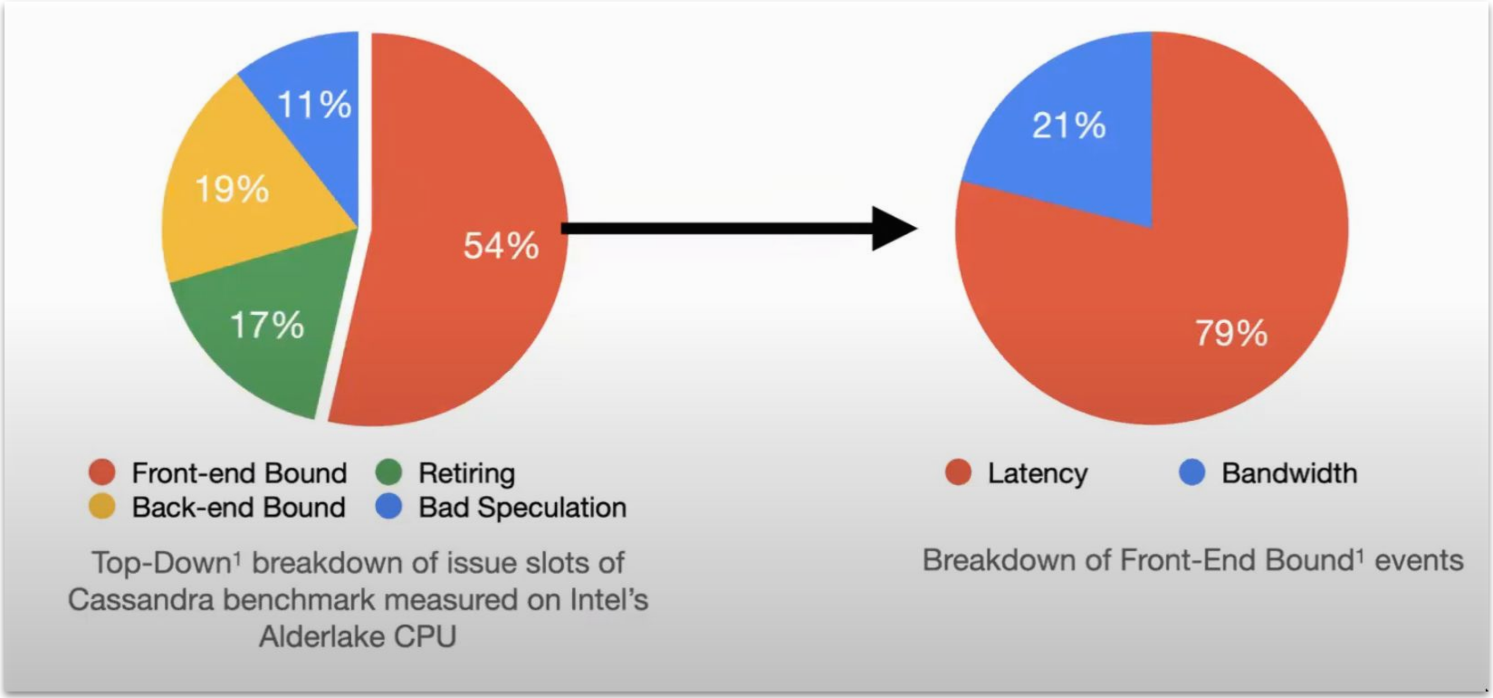
Microarchitecture for Datacenter Workloads

RnD Presentation

Hrishikesh Jedhe Deshmukh
210050073

Overview of the problem

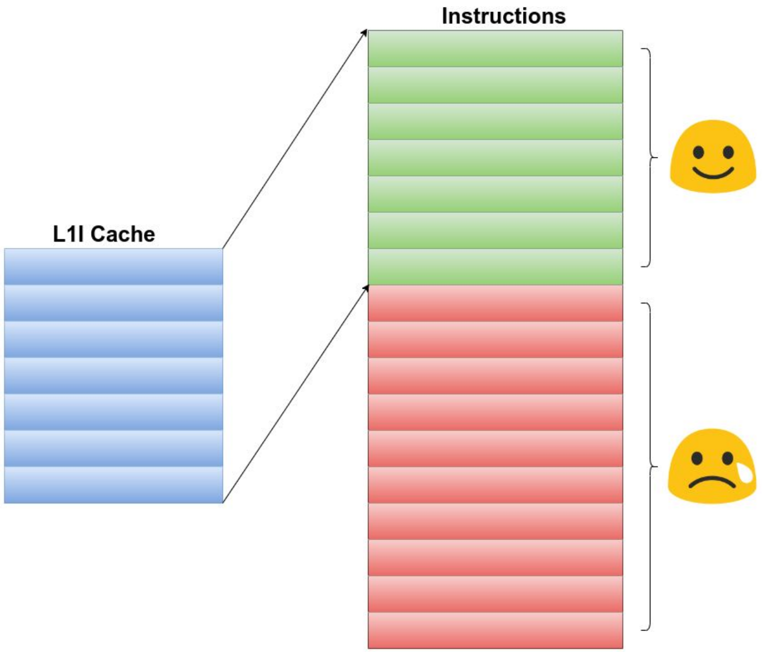


Credits: ISCA'23: The 50th International Symposium on Computer Architecture
Lightning Talks - Session2B: Memory Systems

What's special about Datacenter Workloads?

