The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi, Onur Mutlu, Yale N. Patt

UT Austin/Google, ETH Zürich, UT Austin

October 19th, 2016

Continuous Runahead Outline

- Overview of Runahead
- Runahead Limitations
- Continuous Runahead Dependence Chains
- Continuous Runahead Engine
- Continuous Runahead Evaluation
- Conclusions

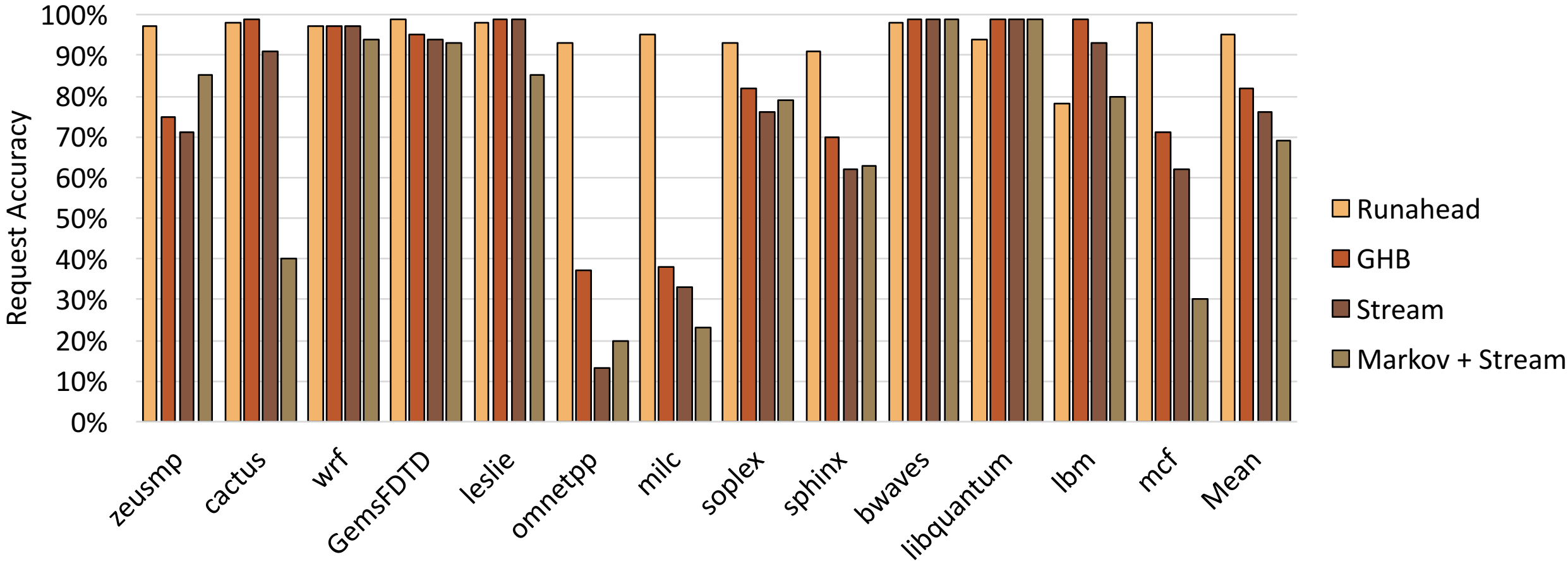
Continuous Runahead Outline

- Overview of Runahead
- Runahead Limitations
- Continuous Runahead Dependence Chains
- Continuous Runahead Engine
- Continuous Runahead Evaluation
- Conclusions

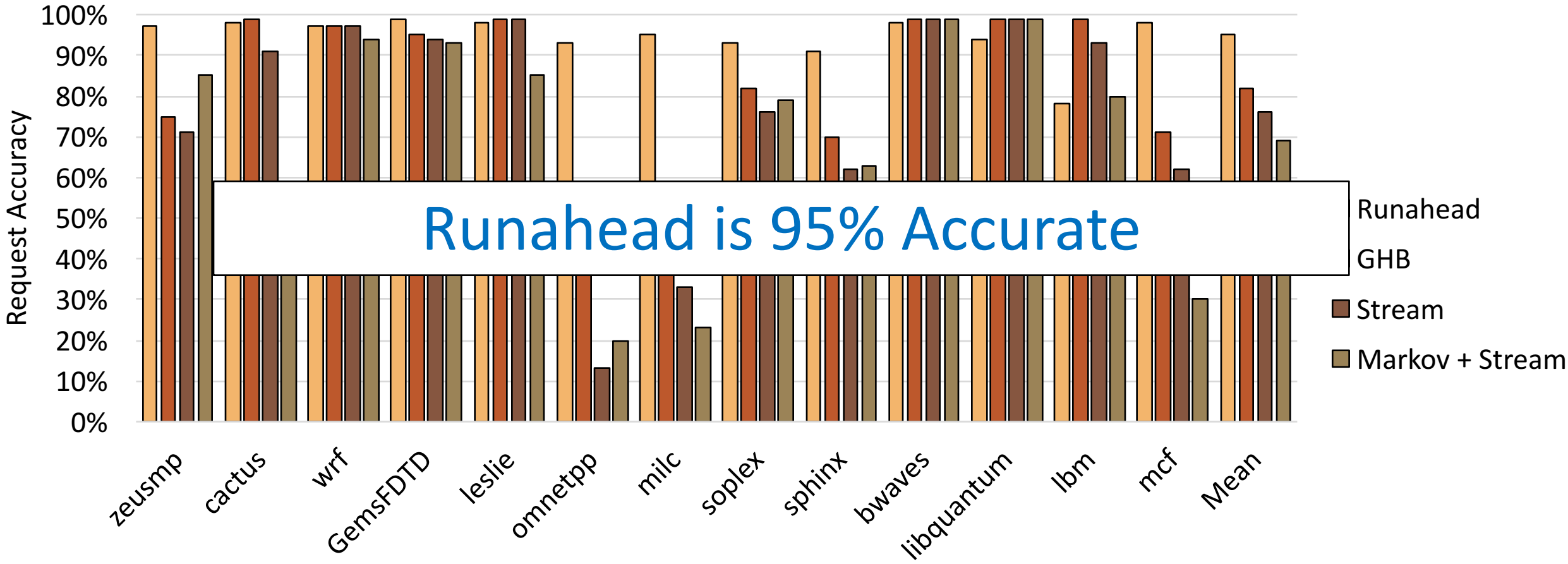
Runahead Execution Overview

- Runahead dynamically expands the instruction window when the pipeline is stalled [Mutlu et al., 2003]
  - The core checkpoints architectural state
  - The result of the memory operation that caused the stall is marked as poisoned in the physical register file
  - The core continues to fetch and execute instructions
  - Operations are discarded instead of retired
  - The goal is to generate new independent cache misses

