

# Memory/Cache hierarchy optimizations for Graph Analytics

---

**ADYASHA - 210050007**  
**DHANANJAY – 210050044**  
**MRIDUL - 210050100**



**EVERYONE**

**THIS IS MY PRESENTATION**

WHY SUCH SPECIAL EMPHASIS ON A PARTICULAR  
KIND OF WORKLOAD? WHAT IS SO DIFFERENT ABOUT  
IT ?



# Graph Analytics

---

## Practical Optimal Cache Replacement for Graphs

Vignesh Balaji  
Carnegie Mellon University  
vigneshb@andrew.cmu.edu

Neal Crago  
Nvidia Research  
ncrago@nvidia.com

Aamer Jaleel  
Nvidia Research  
ajaleel@nvidia.com

Brandon Lucia  
Carnegie Mellon University  
blucia@andrew.cmu.edu

A cache's replacement policy is a key determinant of locality. Decades of work have produced high-performance replacement policies for various workloads. **However, we find that state-of-the-art replacement policies are ineffective for graph processing.** Graph data reuse is dynamically variable and graph-structure-dependent, two properties not captured well by existing replacement policies. Belady's MIN replacement policy is an ideal policy that perfectly captures dynamic, graph-structure-dependent reuse, but it is impractical because it relies on knowledge of future accesses.

## High Performance Big Data Graph Analytics Leveraging Near Memory System

AHSEN TAHIR\*, JAWAD AHMAD†, SYED AZIZ SHAH‡ AND QAMMER H ABBASI§

### A. Graph Workloads are Irregular

The traditional memories are not suitable for graph workloads. Graph workloads are memory bound [10]. They have irregular access patterns and lack temporal locality which effects high performance graph computing [11, 12, 13, 14, 15, 16, 17]. This results in a large number of accesses to off-chip memory which are costly both in terms of latency and energy. **Graph workloads require an efficient memory interface which can provide concurrency of accesses, with lower latency and energy costs.**

## Graphfire: Synergizing Fetch, Insertion, and Replacement Policies for Graph Analytics

Aninda Manocha, *Student Member, IEEE*, Juan L. Aragón, *Member, IEEE*, and Margaret Martonosi, *Fellow, IEEE*

### 1 INTRODUCTION

**F**OR decades, caches have played a significant role in improving CPU performance, reducing off-chip memory access latency as well as processor-memory traffic [16], [43]. Many widespread applications, including linear algebra routines for dense neural networks, demonstrate access regularity that benefits from the logic and structure of modern cache designs. **However, graph analytics remain an important domain of applications where even state-of-the-art cache management techniques continue to struggle.**

## Gretch: A Hardware Prefetcher for Graph Analytics

ANIRUDH MOHAN KAUSHIK, University of Waterloo, Canada

GENNADY PEKHIMENKO, University of Toronto, Canada

HIREN PATEL, University of Waterloo, Canada

Data-dependent memory accesses (DDAs) pose an important challenge for high-performance graph analytics (GA). **This is because such memory accesses do not exhibit enough temporal and spatial locality resulting in low cache performance.** Prior efforts that focused on improving the performance of DDAs for GA are not applicable across various GA frameworks. This is because (1) they only focus on one particular graph representation, and (2) they require workload changes to communicate specific information to the hardware for their effective operation.



# OVERVIEW

- Unpredictable memory access patterns
- Very little temporal and spatial locality.
- Modern datasets, e.g. social networks, are also significantly larger than the last-level cache (LLC), leading to thrashing at all levels of the memory hierarchy.
- Irregular accesses themselves have variable reuse, which troubles the SOTA heuristic and learning-based replacement policies
- As a result, graph applications frequently perform expensive, off-chip memory accesses, whose long latencies can dominate application runtimes and limit scalability.

# Recent Research in this field

---

- Most recent improvements based on software preprocessing or hardware-software interaction.
- Software preprocessing arranges high reuse vertices together in the memory, thereby minimizing the cache misses by improving cache line utilization.
- However given just traces, we can't do any software dependent changes
- Also software preprocessing renders the technique less practical for large graphs, where the input graph is only processed once, or when the graph is not even fully traversed.

What is the plan  
now?



We went about how we could improve performance without accessing the graph structure, and approached our problem with respect to three parameters :

- **Changing the size of caches**
- **Changing the replacement policy**
- **Inclusive/Exclusive/Non-Inclusive Cache heirarchy**

# Baseline Architecture

Table I: Baseline architecture

<b>core</b>	4 cores, ROB = 128-entry, load queue = 48-entry, store queue = 32-entry, reservation station entries = 36, dispatch width = issue width = commit width = 4, frequency = 2.66GHz
<b>caches</b>	3-level hierarchy, inclusive at all levels, writeback, least recently used (LRU) replacement policy, data and tags parallel access, 64B cacheline, separate L1 data and instruction caches
<b>L1D/I cache</b>	private, 32KB, 8-way set-associative, data access time = 4 cycles, tag access time = 1 cycle
<b>L2 cache</b>	private, 256KB, 8-way set-associative, data access time = 8 cycles, tag access time = 3 cycles
<b>L3 cache (LLC)</b>	shared, 8MB, 16-way set-associative, data access time = 30 cycles, tag access time = 10 cycles
<b>DRAM</b>	DDR3, device access latency = 45ns, queue delay modeled





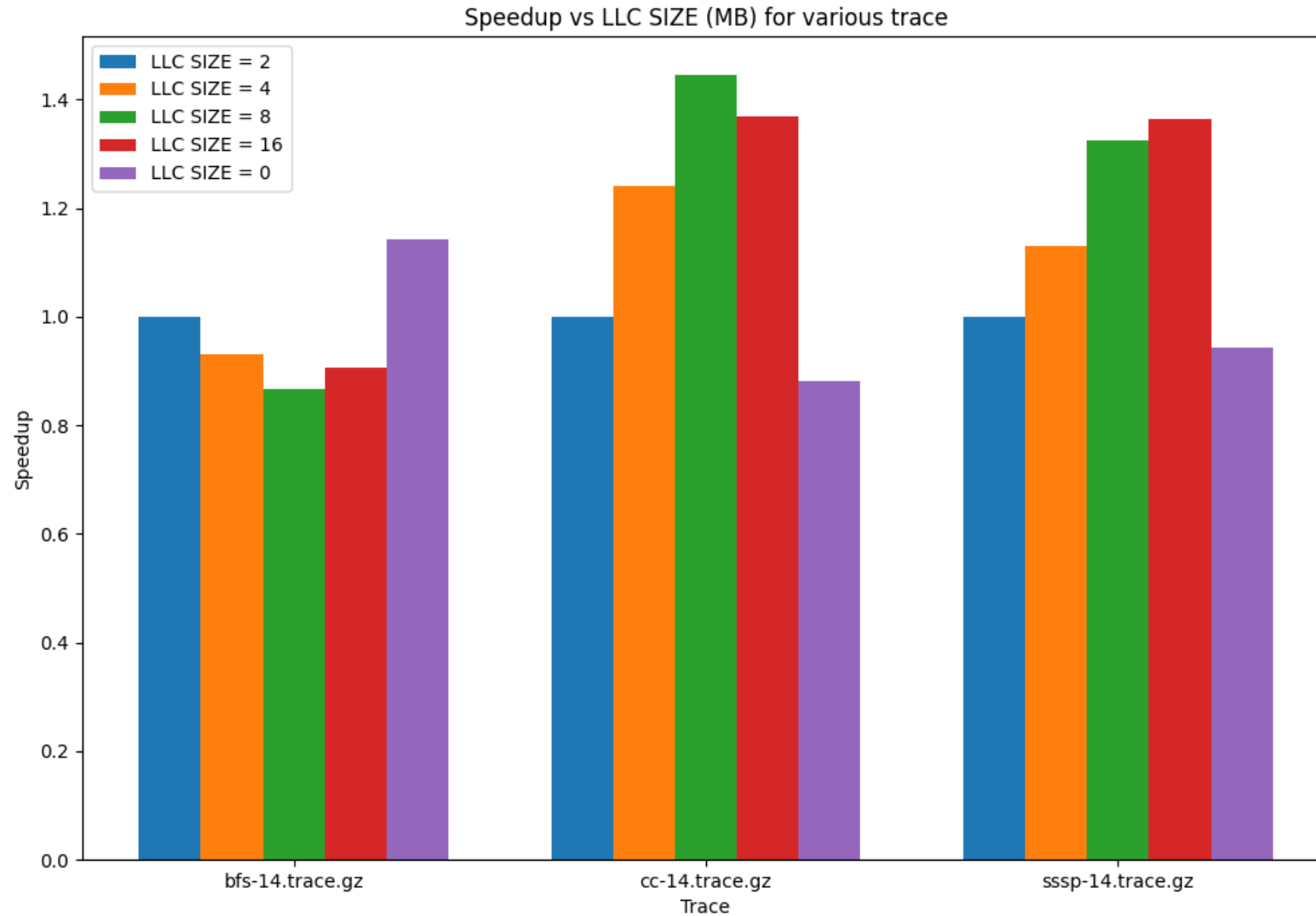
---

# Changing Cache Sizes

## Theory we knew

- In general for most of the workloads, increasing the cache size can improve speedup because it allows more data to be stored in the cache. This results in fewer cache misses and less time spent waiting for data to be fetched from main memory.
- However, increasing the cache size may also result in higher latency and power consumption, which can reduce the overall system performance. Additionally, if the cache size is too large, it may cause cache thrashing, where the cache becomes overwhelmed with data, resulting in a slowdown of system performance.

