TERM PAPER DISCUSSION

Bespoke Cache Enclaves: Fine-Grained and Scalable Isolation from Cache Side-Channels via Flexible Set-Partitioning

GROUP 3

Wamique Zia – 2023CSM1020

PROBLEM STATEMENT

- The problem addressed in the paper is the vulnerability of shared last-level caches (LLCs) to cache side-channel attacks.
- Cache side-channel attacks exploit the shared nature of LLCs to leak sensitive information from victim programs by monitoring their cache accesses.
- These attacks can lead to serious security and privacy breaches, such as leaking encryption keys, user keystrokes, browsing history, and more.

TYPES OF ATTACKS

- **Conflict-based attacks:**
  - These attacks exploit cache set evictions to leak information shared between a victim and a spy process. Example, Prime+Probe.
- **Shared-memory-based attacks:**
  - Information leakage occurs through hits to shared cache lines between a victim and an adversary.
  - Adversaries flush specific cache lines shared with the victim, wait for the victim to reload those lines, and measure the reload time. Example, Flush+Reload.
- **Cache-occupancy based attacks:**
  - These attacks leverage changes in the victim's cache space within the shared LLC to extract information.
  - Adversaries monitor changes in cache line occupancy over time to infer information about the victim's execution behavior or data access patterns. Example, Prime+Probe with cache set contention.

RELATED WORKS

Two mostly used methods to prevent side-channel attacks are:

1. Randomized LLCs:

- Randomized LLCs involve randomizing the mapping of addresses to cache-sets to make conflict-based attacks harder.
- By introducing randomness in the cache mapping, attackers find it more challenging to predict the location of specific data in the cache, thus mitigating certain types of cache side-channel attacks.
- Randomized LLCs aim to increase the unpredictability of cache access patterns, making it more difficult for attackers to exploit cache vulnerabilities.
- This approach enhances security by reducing the effectiveness of cache side-channel attacks that rely on predicting cache access patterns.

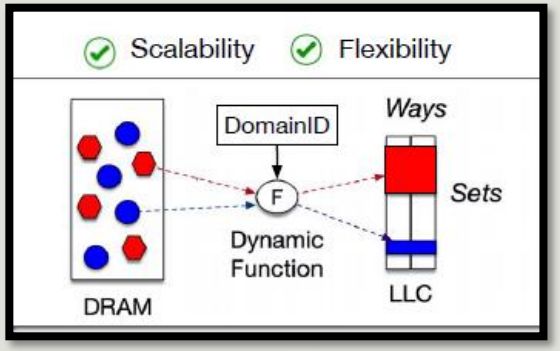
RELATED WORKS

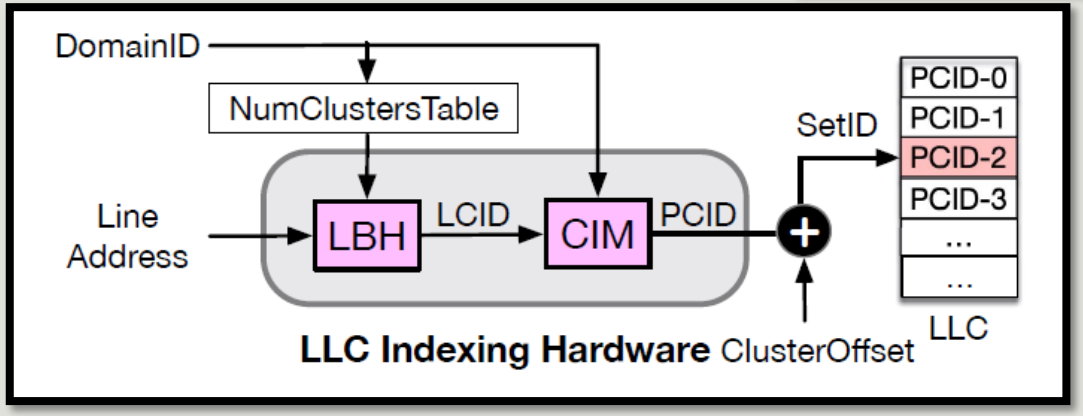
2. Cache Partitioning Techniques:

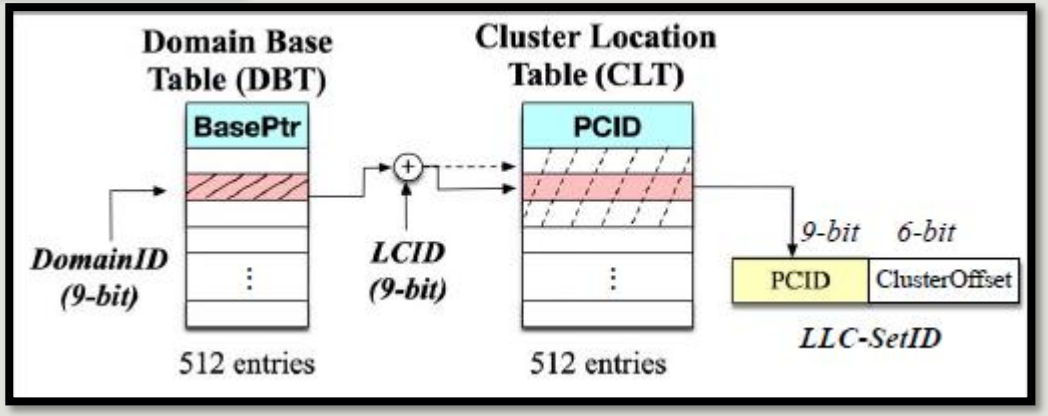
- Cache partitioning involves dividing the cache space between different processes or applications to isolate cache resources and prevent information leakage through cache side channels.
- By allocating non-overlapping cache regions to different entities, cache partitioning aims to provide strong isolation and restrict the impact of shared cache vulnerabilities.
- Cache partitioning can be implemented at different granularities, such as at the way level or page level, to tailor the level of isolation based on specific security requirements.
- Fine-grained cache partitioning allows for precise allocation of cache resources, catering to applications with varying cache working set sizes and memory requirements.
- Cache partitioning techniques provide a principled defense against cache side-channel attacks, enhancing security and privacy in shared cache environments.
- These include Way-Partitioning, Set-Partitioning, Page Coloring methods.

PROPOSED SOLUTION

The Bespoke Cache Enclaves (BCE) architecture works on Set Partitioning technique and consists of mainly two hardware modules, Cluster Indirection Module (CIM) & Load-Balancing Hash (LBH) module.







