3.8
9	80.252	0.389659	3.987

For Trace 3







Prefetch Degree	MPKI	IPC	IPC Ratio (Speedup)
Baseline	155.279	0.14018	1
3	432.457	0.13906	0.992
5	519.891	0.123272	0.879
9	598.464	0.116672	0.832