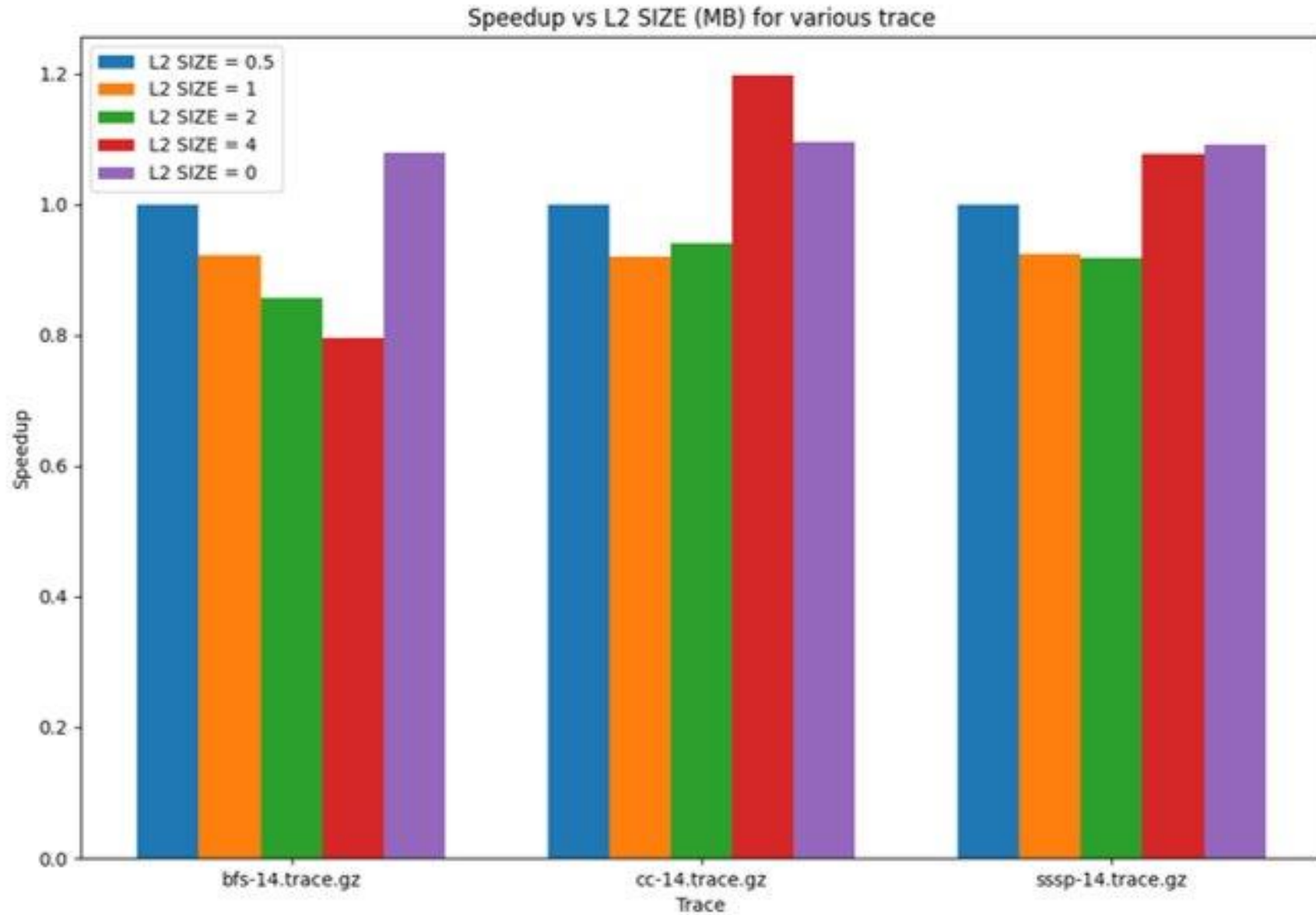
# LLC speedup graph



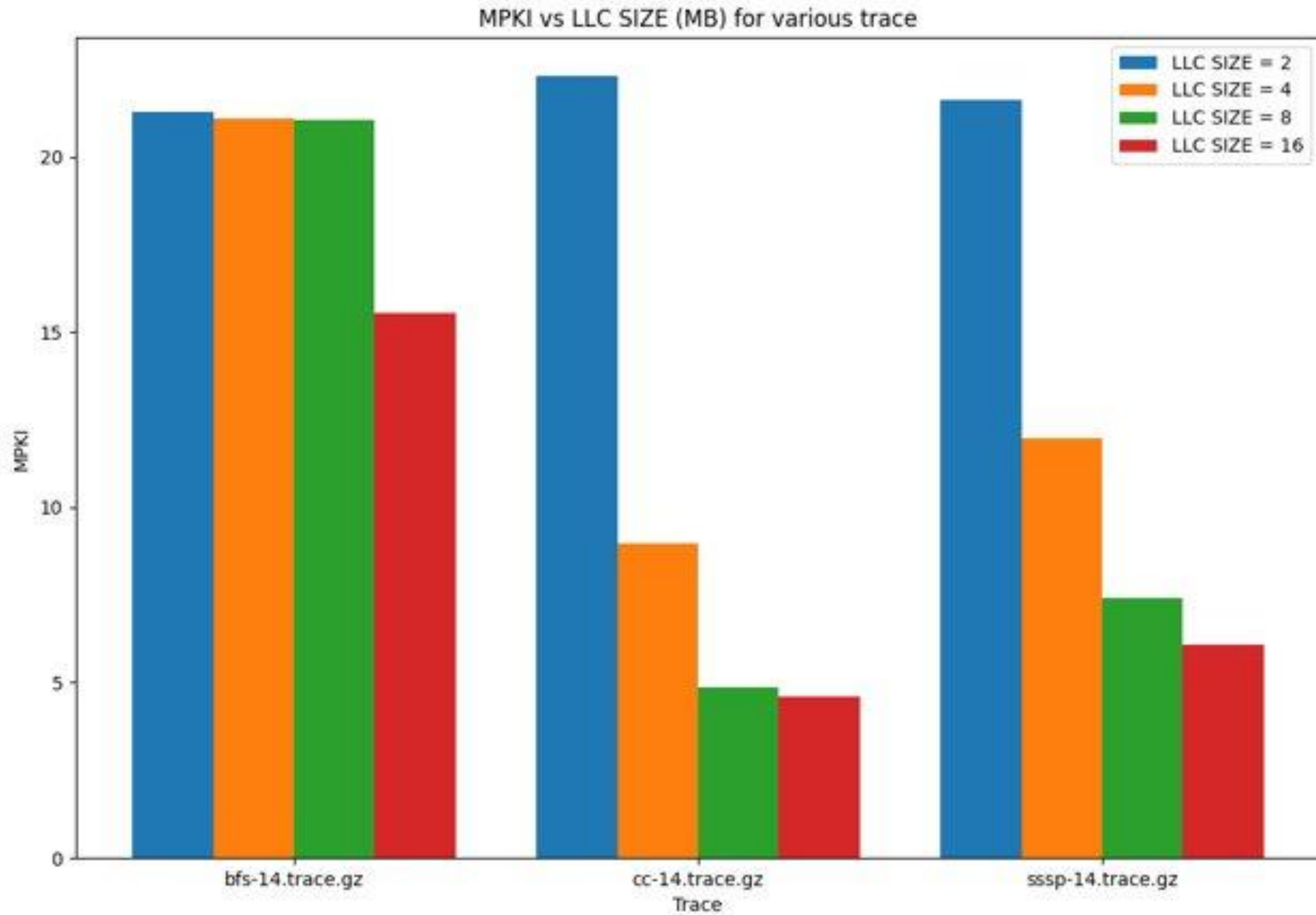
\*\*The MPKI measured everywhere is for LLC cache from now on.



# L2 speedup graph

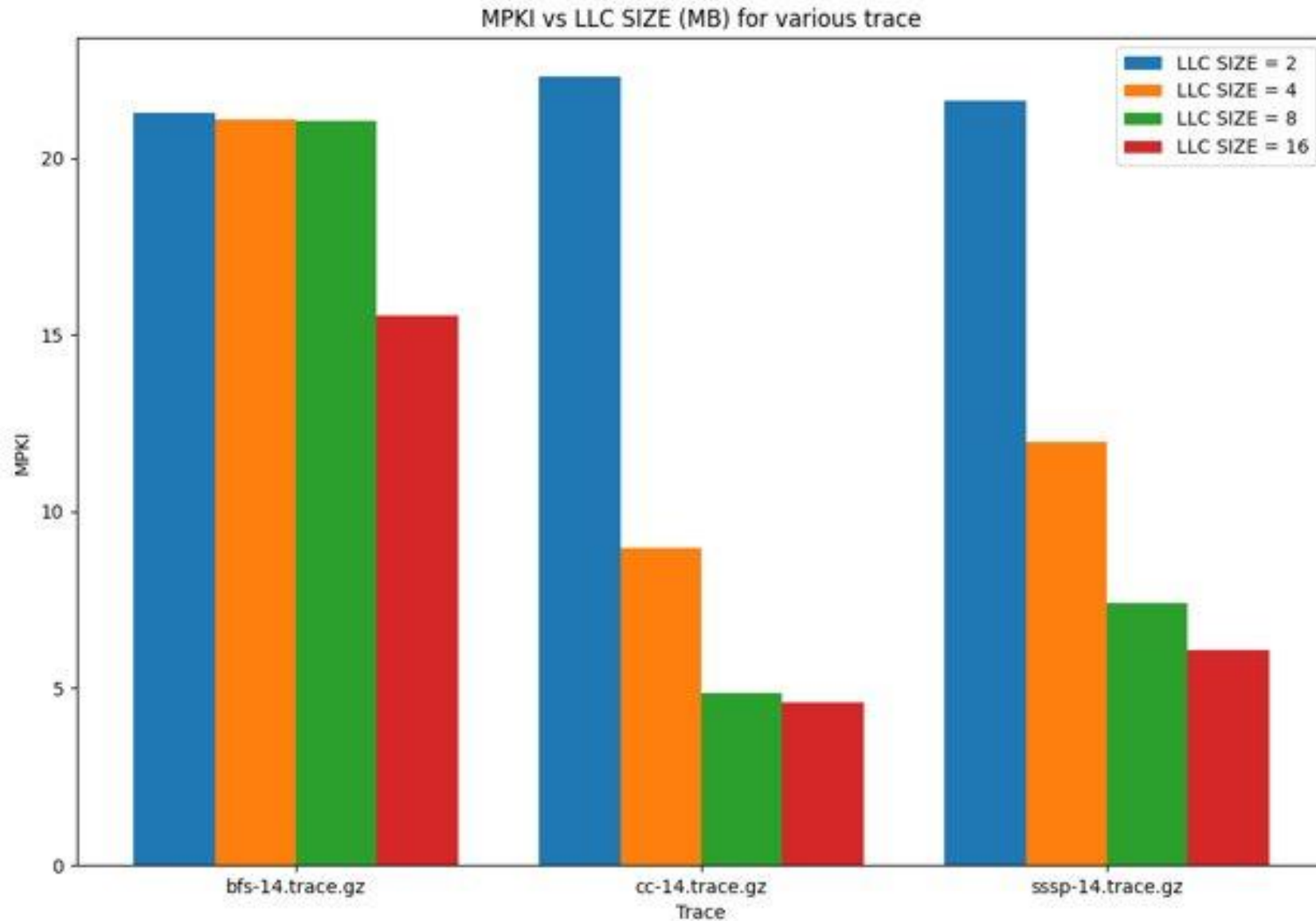


# LLC MPKI graph

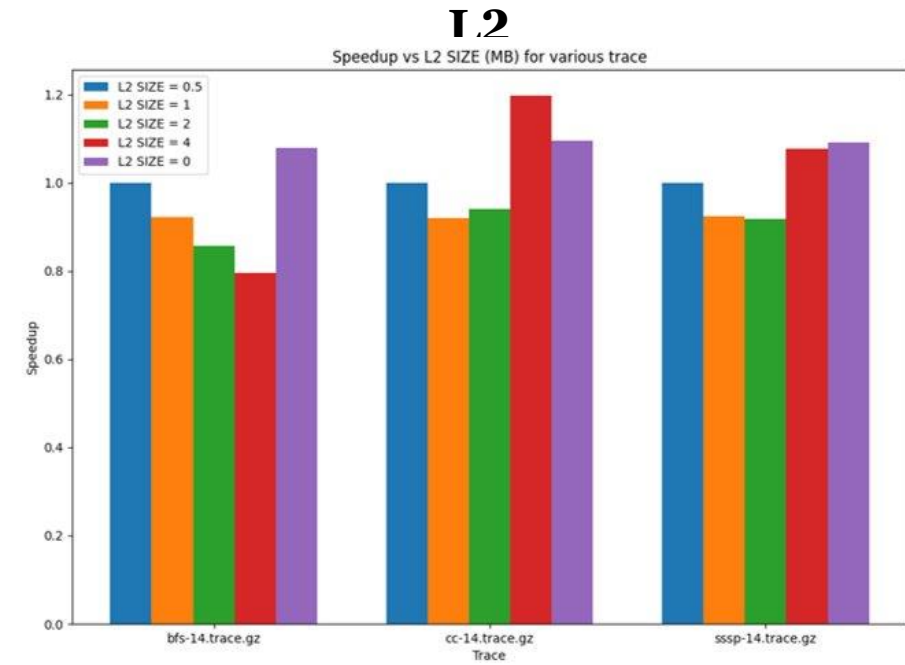
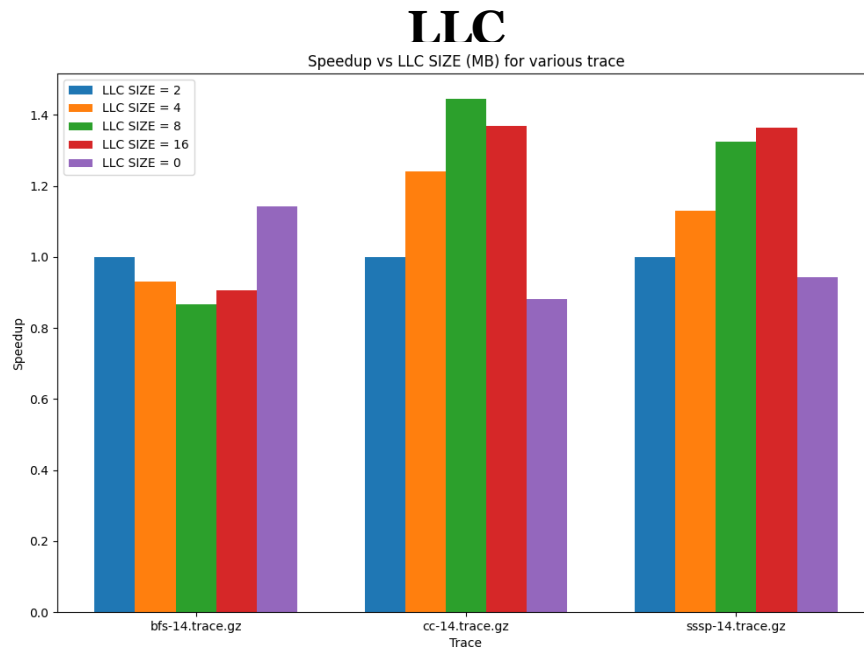