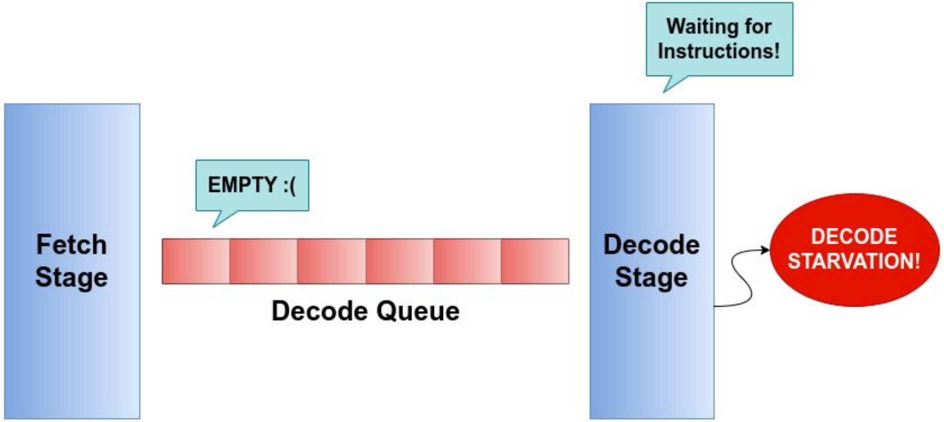
- Datacenter Workloads have very large code footprint i.e. the unique instructions are large in number
- L1I cache is not able to store the working set of instructions
- Leads to frequent high latency misses to fetch instructions from higher level caches or memory

Causes DECODE STARVATION!



What's Decode Starvation?

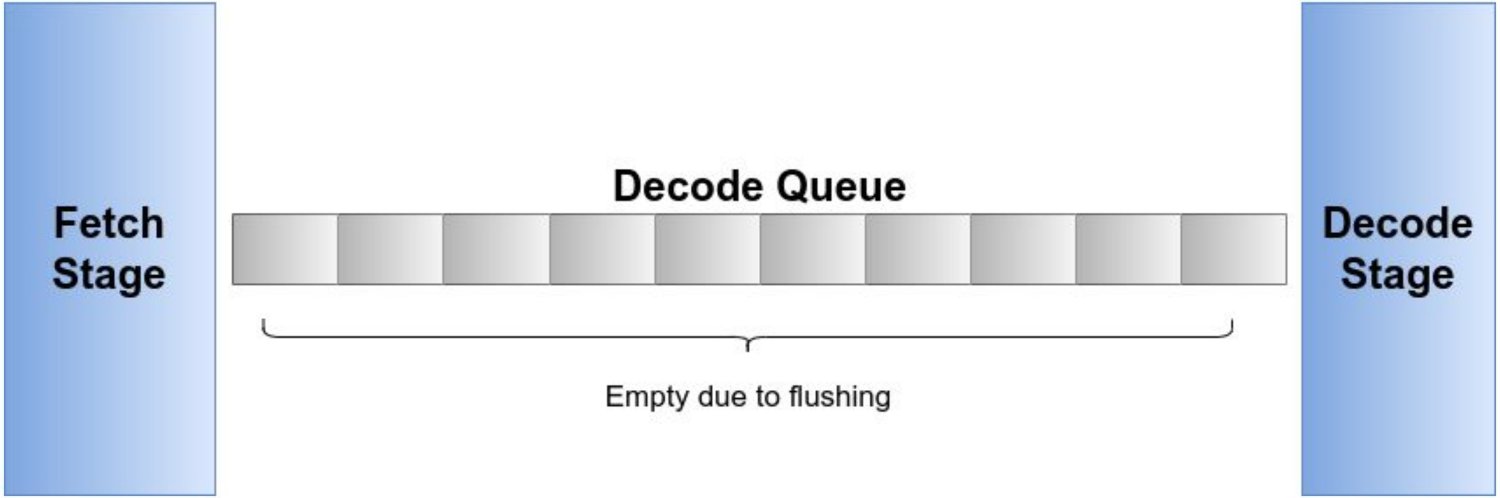
- State where decode stage is waiting for instructions
- Decode Queue is empty :(
- As frontend is stalled, all further stages get affected



Decode Starvation in Modern Processors

- Branch misprediction invalidates early fetch work
- Requires flush of the processor pipeline
- Re-steering of front end takes time
- Even more time is needed for fetch to run far ahead of decode to tolerate L1I misses
- Decode stage isn’t fed with instructions due to this

Decode Starvation on Modern Processors contd.



What to do?

