CS231 : Lab4 - Part2

Thomas Biju Cheeramvelil (22B1073)

Contents

1 Part I : Replacement Policies **2**
1.1 LRU (Least Recently Used) . . . 2
1.2 FIFO (First In First Out) . . . 2
1.3 LFU (Least Frequently Used) . . . 2
1.4 BIP (Binary Insertion Policy) . . . 2
1.5 Measurements for Different Traces . . . 3
1.5.1 602.gcc_s-1850B.champsimtrace.xz . . . 3
1.5.2 603.bwaves_s-1740B.champsimtrace.xz . . . 3
1.5.3 619.lbm_s-2677B.champsimtrace.xz . . . 3
1.5.4 bc-0.trace.gz . . . 4
1.5.5 sssp-0.trace.gz . . . 4
1.6 Comparison graphs . . . 4

2 Part II: Data Prefetcher **6**
2.1 Measurements for Different Traces . . . 6
2.1.1 602.gcc_s-1850B.champsimtrace.xz . . . 6
2.1.2 603.bwaves_s-1740B.champsimtrace.xz . . . 6
2.1.3 619.lbm_s-2677B.champsimtrace.xz . . . 6
2.1.4 bc-0.trace.gz . . . 6
2.1.5 sssp-0.trace.gz . . . 7
2.2 Comparison Graphs . . . 7

1 Part I : Replacement Policies

In cache management, the choice of a replacement policy plays a crucial role in determining which data is retained in the cache and which data is evicted when a set in a cache is full and new data needs to be accommodated.

1.1 LRU (Least Recently Used)

The LRU (Least Recently Used) replacement policy is a cache management strategy that prioritizes the retention of the least recently accessed data in a cache when space is needed for new data. The core idea behind LRU is to keep the data in the cache that has been accessed the most recently and, conversely, to evict data that has not been accessed for a longer time. This approach is based on the assumption that recently accessed data is more likely to be accessed again in the near future, making it a candidate for retention.

To implement LRU, the cache maintains a timestamp or a counter associated with each data element. This timestamp reflects the order of access, with the most recently accessed data having the highest value. When a cache line is needed for a new data item and the cache is full, the LRU policy identifies the cache line with the lowest (or least) timestamp value. Each read cache access updates the timestamp of the accessed data. The timestamp of the accessed data is set to the current time, ensuring that it becomes the most recently used.

LRU is effective in optimizing cache hit rates for workloads with temporal locality, where recently accessed data is likely to be used again. While this may be a good strategy for workloads whose working set is smaller than the available cache size or for workloads that have high temporal locality, such an insertion policy causes thrashing for memory-intensive workloads that have a working set greater than the available cache size.

1.2 FIFO (First In First Out)

The FIFO (First In First Out) replacement policy is a cache management strategy that prioritizes the eviction of the oldest cached data when space is needed for new data. The core idea behind FIFO is to evict the data that has been present in the cache for the longest time, without considering how frequently it has been accessed.

We can implement FIFO by maintaining a queue or buffer structure that holds cached data elements, in which the data elements are ordered in the queue based on their arrival time, with the oldest data at the front of the queue. I have implemented it by maintaining a timestamp associated with each data element, which reflects the order of insertion. When a cache line is needed for a new data item and the cache is full, the FIFO policy identifies the cache line with the lowest timestamp value, and evicts it, similar to the LRU policy.

1.3 LFU (Least Frequently Used)

The LFU (Least Frequently Used) replacement policy is a cache management strategy that prioritizes the eviction of data with the least frequent access when new data needs to be accommodated in the cache. Unlike other policies that focus on recency, LFU looks at the number of accesses of data. It aims to retain data that has been accessed the least frequently, under the assumption that such data might not be as relevant or in-demand.

To implement LFU, each data element in the cache is associated with a frequency count. This count is increased every time the data is accessed. When the cache is full and new data needs to be cached, the LFU policy identifies the data element with the lowest frequency count for eviction.

1.4 BIP (Binary Insertion Policy)

The Binary Insertion Policy (BIP) is a cache management strategy that employs a binary insertion algorithm to prioritize the eviction of cache entries.