# L2 MPKI graph



# What happened when we actually changed cache sizes of L2 and LLC for Graph Analytics workload:



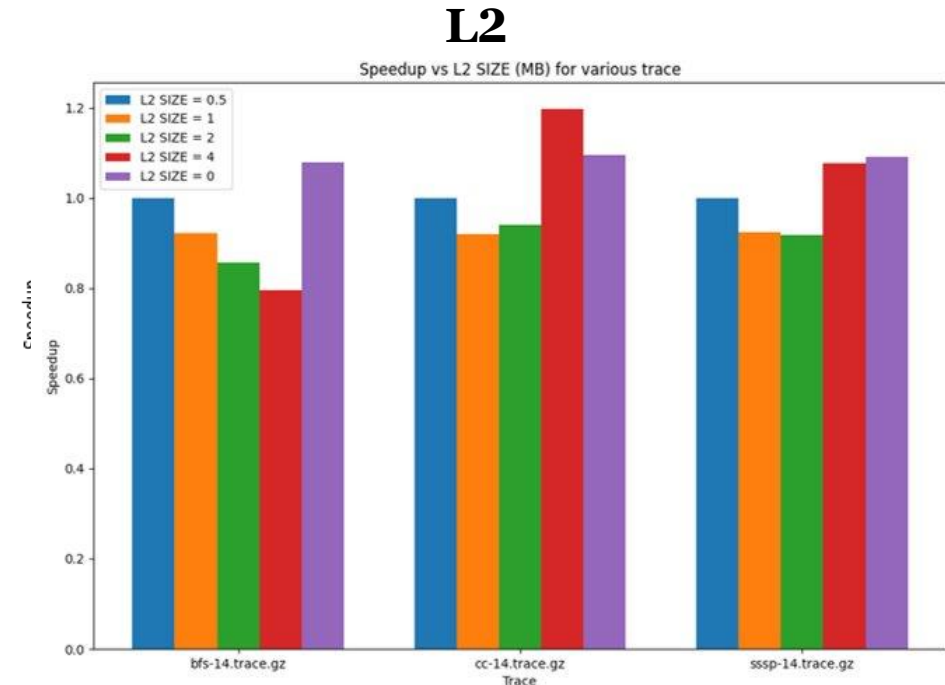
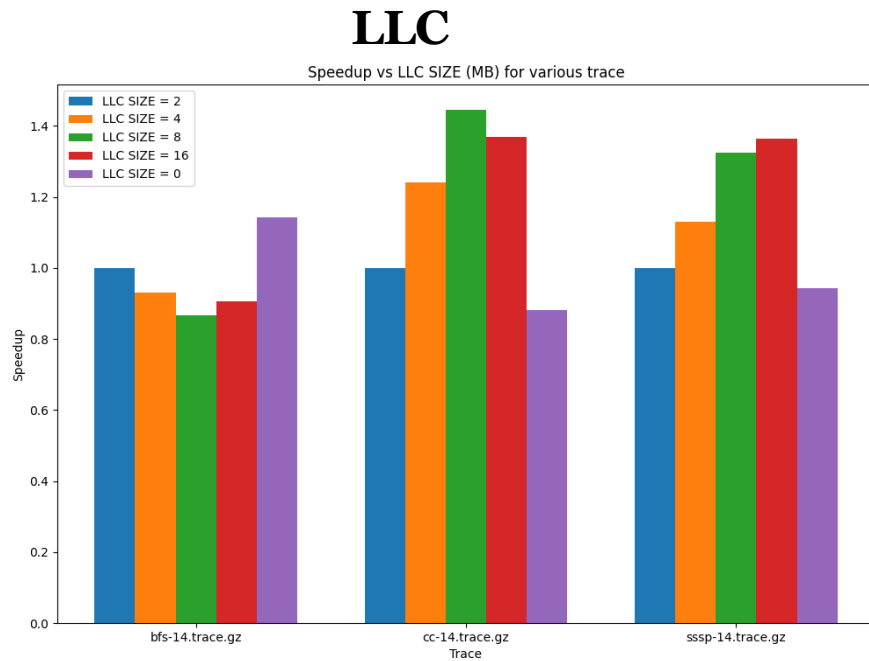
Observation 1: ( For this part we neglect the 0 size caches, that is, the purple lines)

On increasing cache size by increasing number of sets, keeping other things constant, we observe :

- 1) For bfs trace – No speedup for both, instead there is a slowdown indicating high reuse distance and a very high thrashing workload which experiences misses irrespective of the size of the cache. There is a slowdown due to the additional latency involved with increasing the cache sizes.



# What happened when we actually changed cache sizes of L2 and LLC for Graph Analytics workload:

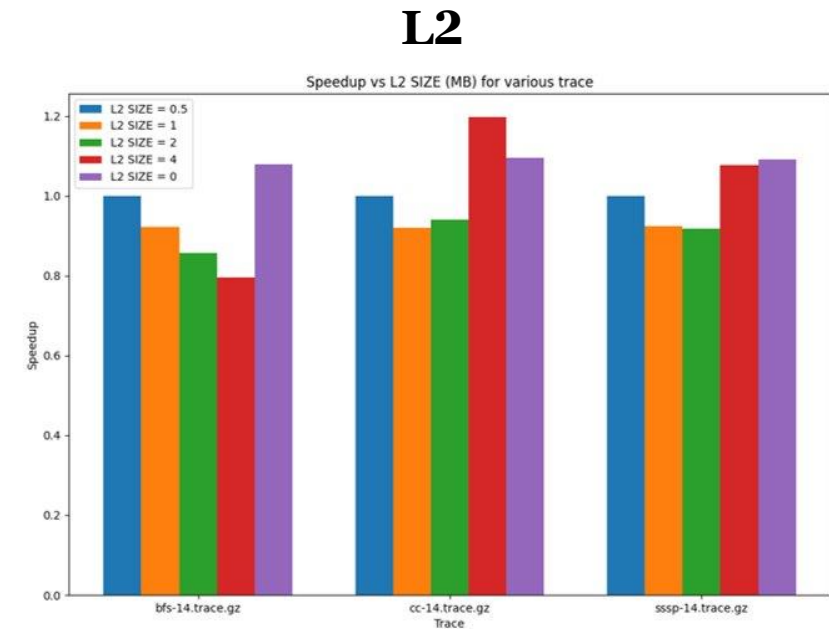
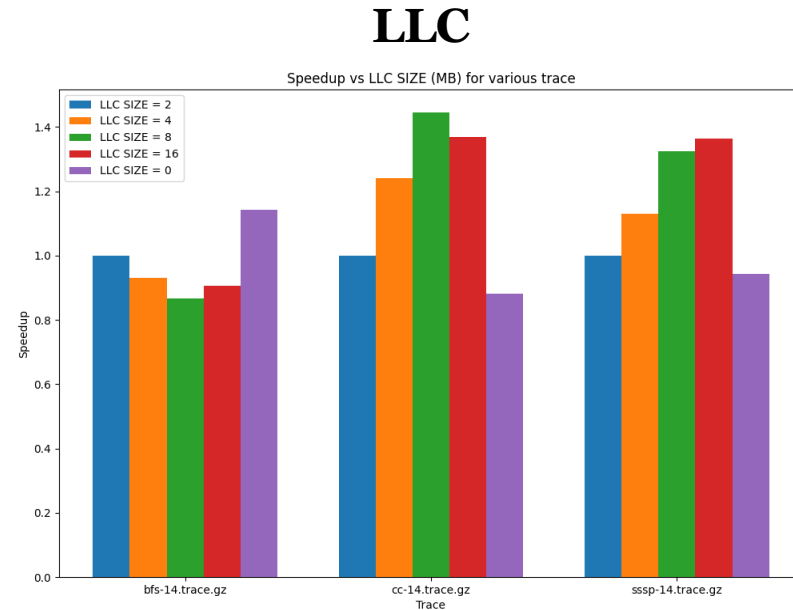


Observation 2: ( For this part we neglect the 0 size caches, that is, the purple lines)

On increasing cache size by increasing number of sets, keeping other things constant, we observe :

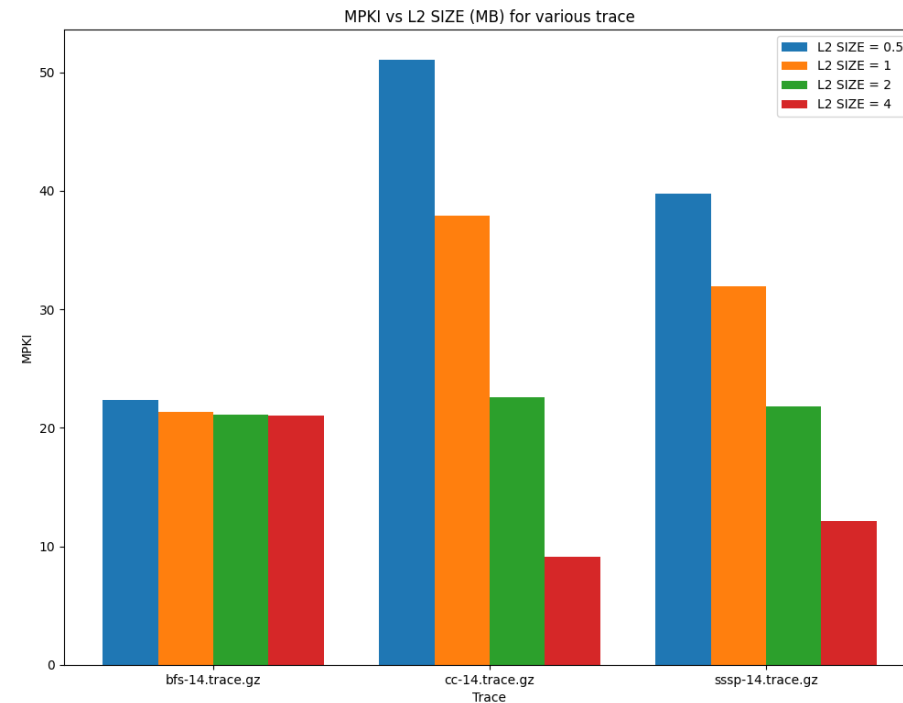
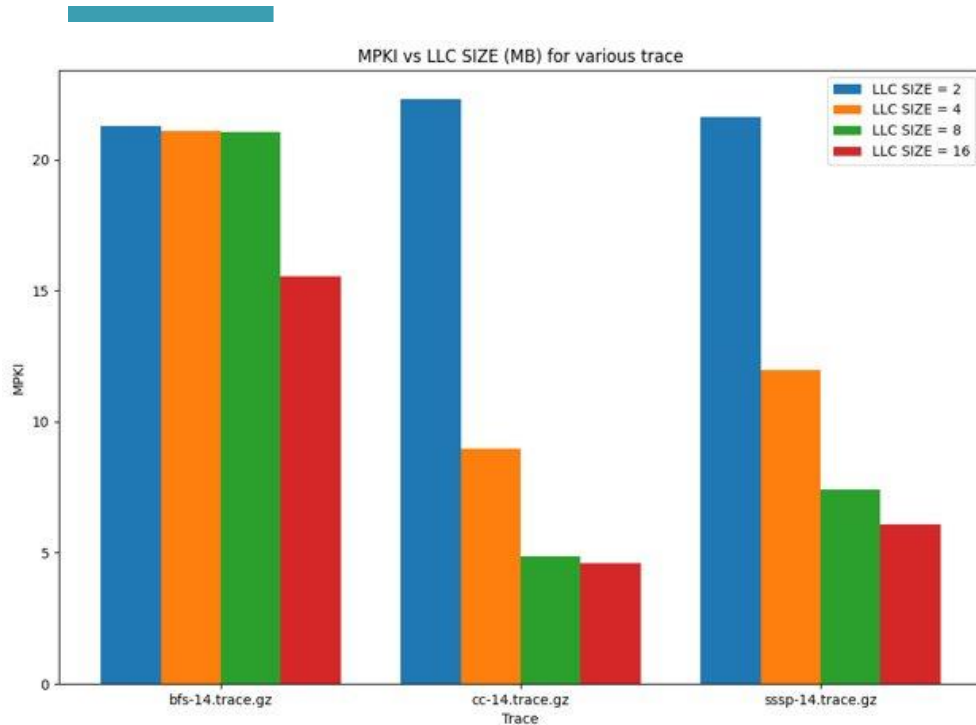
2) For cc and sssp traces – For LLC, we see an expected trend in which speedup increases with size and after a point it decreases slightly due to latency in accesses. It reaches a value of upto 1.4 whereas for L2, we observe that the speedup hardly increases with size , reaching a maximum of 1.2 when the size is increases to a very large value, that is , 8192 bytes.