PROPOSED SOLUTION

The Bespoke Cache Enclaves (BCE) architecture consists of mainly two hardware modules,

1. Cluster-Indirection Module (CIM):

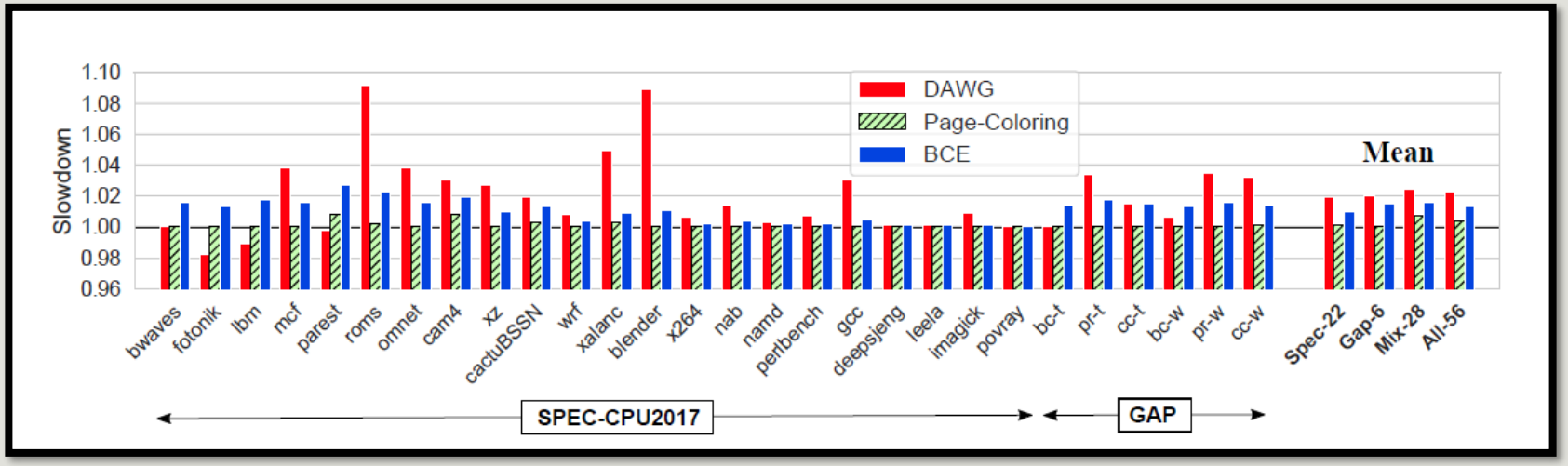
- Responsible for locating the Last-Level Cache (LLC) clusters allocated to a specific domain.
- Ensures secure isolation between clusters of different domains, maintaining data integrity and security.
- Translates Logical Cluster IDs (LCIDs) to Physical Cluster IDs (PCIDs) in the LLC, enabling efficient access to cache sets.
- Utilizes a two-level indirection mechanism to perform the translation in a constant-time manner, preventing timing side-channels.
- Enables each domain to have its own set of isolated cache clusters within the LLC.

PROPOSED SOLUTION

2. Load-Balancing Hash (LBH) Module:

- Responsible for uniformly mapping line addresses to logical clusters within a domain to minimize set-conflicts.
- Utilizes multiple randomizing hash functions to distribute cache lines evenly among logical clusters. Supports a configurable number of clusters per domain, accommodating varying cluster configurations.
- Enhances cache performance by reducing contention for cache sets and improving cache utilization efficiency.
- Designed to provide constant-time access and prevent new timing side-channels, ensuring data security and integrity within the cache system.

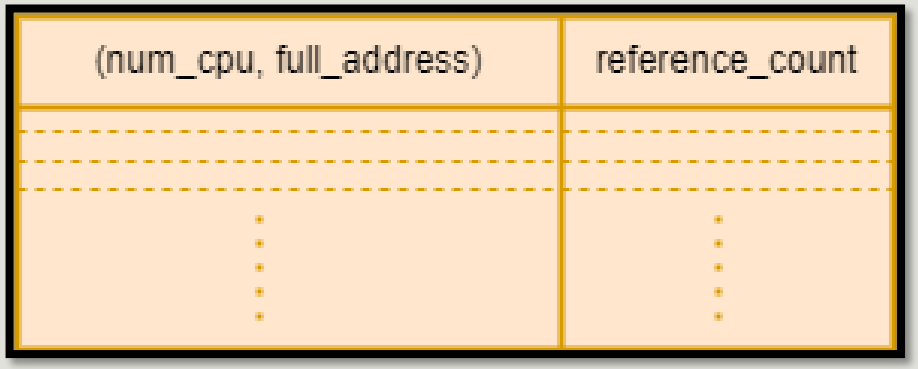
PERFORMANCE



Workloads	Non-Secure	DAWG	Page-Coloring	BCE
Spec-22	8.42	8.73	8.44	8.44
Gap-6	41.64	42.61	41.63	41.63
Mix-28	27.56	29.82	28.53	28.59
All-56	**21.55**	**22.91**	**22.04**	**22.07**

DEAD BLOCK PREDICTION

1. Employed a hash map, called BLOCK_MAP, that uses num_cpu and full_address of the current block as the key.
2. The count of number of times a same block is re-referenced before being evicted is maintained as value part of the map.



DEAD BLOCK PREDICTION

3. Upon each hit, the count is increased.
4. While selecting the victim, the current cpu and the full_address is searched for each blocks stored in the particular, then it is compared to find the minimum referenced block and select that for eviction and making that entry to zero in BLOCK_ACCESS map.

