Memory / Cache Hierarchy optimizations for Graph Analytics

Masada Jaswanthi	210050095
Sanapati Hasini	210050140
Soupati Sri Nithya	210050152

To improve cache performance:

- Evaluate cache sizes: Experiment with different cache sizes for each level of the cache hierarchy, including L1, L2,LLC caches and compare the performance against the baseline cache hierarchy.

- Evaluate inclusivity: Experiment with different inclusivity options for each level of the cache hierarchy, including inclusive, non-inclusive, and exclusive.

- Evaluate replacement policies: Experiment with different replacement policies, including Least Recently Used(LRU), First In First Out(FIFO), and Random replacement policies.

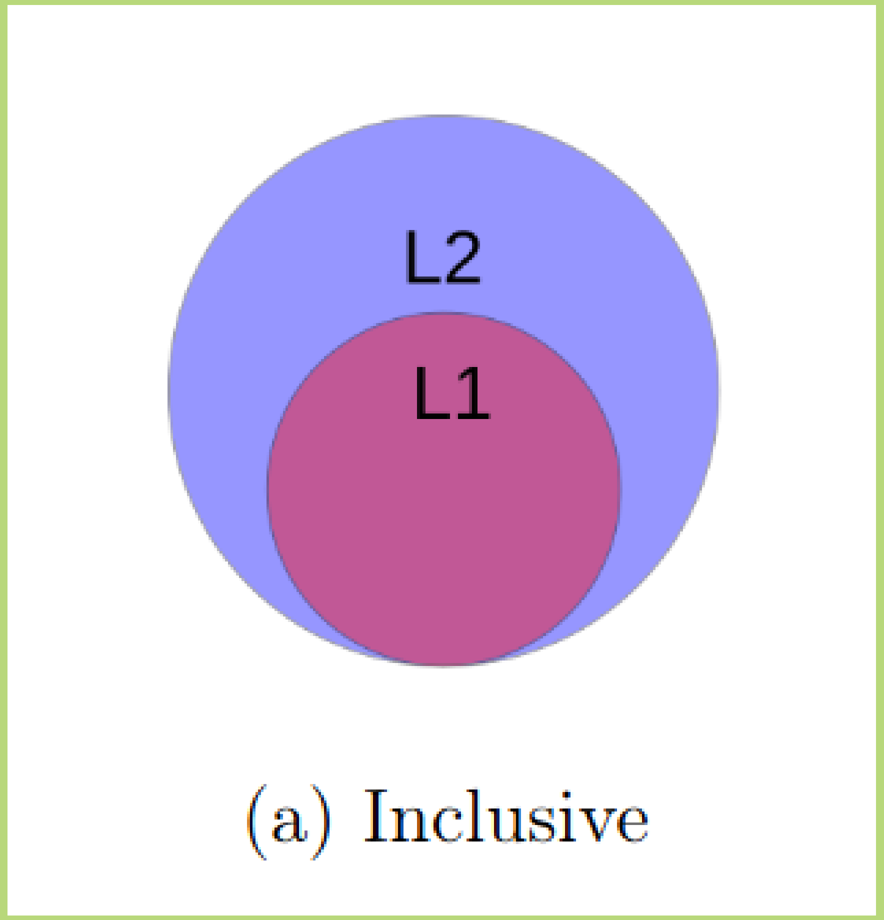
- Evaluate prefetcher: Experiment with different prefetcher settings including Key Prediction-based Cache (KPCP), Instruction Pointer Stride ,Next Line to improve the system's cache performance.

Measure the performance of each cache hierarchy and replacement policy and compare the results to the baseline cache hierarchy to see which configuration performs the best and optimize.

Inclusivity Policies:

- Inclusive
- Non-Inclusive
- Exclusive

Inclusive



intersection is data duplication.
(data block is replicated in both cache levels)

An inclusive cache level contains all data blocks from lower levels plus some other blocks.

"In a 2-level cache hierarchy, the data block will be placed in both cache levels on an L2 miss. If the block is evicted from the L1 and, later, a request comes (L1 miss), the data may still be in the L2, thus avoiding accessing main memory. On an L1 eviction, only write backs of dirty blocks are required. If the block is clean, there is no need to copy it back to the L2 because it is already there as per the inclusion policy. A potential problem is on an L2 eviction: to preserve inclusivity, if the block was present in L1, it must be evicted too."

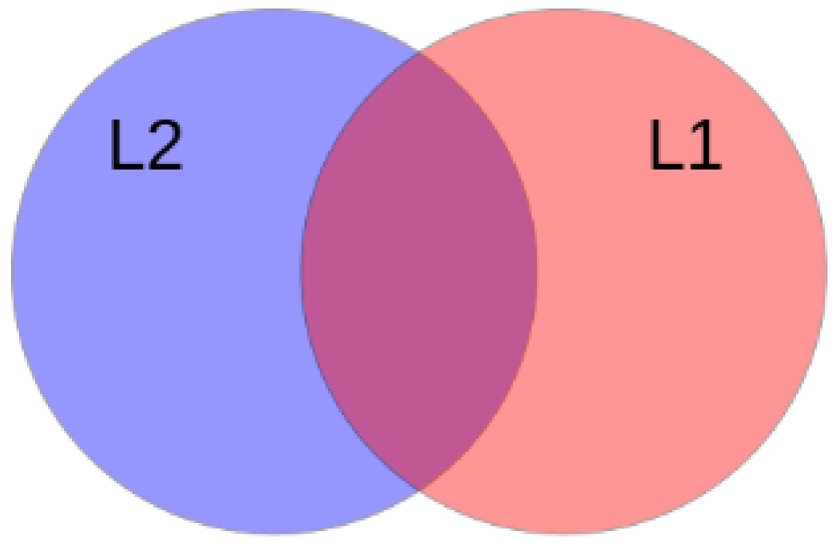
Inclusive caches reduce the number of memory accesses, but they require more cache space.

Disadvantages:

-effective cache size due to data duplication.
The effective size of the cache hierarchy is the size of the LLC.

-back invalidations.
An eviction from the LLC can generate an invalidation in an L1.

Non-Inclusive



(b) Non-inclusive

The data is replicated when there is a miss in a cache level (intersection is data duplication)

A non-inclusive cache level may or may not contain blocks from lower levels.The data is replicated when there is a miss in a cache level, and the block is allocated in that cache level and all higher ones.

"For example, in an L1 miss where the block is in none of the caches, the block will be allocated in L2 and L1. The difference with an inclusive cache is that the inclusivity is not enforced, when a block is evicted from a higher level, it does not generate back invalidations to the lower levels. This simplifies the implementation of this type of caches."

Non-inclusive caches save cache space, but they increase the number of memory accesses.

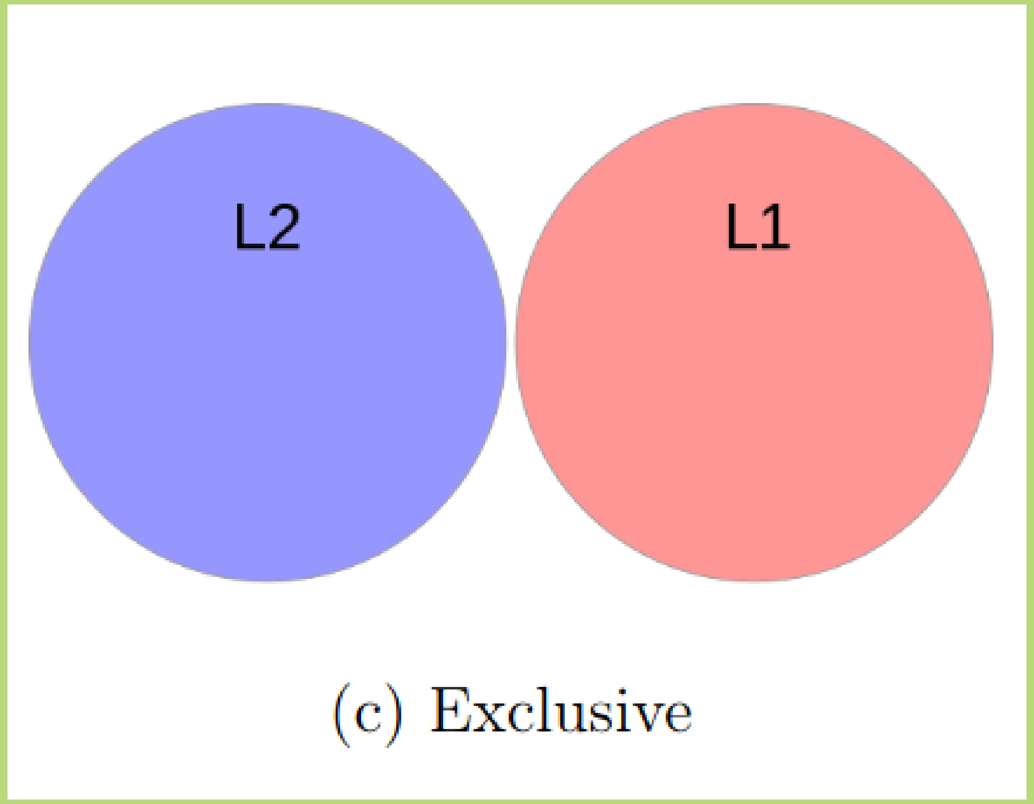
Advantage:

- The effective cache size is higher compared to an inclusive cache.

can be gained by forcing non-inclusion are a higher effective cache and lower conflict misses

"In the best case scenario, the effective cache size is the sum of all caches (when all cache blocks present in the L1 have been replaced in the L2) However, the worst-case scenario is when none of the L1 blocks have been evicted from the L2, equivalent to an inclusive cache."

Exclusive



(c) Exclusive

no data replication

An exclusive cache does not include any replicated block from lower level (increases the total amount of data blocks that can fit in the whole cache hierarchy)

"In the two cache levels (L1 and L2), when a block that is in L1 (and not in L2) is evicted, it will be allocated in L2. When the block is accessed again, it will be invalidated in L2 and allocated in L1. This generates more work to do on an L2 hit. Also, it makes it impossible to use the recency of a block to choose which block to replace when the L2 cache is full, as it only contains data that was evicted from L1 and not accessed again since that eviction."

Exclusive caches offer the best cache performance but require more hardware resources.

Benefits of exclusive caching:

-the extra space of not duplicating the data in the two levels of cache and a higher associativity in the LLC is indeed beneficial, especially for smaller lower-level caches

-Evaluating the performance of exclusive cache hierarchies with respect to inclusive caches

Cache Replacement Policies :

- Least recently used (LRU)
- Least frequently used (LFU)
- First In First Out (FIFO)
- Dynamic Re-Reference Interval Prediction (DRRIP)

Evaluated them to determine which policy helps best for optimizing the cache hierarchy and improving the performance

Least Recently Used (LRU) :

This policy removes the least recently used item from the cache (as the items that haven't been used recently are less likely to be needed again soon)

Least Frequently Used (LFU) :

This policy removes the item that has been used the least number of times (items that have not been used frequently will be removed first)

First-In-First-Out (FIFO) :

This policy removes the oldest item in the cache to make room for new items. (the item that has been in the cache for the longest time is removed)

Dynamic Re-Reference Interval Prediction (DRRIP):

This policy aims to improve upon the accuracy of the Least Recently Used (LRU) policy and also uses a prediction mechanism to estimate the future reuse distance of a block in the cache.