# What happened when we actually changed cache sizes of L2 and LLC for Graph Analytics workload:



Observation 3: ( For this part we FOCUS the 0 size caches, that is, the purple lines)

- Bfs : Since the misses are independent of size of cache , removing the cache level, that is, setting its size to zero, presents a better speedup as we remove the overhead of latency of access associated with these caches.
- Cc and sssp : As performance was not much sensitive to changes in L2 size, we see that removing L2 gives us a speedup. The reason is that the reuse distance in graph workloads is such that L2 can never hold that data unless we use an accurate future predicting replacement policy. As performance was more sensitive to LLC size changes, we observe that turning off LLC cache causes slowdown.



Observation 4:

Bfs trace : As predicted by the speedup graph, we can see for both L2 and LLC, there is little or no change in MPKI with size increase due to the high reuse distance.

cc and sssp traces : Both show a linear decrement in MPKI.

Interesting Question : If both have same trend of MPKI with size, then why is there a difference in speedup?

If both have same trend of MPKI with size, then why is there a difference in speedup?

---

- Miss penalty = L1-Latency + Miss rate-L1\*(L2-Latency + Miss rate-L2\*(LLC-Latency+ Miss rate-LLC x DRAM Latency))
- Miss penalty is more sensitive to changes in L2 latency than LLC latency. Since, miss rate decrease is similar for both, overall changing LLC size is optimal for increasing IPC and hence the speedup is better.



# Analysis and Optimization of the Memory Hierarchy for Graph Processing Workloads

Abanti Basak\*, Shuangchen Li\*, Xing Hu\*, Sang Min Oh\*, Xinfeng Xie\*, Li Zhao<sup>†</sup>, Xiaowei Jiang<sup>†</sup>, Yuan Xie\*  
University of California, Santa Barbara\*    Alibaba, Inc.<sup>†</sup>

## B. Analysis of the Cache Hierarchy

**Observation#4:** *The private L2 cache shows negligible performance sensitivity, whereas the shared LLC shows higher performance sensitivity.* As shown in Fig. 4a, we vary the LLC size from 8MB to 64MB and find the optimal point of 17.4% (max 3.25X) performance improvement for a 4X increase in the LLC capacity. The mean LLC MPKI (misses per kilo instructions) is reduced from 20 in the baseline to 16 (16MB) to 12 (32MB) to 10 (64MB). The corresponding speedups are 7%, 17.4%, and 7.6%. The optimal point is a balance between a reduced miss rate and a larger LLC access latency.

## Validating our results

•Our observations are in agreement with Abanti et. al. This was our initial motivation for comparing L2 and LLC size changes. We changed the latency of cache levels wrt their size by extracting data from left paper.

## Exploring Core and Cache Hierarchy Bottlenecks in Graph Processing Workloads

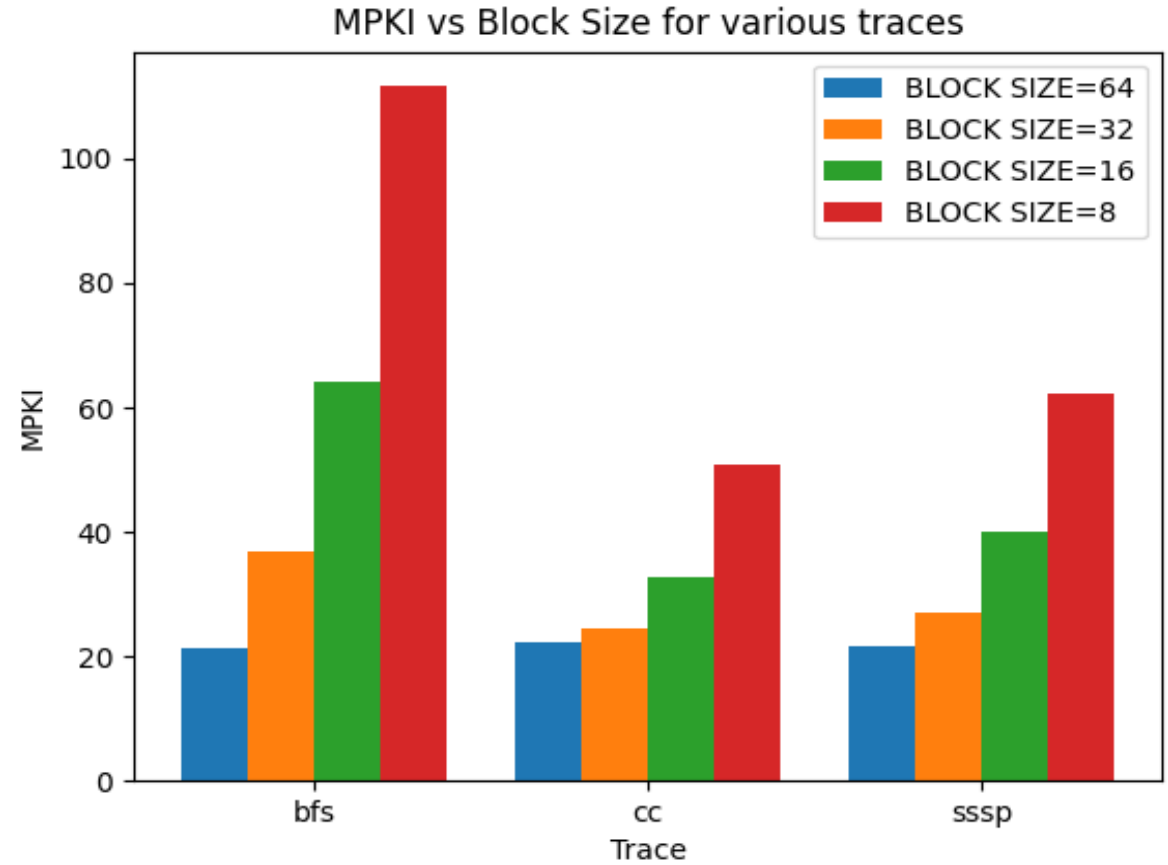
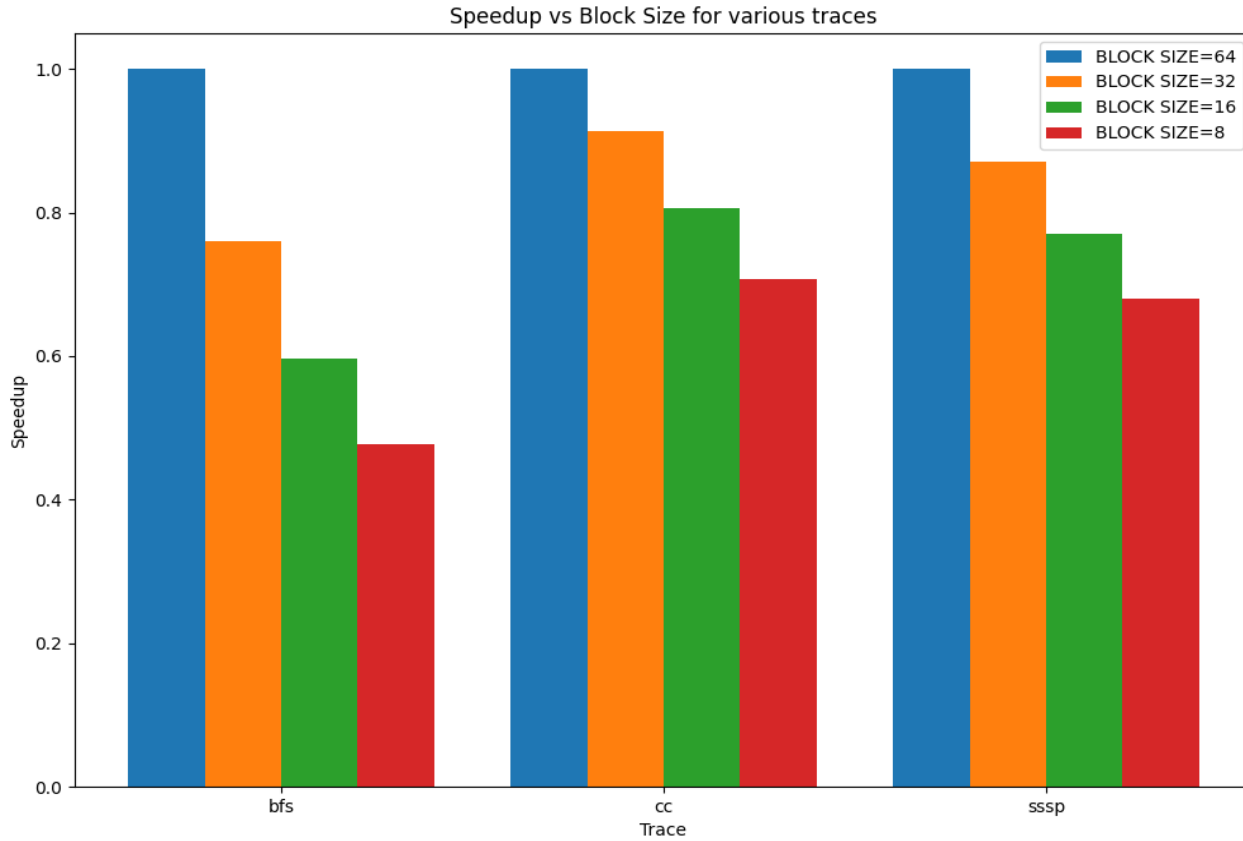
Abanti Basak<sup>ID</sup>, Xing Hu, Shuangchen Li,  
Sang Min Oh<sup>ID</sup>, and Yuan Xie

The leftmost bar in Fig. 5b(ii) represents an architecture with no private L2 caches and no slowdown compared to a 256 KB cache. Therefore, an architecture without private L2 caches is just as fine for graph processing.