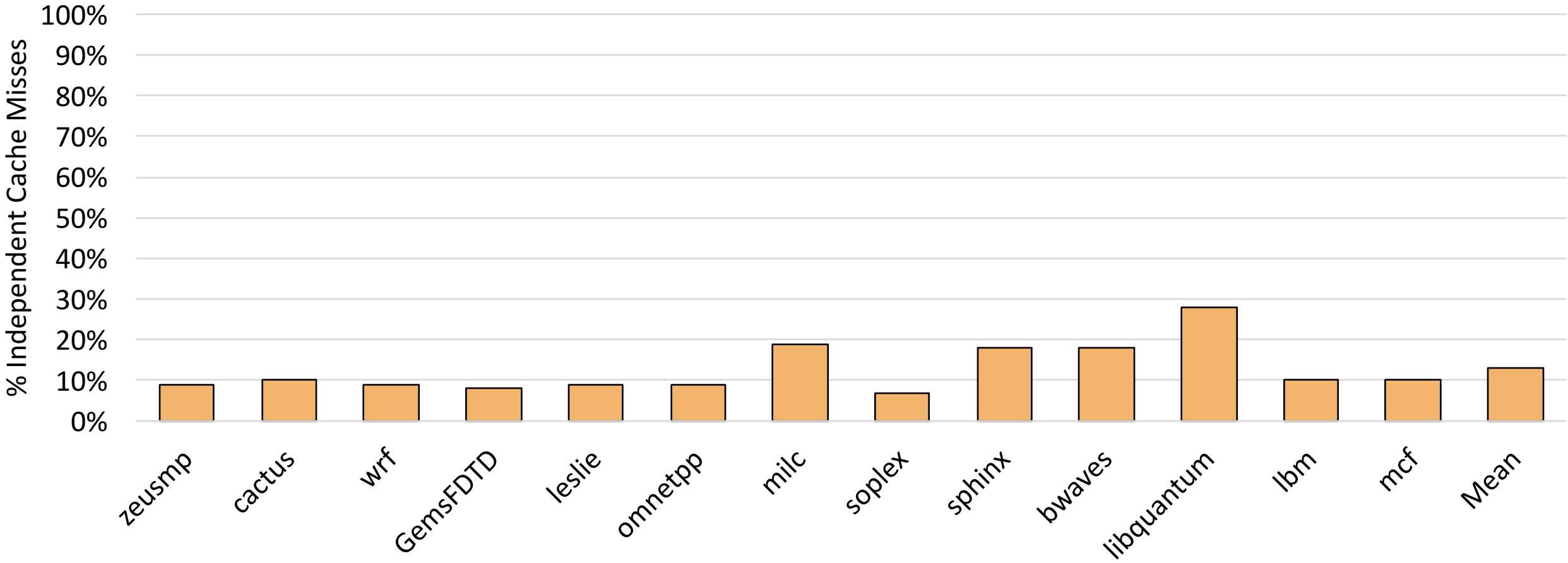
Traditional Runahead Accuracy



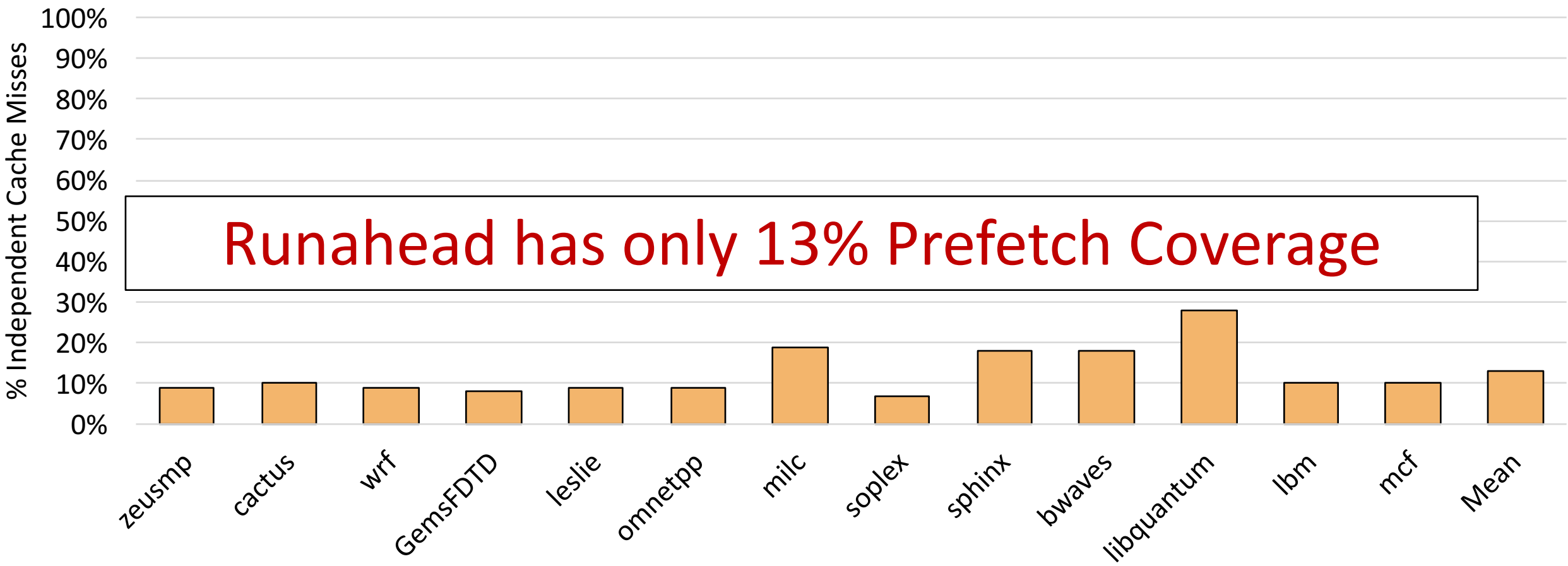
Traditional Runahead Accuracy



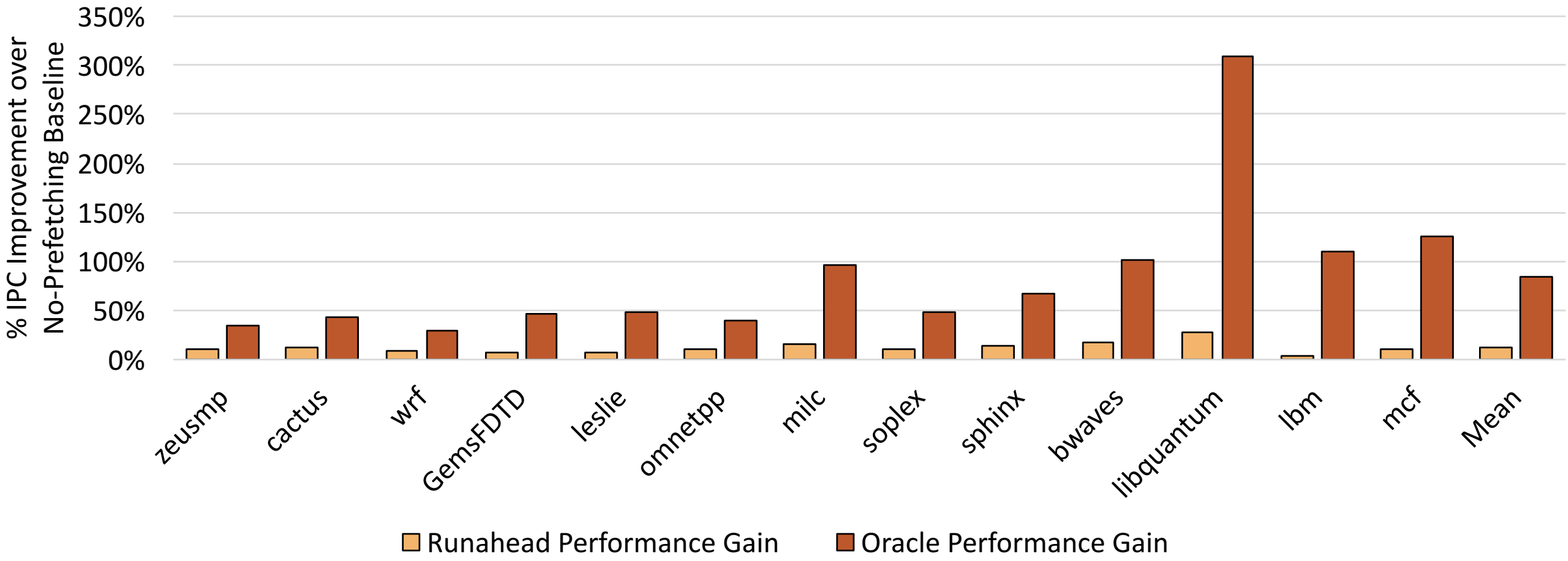
Traditional Runahead Prefetch Coverage



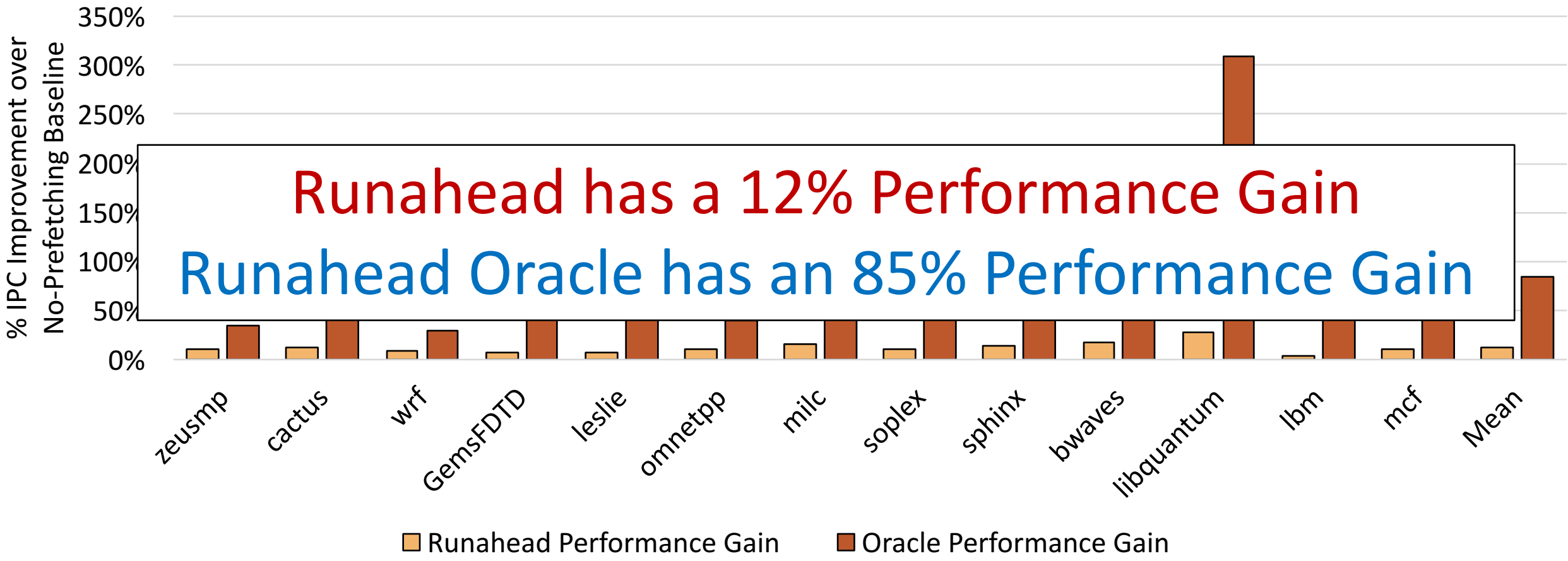
Traditional Runahead Prefetch Coverage



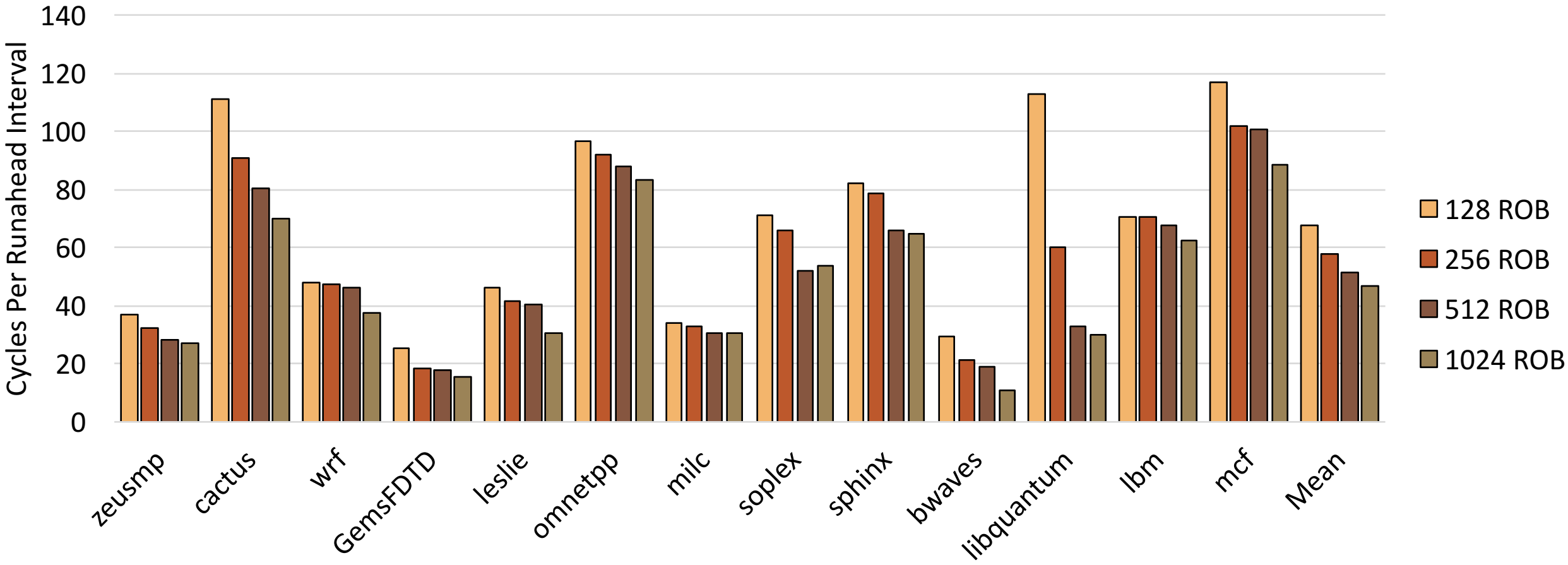
Traditional Runahead Performance Gain



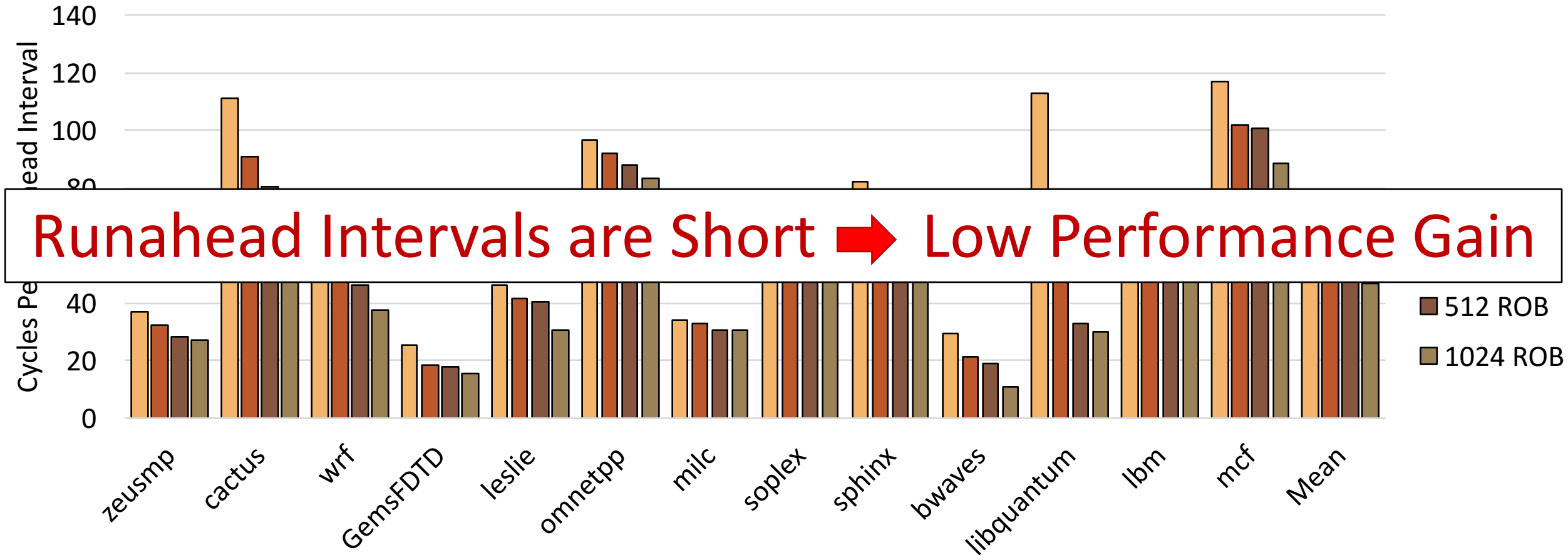
Traditional Runahead Performance Gain



Traditional Runahead Interval Length



Traditional Runahead Interval Length

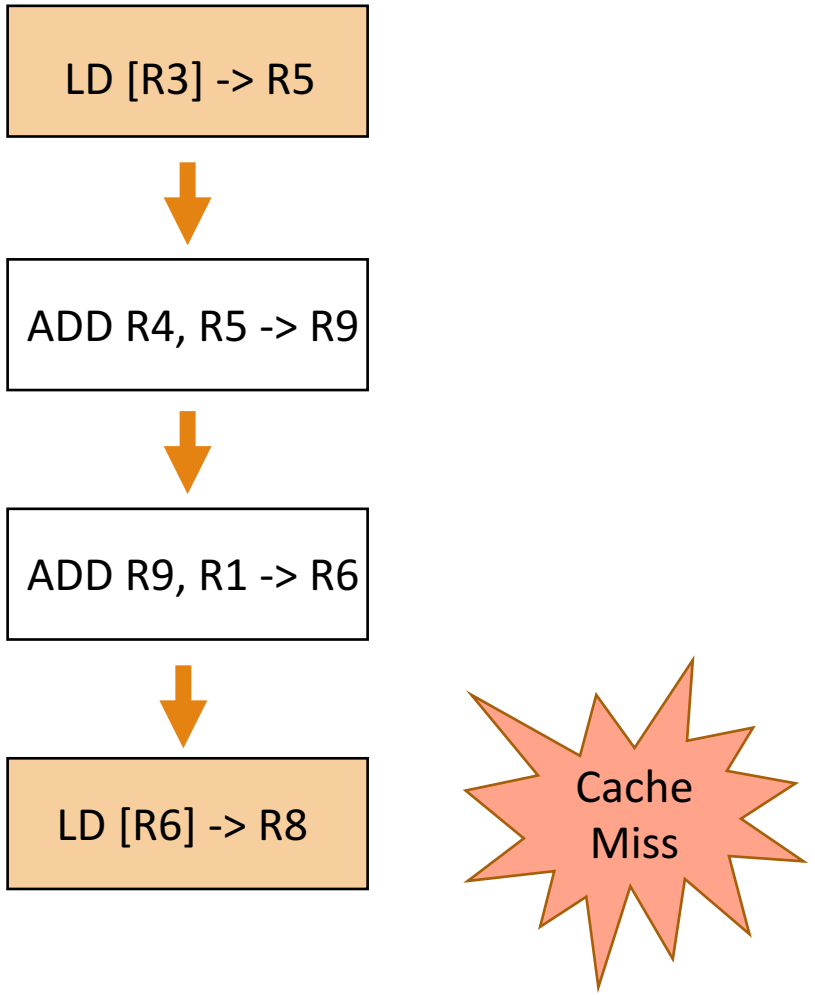


Runahead Intervals are Short ➡ Low Performance Gain

Continuous Runahead Challenges

- Which instructions to use during Continuous Runahead?