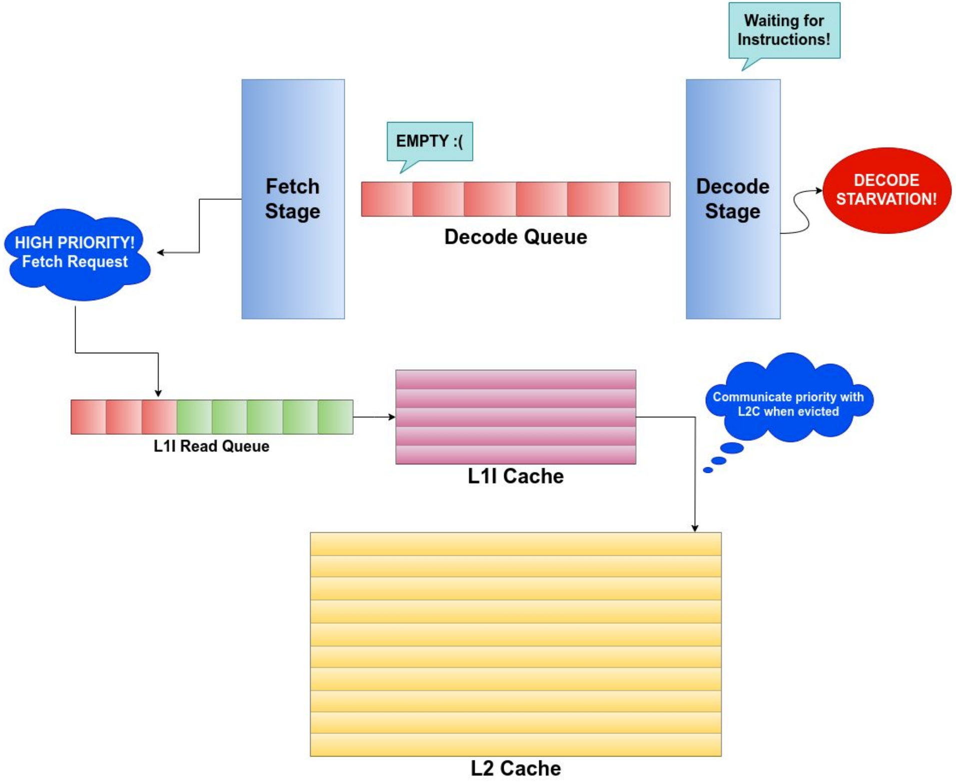
- Not all cache misses are equal!
- Some instruction line misses cause decode starvation while some don’t
- Preserve the instruction lines whose misses cause decode starvation

EMISSARY

- **E**nhanced **Miss A**wareness **R**eplacement Polic**y** for L2 Instruction Caching
- Prioritizes instruction lines whose miss causes decode starvation
- These high priority lines are preserved in L2 upon eviction from L1I cache

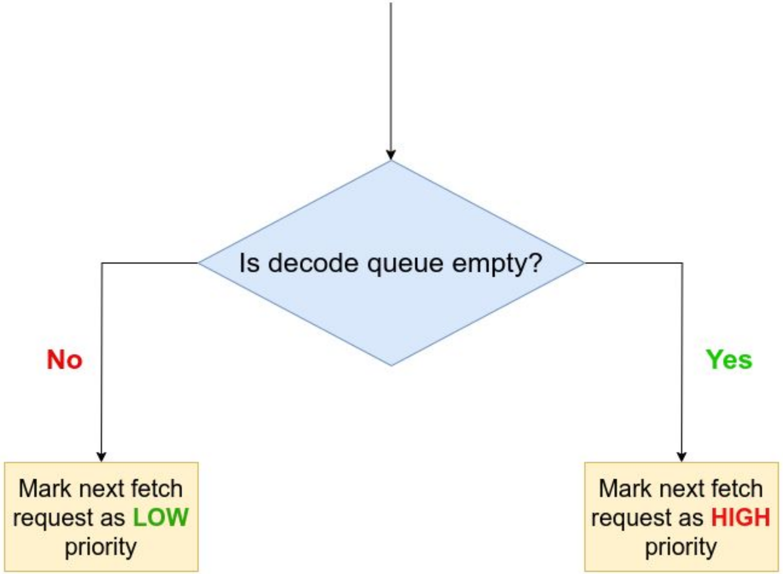
Source - ISCA '23: Proceedings of the 50th Annual International Symposium on Computer Architecture
https://dl.acm.org/doi/abs/10.1145/3579371.3589097

EMISSARY contd.



How is Priority decided?

- If the processor is currently in decode starvation i.e. the decode queue is empty, the next fetch request is marked as high priority
- Priority is then communicated with L1I and then subsequently with L2 on eviction from L1I



How are High Priority lines preserved in L2?

- We try to maintain *N* high priority lines in every set of L2 cache
- Rest of the lines in cache set can be used for low priority instruction lines and data lines
- *N* is usually 4 or 8 for a 16 way associative L2 cache