*Property data is the primary beneficiary of LLC capacity:* To understand which data type benefits from a larger LLC, Fig. 5c shows, for each data type, the percentage of memory references that ends up getting data from the DRAM. We observe that the most reduction in off-chip accesses comes from the property data. Structure and intermediate data do not benefit from a higher capacity. Intermediate data is already mostly accessed in on-chip caches since only 1.9 percent of the accesses to this data type are DRAM-bound in the baseline. Structure data, on the other hand, has a higher percentage of off-chip accesses (7.5 percent) which remains mostly unresponsive to a larger LLC capacity.

*Graph structure cacheline has the largest reuse distance among all the data types. Graph property cacheline has a larger reuse distance than that serviced by the L2 cache:* To further understand the different per-

# Changing Block Size keeping constant size of cache.



Didn't work out :\*(

## Motivation

- The indirect memory access that are part of dependency chain in the graph processing kernel, are not streaming accesses. They hence exhibit poor cache locality and utilization. So we expected that reducing the block size ( keeping cache size same by increasing ways ) might give us an improvement

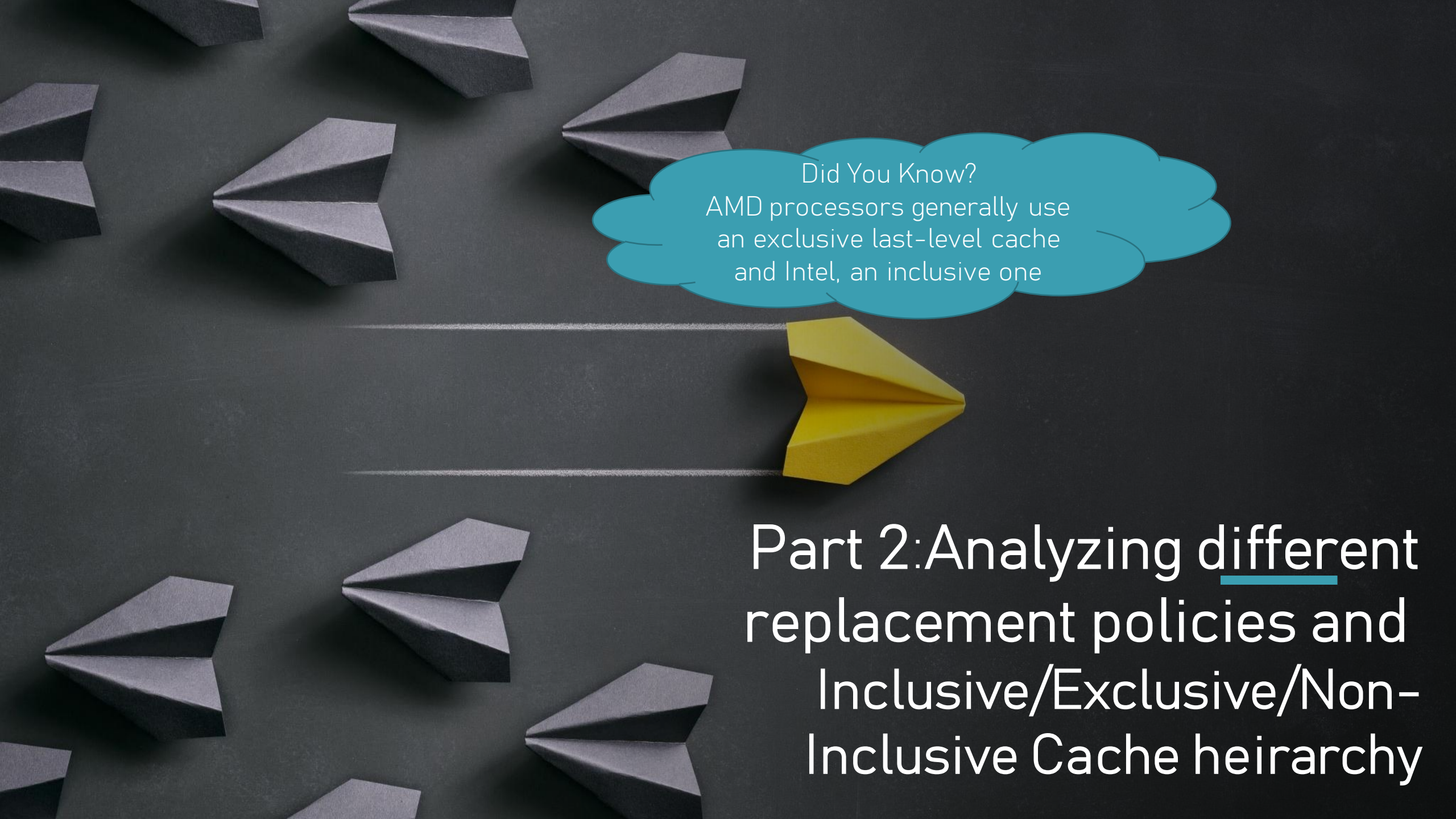
## This optimization failed

- However, we failed, and as expected for a regular workload, on decreasing block size, speedup decreased and MPKI increased significantly.

## Why did we fail

- A possible reason is that although their access frequency is high, the total portion of memory that is accessed indirectly is quite less (3-5%)



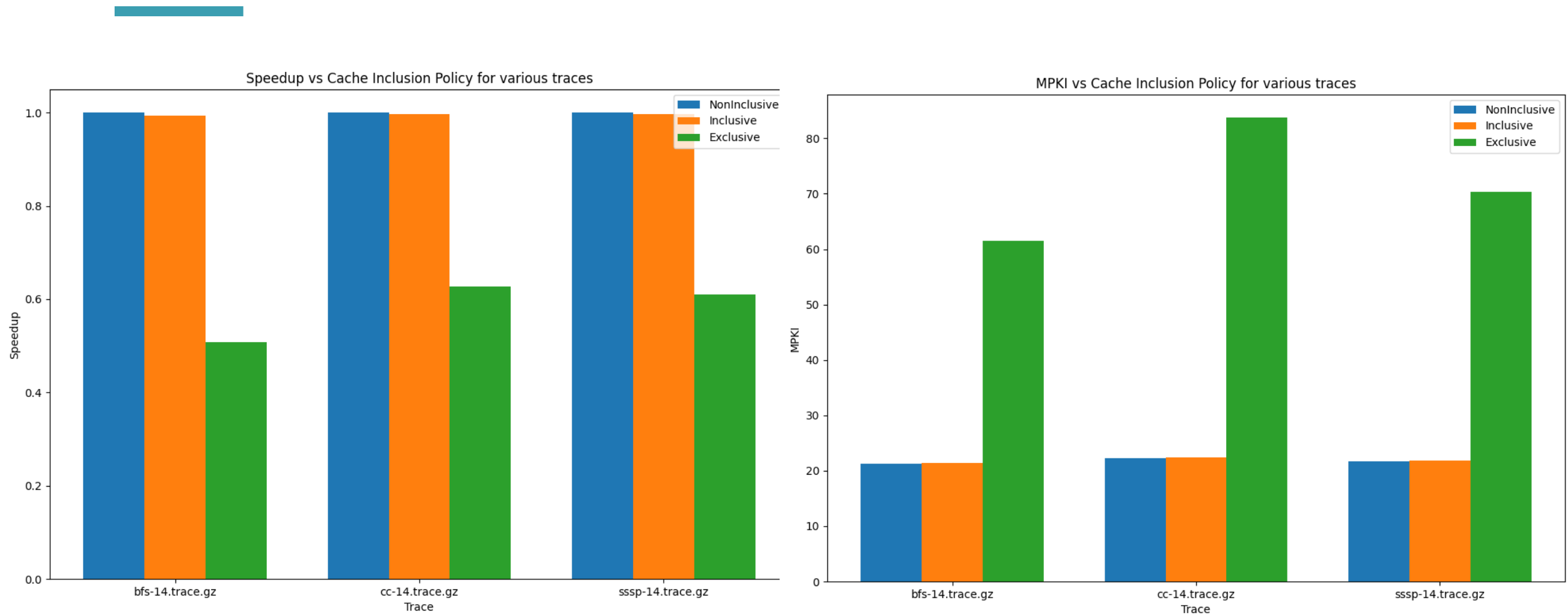


Did You Know?  
AMD processors generally use  
an exclusive last-level cache  
and Intel, an inclusive one

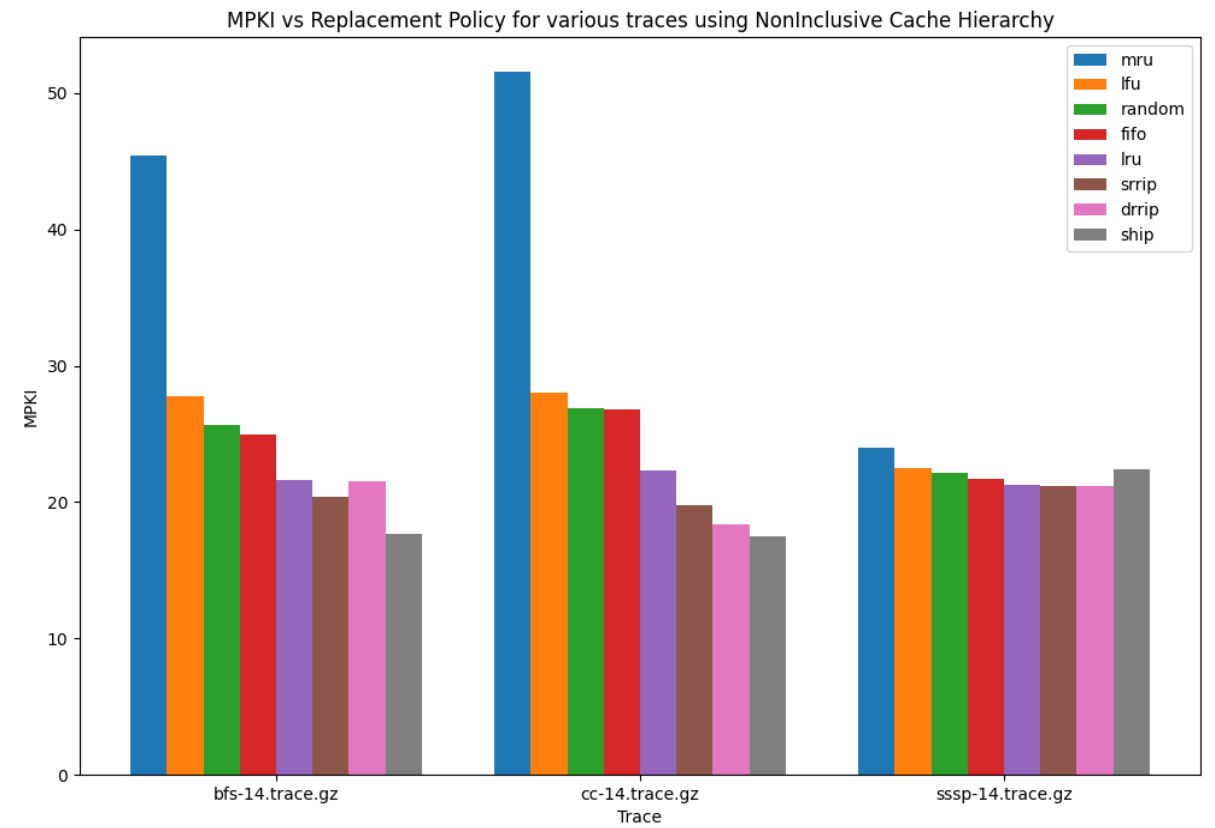
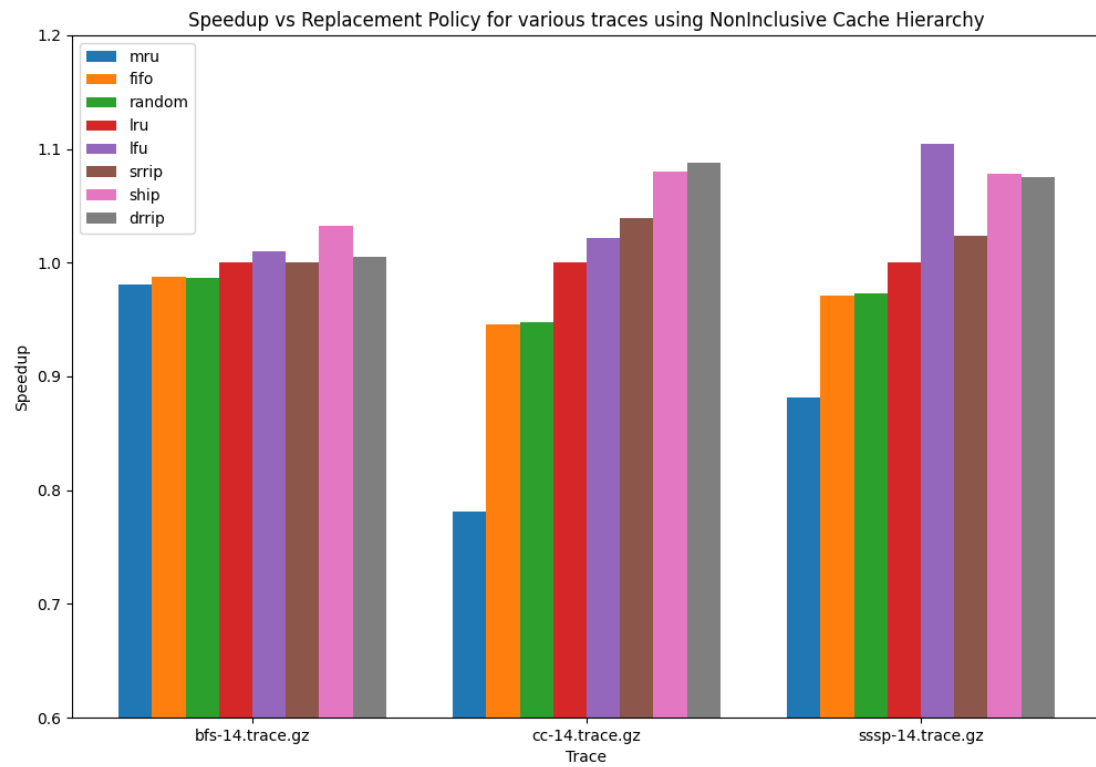
## Part 2: Analyzing different replacement policies and Inclusive/Exclusive/Non- Inclusive Cache hierarchy