Prefetching Techniques :

- Next Line Prefetcher
- Instruction Pointer Stride Prefetcher
- Key Prediction-based Cache Prefetcher (KPCP)

Evaluated them to determine which prefetcher helps best for optimizing the cache hierarchy and improving the performance

Key Prediction-based Cache Prefetcher (KPCP) :

KPCP uses a history of previous cache accesses to identify a pattern or key, which is then used to prefetch the data before it is actually needed. uses a cache history table to store information about previous cache accesses, identifying patterns in the access history, and generating a prediction key for future cache accesses.

Instruction Pointer Stride Prefetcher:

The prefetcher uses the Instruction Pointer(IP) to detect a fixed pattern of instruction addresses and prefetches the next instruction based on the fixed stride distance. When the processor detects a fixed pattern of instruction addresses in the IP, it prefetches the next instruction in memory at a fixed distance from the current instruction. (IP is a register that stores the address of the currently executing instruction)

Next Line Prefetcher:

The prefetcher predicts the next line of code based on the current instruction being executed, fetches it from memory, and stores it in a cache for quick access. When the next instruction is executed, the prefetcher checks if the predicted line of code is already in the cache and uses it instead of fetching it again from memory.

- We have considered three traces namely bfs-8, bc-0, bc-5.

- Two kinds of plots for each trace

  - a) Base prefetcher (No prefetcher)

"Compared the effect of different replacement policies on the baseline hierarchy with the base being no prefetcher" i.e (over cache size, inclusivity, replacement policies)

  - b) Base Replacement policy (LRU)

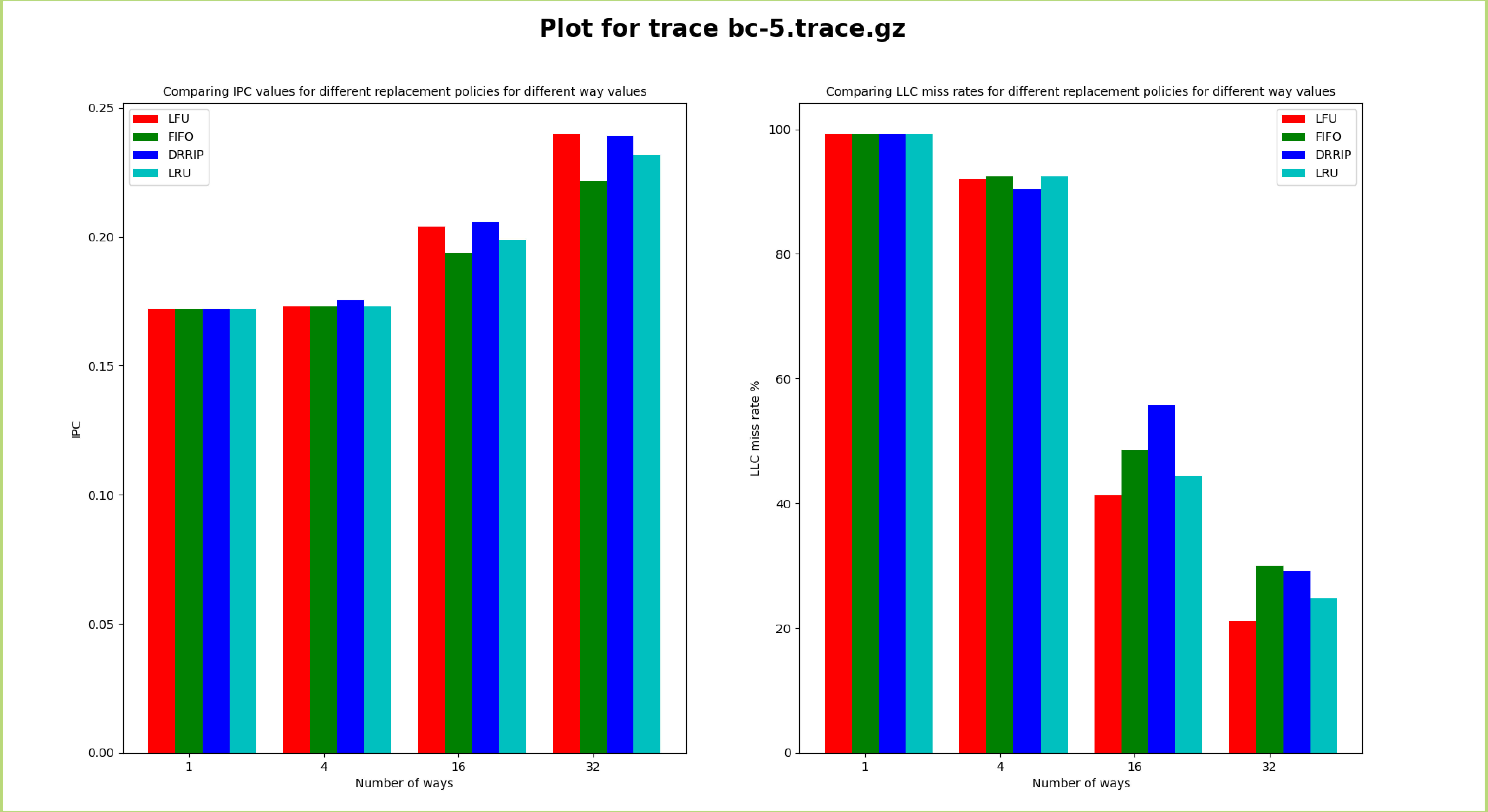
"In our experiment, we considered lru as the base and compared the effect of different prefetchers on baseline hierarchy" i.e (over cache size, inclusivity, prefetchers)

Conclusions of the trace "bc-5":

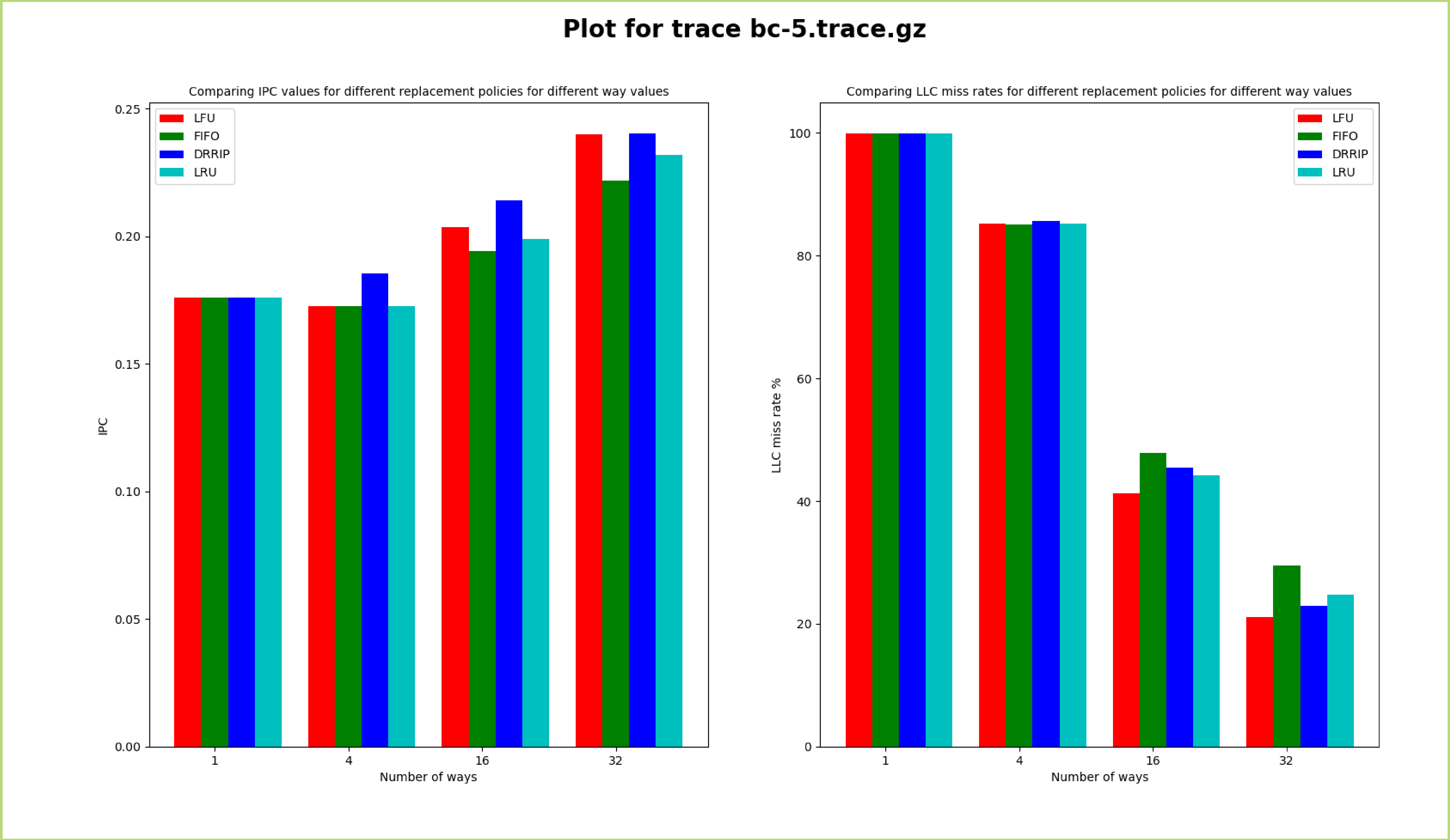
- For inclusive and non-inclusive caches, it is observed that the best replacement policy across the different combinations taken is LFU and best prefetcher being "IP stride prefetcher"

- For exclusive caches, it is observed that the best replacement policy across the different combinations taken are LRU, FIFO (almost similar) with no prefetcher.