  - Dynamically target the dependence chains that lead to critical cache misses
- What hardware to use for Continuous Runahead?
- How long should chains pre-execute for?

Dependence Chains

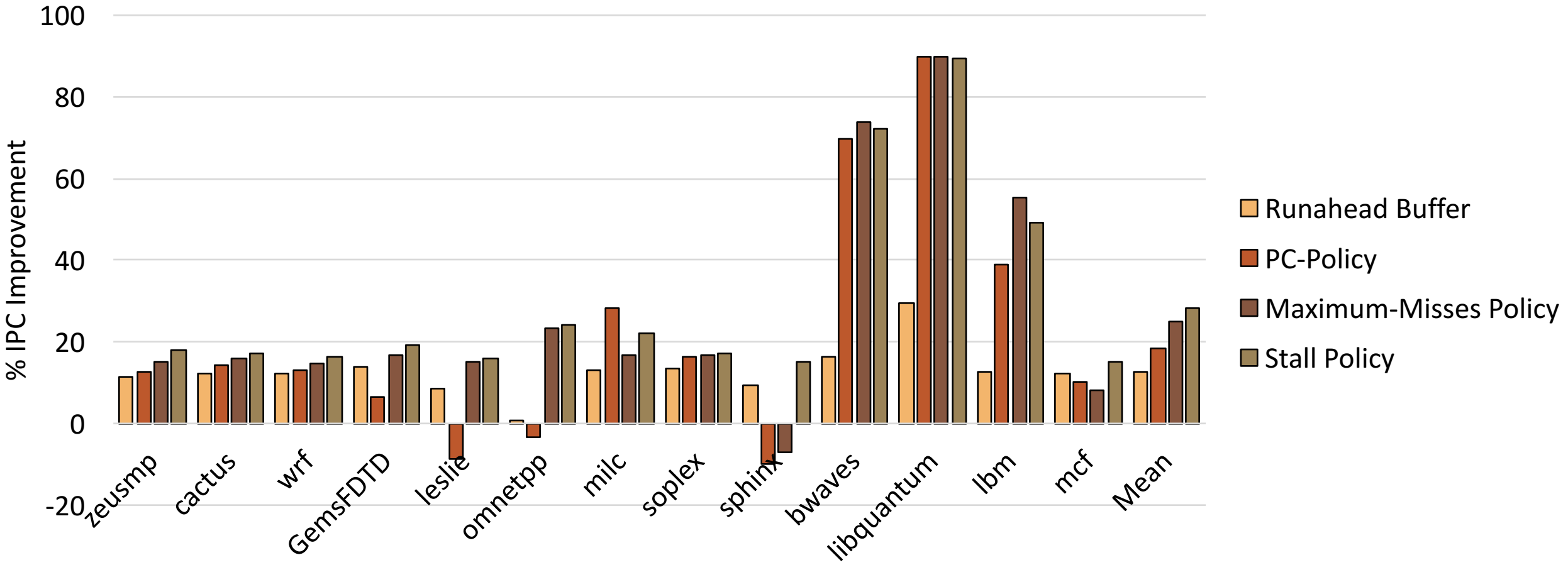


Dependence Chain Selection Policies

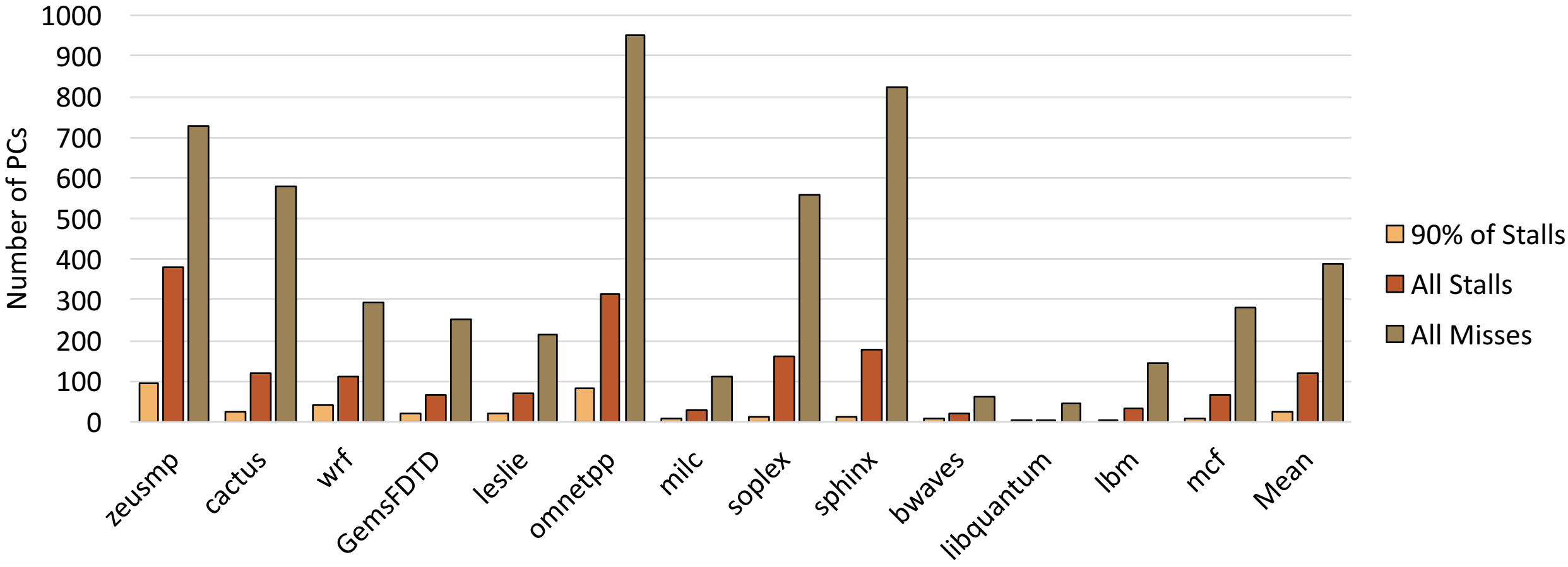
Experiment with 3 policies to determine the best policy to use for Continuous Runahead:

- PC-Based Policy
  - Use the dependence chain that has caused the most misses for the PC that is blocking retirement
- Maximum Misses Policy
  - Use a dependence chain from the PC that has generated the most misses for the application
- Stall Policy
  - Use a dependence chain from the PC that has caused the most full-window stalls for the application