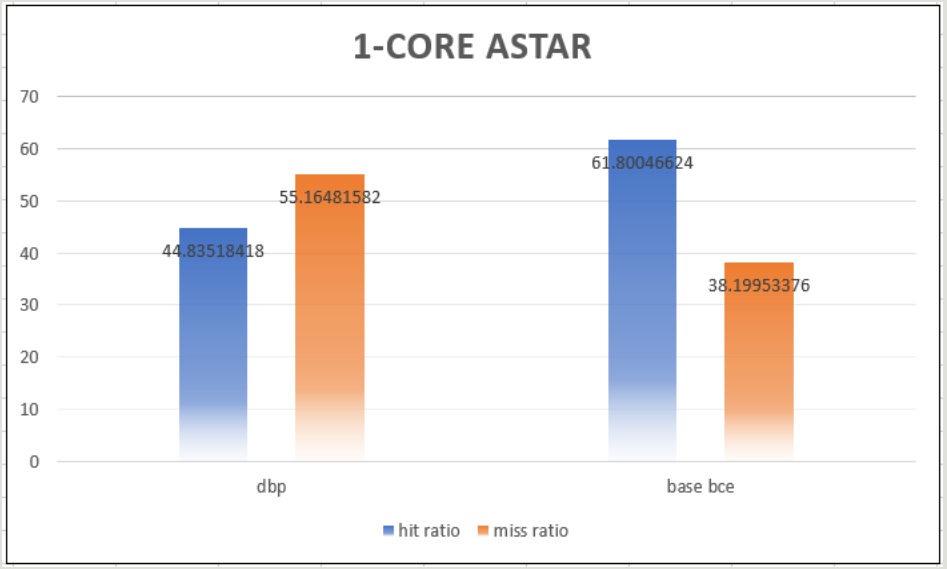
EXPERIMENTAL SETUP

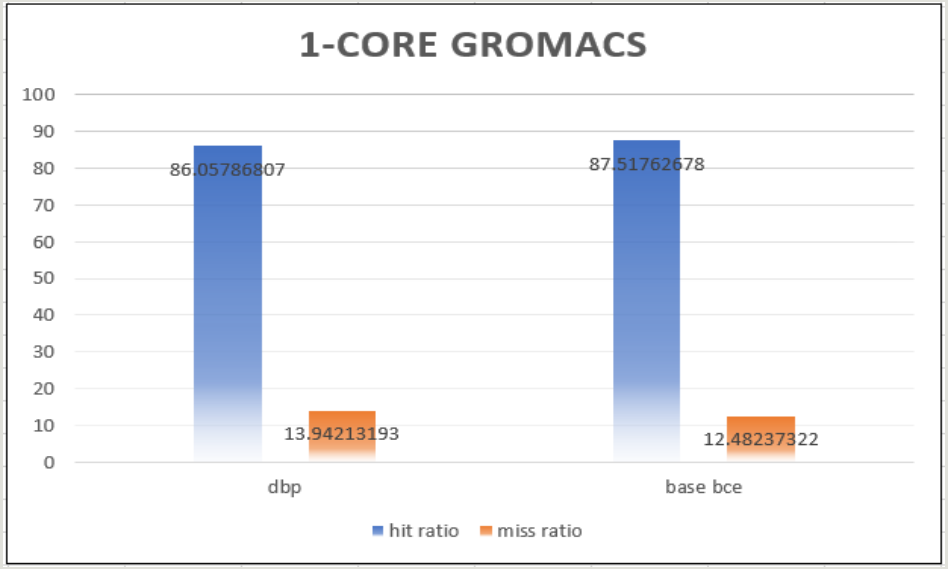
1. L1 (instruction) cache: 32KB 8-way
2. L2 cache: 512KB 8-way
3. L3 (shared) cache: 32MB 16-way
4. Simulator: ChampSim Multicore Out-of-Order simulator

WORKLOADS TESTED

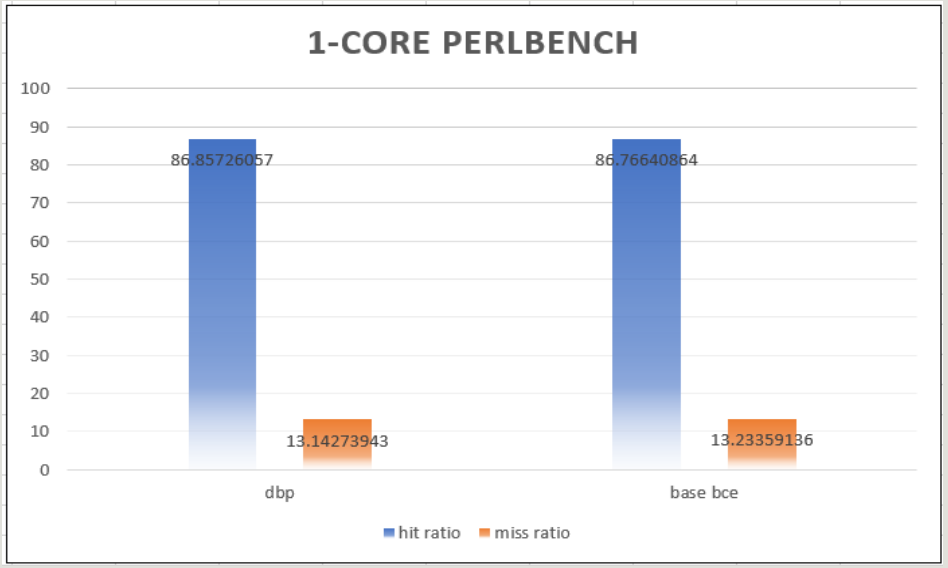
1. Astar
2. Gromacs
3. Perlbench
4. Gcc
5. Bzip2
6. Libquantum
7. CactusADM
8. Calculix
9. Gobmk
10. Gamess
11. Bwaves
12. Omnetpp