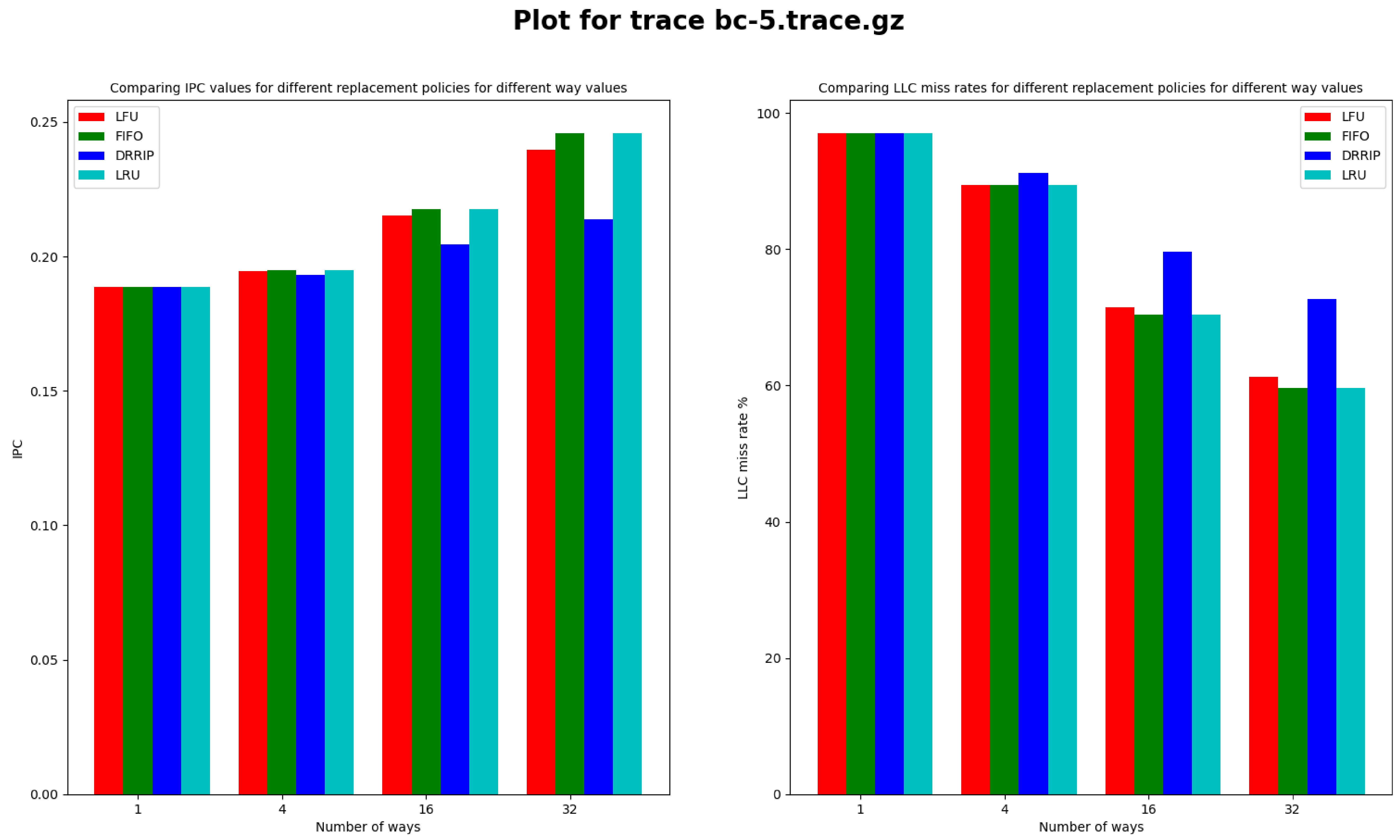
a) Non- Inclusive



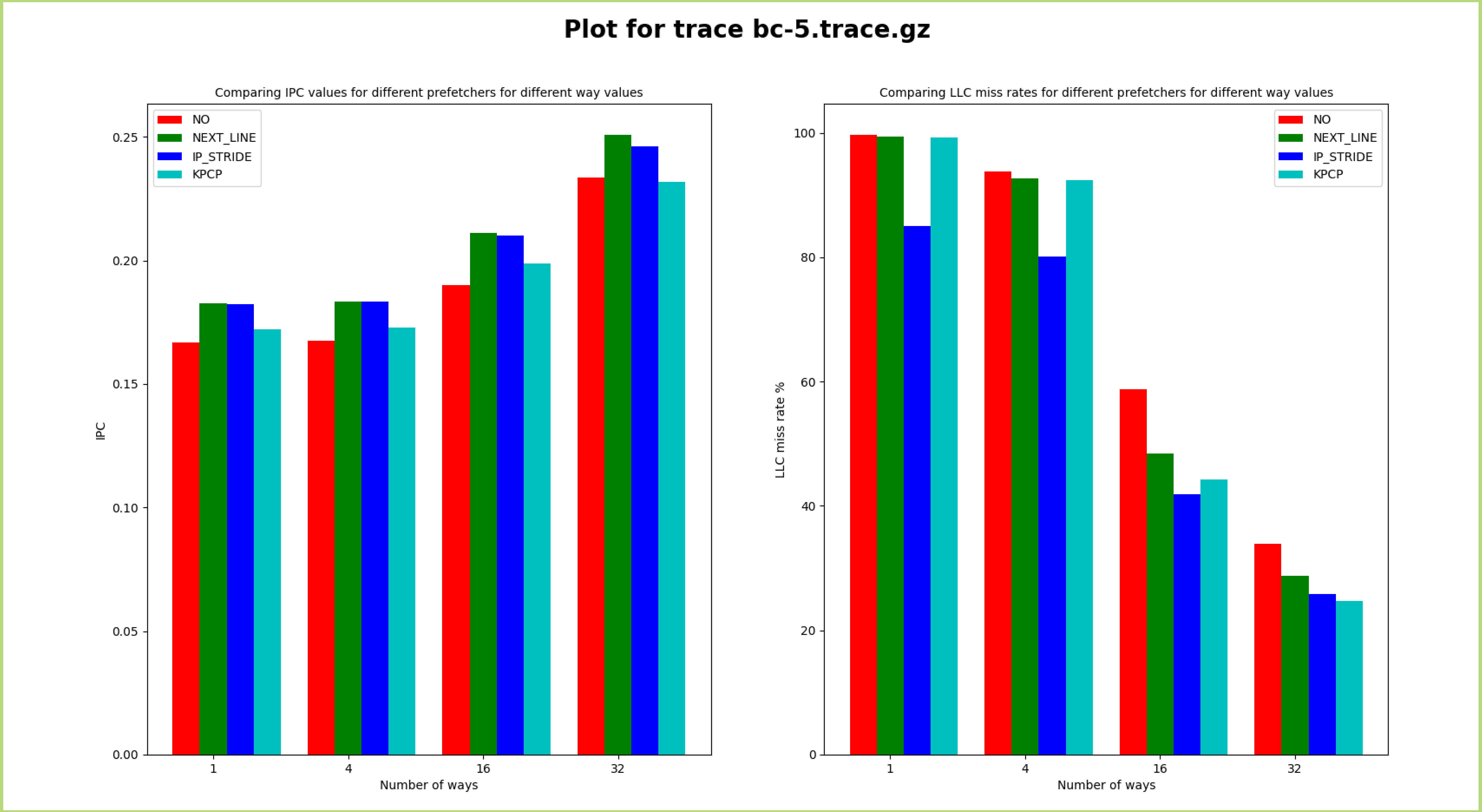
a) Inclusive


Plot for trace bc-5.trace.gz
Comparing IPC values for different replacement policies for different way values
IPC
0.00
0.05
0.10
0.15
0.20
0.25
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU
Comparing LLC miss rates for different replacement policies for different way values
LLC miss rate %
0
20
40
60
80
100
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU

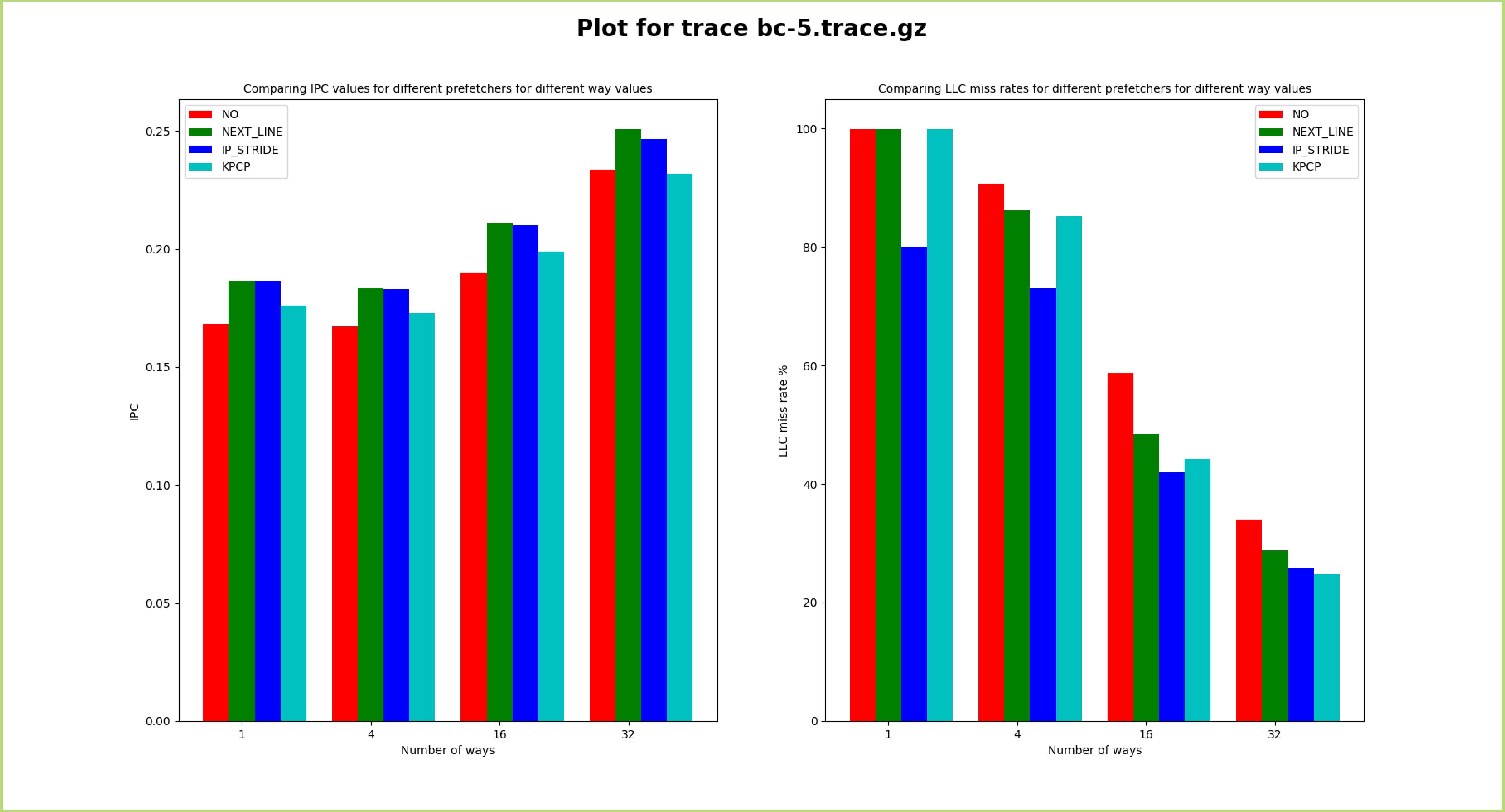
a) Exclusive


Plot for trace bc-5.trace.gz
Comparing IPC values for different replacement policies for different way values
LFU
FIFO
DRRIP
LRU
IPC
Number of ways
0.00
0.05
0.10
0.15
0.20
0.25
1
4
16
32
Comparing LLC miss rates for different replacement policies for different way values
LLC miss rate %
0
20
40
60
80
100