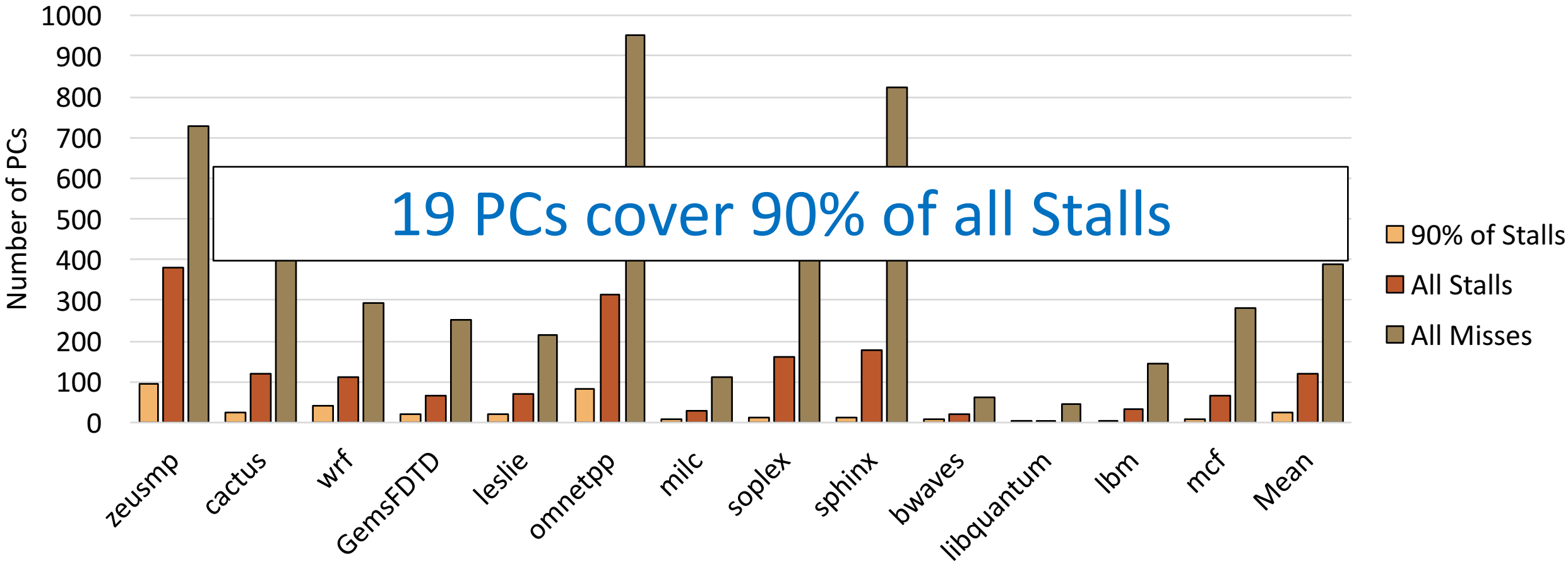
Dependence Chain Selection Policies



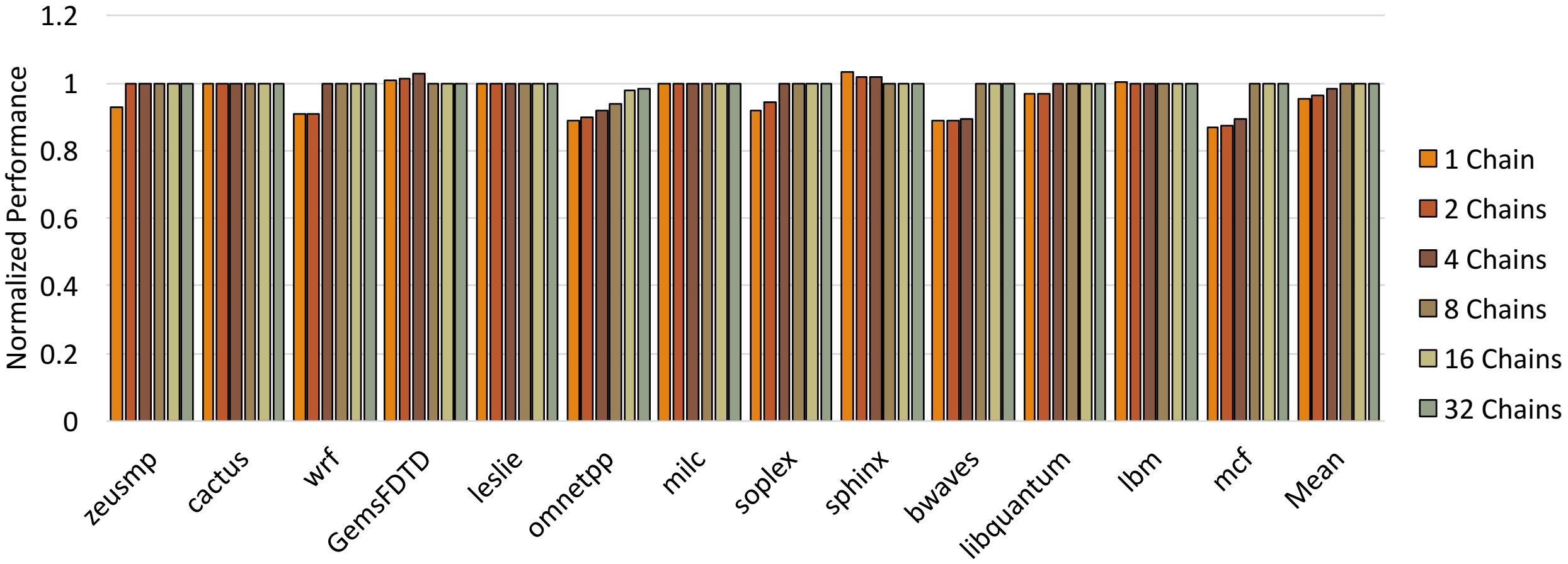
Why does Stall Policy Work?



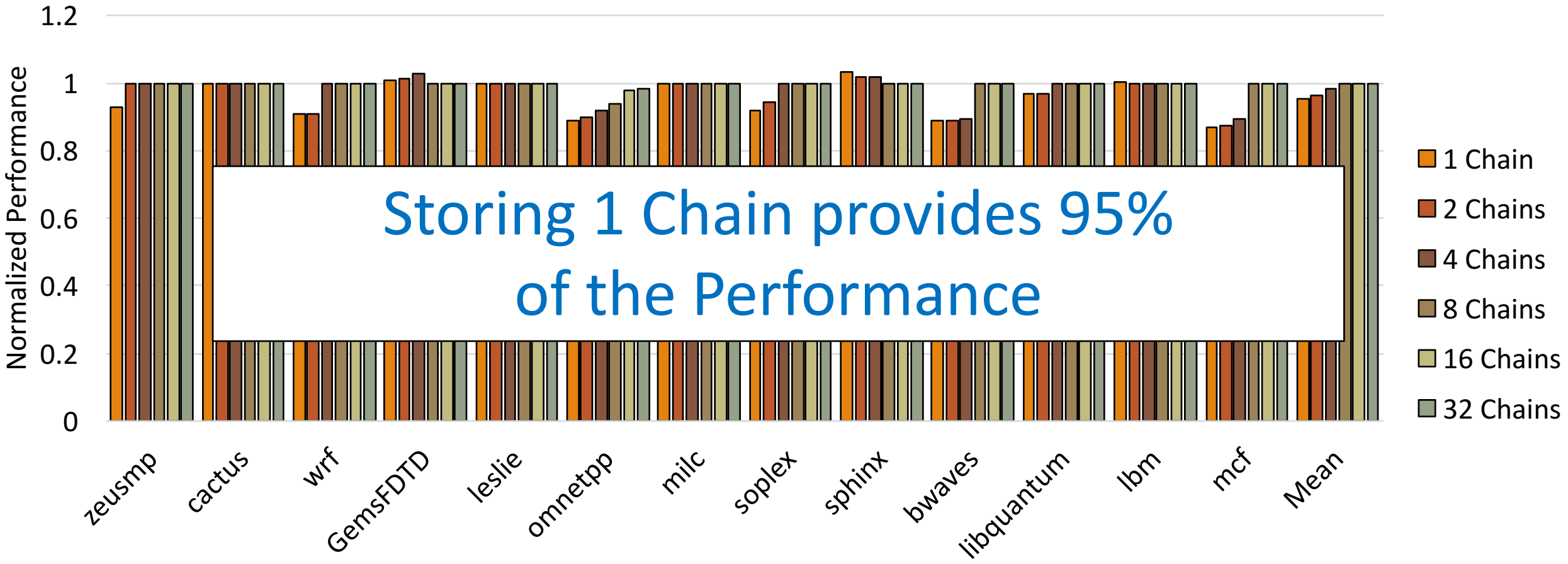
Why does Stall Policy Work?



Constrained Dependence Chain Storage



Constrained Dependence Chain Storage



Continuous Runahead Chain Generation

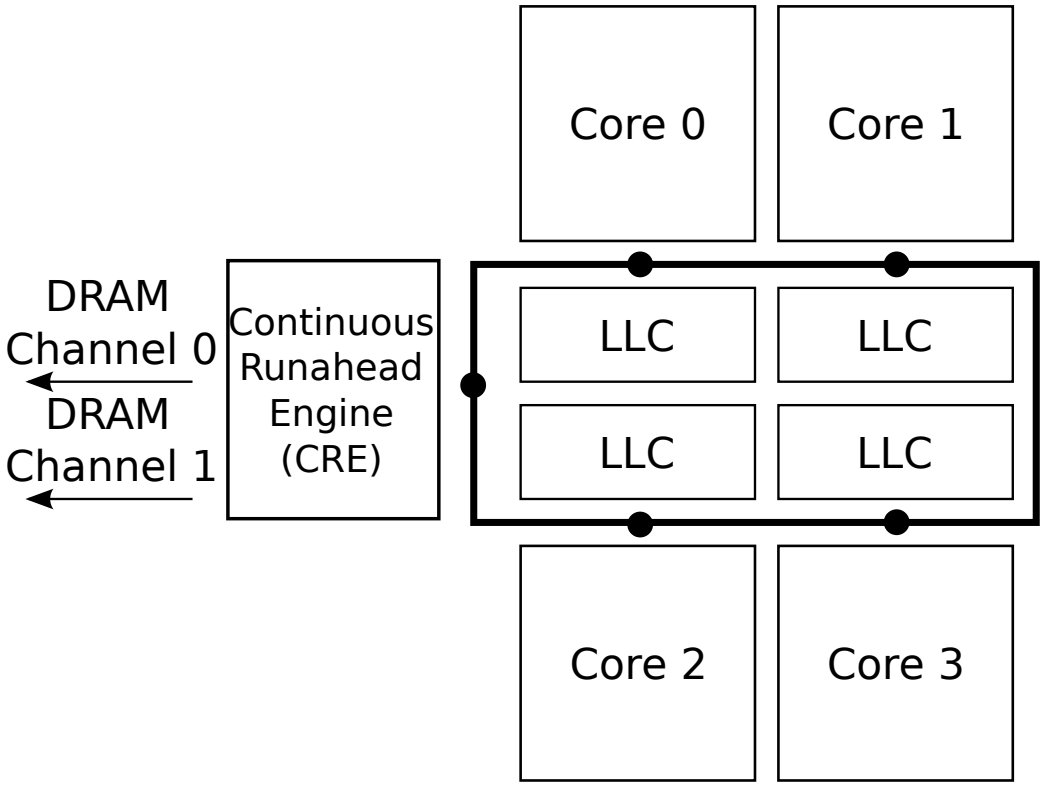
Maintain two structures:

- 32-entry cache of PCs to track the operations that cause the pipeline to frequently stall
- The last dependence chain for the PC that has caused the most full-window stalls

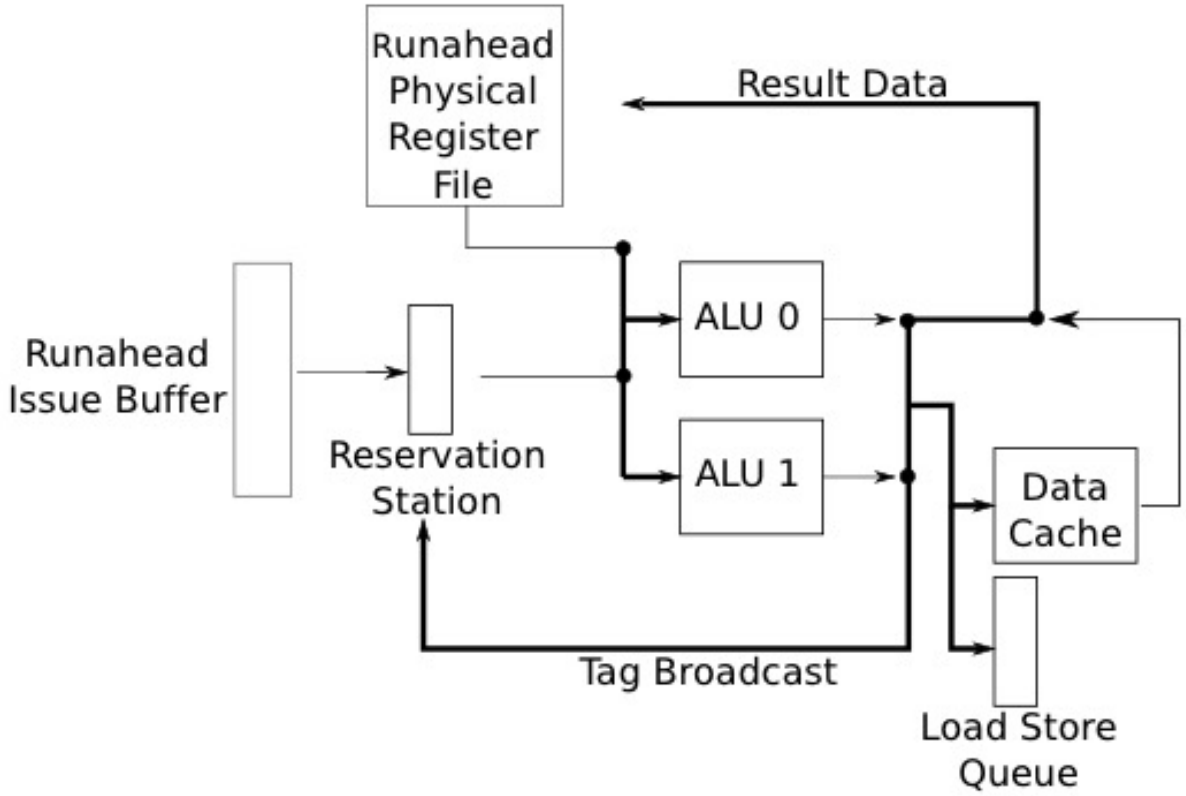
At every full window stall:

- Increment the counter of the PC that caused the stall
- Generate a dependence chain for the PC that has caused the most stalls

Runahead for Longer Intervals



CRE Microarchitecture



- No Front-End
- No Register Renaming Hardware
- 32 Physical Registers
- 2-Wide
- No Floating Point or Vector Pipeline
- 4kB Data Cache