Cache lines are inserted in MRU position with some probability ϵ and in LRU position with probability 1 - ϵ . Cache lines in LRU position are promoted to MRU position only if they are accessed while being in LRU position. Any line in LRU position is evicted to make space for incoming line. For small values of ϵ, BIP can respond to changes in the working set while retaining the thrashing protection of LIP (LRU Insertion Policy), which is a special case of BIP with $\epsilon = 0$, that is, all the incoming lines are placed in the LRU position and are promoted to the MRU position only if they are accessed again.

It can be observed that miss rate decreases with increasing ϵ from 0 to 1. As ϵ increases, BIP becomes more effective at distinguishing between cyclic and linear access patterns. It better accommodates workloads with a mix of these patterns, reducing cache misses as it adapts to the workload's characteristics. This results in the miss rate decreasing.

1.5 Measurements for Different Traces

1.5.1 602.gcc_s-1850B.champsimtrace.xz

Replacement	IPC	Speedup	Accesses	Misses	L2C Miss Rate (%)
LRU	2.118	1.0	603867	445984	73.85
FIFO	2.121	1.0014	603861	459075	76.02
LFU	2.131	1.0061	603873	446594	73.95
BIP (eps = 0)	2.136	1.0085	603866	446594	73.96
BIP (eps = 0.25)	2.116	0.9991	603868	446478	73.94
BIP (eps = 0.5)	2.118	1.0000	603876	446441	73.93
BIP (eps = 0.75)	2.117	0.9995	603871	446315	73.91
BIP (eps = 1.0)	2.118	1.0000	603864	446082	73.87

1.5.2 603.bwaves_s-1740B.champsimtrace.xz

Replacement	IPC	Speedup	Accesses	Misses	L2C Miss Rate (%)
LRU	0.6344	1.0	517534	475120	91.8
FIFO	0.6329	0.9976	517487	475805	91.95
LFU	0.6346	1.0003	517446	506462	97.88
BIP (eps = 0)	0.6368	1.0038	517497	506225	97.82
BIP (eps = 0.25)	0.635	1.0009	517511	495513	95.75
BIP (eps = 0.5)	0.6349	1.0008	517522	486549	94.02
BIP (eps = 0.75)	0.6349	1.0008	517534	479050	92.56
BIP (eps = 1.0)	0.6345	1.0002	517535	475127	91.81

1.5.3 619.lbm_s-2677B.champsimtrace.xz

Replacement	IPC	Speedup	Accesses	Misses	L2C Miss Rate (%)
LRU	0.2125	1.0	3651912	1229395	33.66
FIFO	0.2127	1.0009	3651903	1185023	32.45
LFU	0.2134	1.0042	3651916	3266904	89.46
BIP (eps = 0)	0.21	0.9882	3651925	3059114	83.77
BIP (eps = 0.25)	0.2113	0.9944	3651932	2402836	65.8
BIP (eps = 0.5)	0.2122	0.9986	3651910	1922251	52.64
BIP (eps = 0.75)	0.2125	1.0000	3651915	1473849	40.36
BIP (eps = 1.0)	0.212	0.9976	3651929	1294073	35.44

1.5.4 bc-0.trace.gz

Replacement	IPC	Speedup	Accesses	Misses	L2C Miss Rate (%)
LRU	0.158	1.0	2878955	1897026	65.89
FIFO	0.1583	1.0019	2878979	1942143	67.46
LFU	0.1715	1.0854	2879208	1965425	68.26
BIP (eps = 0)	0.1615	1.0222	2878884	2209340	76.74
BIP (eps = 0.25)	0.1586	1.0038	2878931	2131855	74.05
BIP (eps = 0.5)	0.1577	0.9981	2878923	2041650	70.92
BIP (eps = 0.75)	0.1579	0.9994	2878952	1981097	68.81
BIP (eps = 1.0)	0.158	1.0000	2878971	1904676	66.16

1.5.5 sssp-0.trace.gz

Replacement	IPC	Speedup	Accesses	Misses	L2C Miss Rate (%)
LRU	0.3945	1.0	1609248	965982	60.03
FIFO	0.3944	0.9997	1609245	1002020	62.27
LFU	0.4112	1.042	1609227	1183252	73.53
BIP (eps = 0)	0.4084	1.0352	1609241	1029510	63.97
BIP (eps = 0.25)	0.3956	1.0028	1609229	1008991	62.7
BIP (eps = 0.5)	0.3936	0.9977	1609243	1000425	62.17
BIP (eps = 0.75)	0.3936	0.9977	1609255	987195	61.34
BIP (eps = 1.0)	0.3944	0.9997	1609253	969762	60.26