Algorithm 1 The EMISSARY Eviction Policy

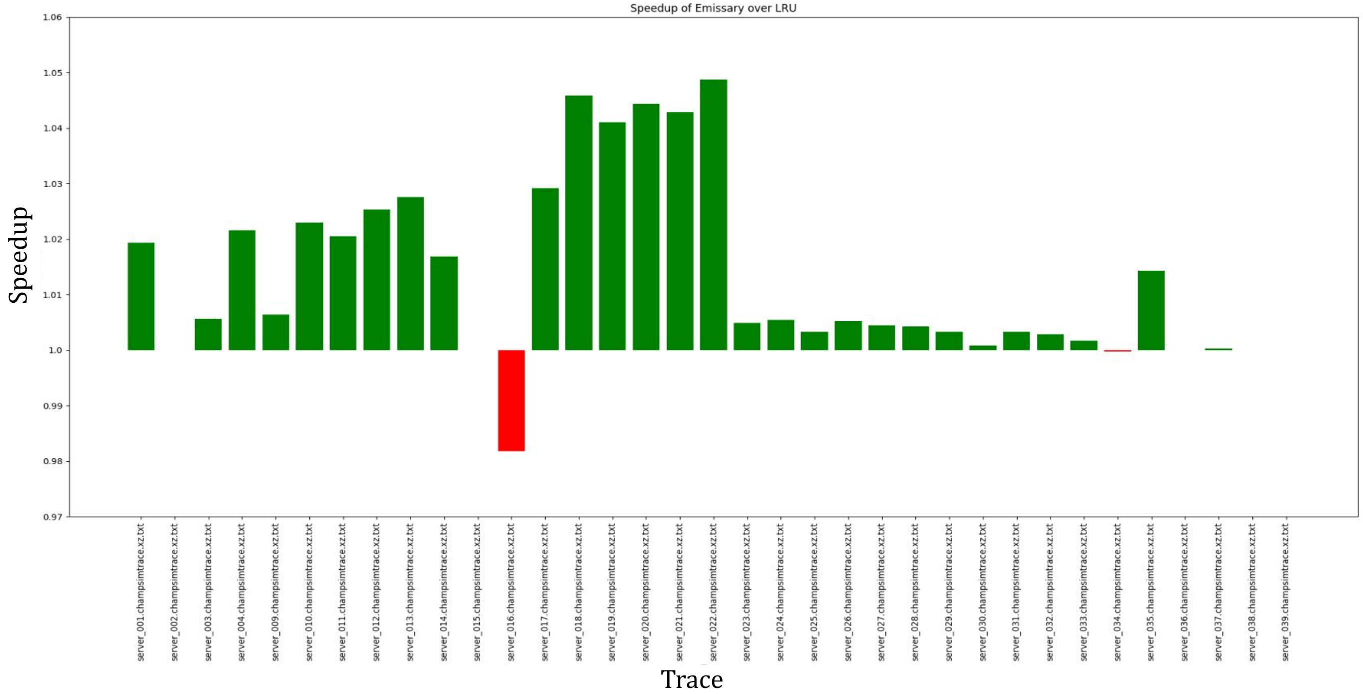
```
if number of high-priority (P = 1) lines <= N then
    Evict the LRU among the low-priority (P = 0) lines
else
    Evict the LRU among high-priority lines
end if
```

Why L2? Why not L1I?

- L1I misses are well tolerated by modern processors
- Fetch stage runs quite ahead of decode stage so L1I miss latency gets hidden
- Longer reuse intervals make L2 more appropriate than L1I

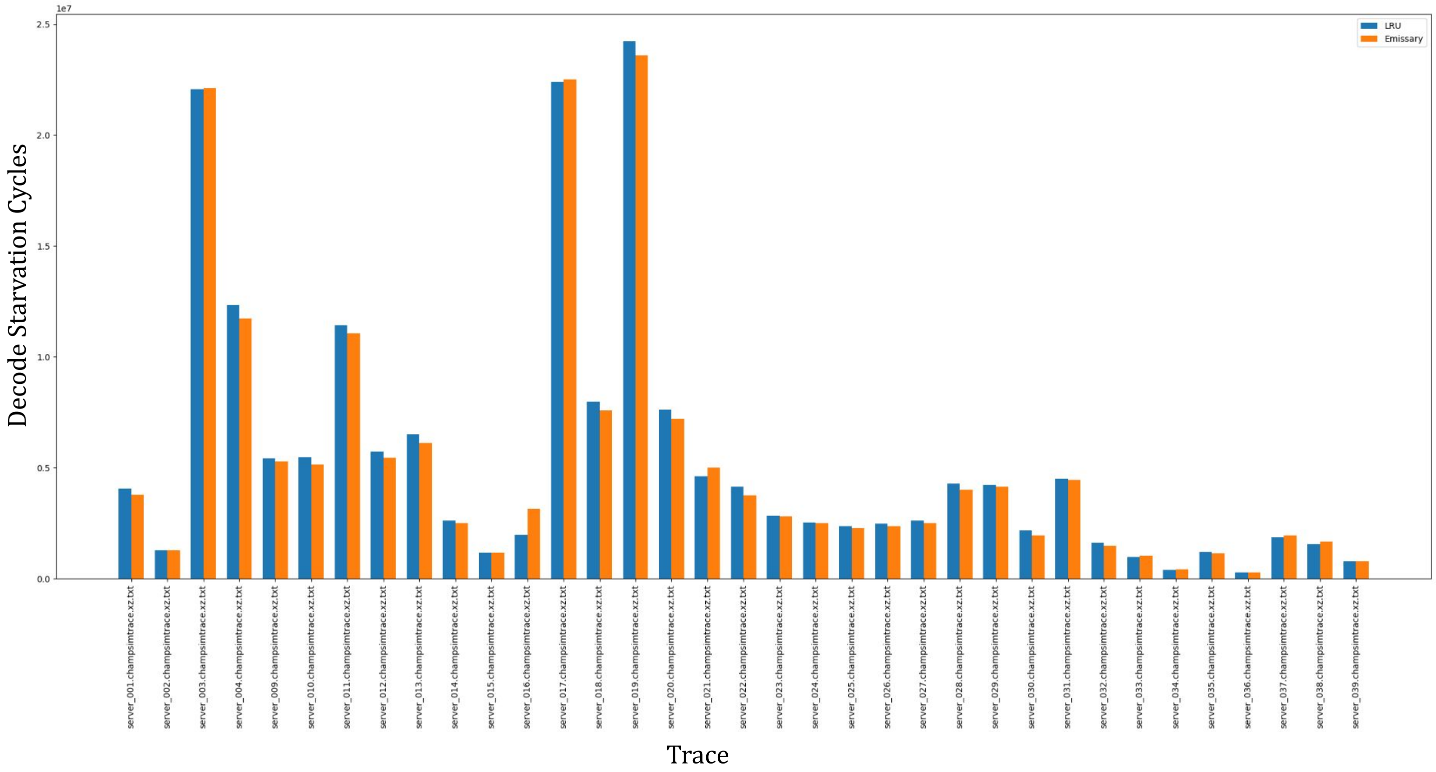
EMISSARY vs LRU - Speedup

CVP server traces simulated on ChampSim with 50M warmup and 50M simulation instructions