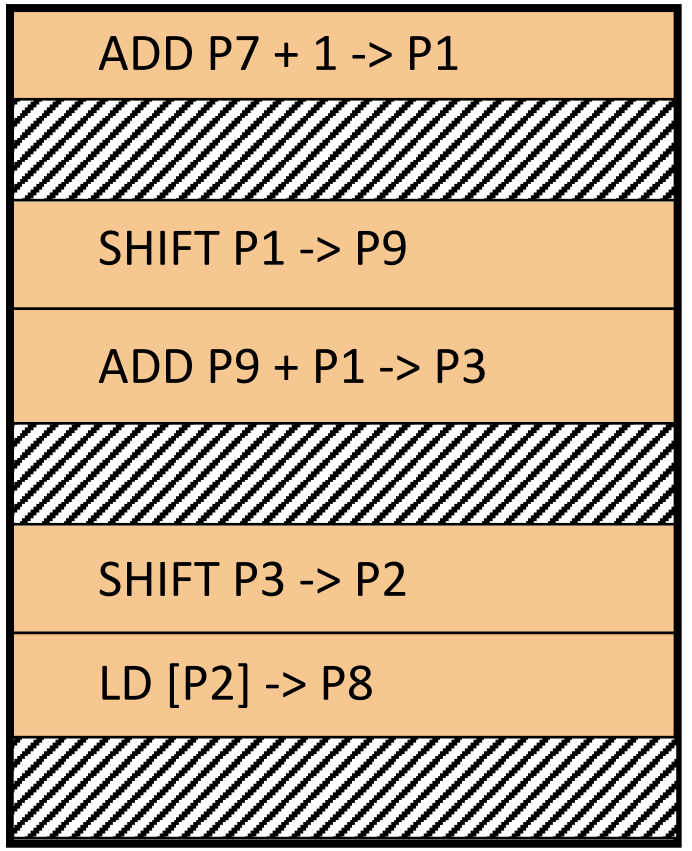
Dependence Chain Generation

Cycle: 4

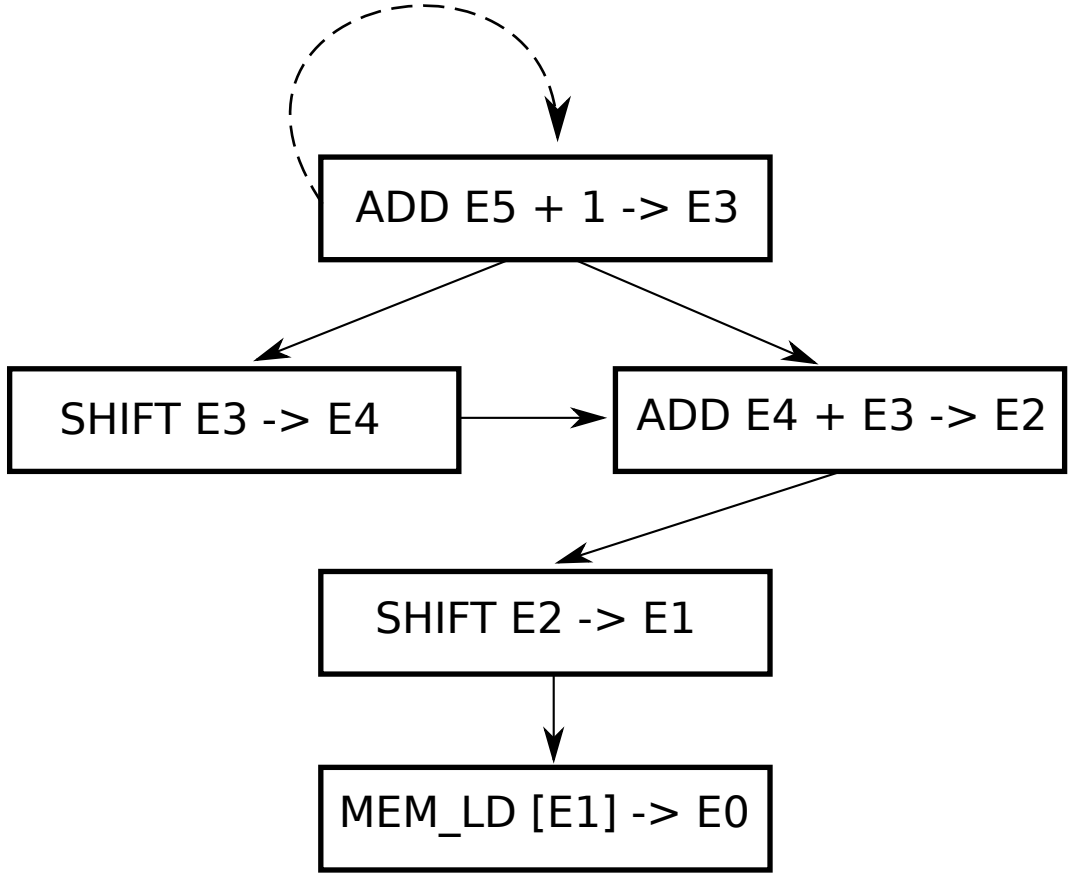
Search List: P9, P1

Register Remapping Table:

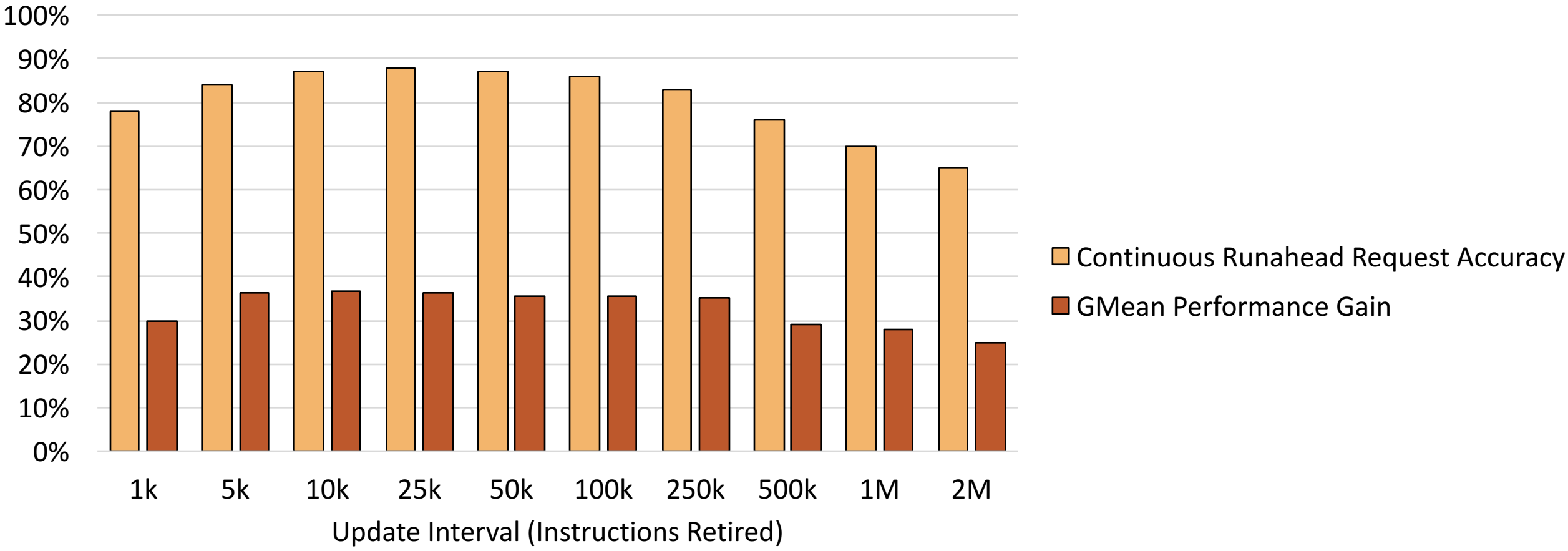
	EAX	EBX	ECX
Core Physical Register	P7	P8	P9
CRE Physical Register	E3	E0	E4
First CRE Physical Register	E3	E0	E1



Dependence Chain Generation



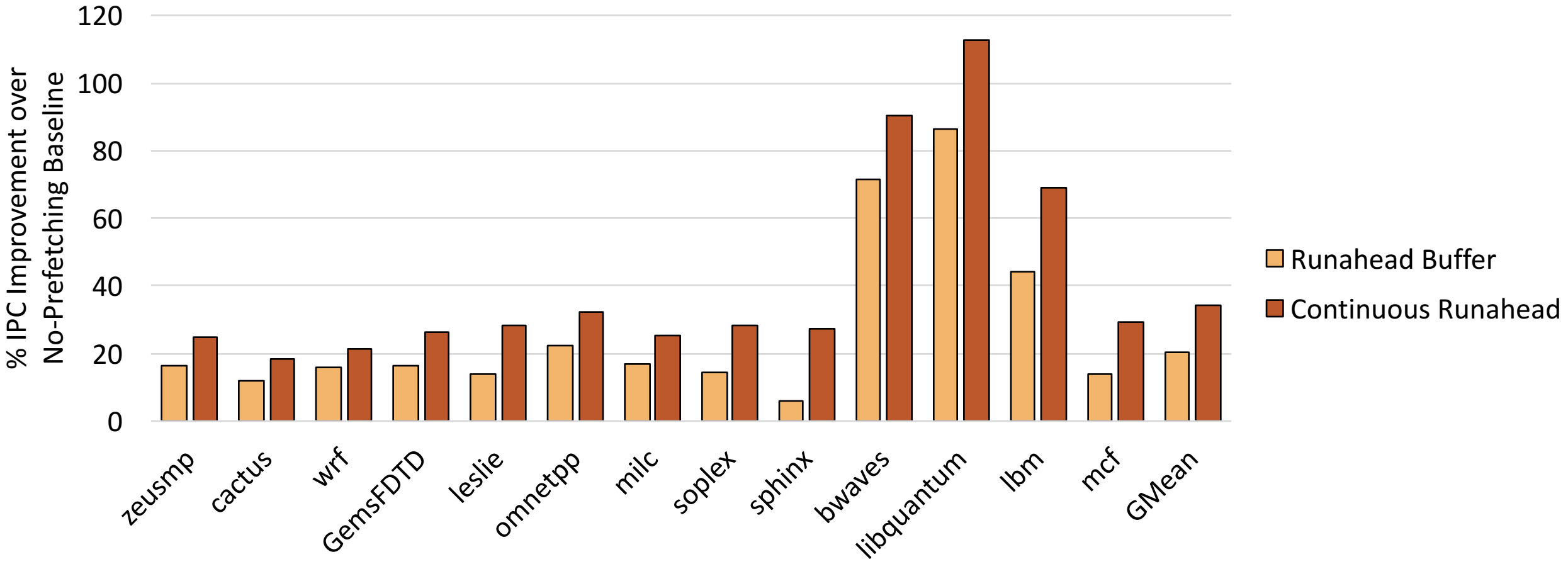
Interval Length



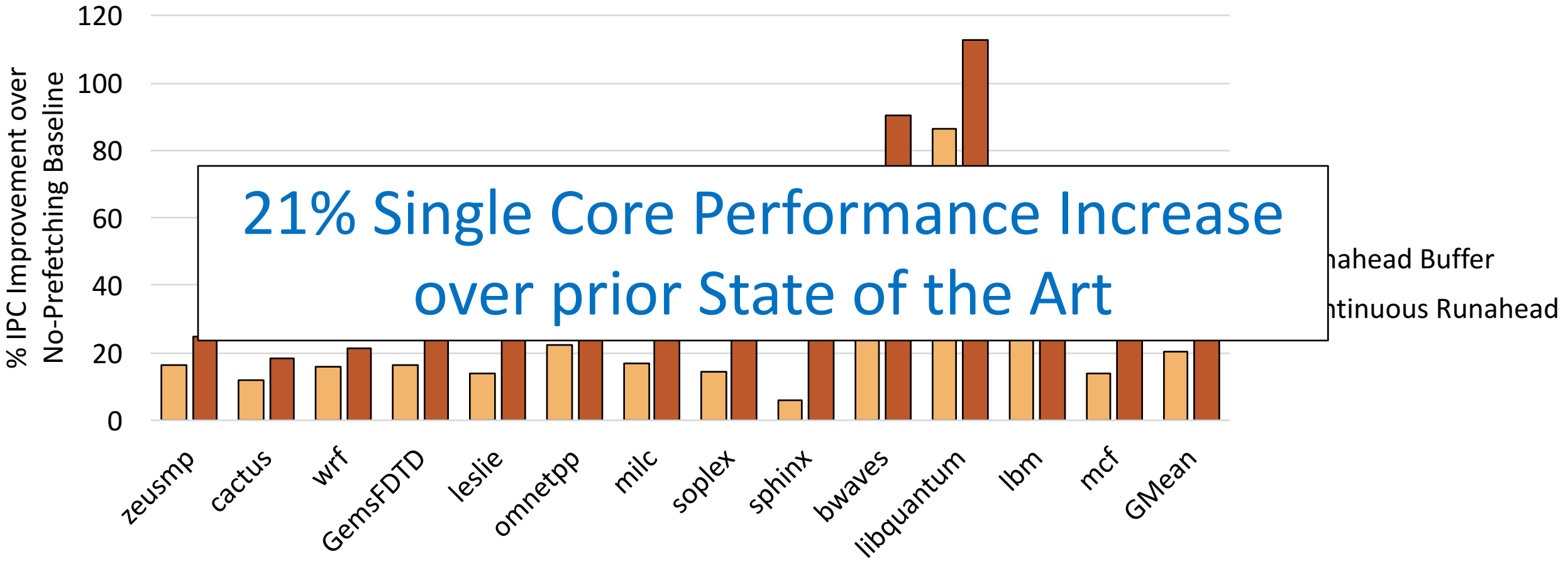
System Configuration

- Single-Core/Quad-Core
  - 4-wide Issue
  - 256 Entry Reorder Buffer
  - 92 Entry Reservation Station
- Caches
  - 32 KB 8-Way Set Associative L1 I/D-Cache
  - 1MB 8-Way Set Associative Shared Last Level Cache per Core
- Non-Uniform Memory Access Latency DDR3 System
  - 256-Entry Memory Queue
  - Batch Scheduling
- Prefetchers
  - Stream, Global History Buffer
  - Feedback Directed Prefetching: Dynamic Degree 1-32
- CRE Compute
  - 2-wide issue
  - 1 Continuous Runahead issue context with a 32-entry buffer and 32-entry physical register file
  - 4 kB Data Cache