1.6 Comparison graphs

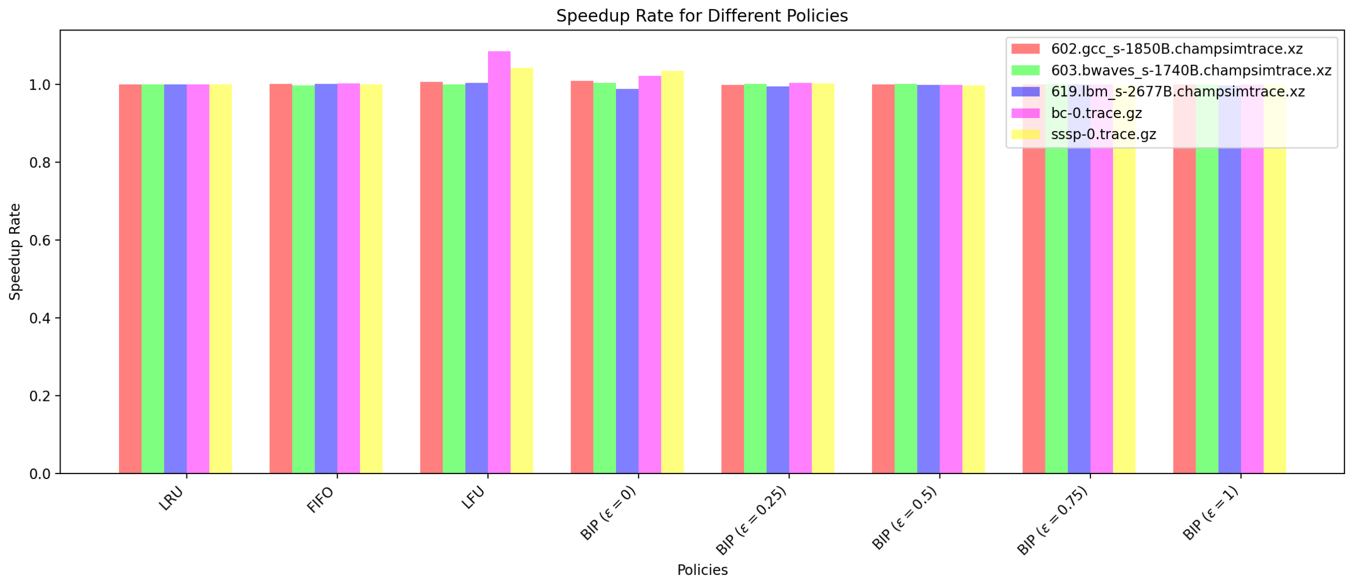

Figure 1: Speedup

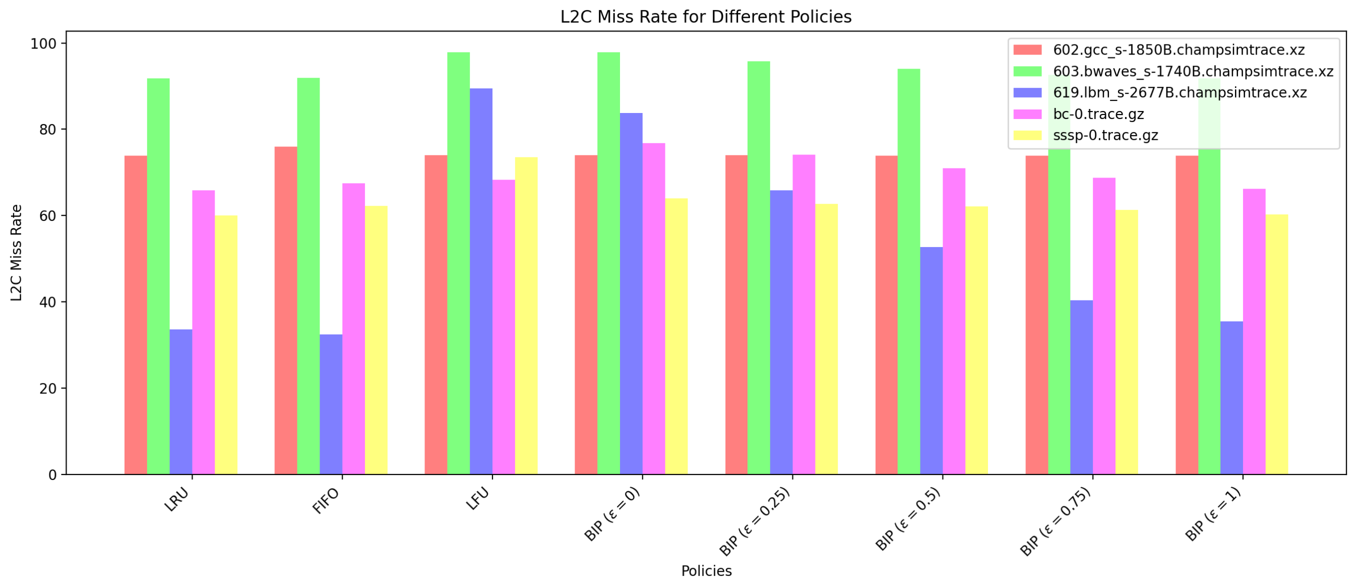

Figure 2: L2C Miss Rate

2 Part II: Data Prefetcher

The stream prefetcher implemented works as follows:

The first miss, say to cache line X, initiates a stream. The second miss to cache line X+Y (or X-Y). defines the direction of the stream in this case. The third miss, at X+Z (or X-Z) (where $Z > Y$), confirms the direction. Here Z < PREFETCH_DISTANCE. If $X- > X + Y- > X + Z$ set stream direction as 1. If $X- > X - Y- > X - Z$ set stream direction as -1. Now we mark X as start_addr and X+stream_dirn*PREFETCH_DISTANCE as end_addr. This forms a monitoring region between either (start_addr, end_addr) or (end_addr, start_addr). Whenever we get miss in this monitoring region, we prefetch lines end_addr+1, end_addr+2, ... end_addr+PREFETCH_DEGREE or end_addr-1, end_addr-2, ... end_addr-PREFETCH_DEGREE depending on the stream direction. Also now we move monitoring region to either (start_addr+PREFETCH_DEGREE, end_addr+PREFETCH_DEGREE) or (end_addr-PREFETCH_DEGREE, start_addr-PREFETCH_DEGREE) depending on stream direction. I have implemented the table of monitoring regions by creating a `struct lru_table_custom` which stores atmost 64 monitoring regions, and if the table is full and if a monitoring region is to be inserted, the least recently used monitoring regio is evicted.