# Comparison of Inclusive, Exclusive and Non-inclusive

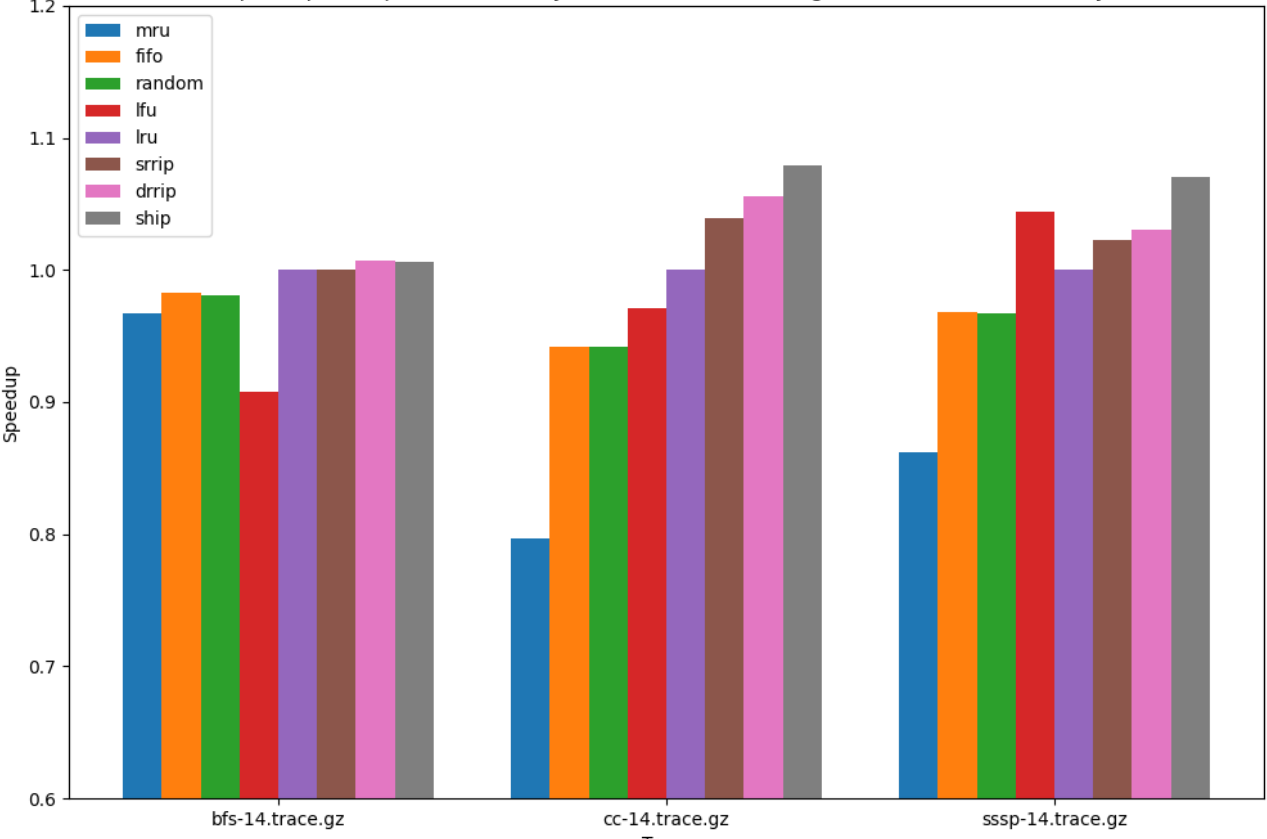


# Non-inclusive

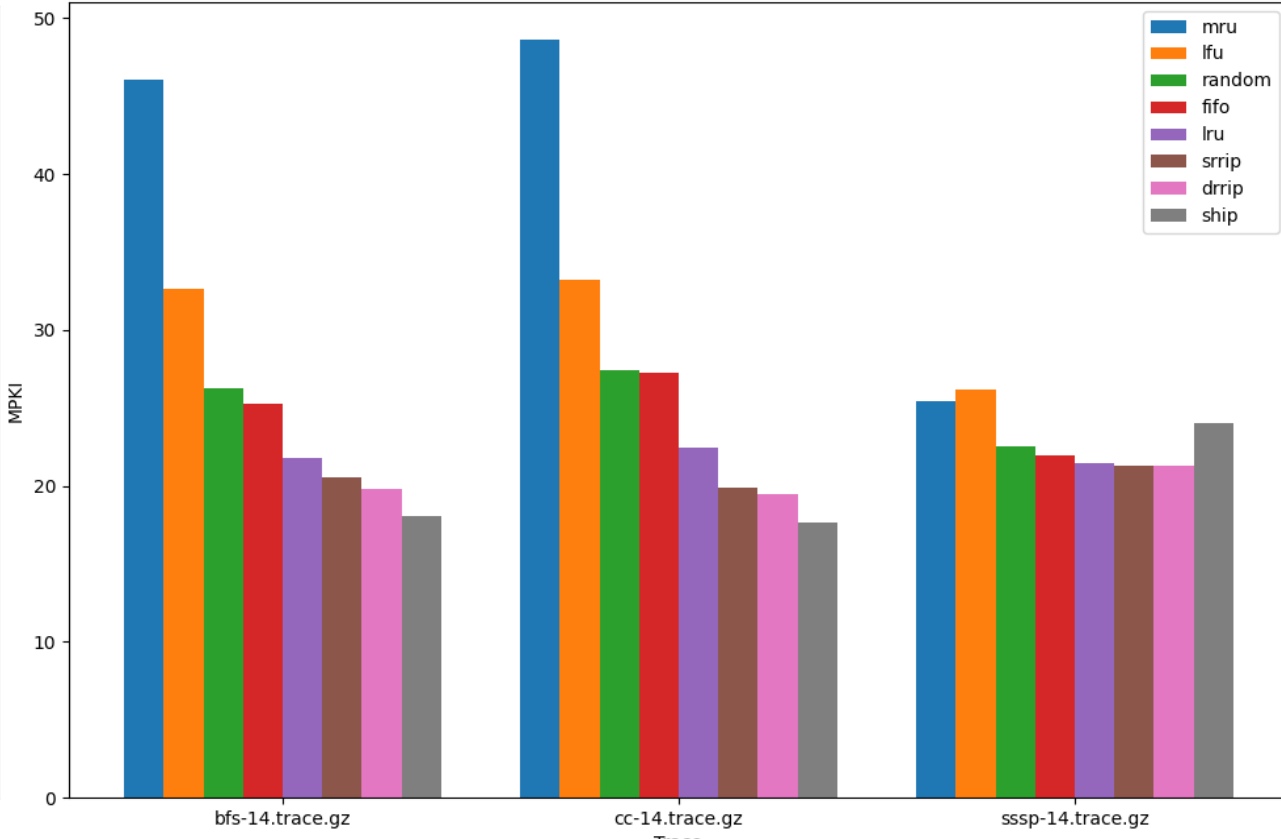


# Inclusive

Speedup vs Replacement Policy for various traces using Inclusive Cache Hierarchy

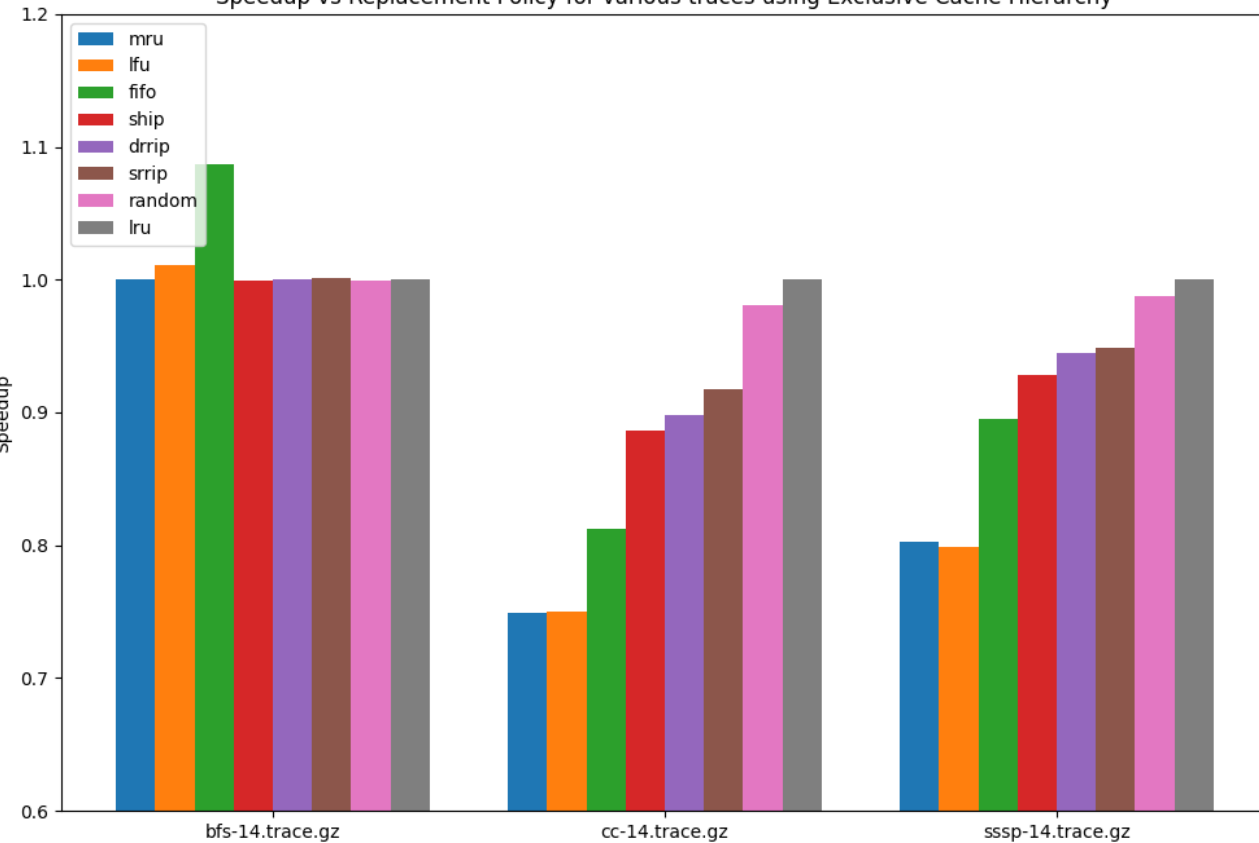


MPKI vs Replacement Policy for various traces using Inclusive Cache Hierarchy

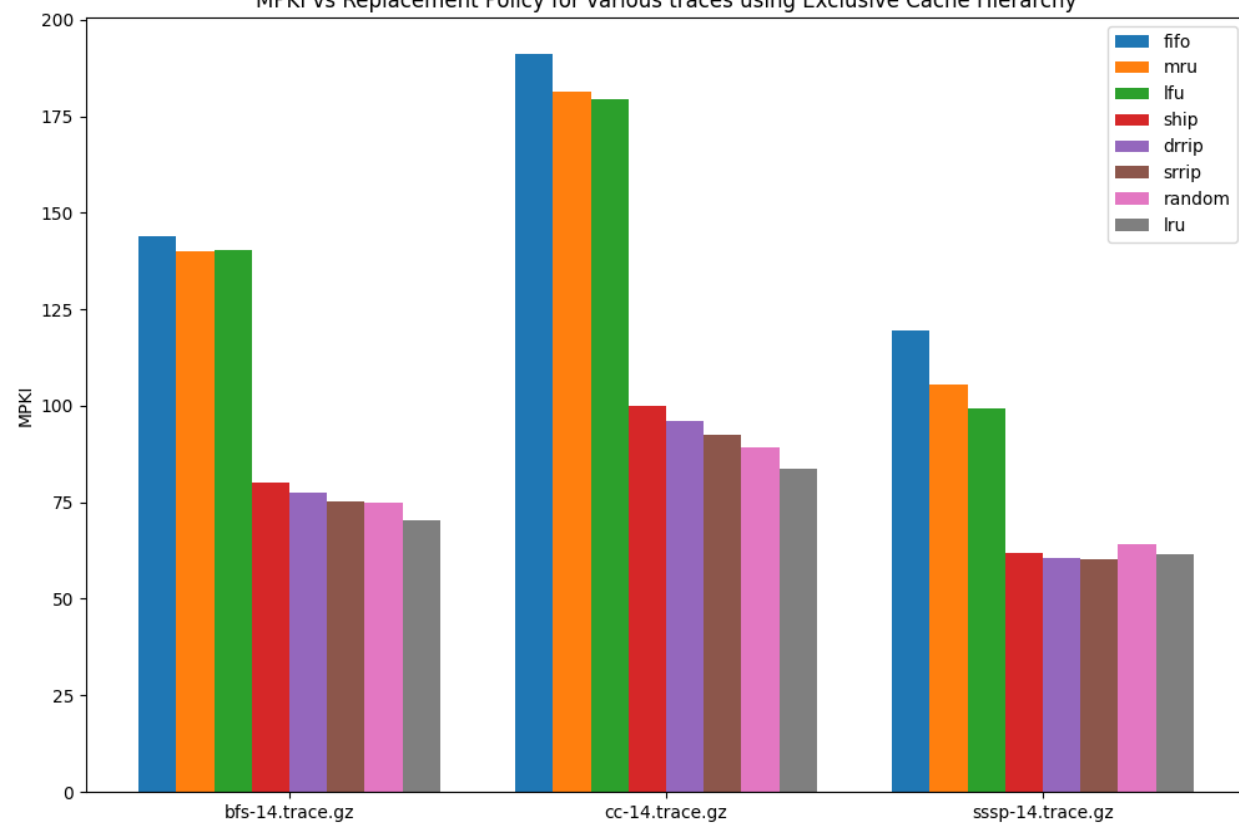


# Exclusive

Speedup vs Replacement Policy for various traces using Exclusive Cache Hierarchy



MPKI vs Replacement Policy for various traces using Exclusive Cache Hierarchy





# EXPECTATION

- We expect frequency based replacement policies to work better than recency based, because the latency **bottleneck in most graph workloads are the high reuse distance, repeatedly accessed memory locations.**

# REALITY

- Our expectation matches in case of non inclusive hierarchy
- For exclusive, all replacement policies perform poorly except LRU.
- For inclusive, SHIP works the best for the replacement policy.



# Validation of Results

## EVALUATION OF CACHE INCLUSION POLICIES IN CACHE MANAGEMENT

A Thesis