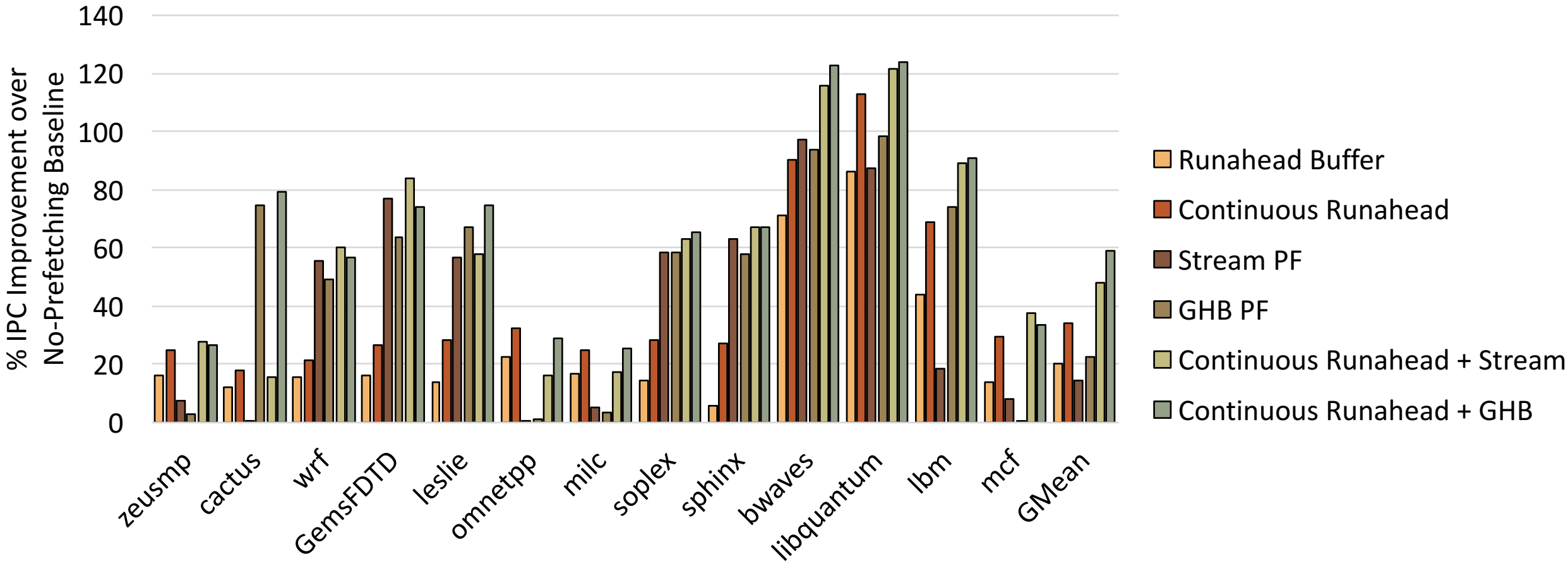
Single-Core Performance



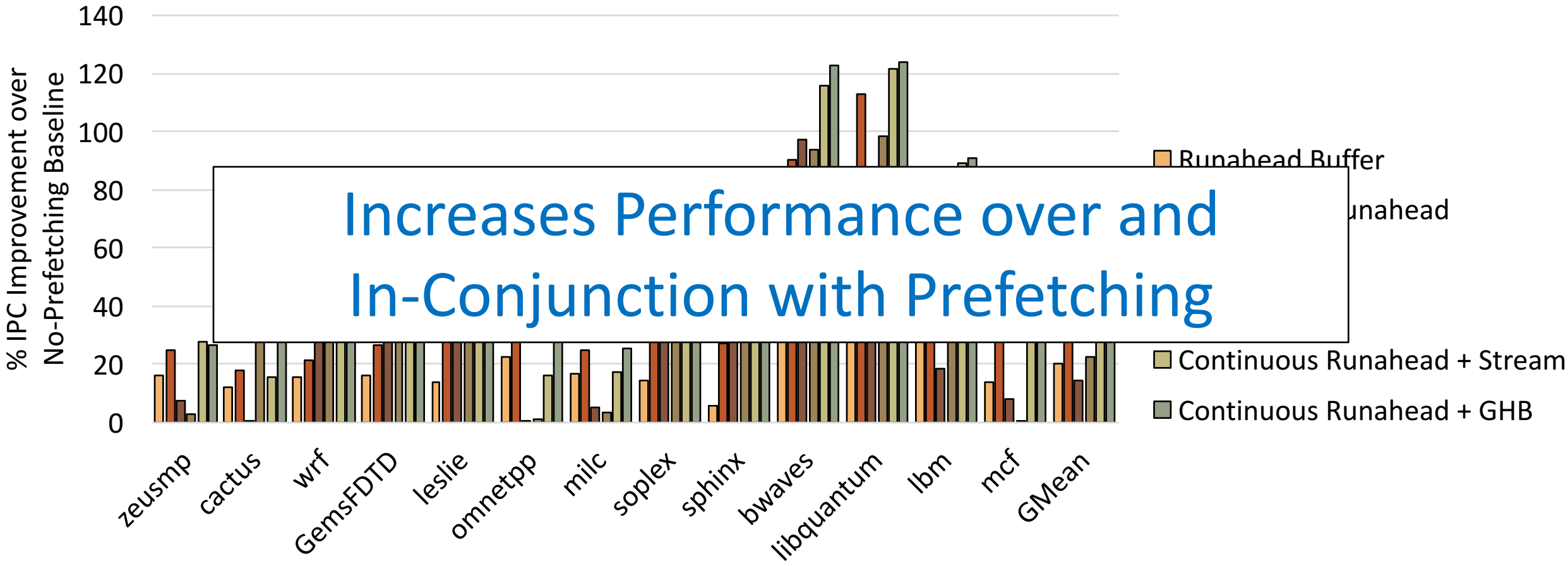
Single-Core Performance



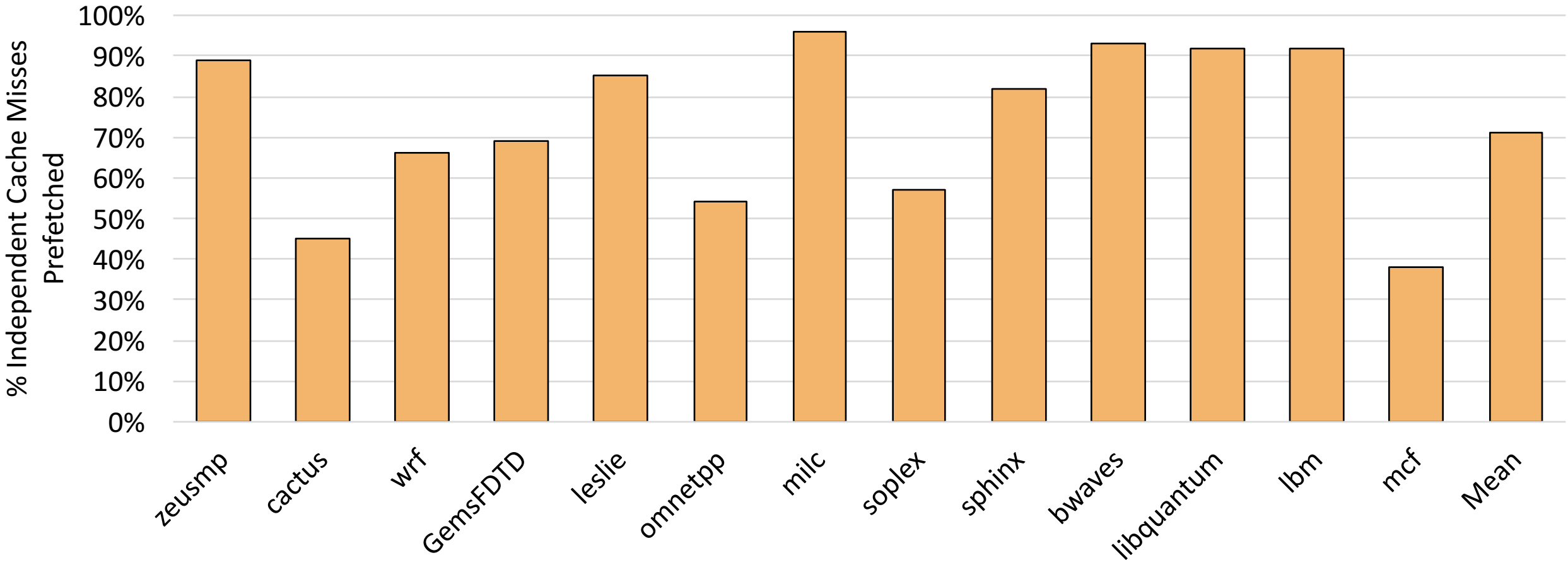
Single-Core Performance + Prefetching



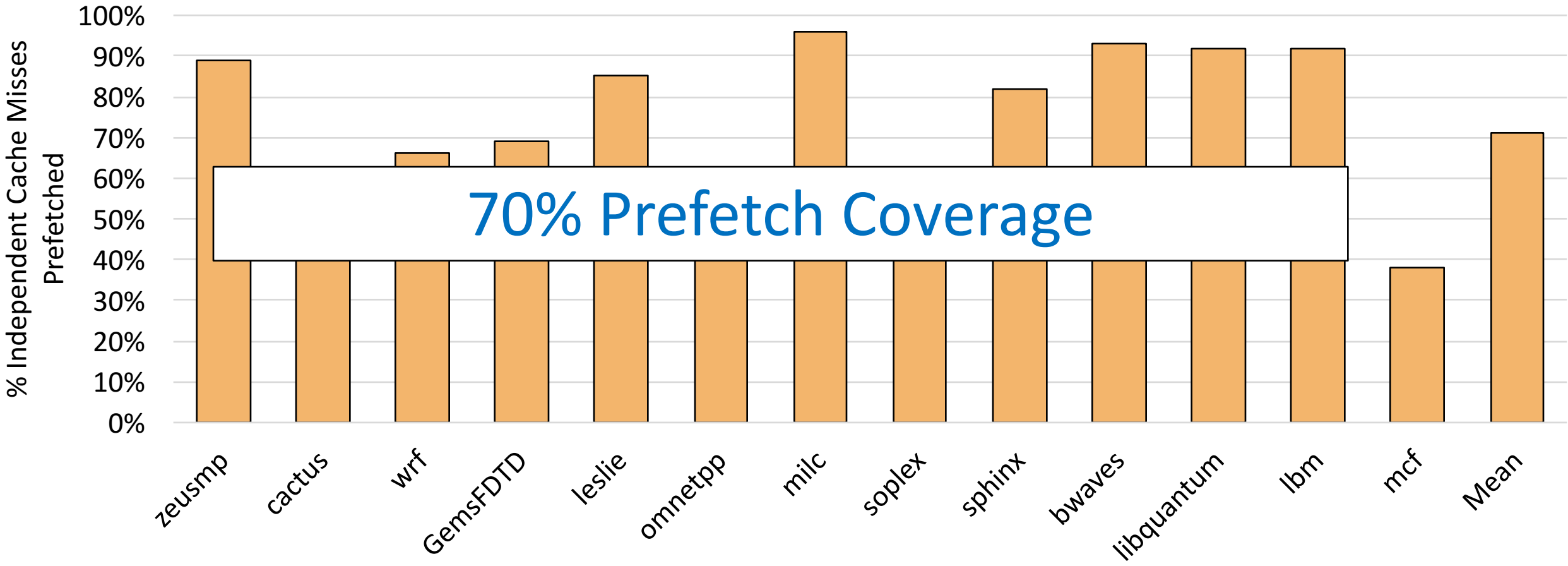
Single-Core Performance + Prefetching



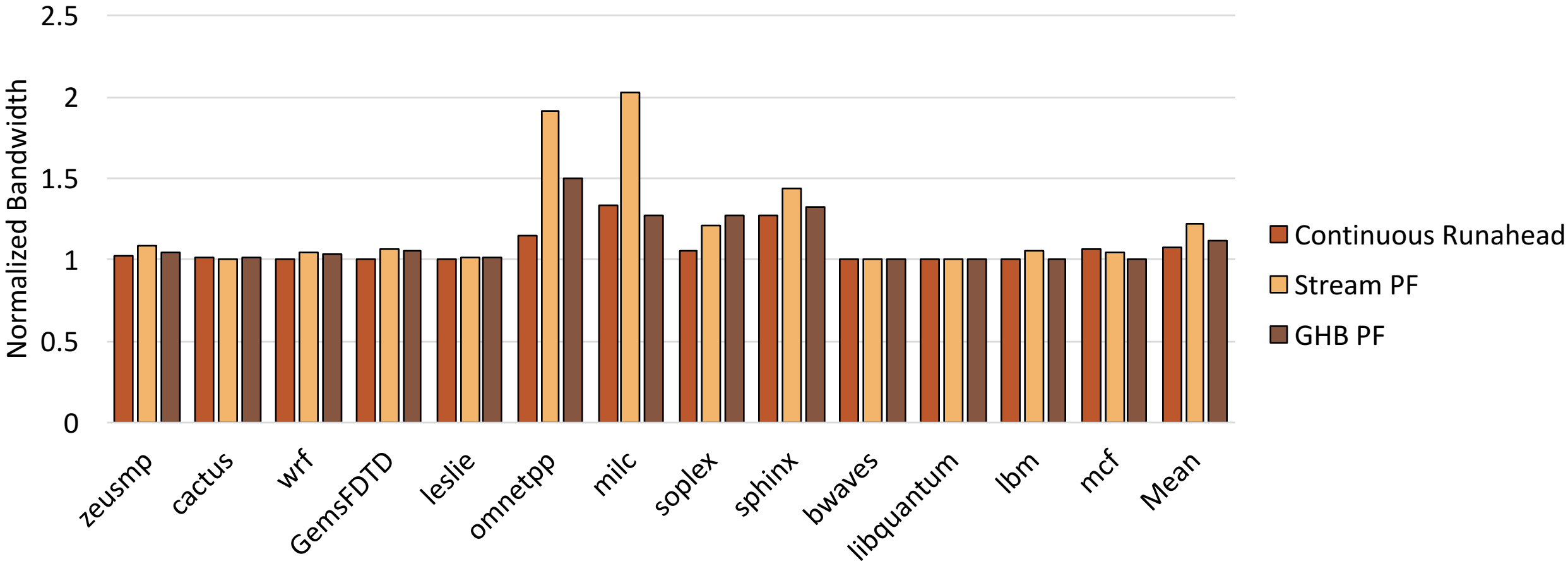
Independent Miss Coverage



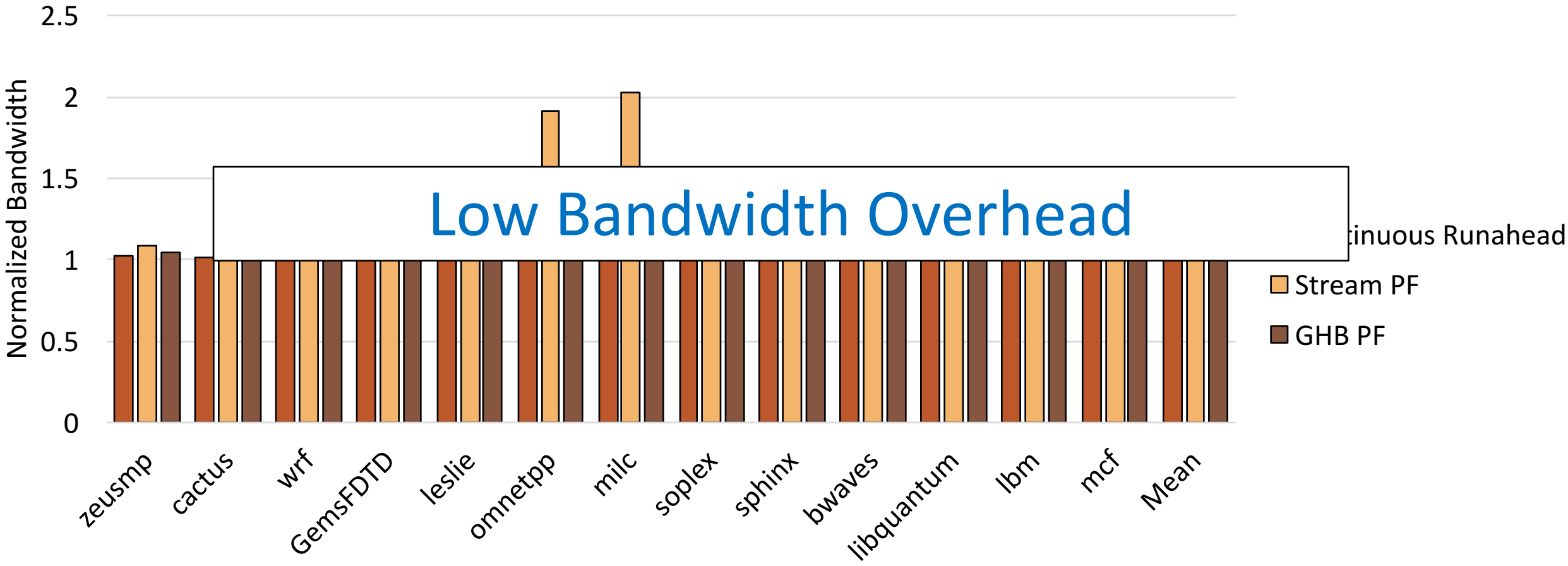
Independent Miss Coverage



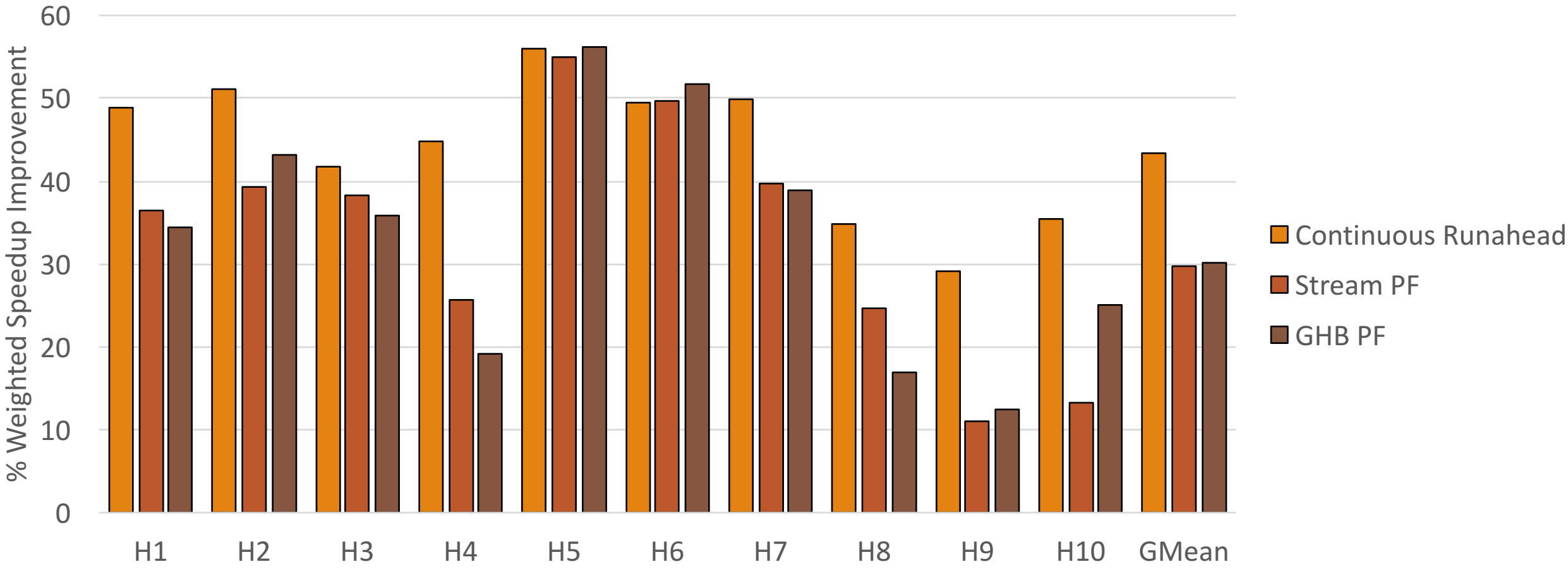
Bandwidth Overhead



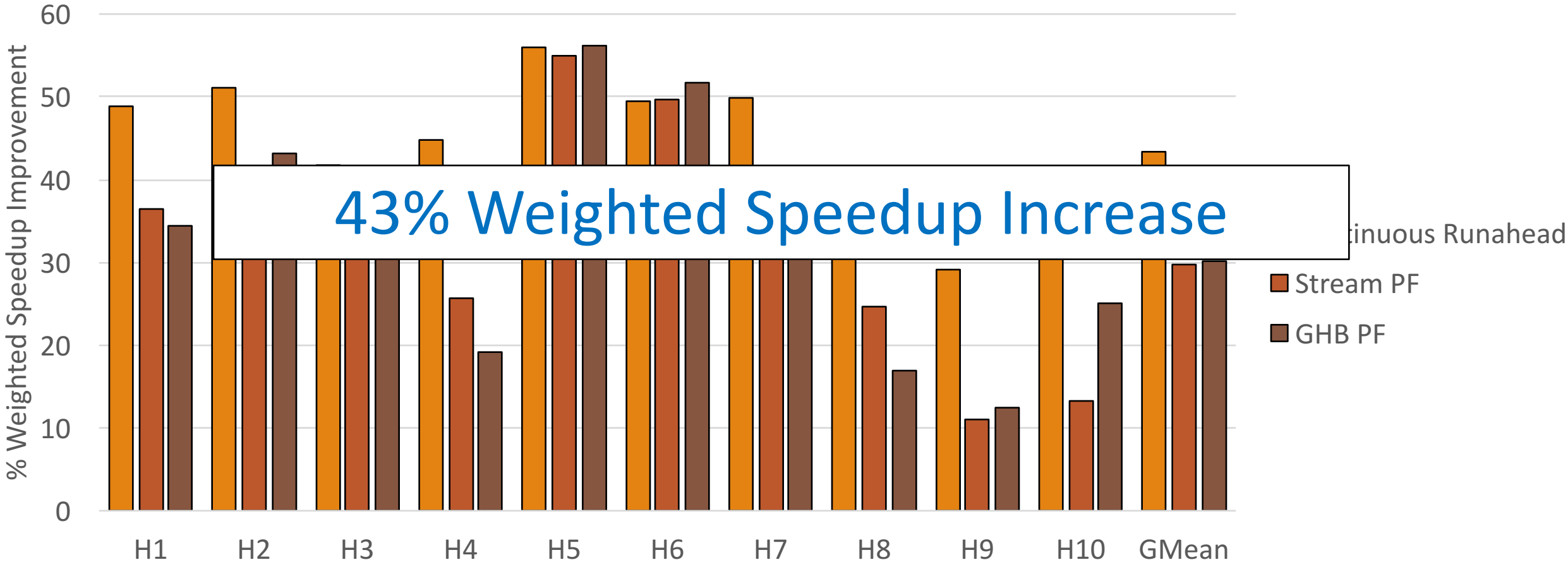
Bandwidth Overhead