The prefetcher was then compared against the ip-stride prefetcher which was provided with Champsim, based on the metrics: SpeedUp, Prefetcher Accuracy, L1D MPKI and L2C Load MPKI.

2.1 Measurements for Different Traces

2.1.1 602.gcc_s-1850B.champsimtrace.xz

Prefetcher	**Speedup**	**Prefetcher Accuracy**	**L1D MPKI**	**L2C Load MPKI**
No	1.0	-	365.85	742.09
IP Stride	1.0505	39.48%	326.45	674.55
Stream	1.0529	31.12%	277.97	378.93

2.1.2 603.bwaves_s-1740B.champsimtrace.xz

Prefetcher	**Speedup**	**Prefetcher Accuracy**	**L1D MPKI**	**L2C Load MPKI**
No	1.0	-	172.86	949.28
IP Stride	1.8033	44.58%	139.38	473.69
Stream	1.0388	97.54%	169.68	906.81

2.1.3 619.lbm_s-2677B.champsimtrace.xz

Prefetcher	**Speedup**	**Prefetcher Accuracy**	**L1D MPKI**	**L2C Load MPKI**
No	1.0	-	420.23	1000.0
IP Stride	0.9915	41.58%	418.0	757.25
Stream	1.0847	34.23%	421.03	595.37

2.1.4 bc-0.trace.gz

Prefetcher	**Speedup**	**Prefetcher Accuracy**	**L1D MPKI**	**L2C Load MPKI**
No	1.0	-	292.55	823.84
IP Stride	1.0715	31.18%	274.48	708.6
Stream	1.0019	17.39%	328.52	885.77

2.1.5 sssp-0.trace.gz

Prefetcher	Speedup	Prefetcher Accuracy	L1D MPKI	L2C Load MPKI
No	1.0	-	327.96	893.61
IP Stride	1.0253	27.9%	326.06	845.76
Stream	1.0185	21.16%	291.14	775.91