Geo Mean speedup of 1.012

EMISSARY vs LRU - Decode Starvation Cycles



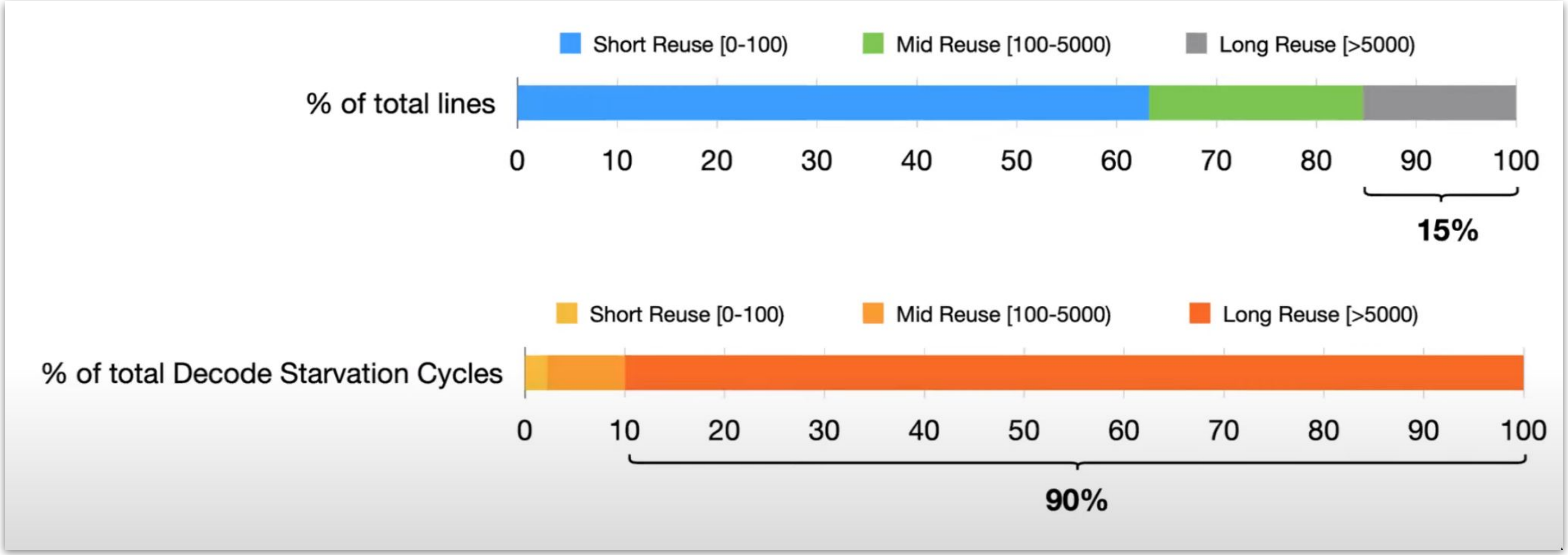
Reuse Distance

- Number of unique lines accessed between two consecutive accesses of a line is the Reuse Distance of that line

a b c b d c e a

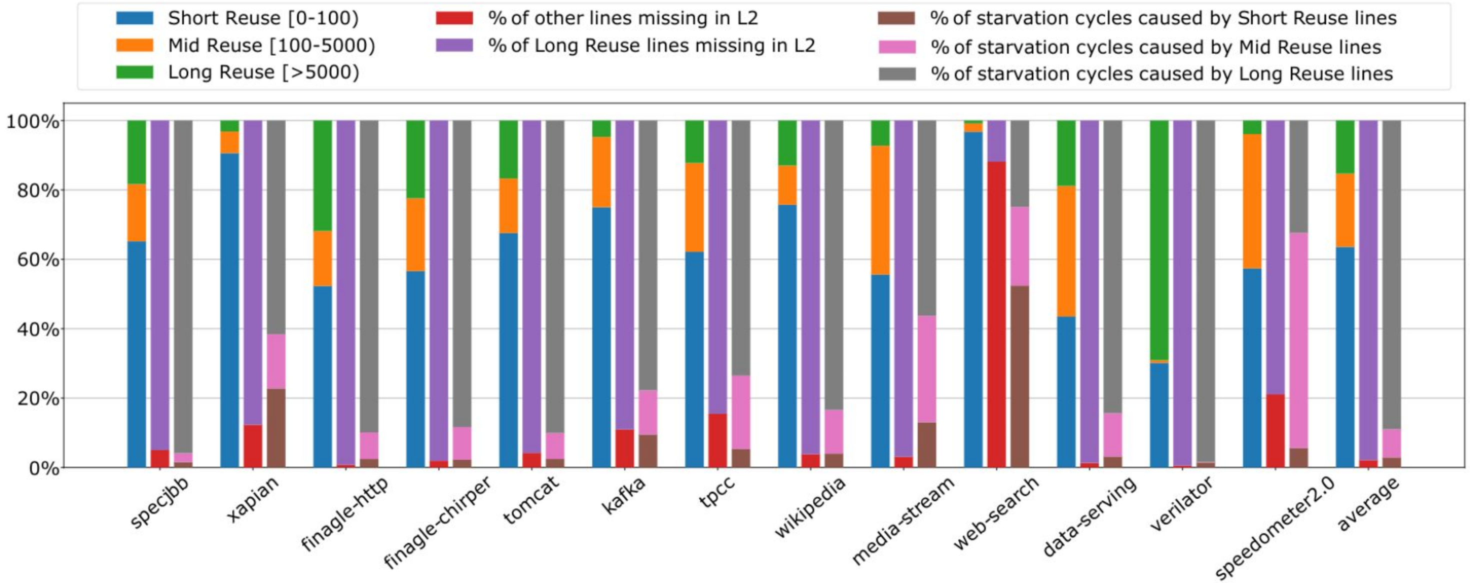
- a : 4
- b : 1
- c : 2

Observations from EMISSARY paper



Credits: ISCA'23: The 50th International Symposium on Computer Architecture
Lightning Talks - Session2B: Memory Systems