by

LUNA BACKES DRAULT

Submitted to the Office of Graduate and Professional Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE

SHiP stands out in inclusive caches. In many prefetcher configurations in an inclusive cache, SHiP is the best compared to other replacement policies. It is also

### 6.1.3 Replacement Policy Impact

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.

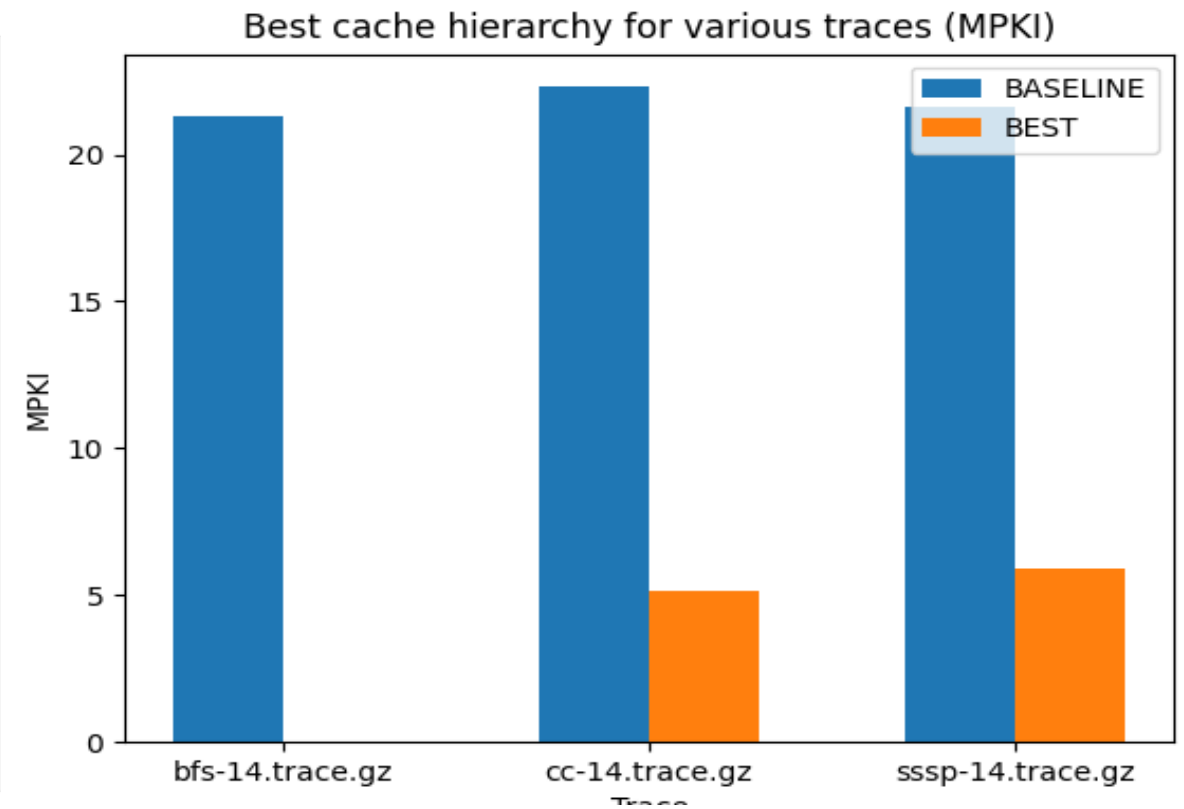
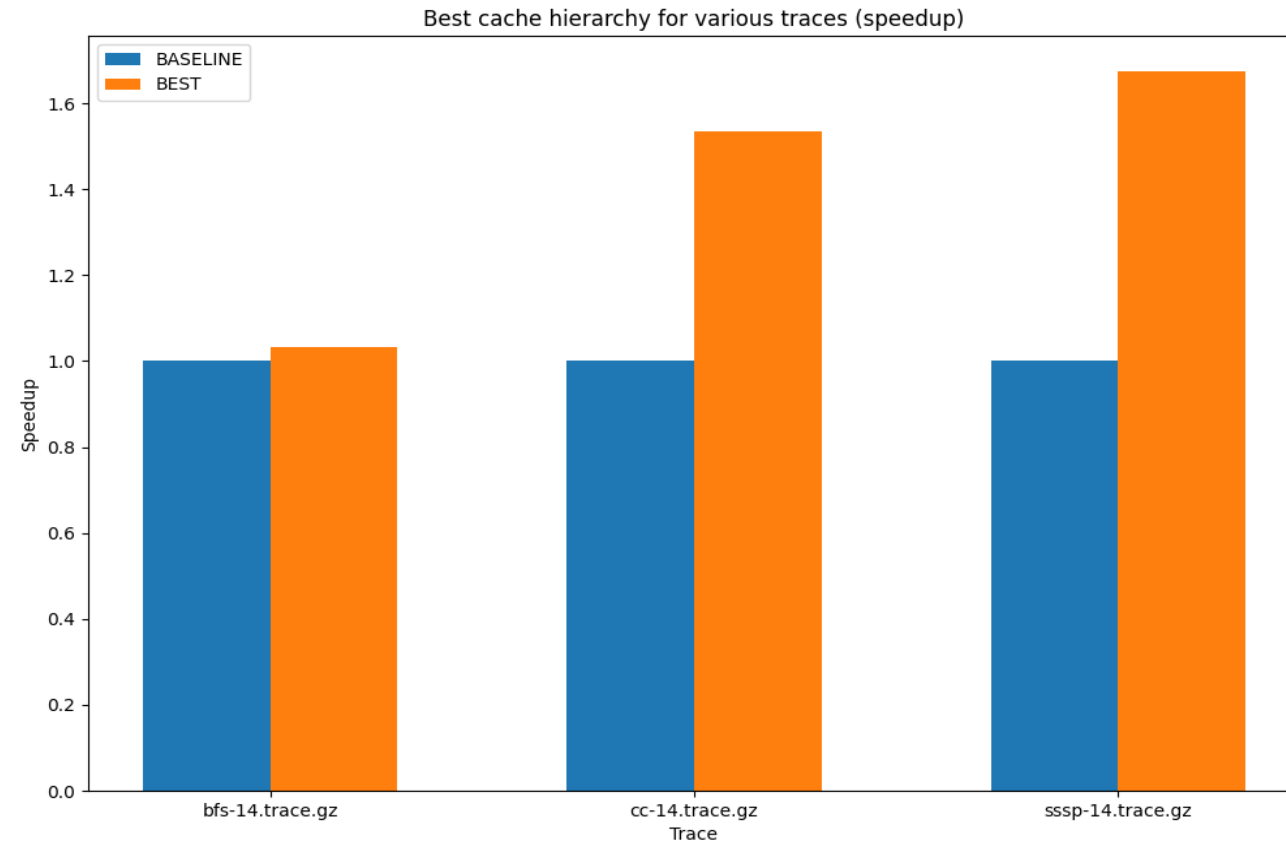
## Graphfire: Synergizing Fetch, Insertion, and Replacement Policies for Graph Analytics

Aninda Manocha, *Student Member, IEEE*, Juan L. Aragón, *Member, IEEE*, and Margaret Martonosi, *Fellow, IEEE*

**Replacement:** Despite their poor locality, a subset of PIAs have high reuse and benefit from caching [7], but their reuse distances are often too long to protect them from eviction. When the LLC is reserved for PIAs cached at word granularity, frequency-based replacement is significantly more effective at learning and retaining PIAs with high reuse, unlike across-the-board temporal locality.