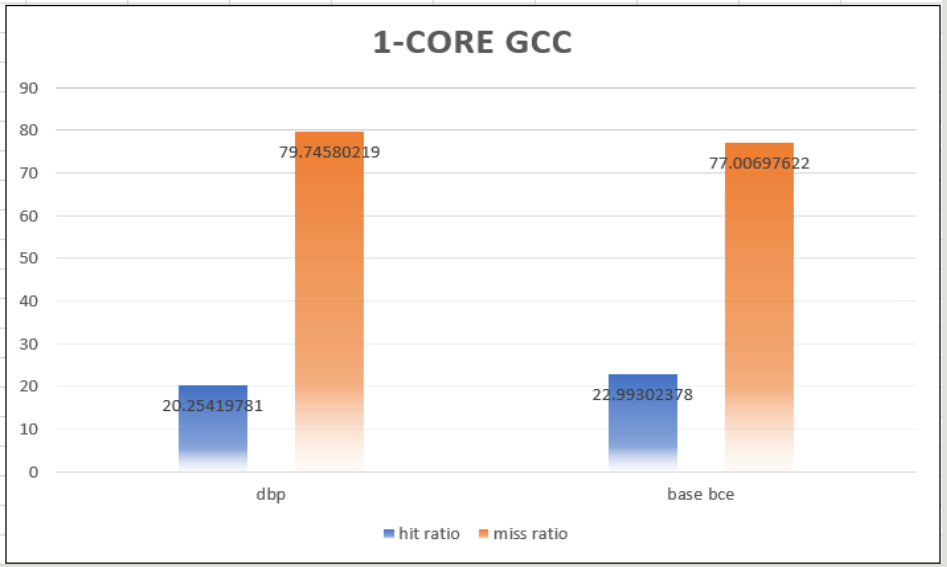
RESULTS – 1 core



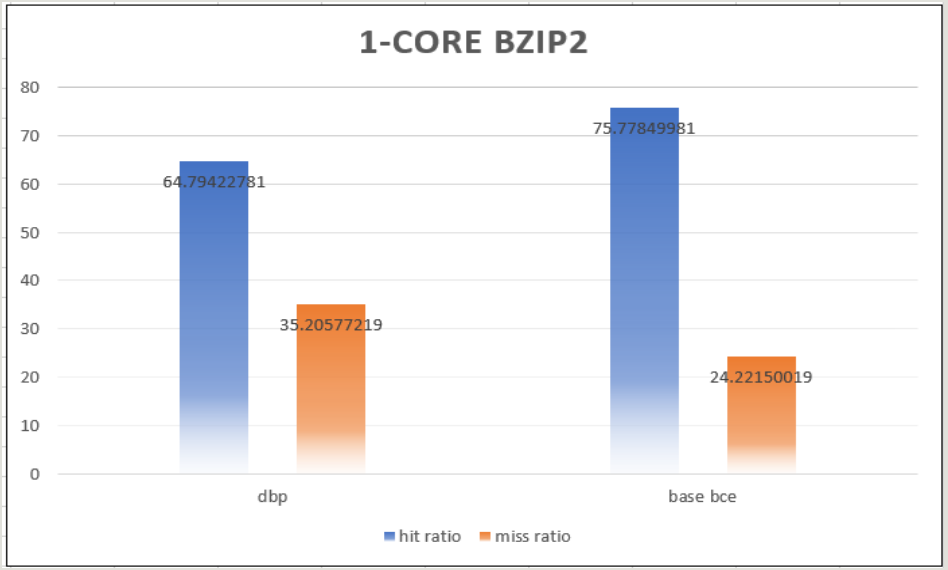


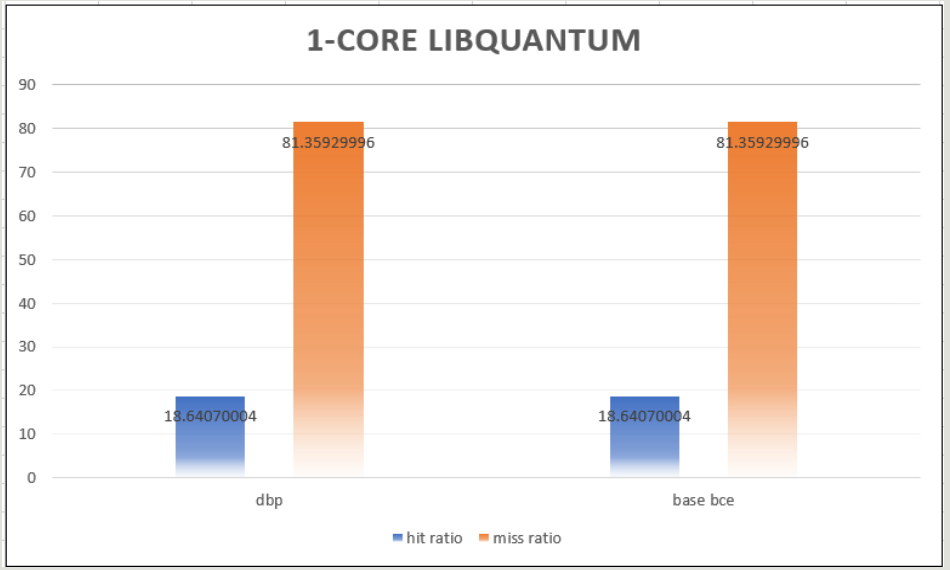
RESULTS – 1 core



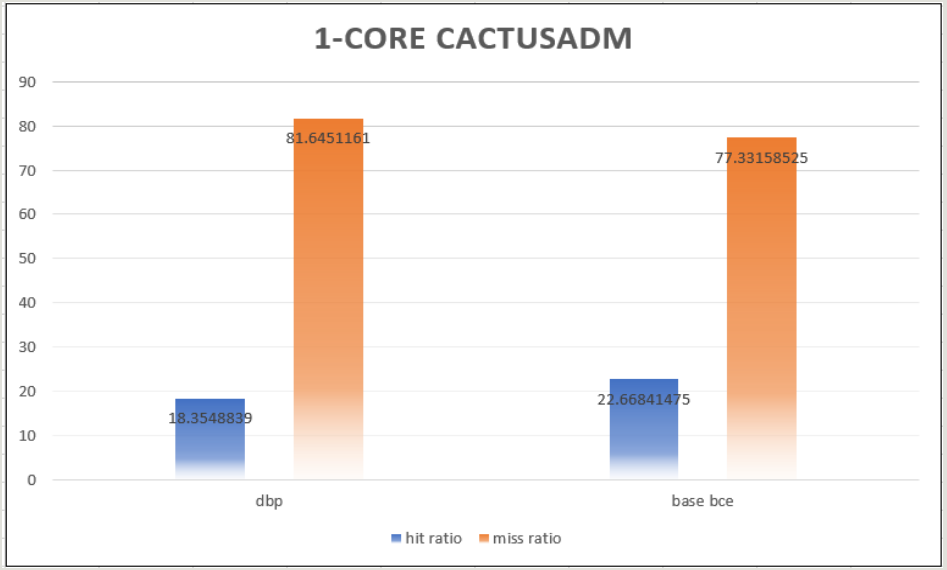


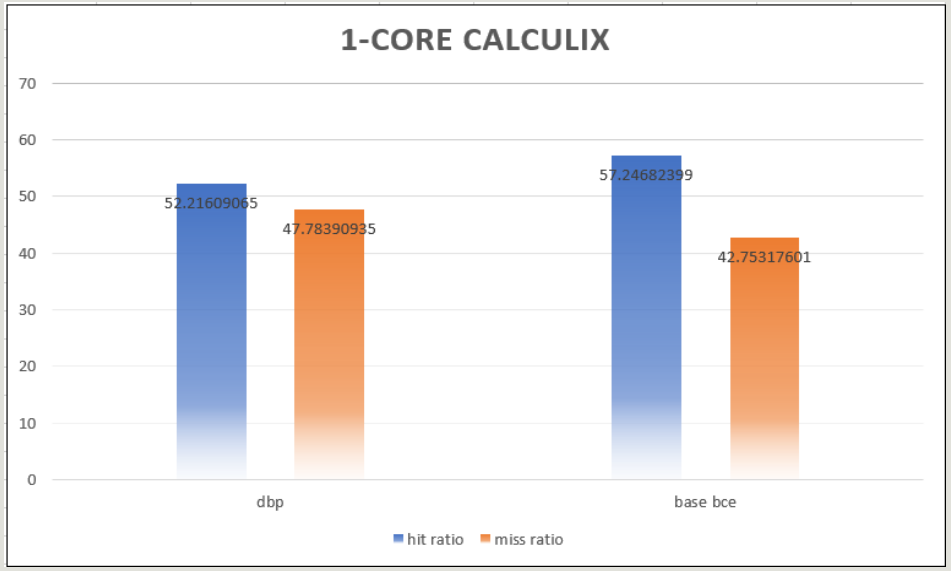
RESULTS – 1 core



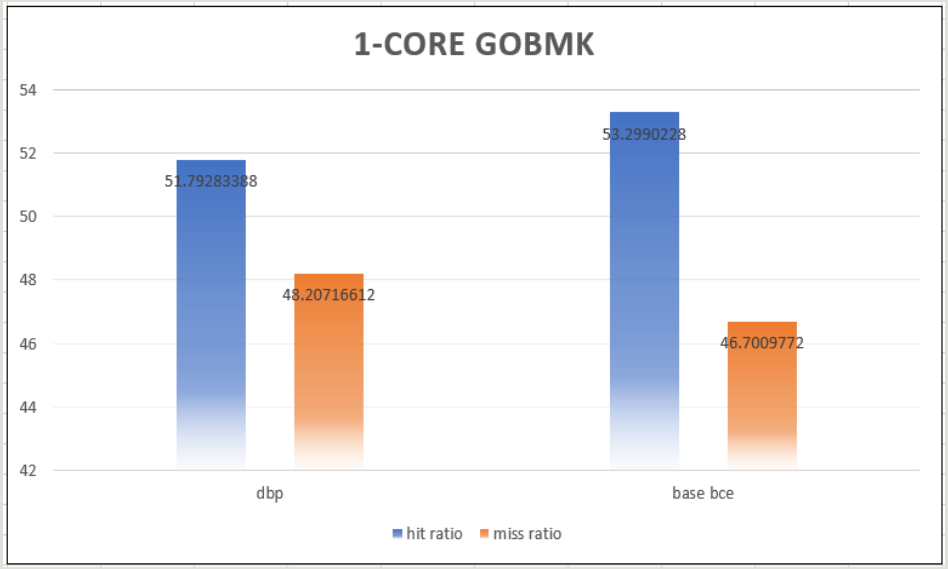