Observations from EMISSARY paper

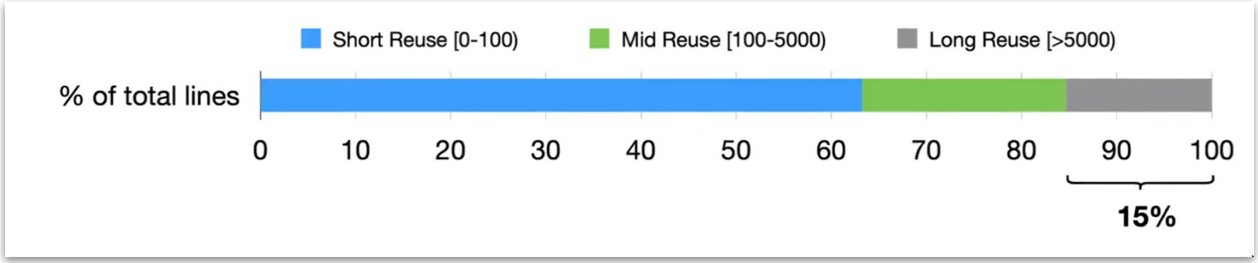


Long reuse lines account for >90% of decode starvation cycles while being only 15% in proportion

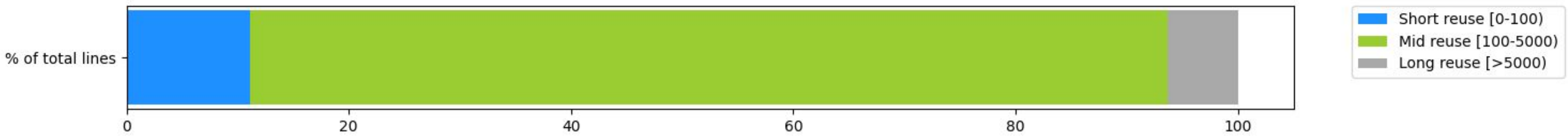
Source : https://dl.acm.org/doi/abs/10.1145/3579371.3589097

Comparing observations

From EMISSARY paper:



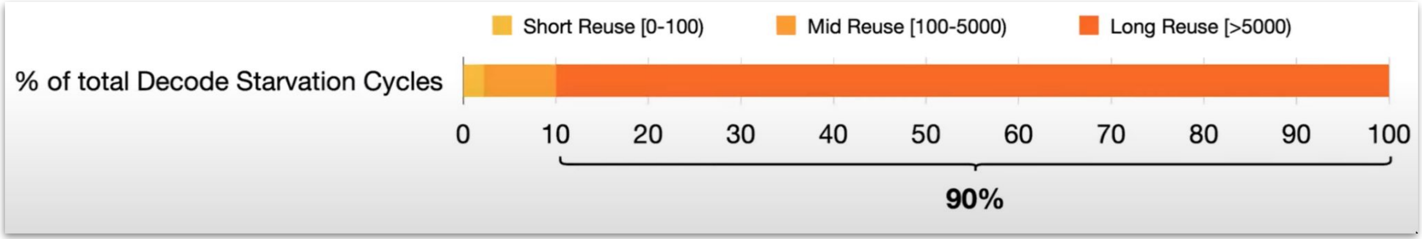
Our observations:



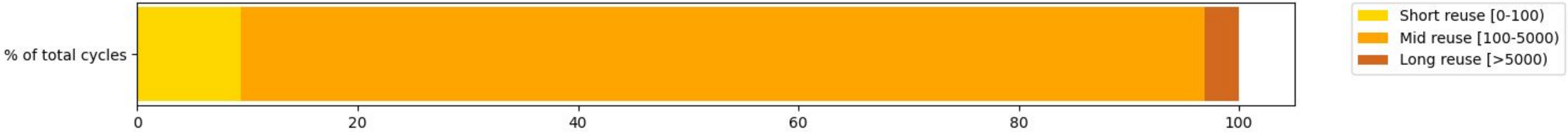
Note: Traces are different

Comparing observations

From EMISSARY paper:

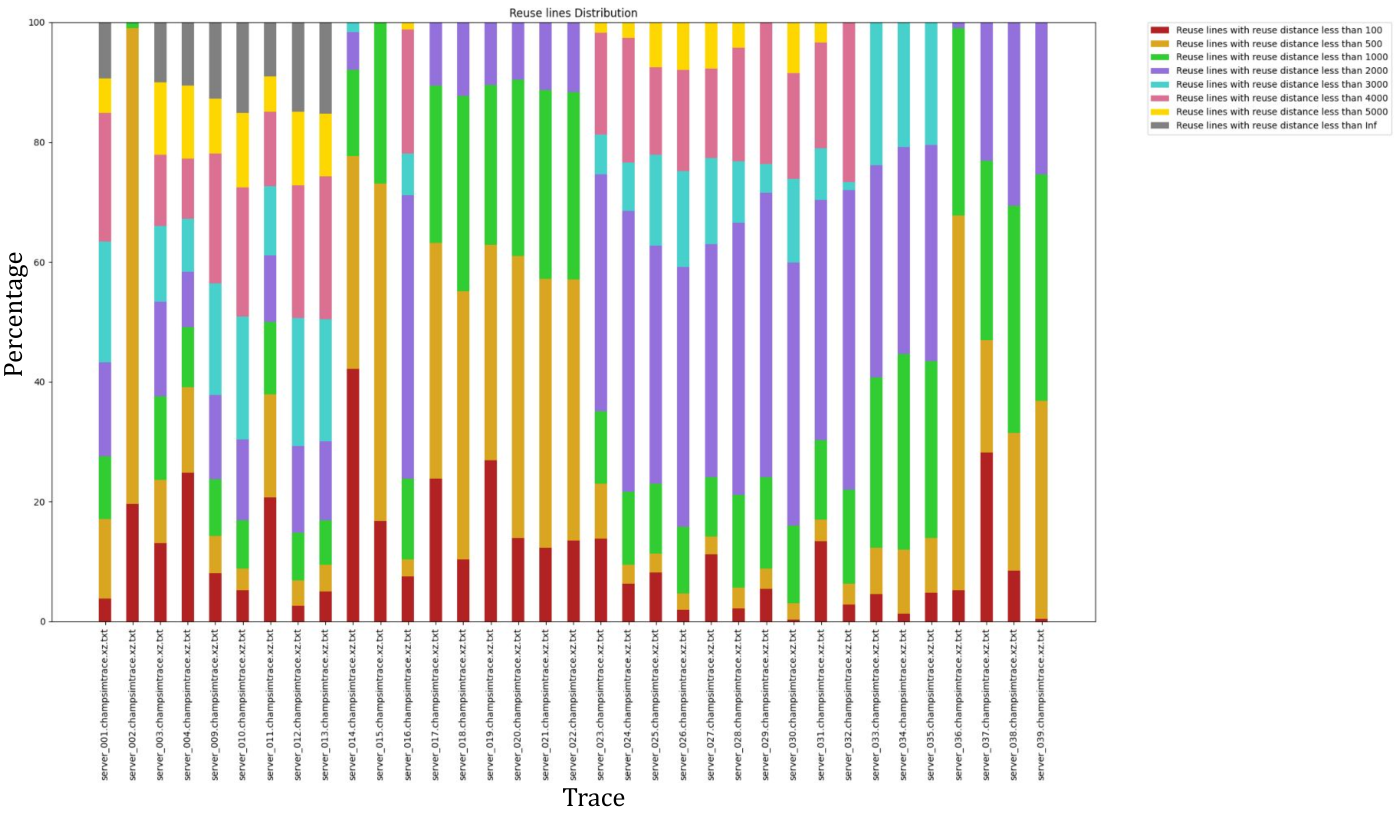


Our observations:

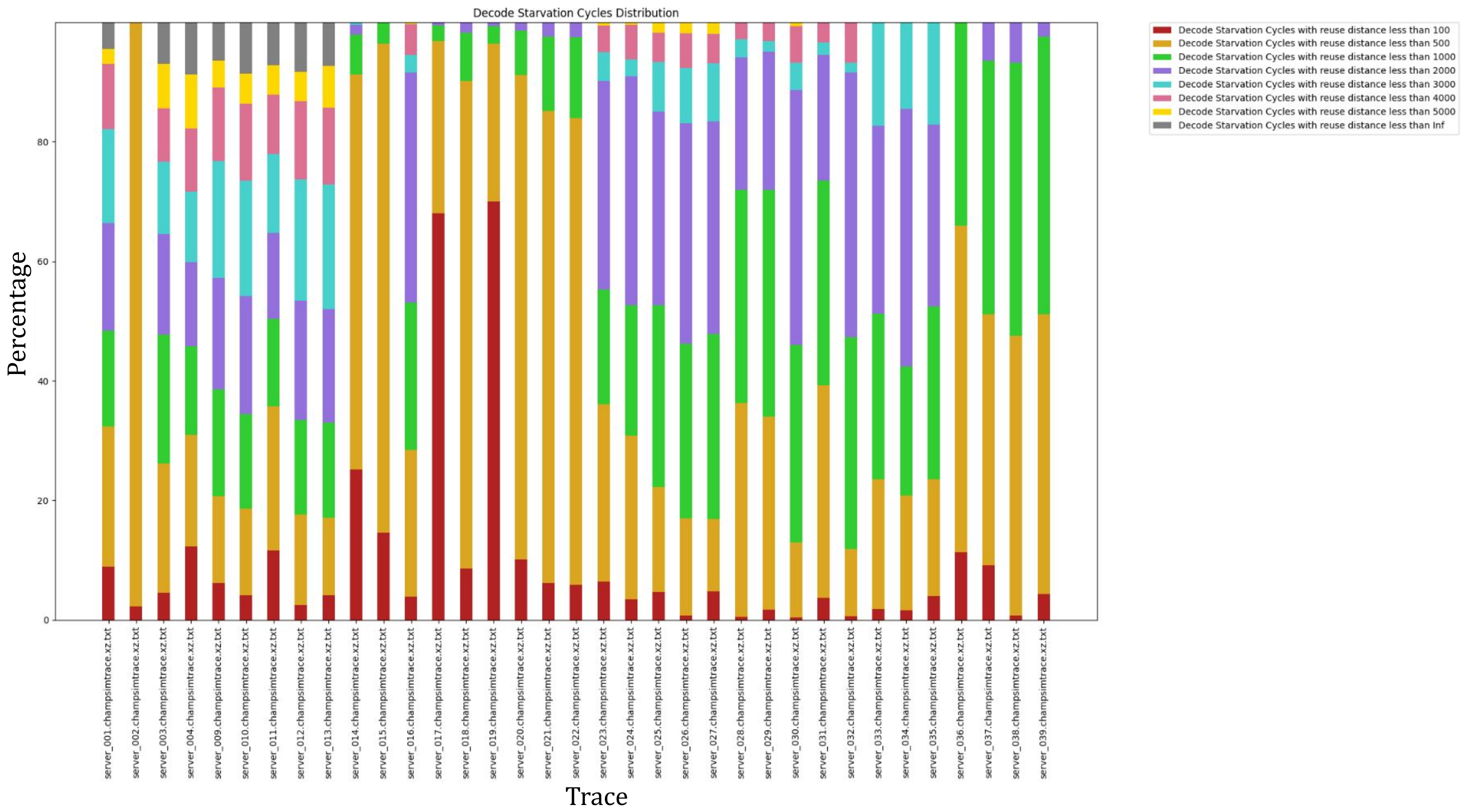


Note: Traces are different

Reuse Line Distribution - Our Observation



Decode Starvation Cycles - Our Observation

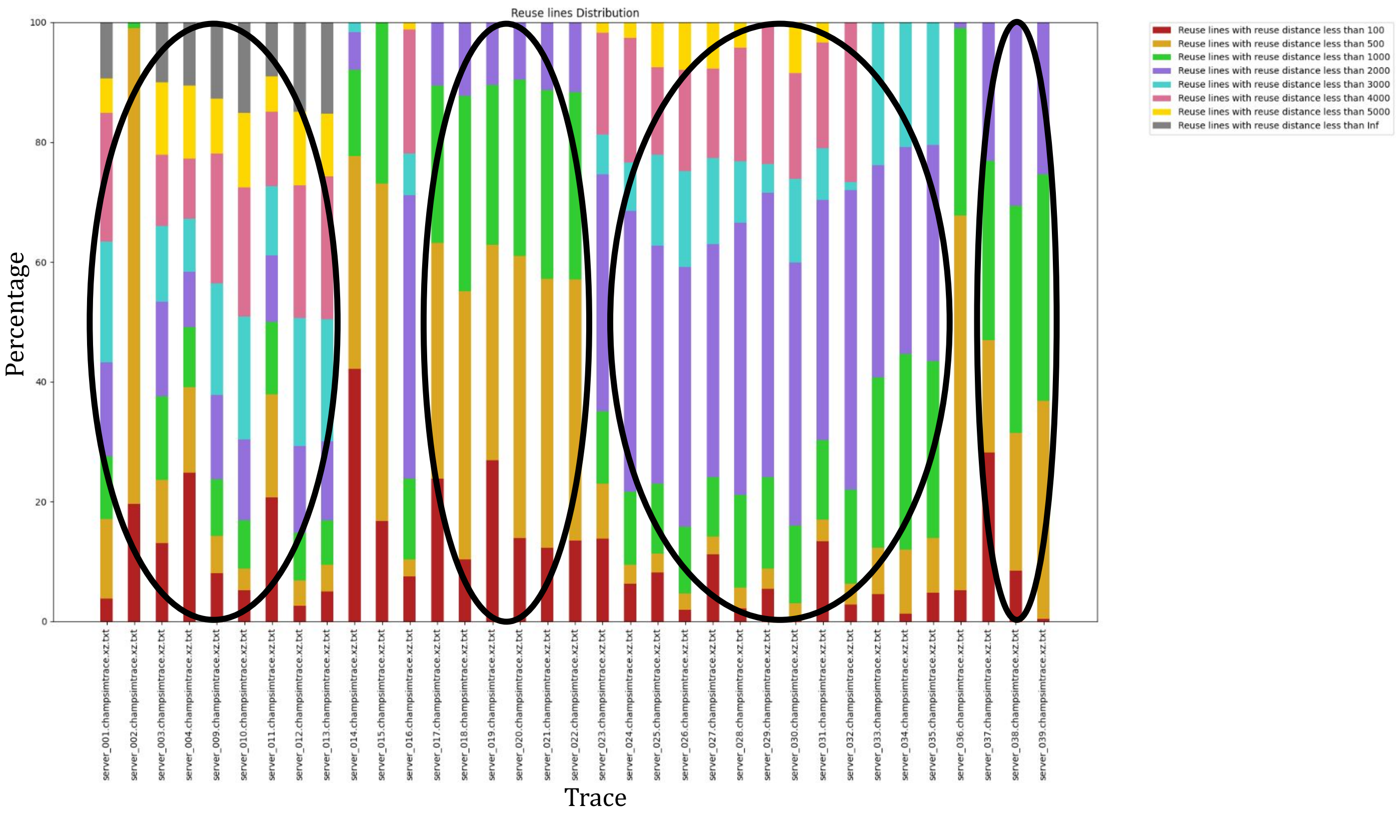


What's different?

- ~~Small % of lines cause majority of decode starvations~~
- Decode starvation cycles are distributed over large proportion of instruction lines
- Mid reuse lines cause the most decode starvation

Need to dynamically detect reuse distance of instruction lines and prioritize them appropriately

Further scope - Cluster analysis?



Further scope - Cluster analysis?