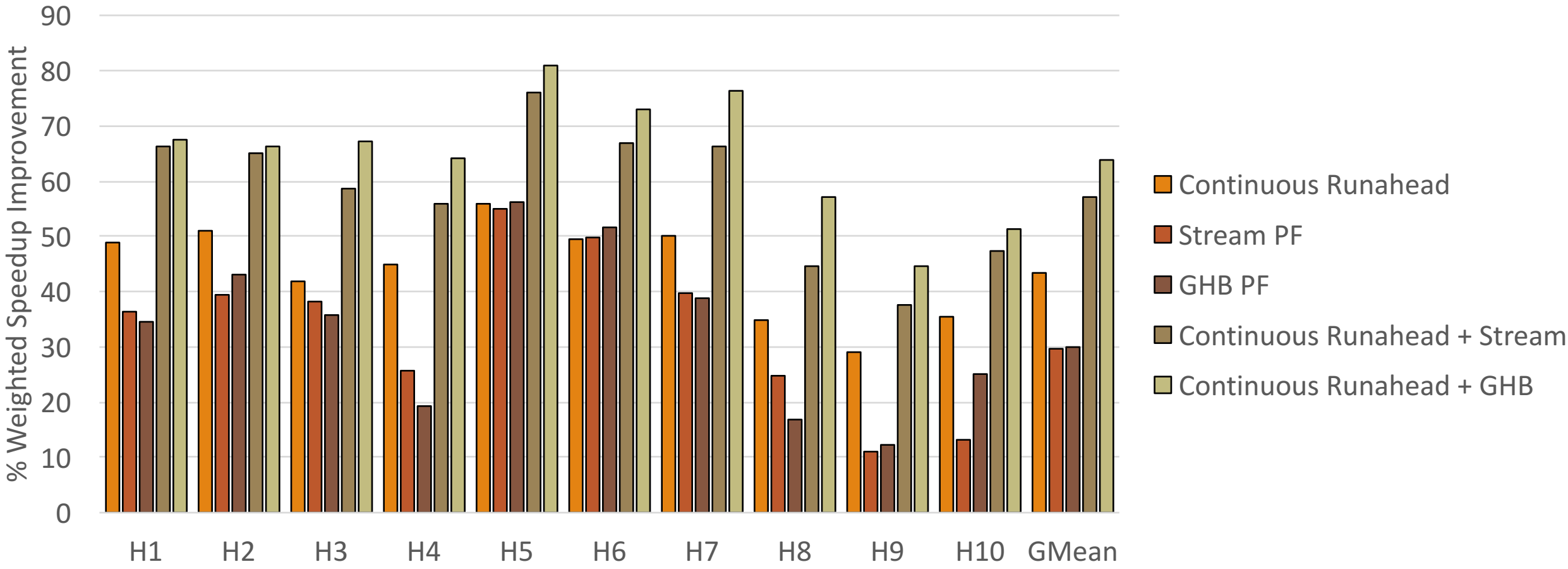
Multi-Core Performance



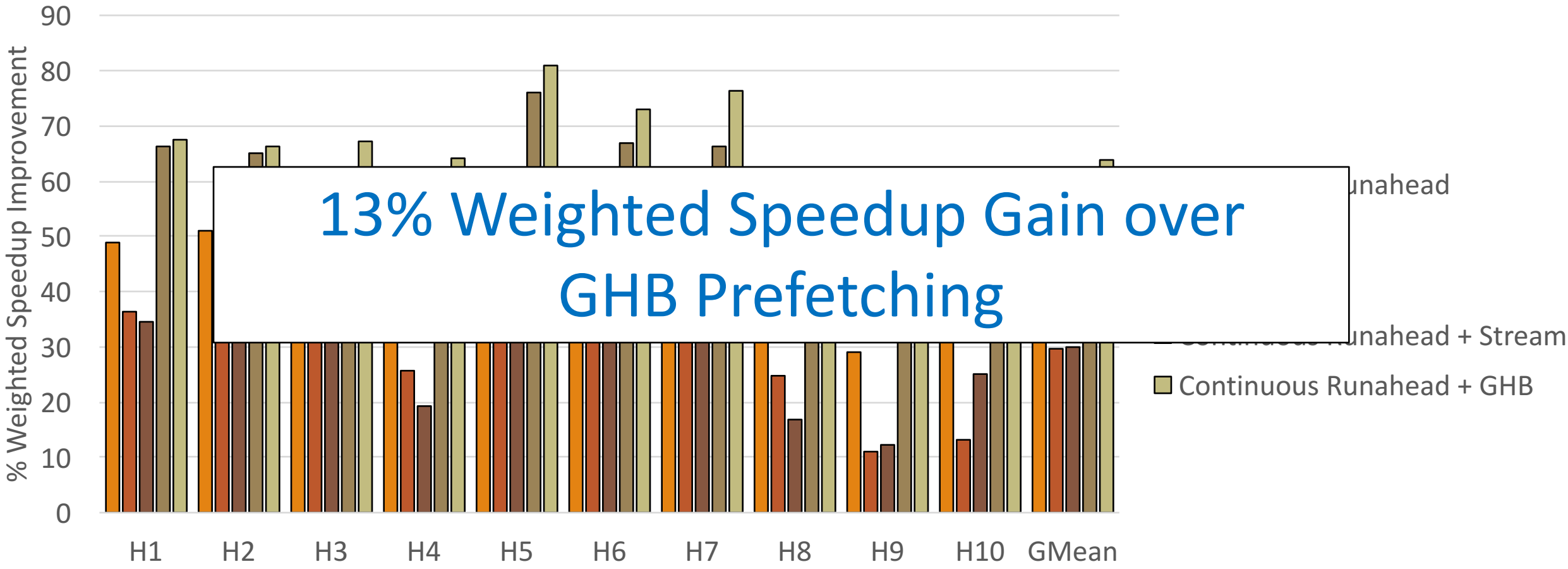
Multi-Core Performance



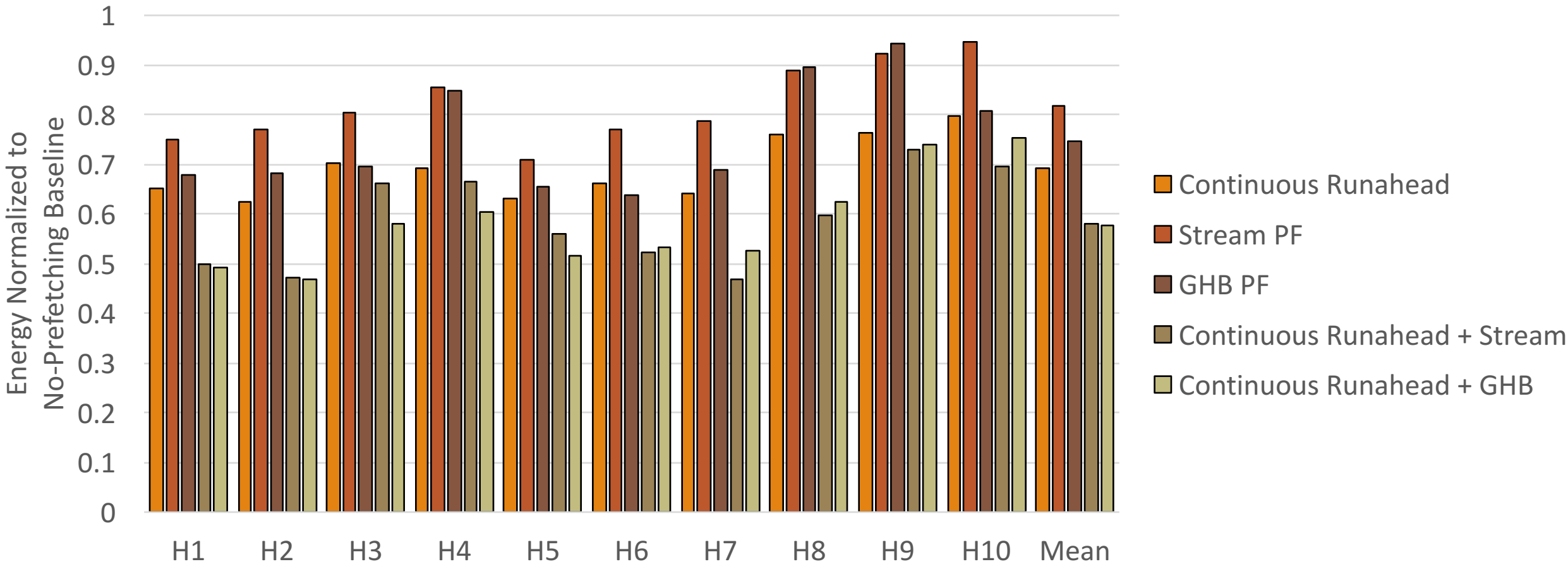
Multi-Core Performance + Prefetching



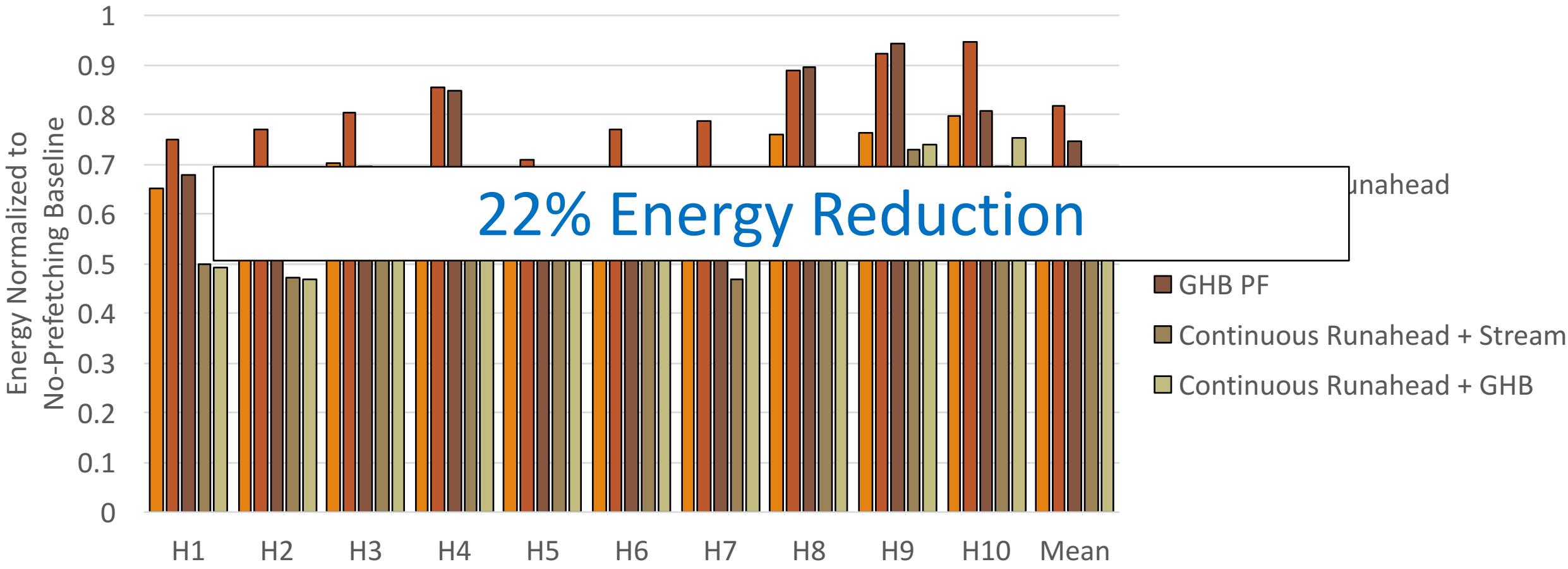
Multi-Core Performance + Prefetching



Multi-Core Energy Evaluation



Multi-Core Energy Evaluation



Conclusions

- Runahead prefetch coverage is limited by the duration of each runahead interval
- To remove this constraint, we introduce the notion of Continuous Runahead
- We can dynamically identify the most critical LLC misses to target with Continuous Runahead by tracking the operations that cause the pipeline to frequently stall
- We migrate these dependence chains to the CRE where they are executed continuously in a loop

Conclusions

- Continuous Runahead greatly increases prefetch coverage
- Increases single-core performance by 34.4%
- Increases multi-core performance by 43.3%
- Synergistic with various types of prefetching

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi, Onur Mutlu, Yale N. Patt
UT Austin/Google, ETH Zürich, UT Austin

October 19th, 2016