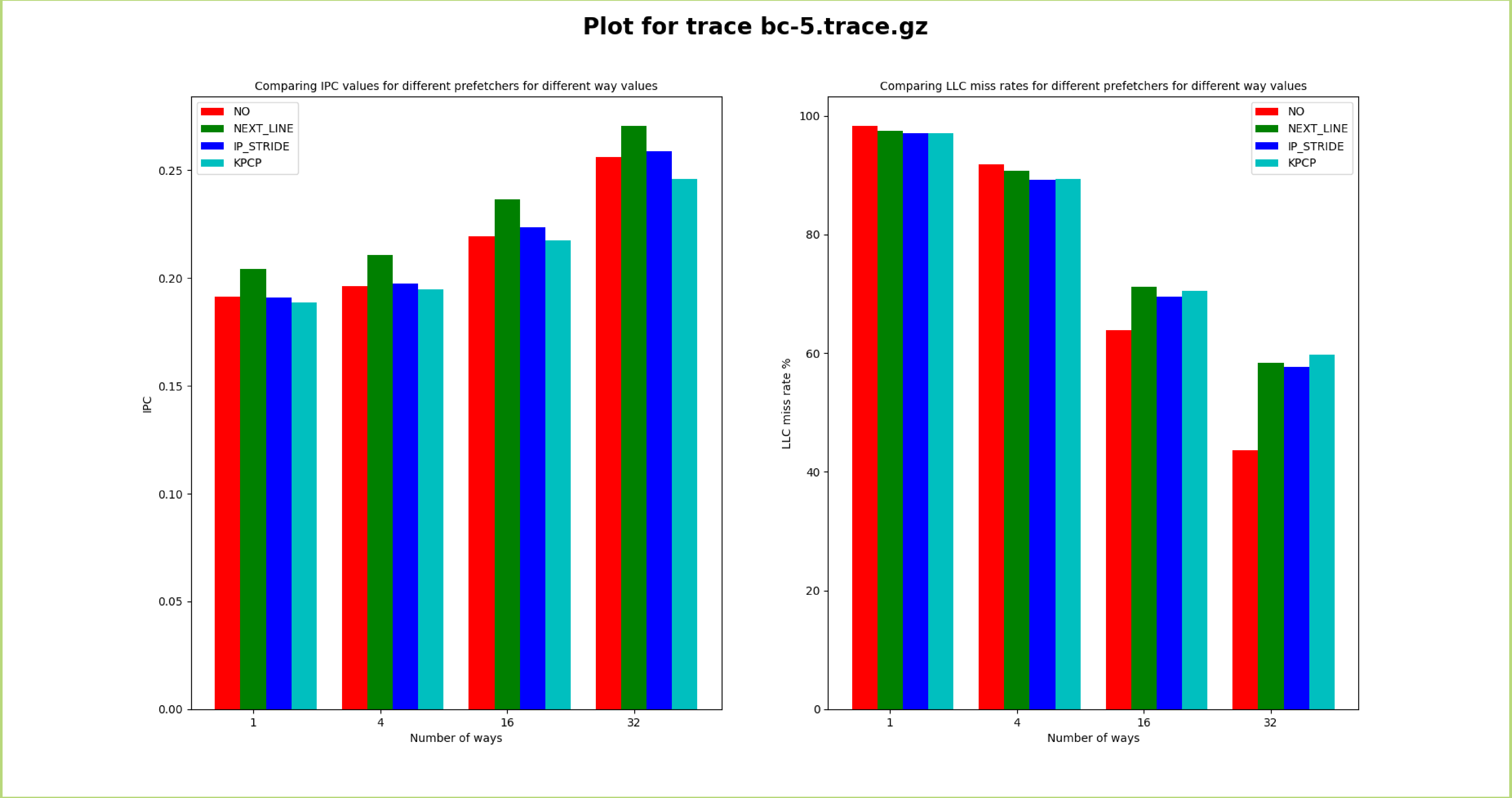
b) Non- Inclusive



b) Inclusive


Plot for trace bc-5.trace.gz
Comparing IPC values for different prefetchers for different way values
NO
NEXT_LINE
IP_STRIDE
KPCP
IPC
0.00
0.05
0.10
0.15
0.20
0.25
1
4
16
32
Number of ways
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100
Number of ways

b) Exclusive

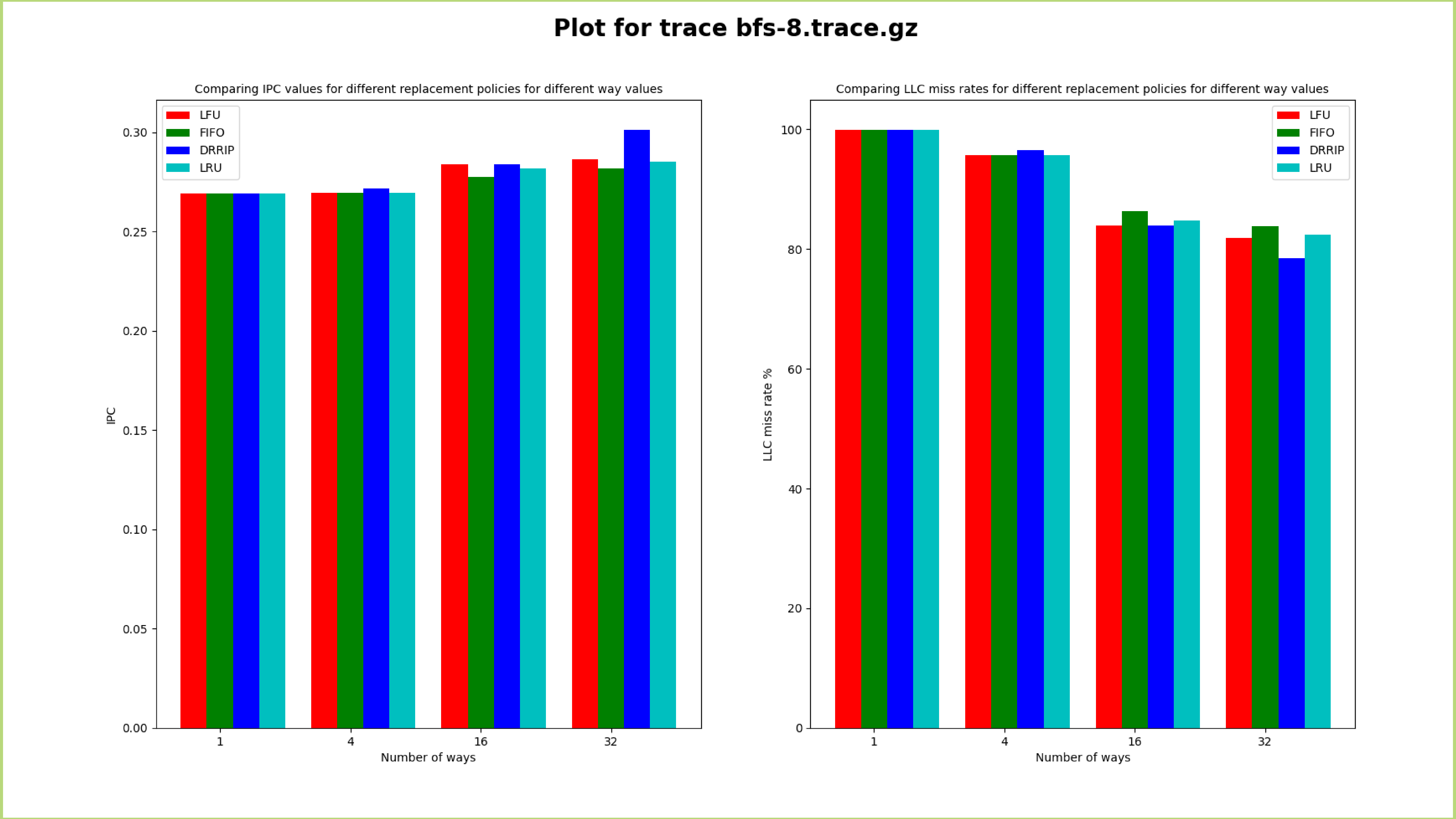

Plot for trace bc-5.trace.gz
Comparing IPC values for different prefetchers for different way values
NO
NEXT_LINE
IP_STRIDE
KPCP
IPC
0.00
0.05
0.10
0.15
0.20
0.25
1
4
16
32
Number of ways
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100

Conclusions of the trace "bfs-8":

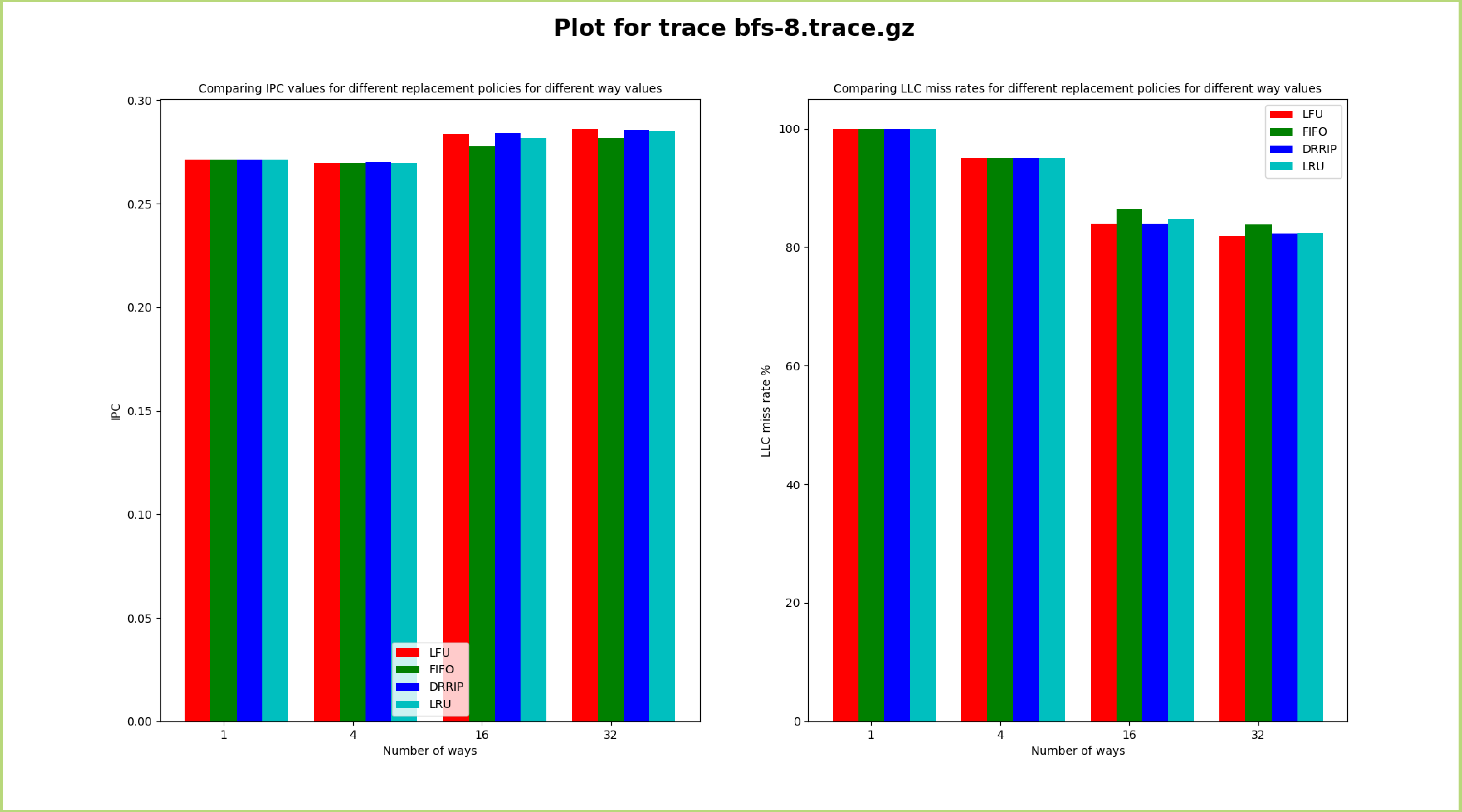
- For inclusive and non-inclusive caches, it is observed that the best replacement policy across the different combinations taken is "LFU" and the best prefetcher being "IP stride prefetcher".