# Finally we take our best results from all these observations and combine them to generate our final cache hierarchy. Final result !



**\*\*The MPKI measured is for LLC cache**

bfs : LLC = 0, Non Inclusive, L2C=512kB,  
cc: L2 = 0, Non Inclusive, LLC=8MB , DRRIP  
sssp : L2 = 0, Non Inclusive, LLC=8MB, LFU



---

Additional  
Exploration (Not  
Implemented)

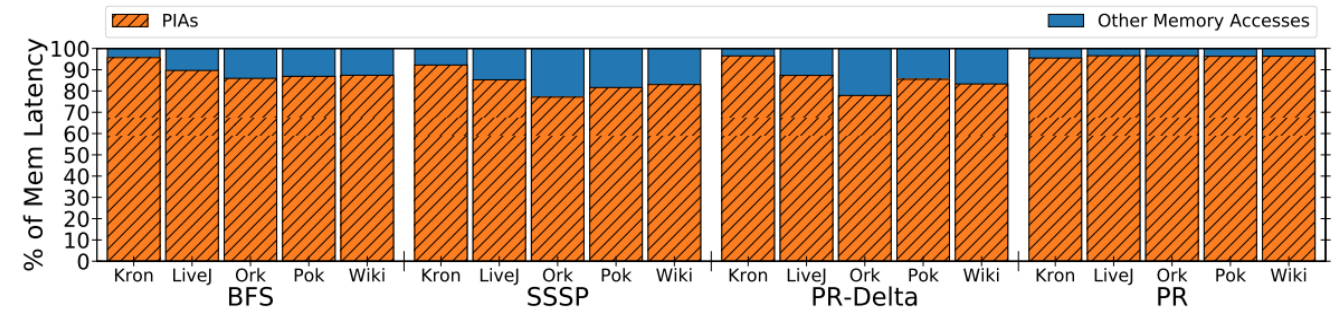
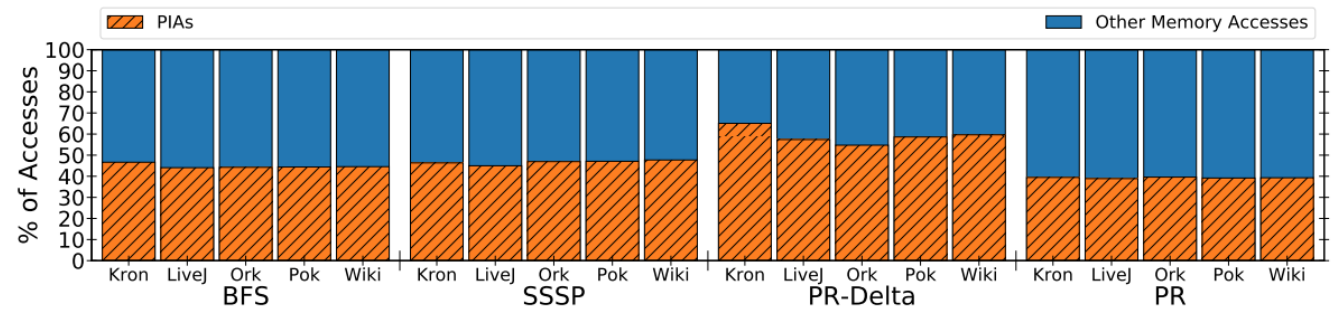


# Graphfire

A flexible, hardware-based memory hierarchy approach that

- (i) learns when PIA(Primary Indirect Accesses) occur in graph applications and
- (ii) optimizes their performance through tailored fetch, insertion, and replacement policies.

Four different access patterns in any graph workload : Infrequent/Primary  
Streaming/Indirect





# Re-Reference Table

learns which instructions are responsible for what type of access pattern

- Contains 4 fields :
  - (1)the PC (2)a Utilization Value (UV) (3)L1 cache set ID (4) a frequency value
- Cacheline Utilization : inferred by UV value
- PLA Identification : identified using the frequency value ( a saturating counter )
- Negligible hardware overhead

# Cache Policies for Primary Indirect Accesses

---

## Fetch: Tailored Access Granularity

- Unlike streaming accesses, PIAs poorly utilize cachelines. So, Graphfire fetches data at word granularity

## Insertion: Data-Aware Caching

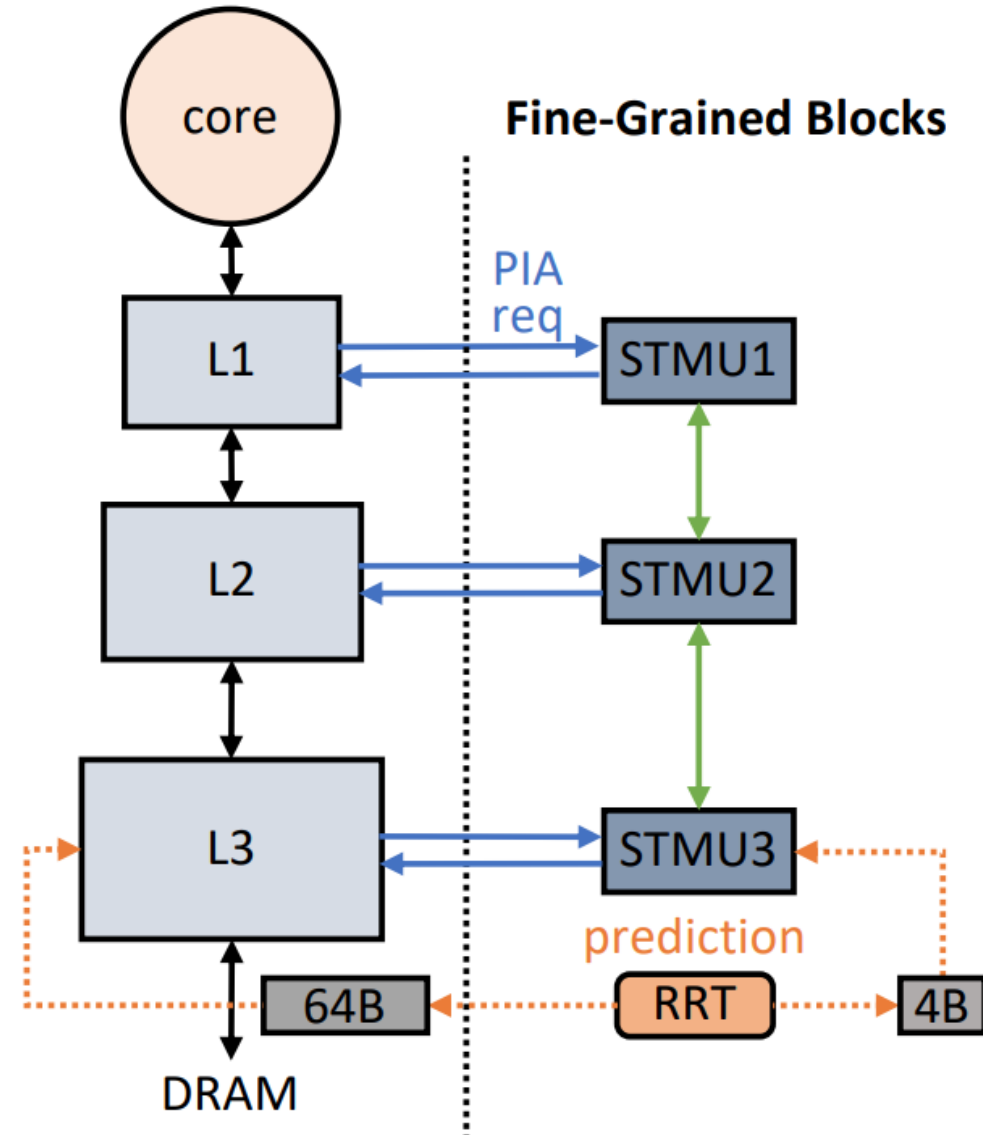
- Thus, all non-PIA accesses bypass these caches because they do not gain much benefit from them. This prevents other accesses from evicting high-reuse PIAs from the LLC.

## Replacement: Frequency-Based Eviction

- Graphfire applies Frequency Based Replacement in the lower cache levels instead of relying on recency.

# Memory Hierarchy Design

- Two types of cache blocks :
  - normal 64B cache blocks
  - merged blocks, of coalesced, non-contiguous sub-blocks that store data for fine-grained accesses
- STMU (sub-tag matching unit) : perform operations on these specialized merged blocks





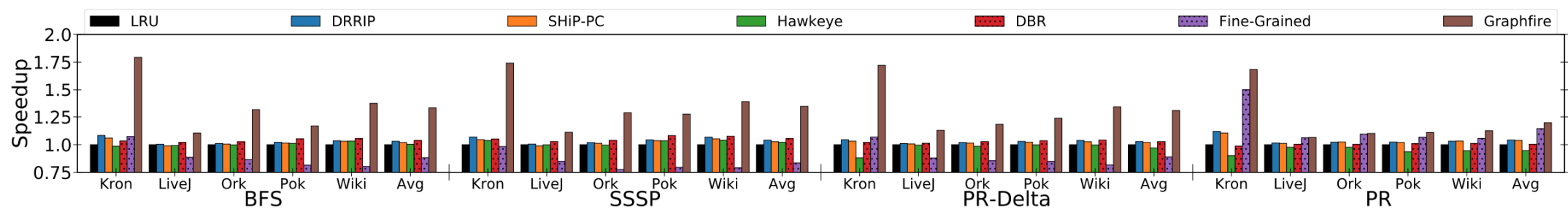


Fig. 14. Speedup comparisons of Graphfire with state-of-the-art and graph-specialized cache management policies. Through automatically identifying PIAs and exploiting synergies between its techniques, Graphfire significantly outperforms all prior cache management policies without relying on software preprocessing.

# THANK YOU



Deprived of Sleep, High on caffeine :)



# References

## 1) Analysis and Optimization of the Memory Hierarchy for Graph Processing Workloads

-Abanti Basak, Shuangchen Li, Xing Hu, Sang Min Oh, Xinfeng Xie, Li Zhao, Xiaowei Jiang, Yuan Xie

## 2) Data-Aware Cache Management for Graph Analytics

-Neelam Sharma, Varun Venkitaraman, Newton, Vikash Kumar, Shubham Singhania, Chandan Kumar Jha

## 3) Graphfire: Synergizing Fetch, Insertion, and Replacement Policies for Graph Analytics

-Aninda Manocha, Student Member, IEEE, Juan L. Aragón, Member, IEEE, and Margaret Martonosi, Fellow, IEEE

## 4) Gretch: A Hardware Prefetcher for Graph Analytics

-ANIRUDH MOHAN KAUSHIK, GENNADY PEKHIMENKO, HIREN PATEL

## 5) Exploring Core and Cache Hierarchy Bottlenecks in Graph Processing Workloads

-Abanti Basak , Xing Hu, Shuangchen Li, Sang Min Oh , and Yuan Xie

## 6) EVALUATION OF CACHE INCLUSION POLICIES IN CACHE MANAGEMENT

-A Thesis by LUNA BACKES DRAULT