2.2 Comparison Graphs

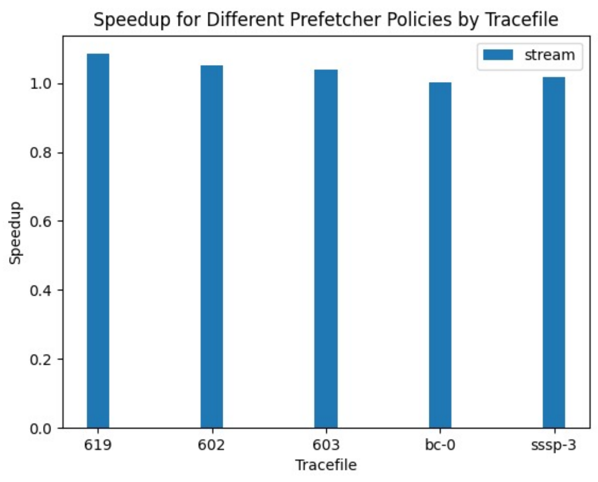

Figure 3: Speedup - Stream prefetcher

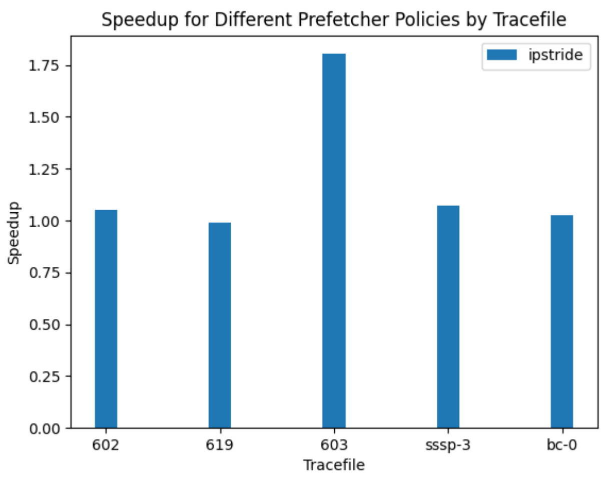

Figure 4: Speedup - IP Stride

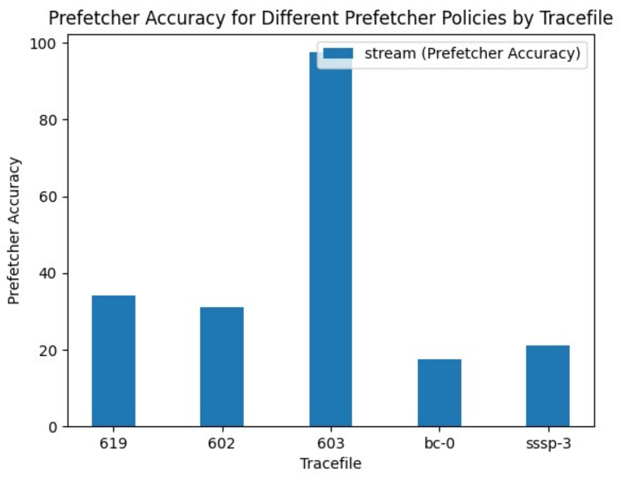

Figure 5: Prefetcher Accuracy - Stream prefetcher

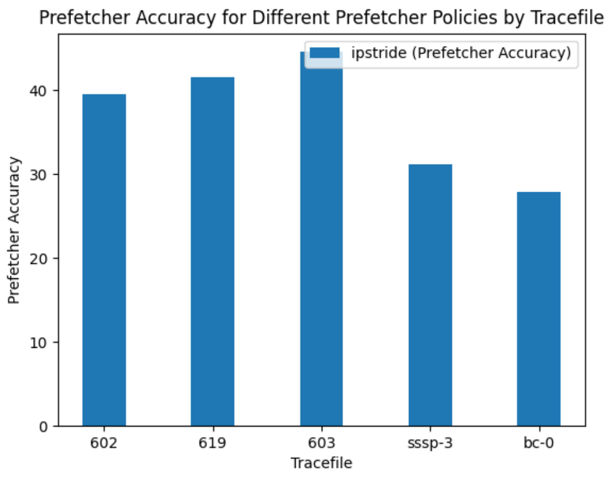

Figure 6: Prefetcher Accuracy - IP Stride