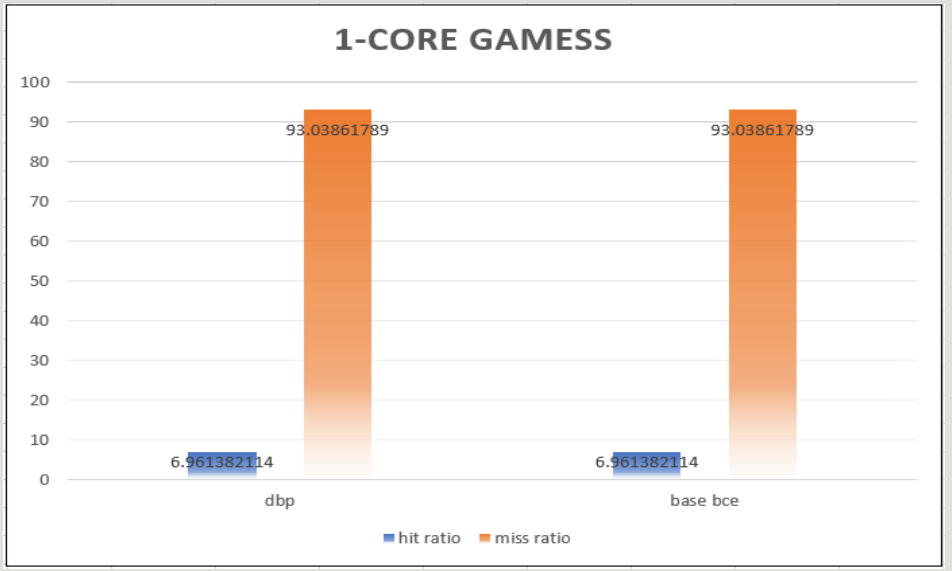
RESULTS – 1 core



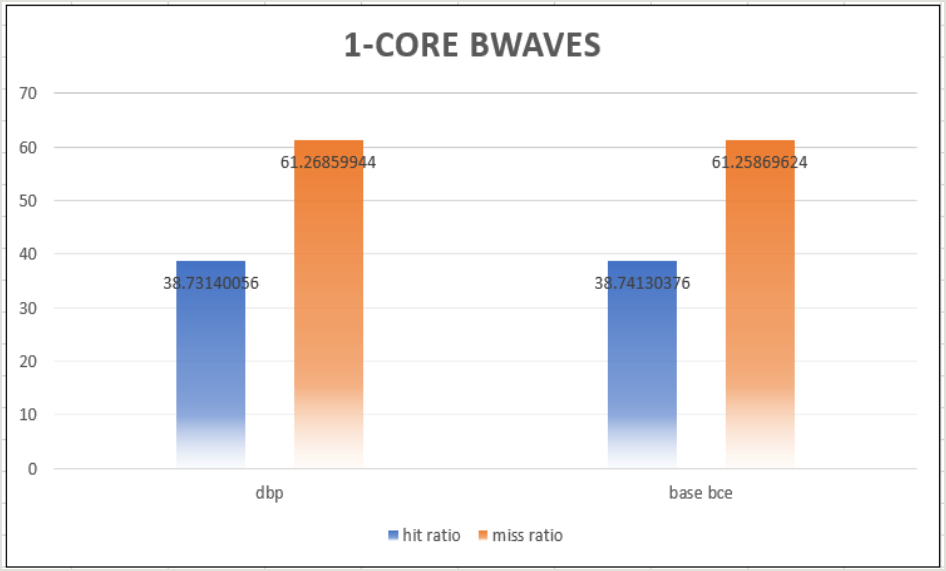


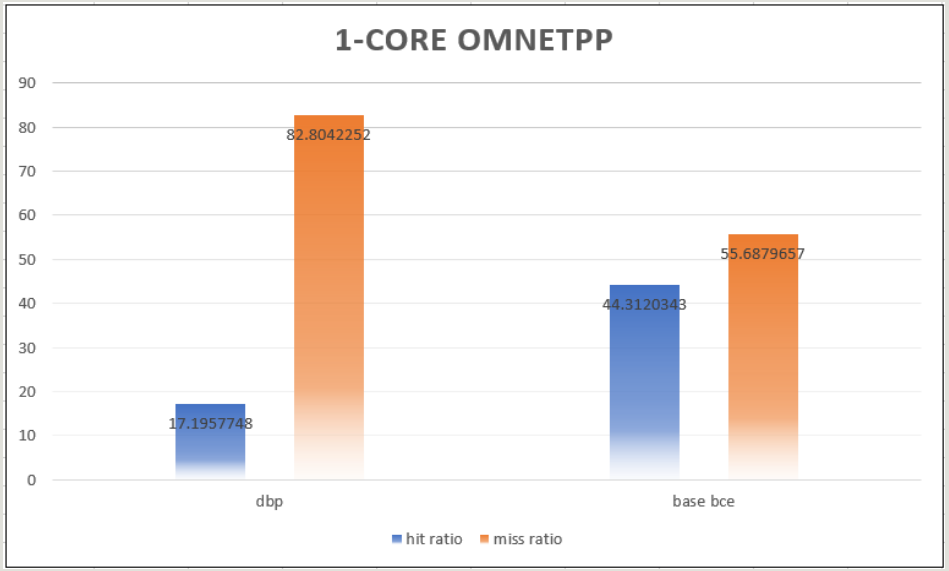
RESULTS – 1 core



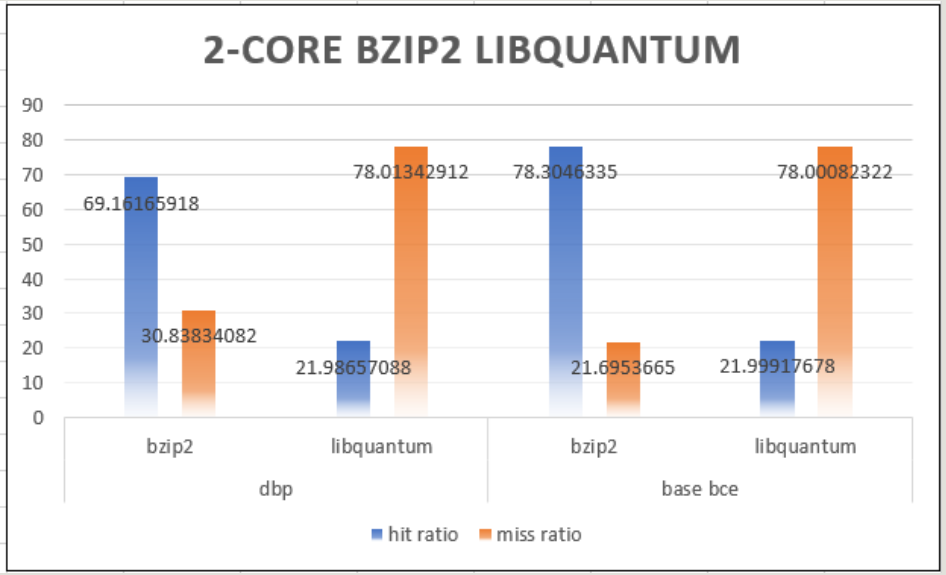


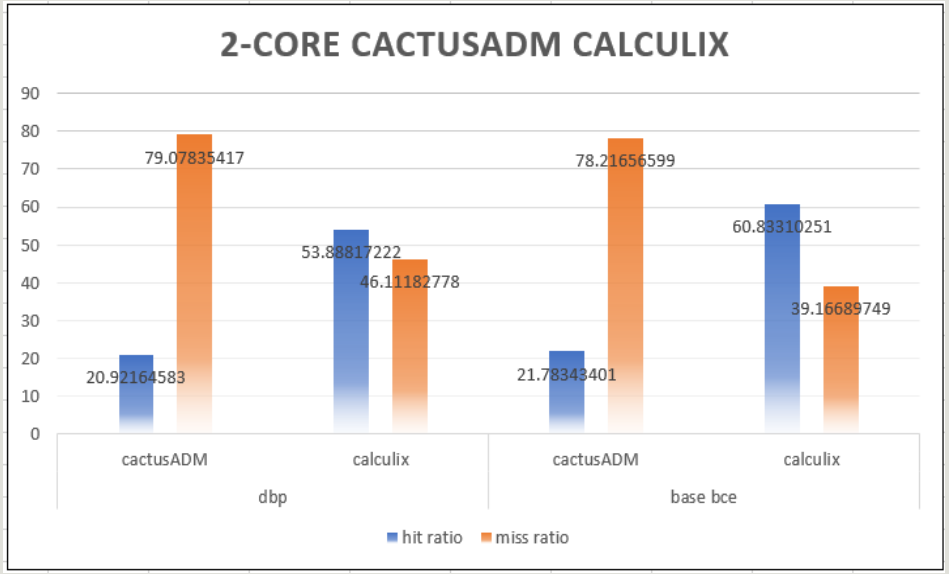
RESULTS – 1 core