- For exclusive caches, it is observed that the best replacement policy across the different combinations taken is "LRU" and the best prefetcher being "no prefetcher".

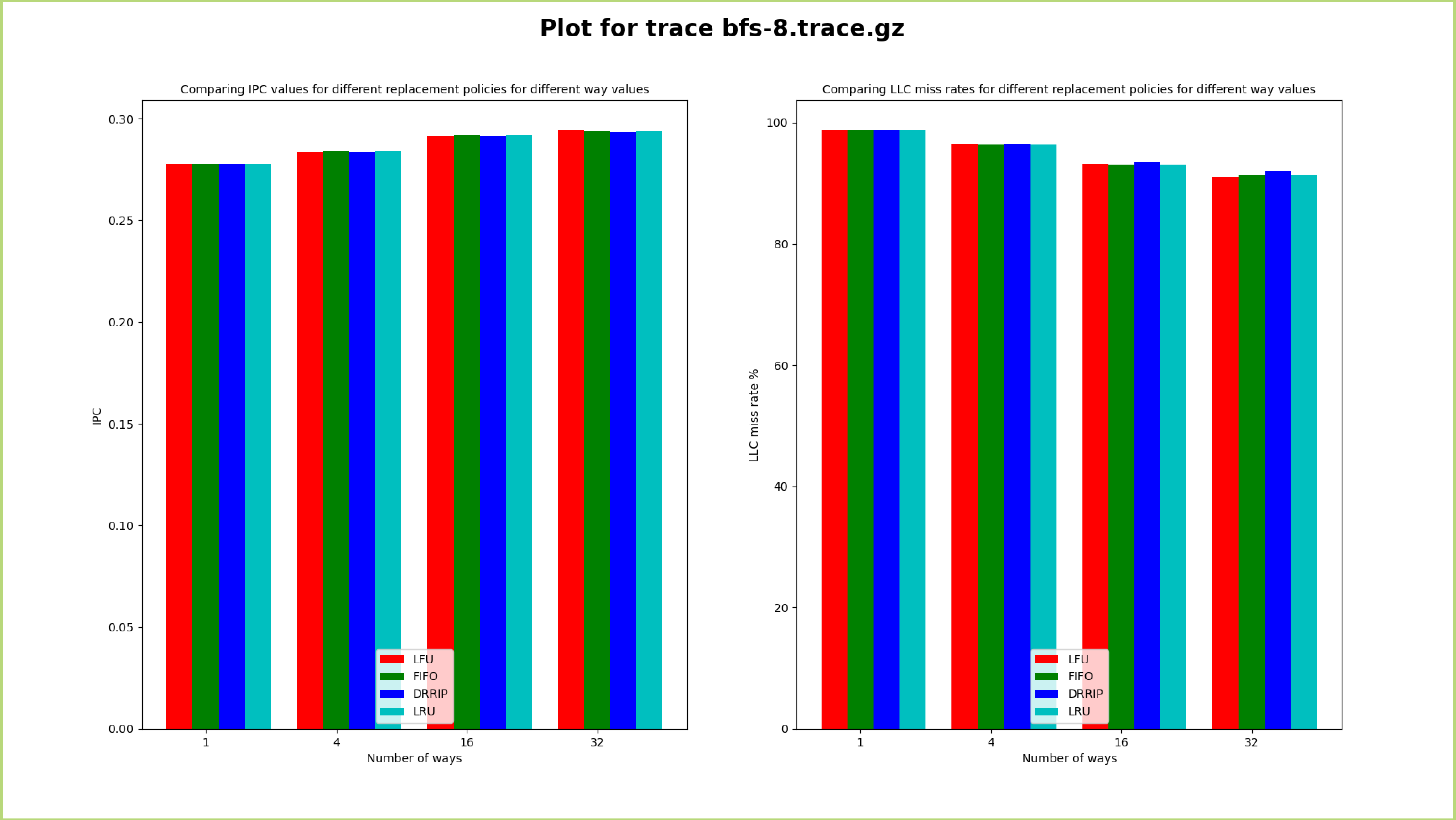
a) Non- Inclusive



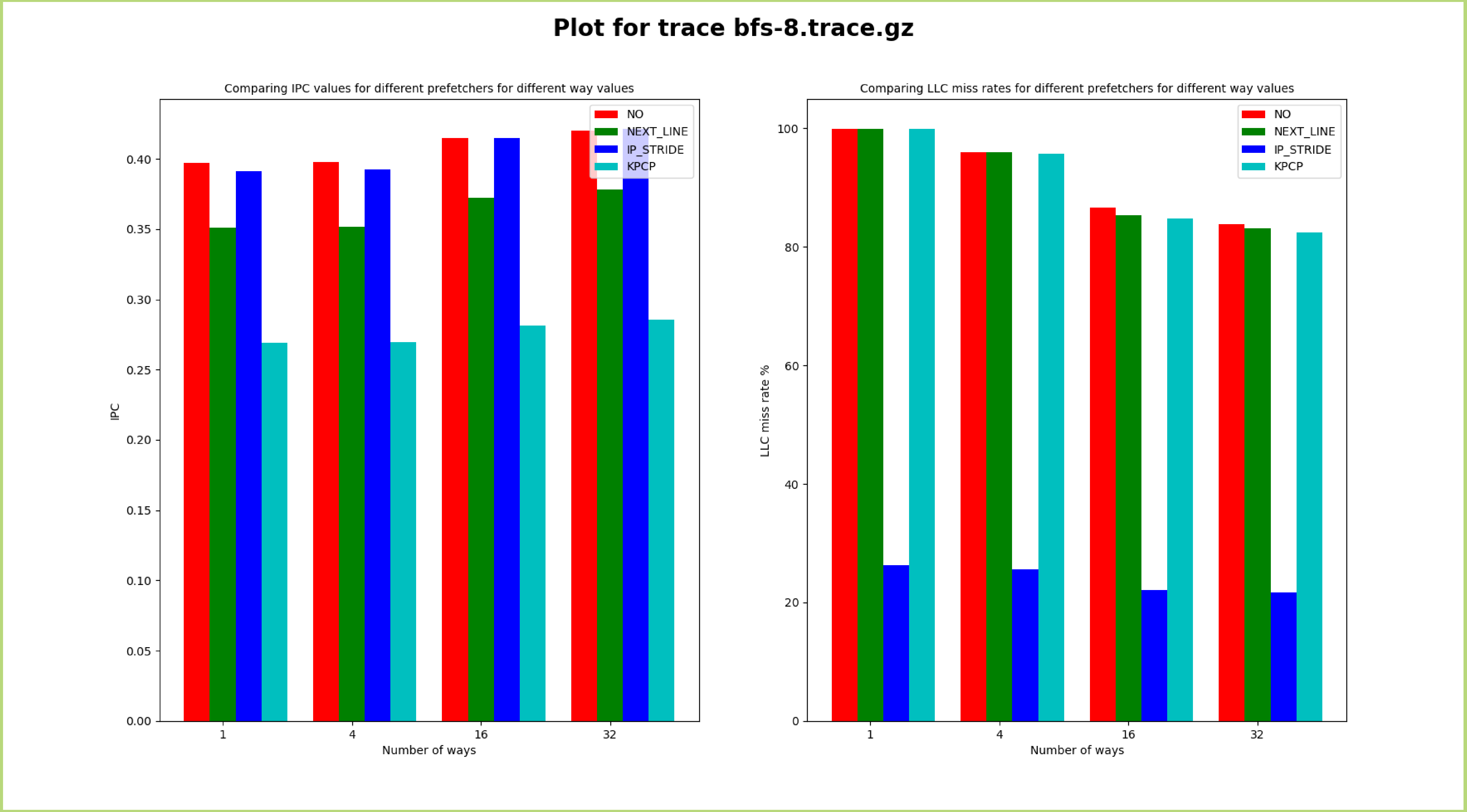
a) Inclusive


Plot for trace bfs-8.trace.gz
Comparing IPC values for different replacement policies for different way values
IPC
0.00
0.05
0.10
0.15
0.20
0.25
0.30
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU
Comparing LLC miss rates for different replacement policies for different way values
LLC miss rate %
0
20
40
60
80
100
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU

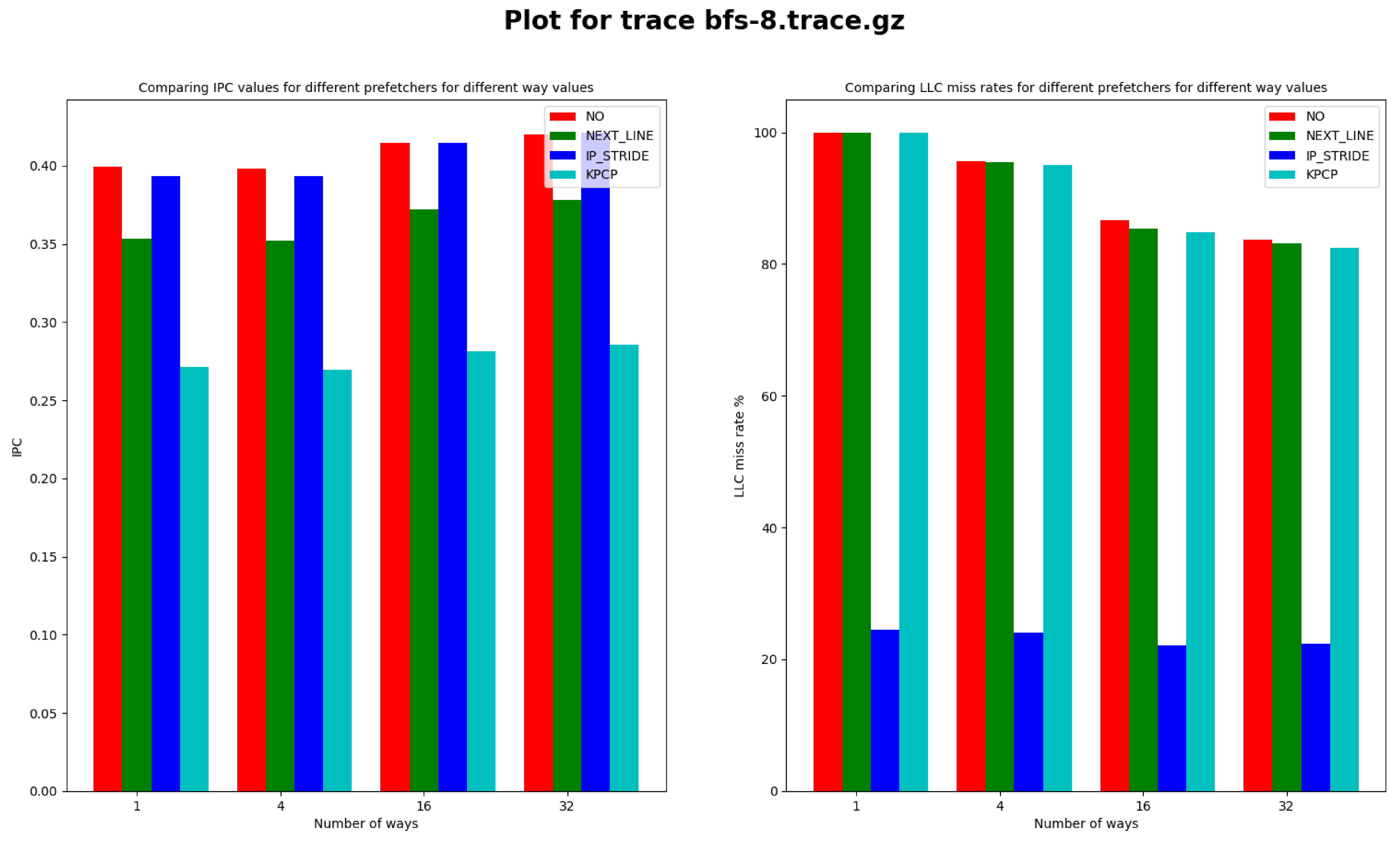
a) Exclusive


Plot for trace bfs-8.trace.gz
Comparing IPC values for different replacement policies for different way values
IPC
0.00
0.05
0.10
0.15
0.20
0.25
0.30
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU
Comparing LLC miss rates for different replacement policies for different way values
LLC miss rate %
0
20
40
60
80
100
1
4
16
32
Number of ways
LFU
FIFO
DRRIP
LRU

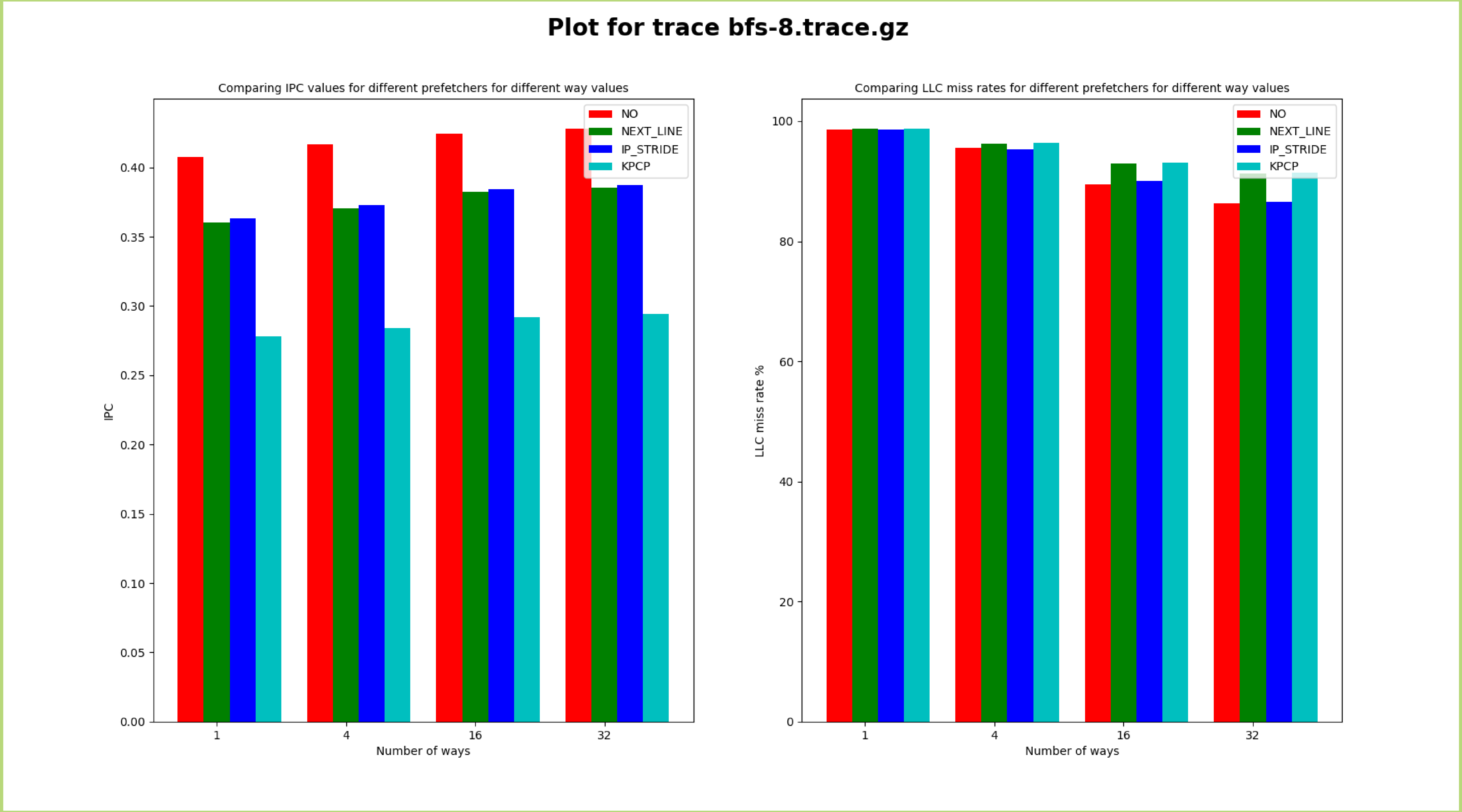
b) Non- Inclusive



b) Inclusive


Plot for trace bfs-8.trace.gz
Comparing IPC values for different prefetchers for different way values
IPC
Number of ways
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
1
4
16
32
NO
NEXT_LINE
IP_STRIDE
KPCP
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100

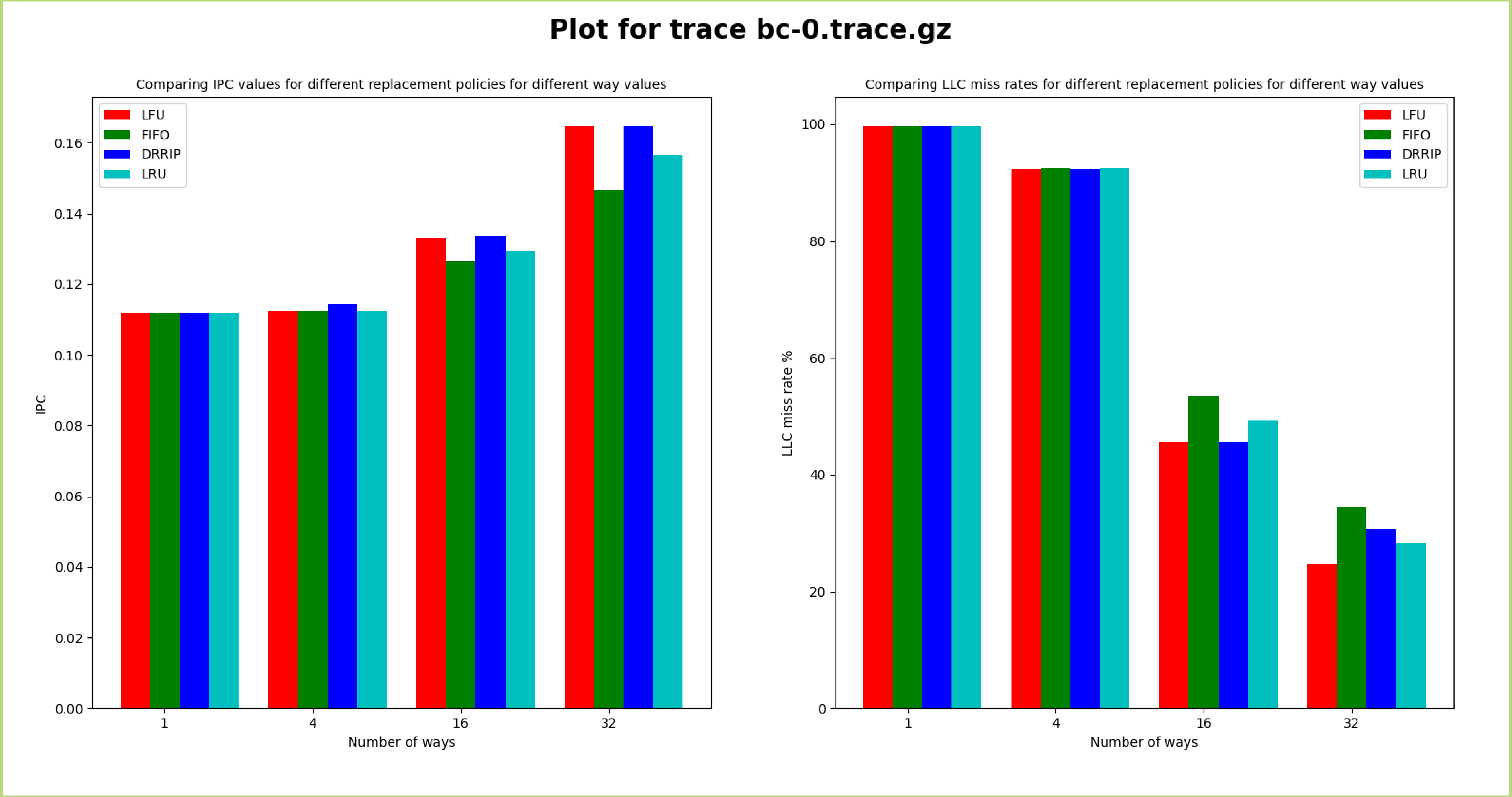
b) Exclusive


Plot for trace bfs-8.trace.gz
Comparing IPC values for different prefetchers for different way values
IPC
Number of ways
1
4
16
32
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
NO
NEXT_LINE
IP_STRIDE
KPCP
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100