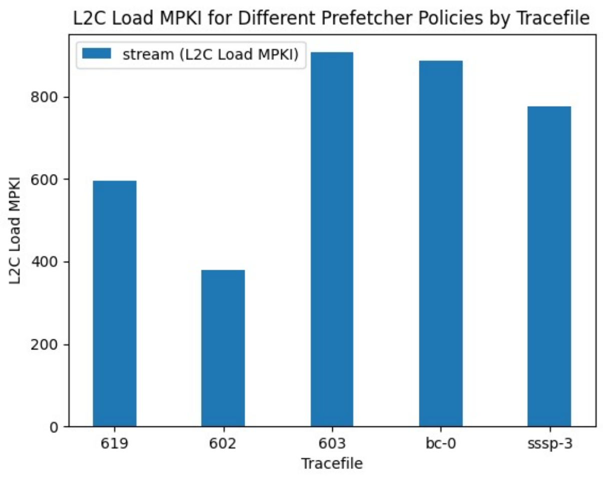

Figure 7: L1D MPKI - Stream prefetcher

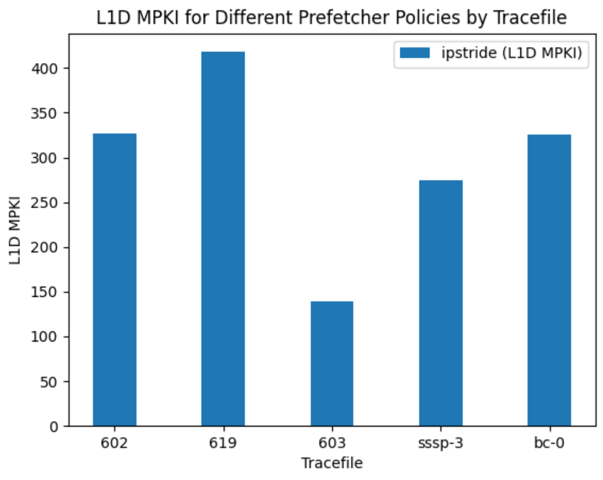

Figure 8: L1D MPKI - IP Stride

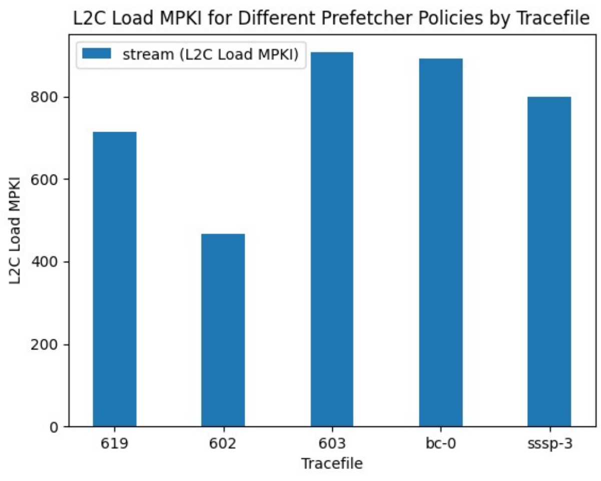

Figure 9: L2C Load MPKI - Stream Prefetcher

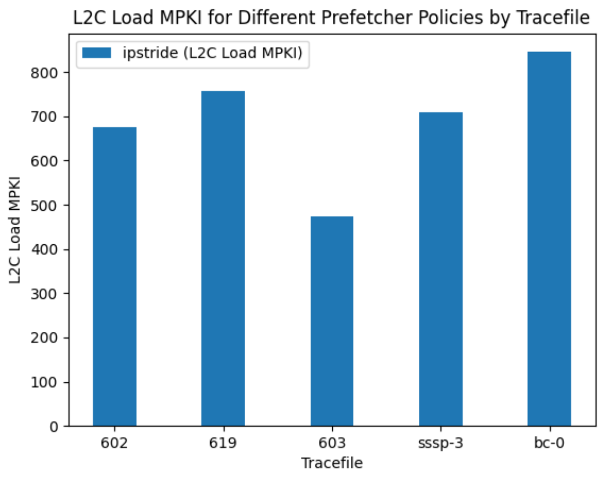

Figure 10: L2C Load MPKI - IP Stride