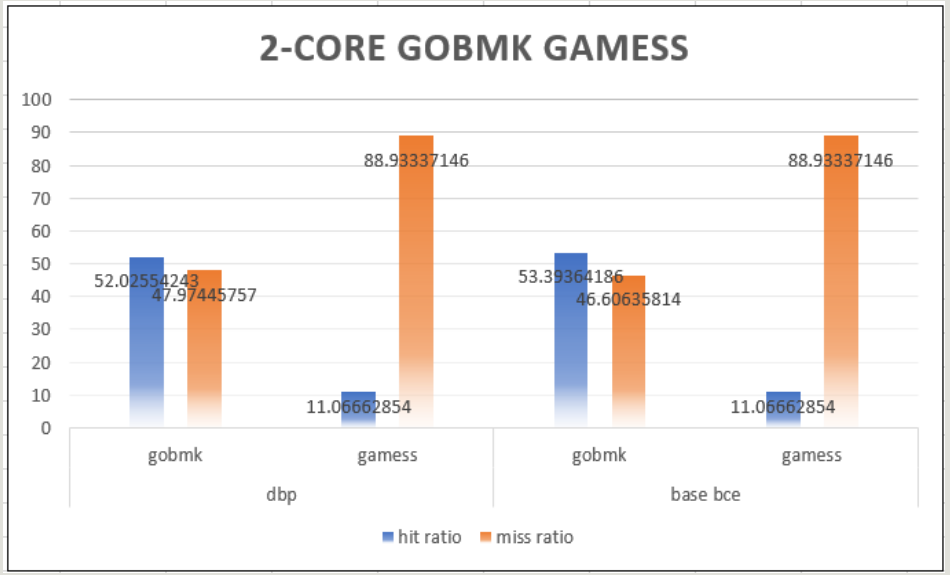


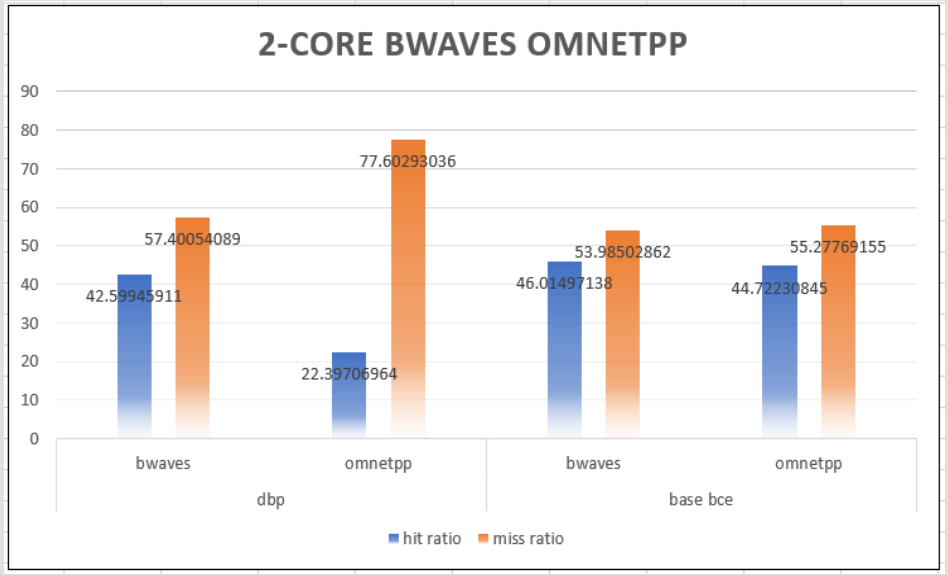
RESULTS – 2 core



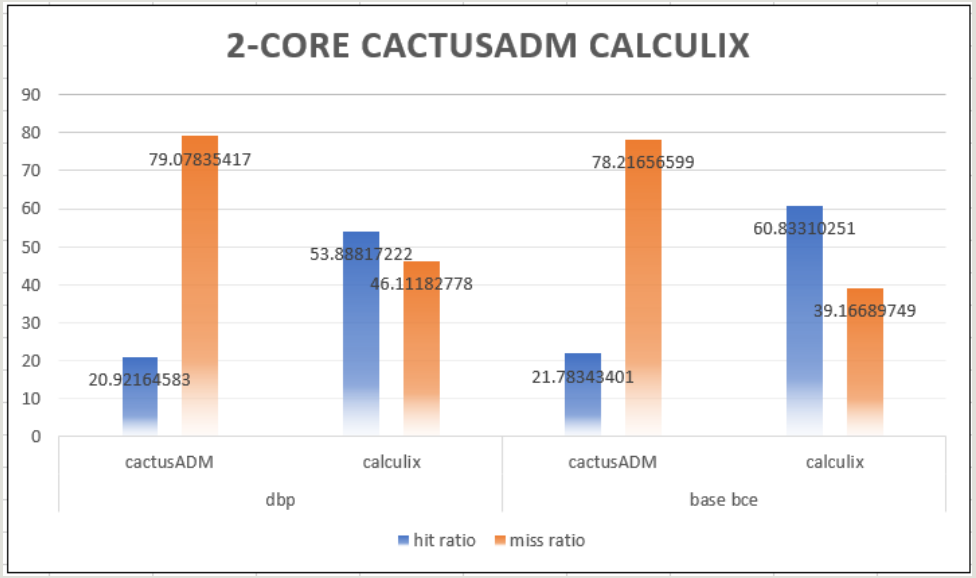


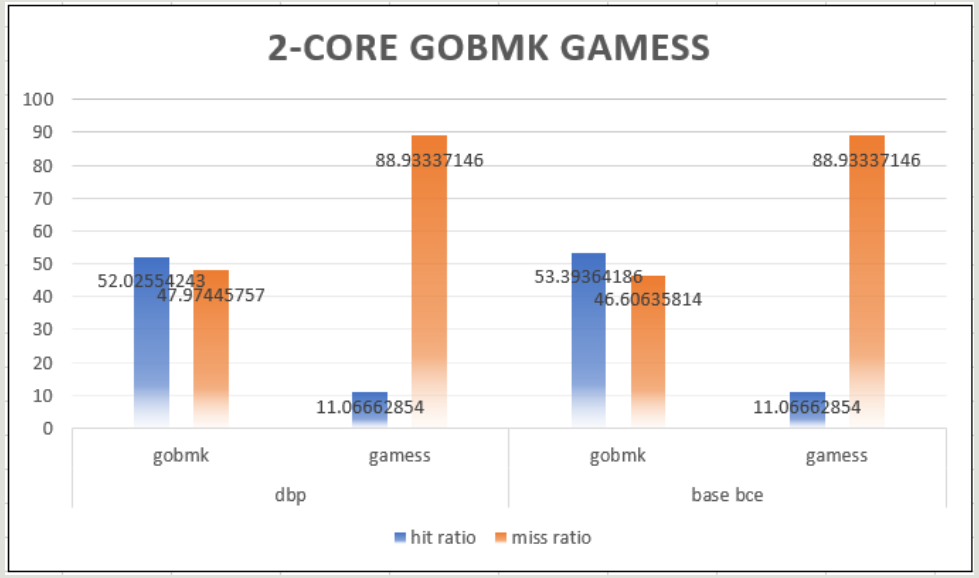
RESULTS – 2 core



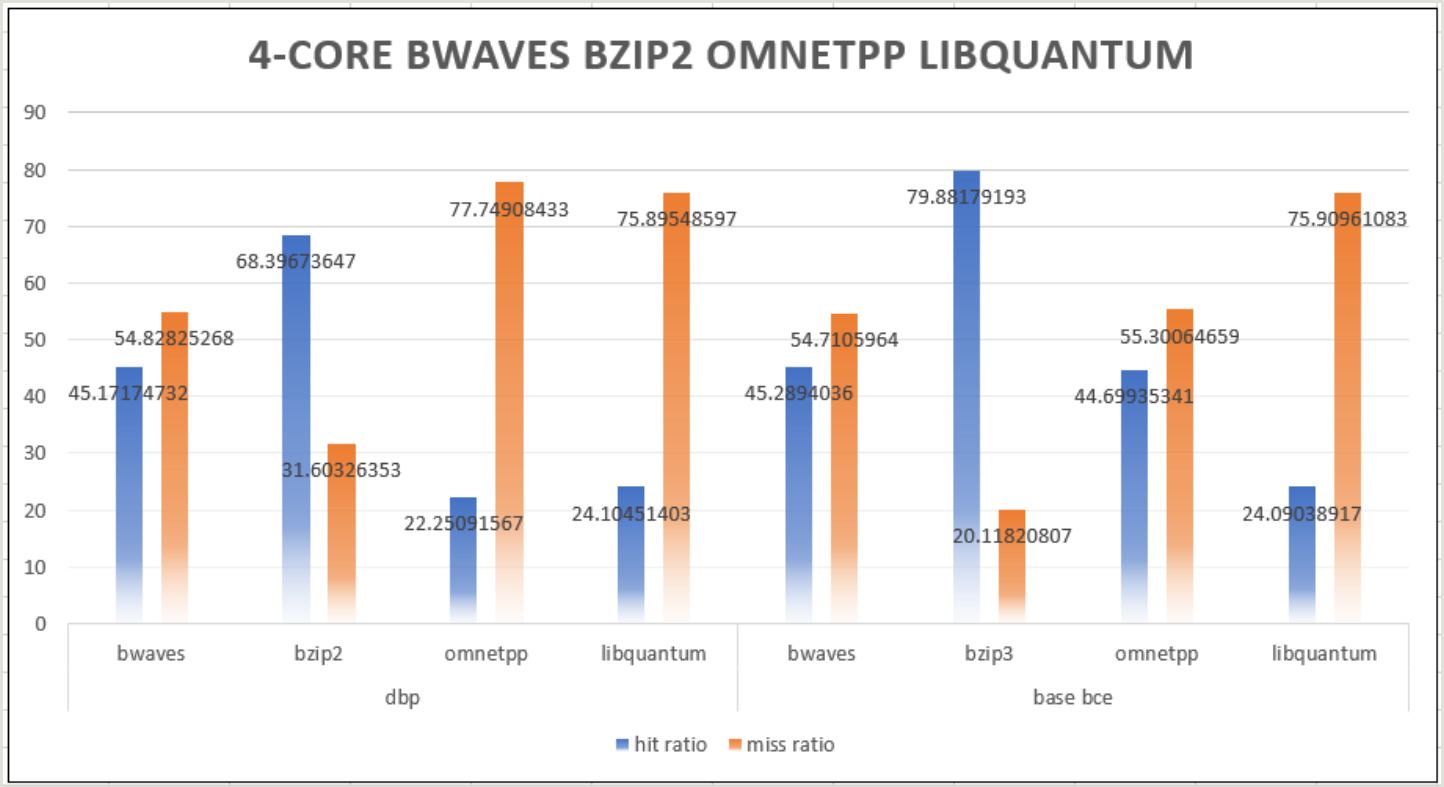


RESULTS – 2 core

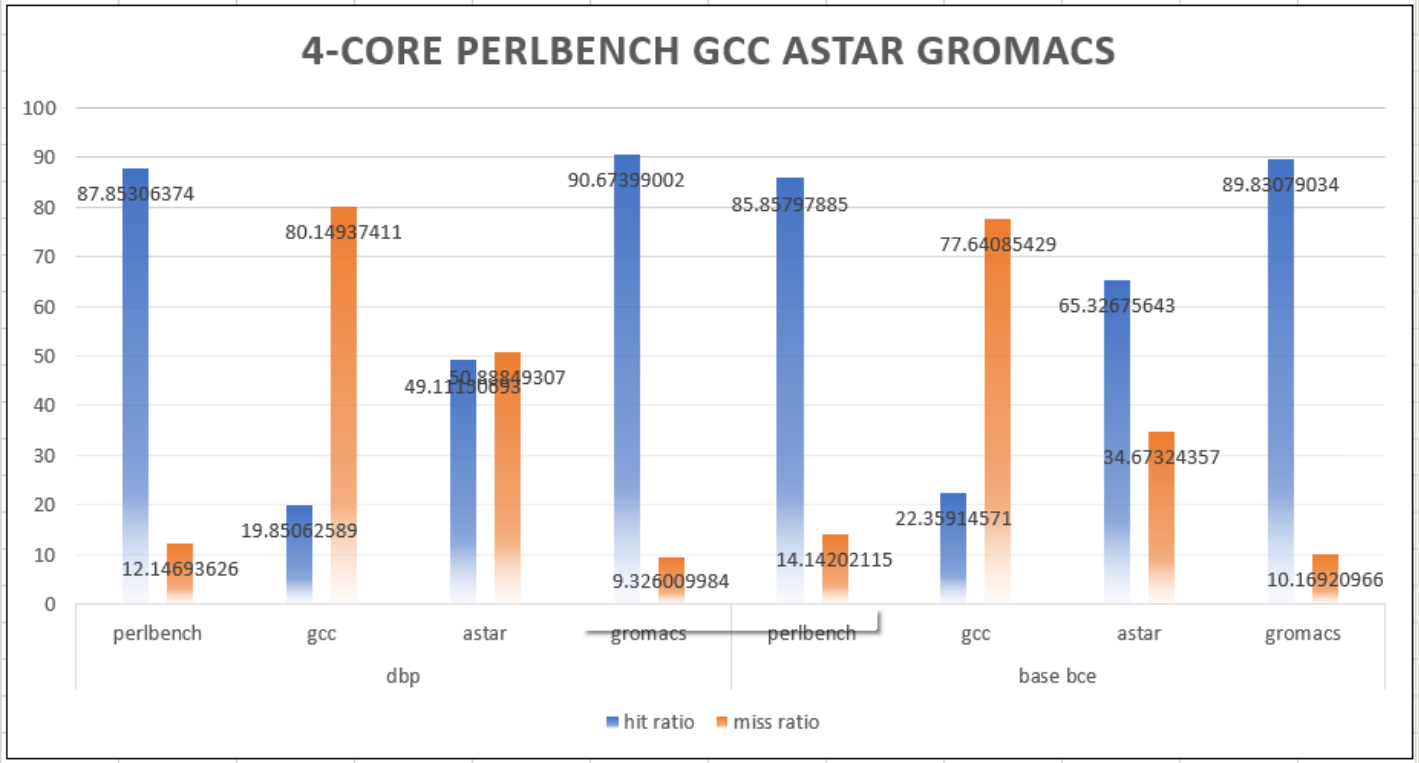