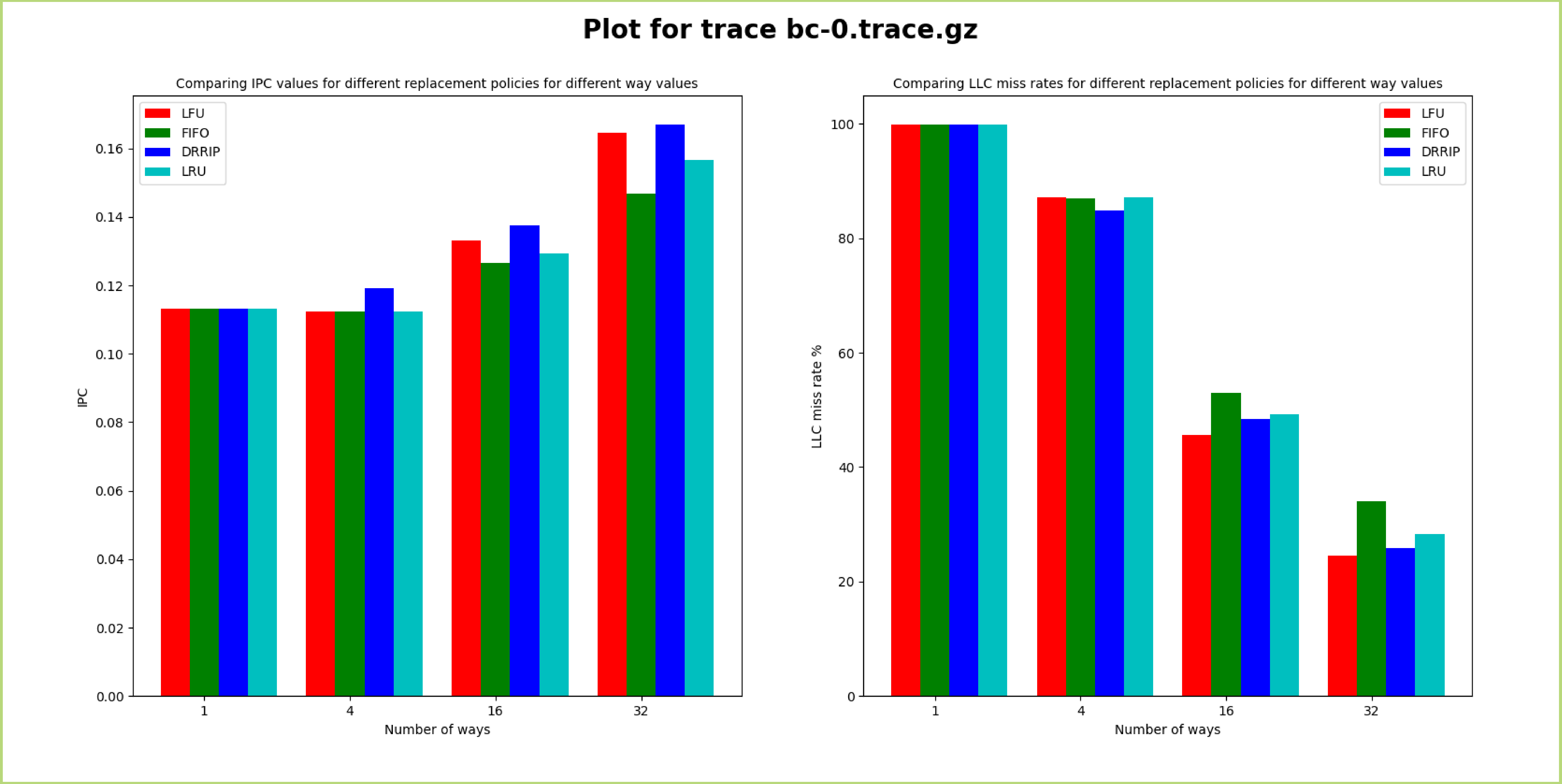
Conclusions of the trace "bc-0":

- For inclusive and non-inclusive caches, it is observed that the best replacement policy across the different combinations taken is "LFU" and the best prefetcher being "IP stride prefetcher".
- For exclusive caches, it is observed that the best replacement policy across the different combinations taken is "LRU" and the best prefetcher being "no prefetcher".

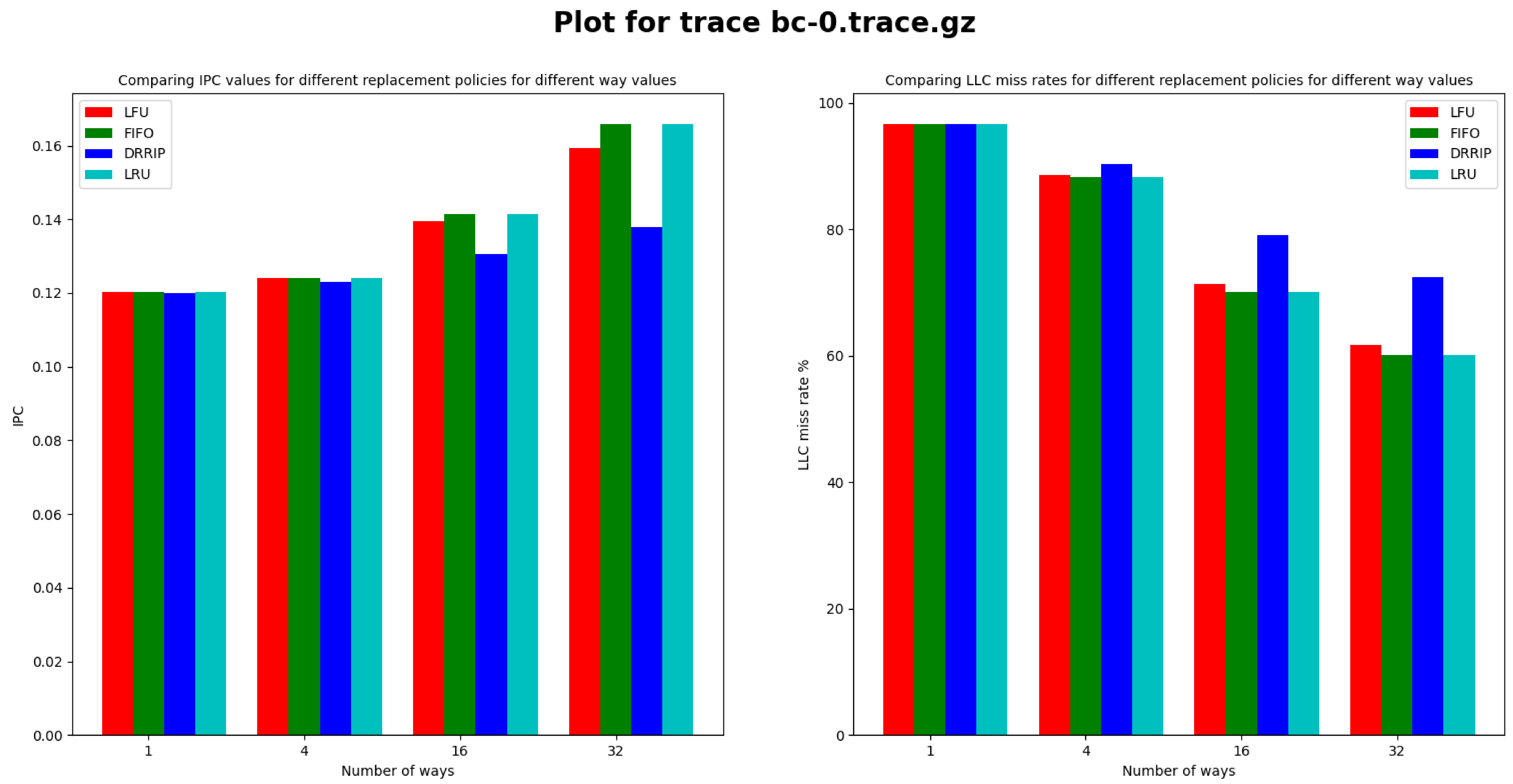
a) Non- Inclusive



a) Inclusive

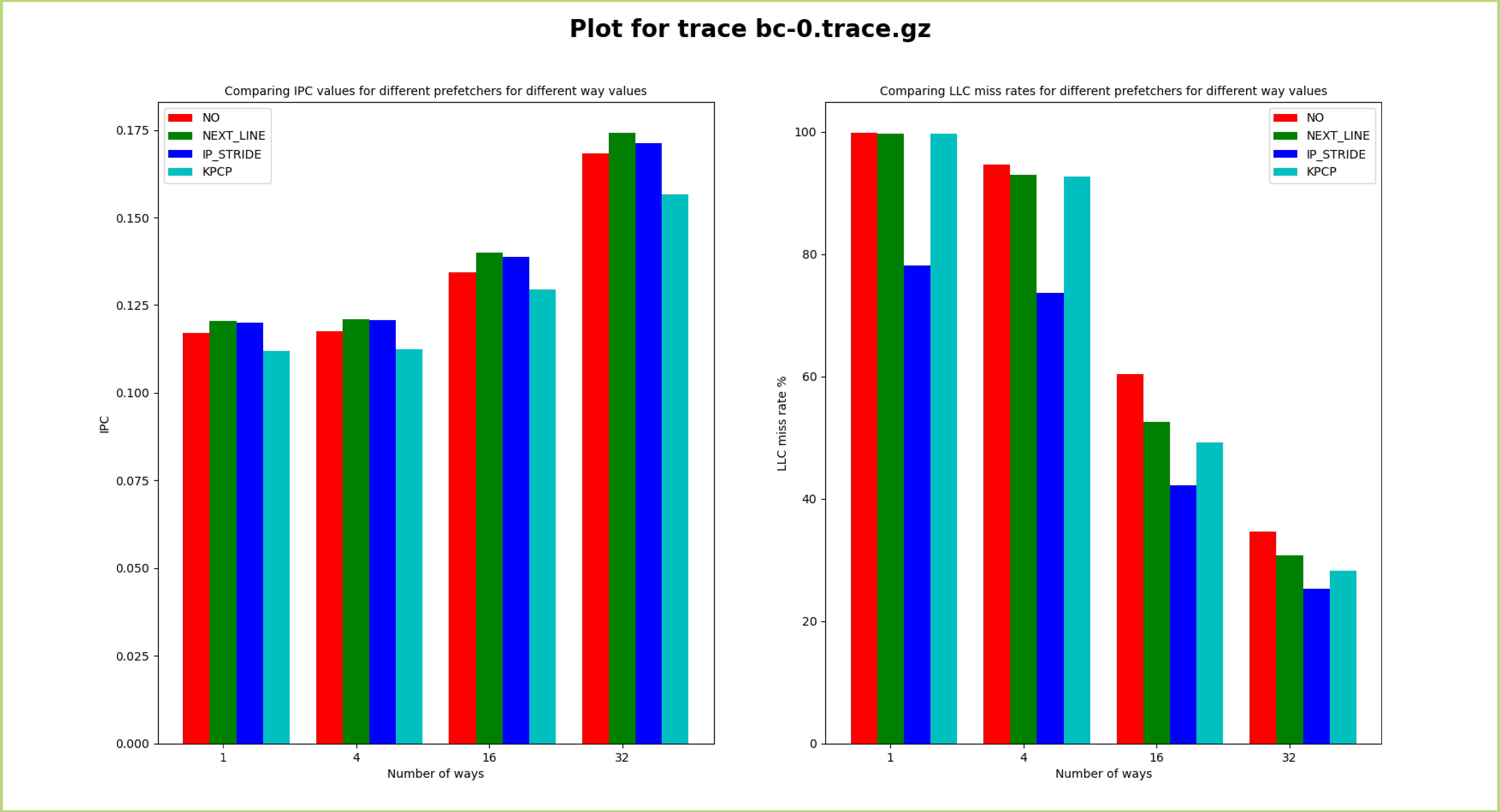

Plot for trace bc-0.trace.gz
Comparing IPC values for different replacement policies for different way values
IPC
Number of ways
LFU
FIFO
DRRIP
LRU
Comparing LLC miss rates for different replacement policies for different way values
LLC miss rate %
Number of ways

a) Exclusive

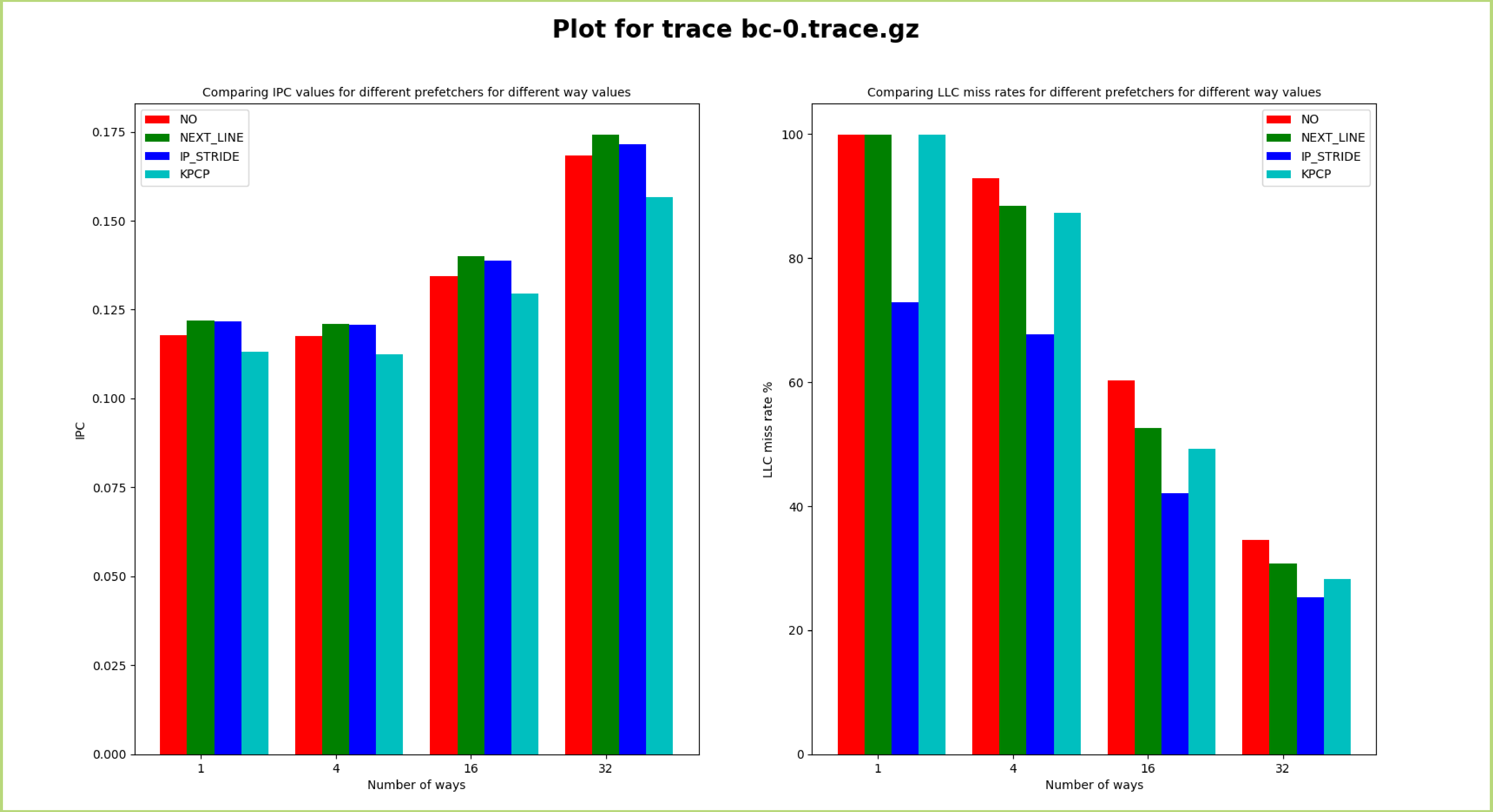


Plot for trace bc-0.trace.gz
Comparing IPC values for different replacement policies for different way values
Comparing LLC miss rates for different replacement policies for different way values
LFU
FIFO
DRRIP
LRU
IPC
LLC miss rate %
Number of ways
1
4
16
32
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
20
40
60
80
100

b) Non- Inclusive

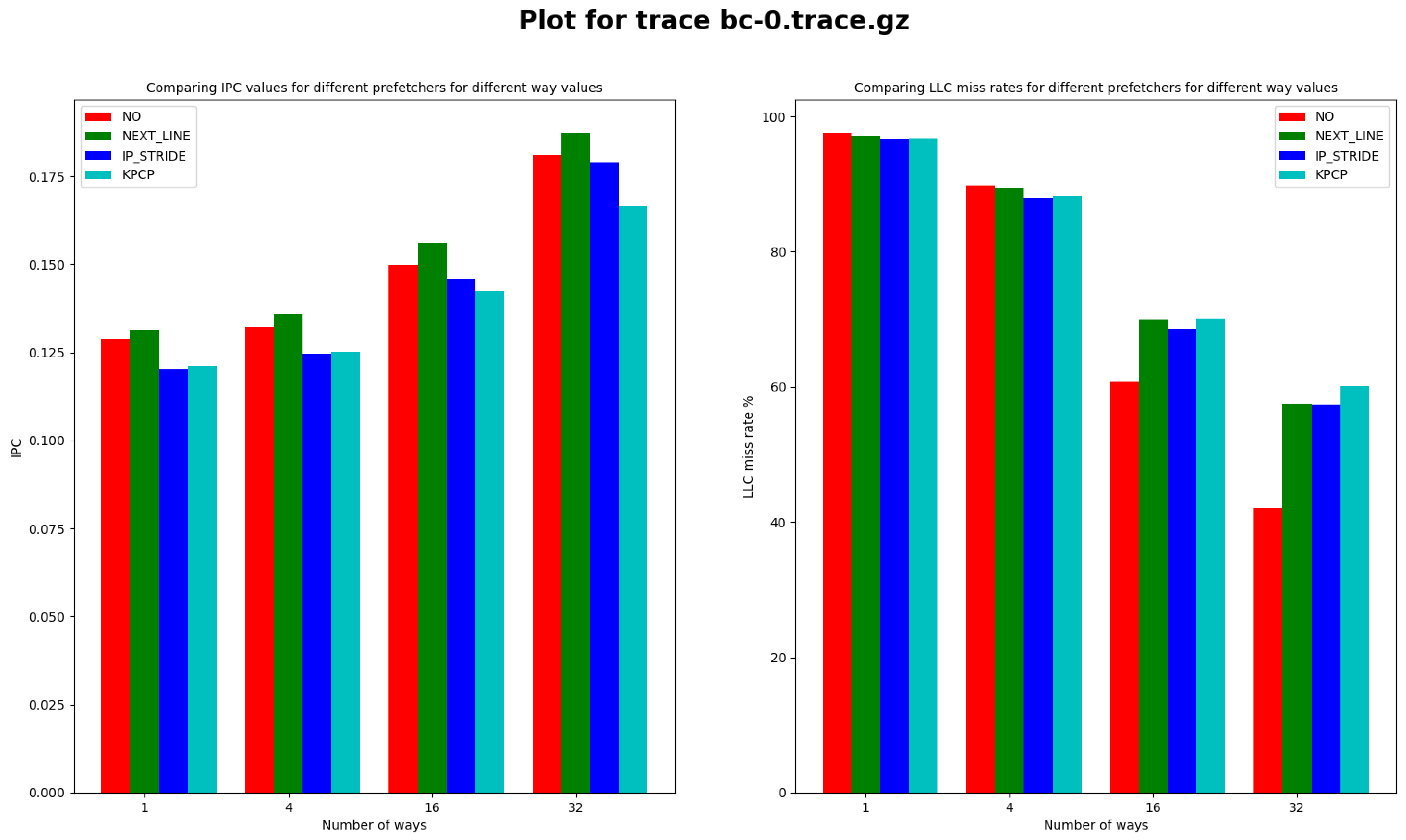


b) Inclusive



Plot for trace bc-0.trace.gz
Comparing IPC values for different prefetchers for different way values
NO
NEXT_LINE
IP_STRIDE
KPCP
IPC
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
1
4
16
32
Number of ways
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100
Number of ways

b) Exclusive


Plot for trace bc-0.trace.gz
Comparing IPC values for different prefetchers for different way values
NO
NEXT_LINE
IP_STRIDE
KPCP
IPC
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
1
4
16
32
Number of ways
Comparing LLC miss rates for different prefetchers for different way values
LLC miss rate %
0
20
40
60
80
100

Inclusions Impact:

- Best configurations for non-inclusive and inclusive caches, and overall, are those without L1 prefetcher and with the IP stride prefetcher in the L2.
- Best two configurations for both inclusive and non inclusive are using DRRIP and LFU.

- Best configuration for an exclusive cache are those without prefetcher for both L1 and L2.
- Best one uses the LRU replacement policy and the next best configurations are with FIFO, then LFU and DRRIP .

- The exclusive policy, in general, performs worse than inclusive and non-inclusive policies.

Prefetcher Impact:

- Prefetching has an impact on performance.The worst configurations are the ones without prefetching in any of the cache levels.
- Not using a prefetcher in L2 seems to get better results than using the next-line prefetcher.This might be due to interference between prefetchers.
- The best performing configurations for all inclusion types (given from the prefetcher combination) are all with IP stride prefetching.
- Benefits of using different replacement policies are shadowed by the prefetcher.

Replacement Policy Impact:

- In all the replacement policies, the inclusive and non-inclusive have more similar behavior.
- For inclusive and non-inclusive policies, the best replacement policy was different depending on the case.
- There is no clear winner among replacement policies, but there are a few patterns that indicate that the LRU replacement policy is among the best options to use in general for an exclusive cache independently of the prefetchers.
- Furthermore, exclusive caches do not show much sensitivity to the replacement policy for a prefetcher combination. In contrast, inclusive and non-inclusive are more variable.
- Among a combination of prefetchers, the LRU replacement policy gets typically similar or worse performance for many inclusive and non-inclusive cache configurations.

Key results:

- Different cache sizes affect the speedup achieved by different configurations, in our case, for exclusive caches on single core in particular. Best cache management technique can be different depending on cache size.

- Exclusive caches benefit from a larger effective capacity, so they might become more popular when the number of cores per last-level cache increases.

- For an exclusive cache, even with no prefetchers at all, LRU performs better than for the small cache in comparison to the rest of replacement policies.The single-core results demonstrate that the prefetchers and replacement policies that work best for non-inclusive and inclusive caches are not the best ones for an exclusive cache.

- The most interesting property of exclusive caches is that they increase the effective capacity of the cache.The combinations of prefetchers have a big impact on performance while in replacement policies the impact was lower, specifically for exclusive cache.

Thank you!