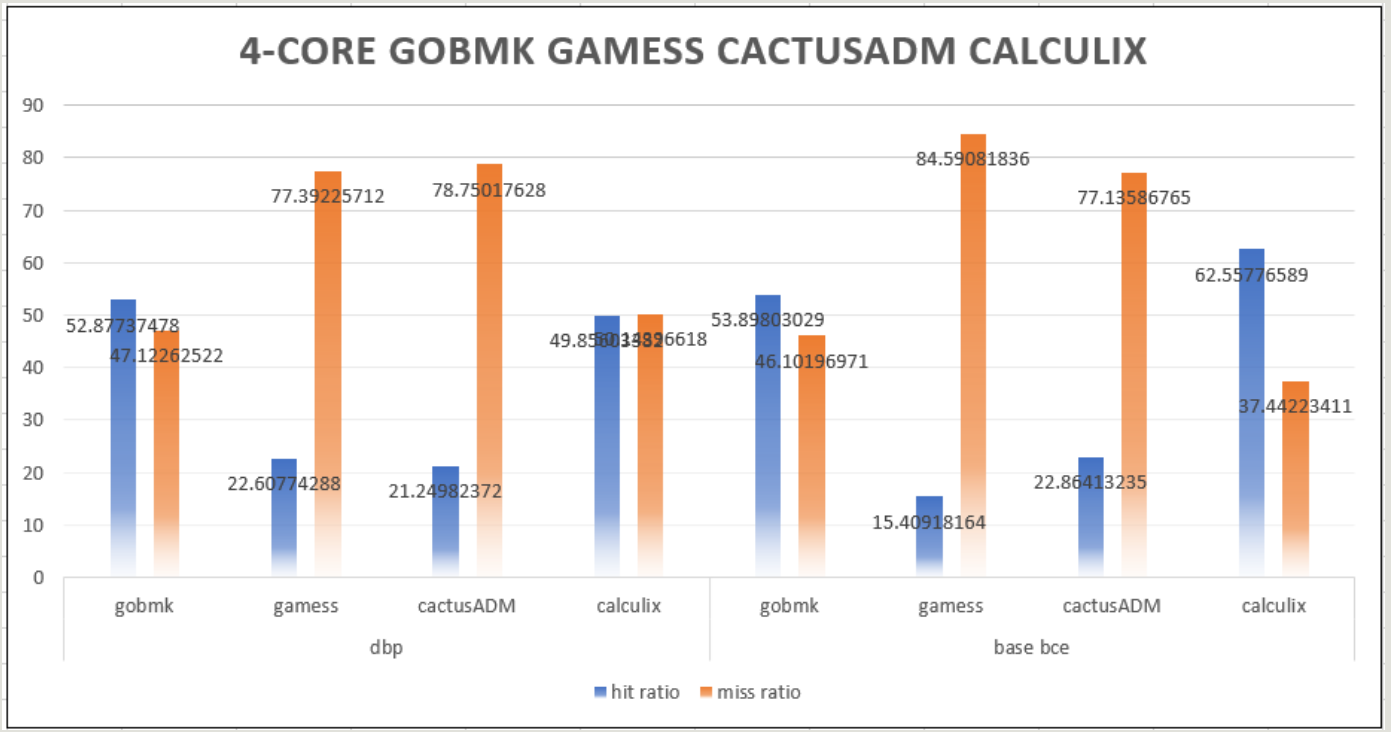
RESULTS – 4 core



RESULTS – 4 core



RESULTS – 4 core



OBSERVATIONS

1. In many of the workloads, the Hit Rate is seen to be decreased, such as in astar, gcc, bzip2.
2. In some other workloads, the Hit Rate is seen to be approximately same, such as in gromacs, libquantum.
3. While running on 2-core with perlbench and gcc workloads, a little ~2% increase can be seen for perlbench.
4. One side effect is observed, even though the *dbp (dead block prediction)* is unable to increase the Hit Rate, but it reduces the LLC Average Miss Latency.

THANK YOU