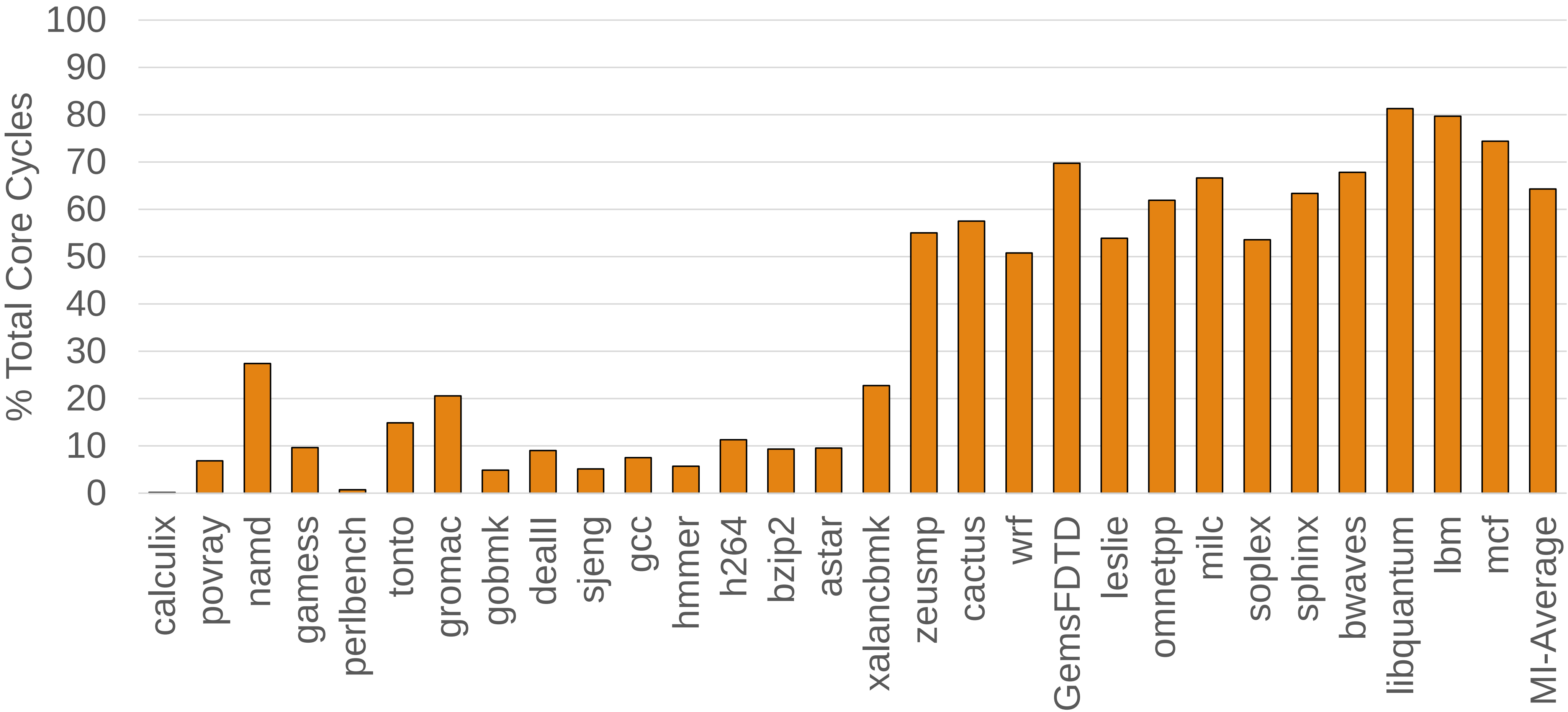
Filtered Runahead Execution with a Runahead Buffer

Milad Hashemi
Yale N. Patt
December 8, 2015

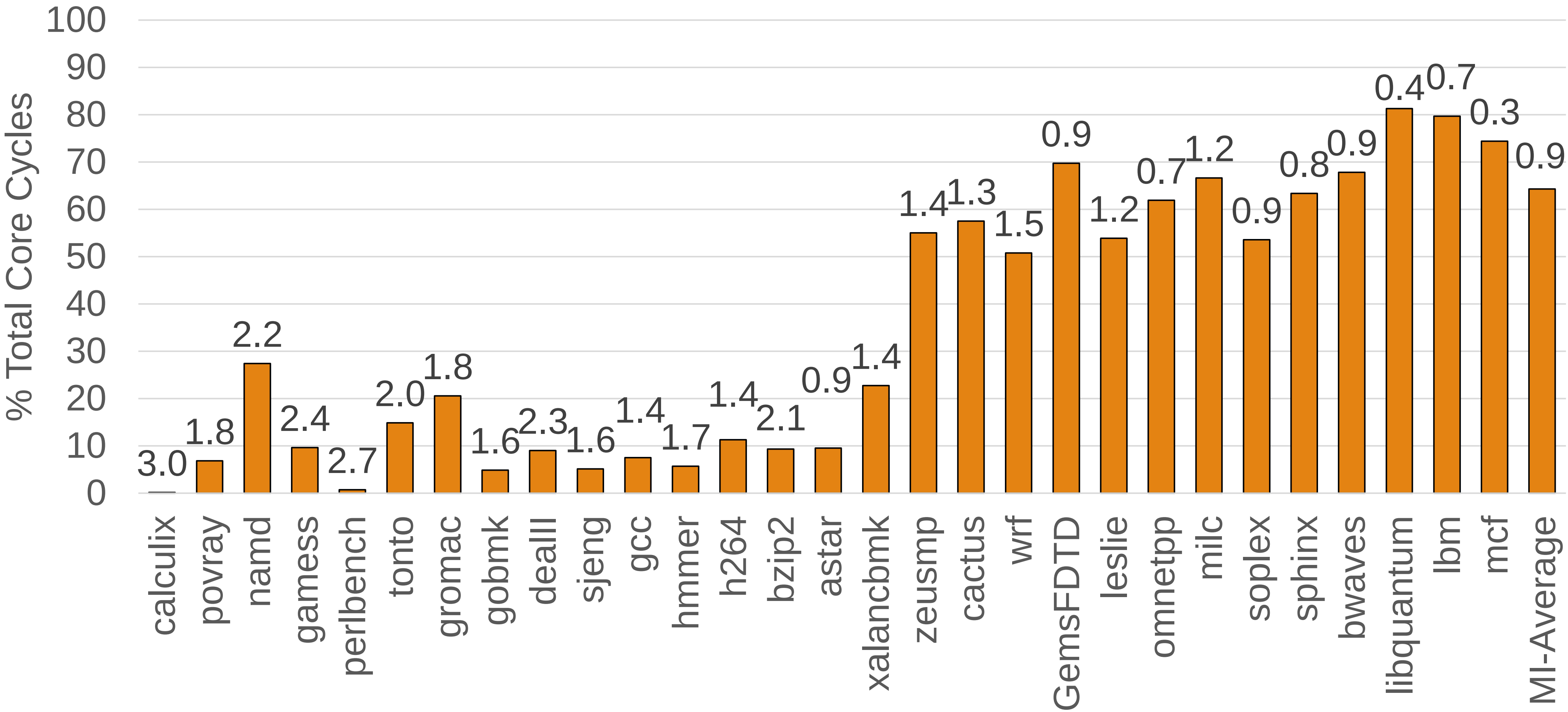
Runahead Execution Overview

- 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

Core Stall Cycles



Core Stall Cycles



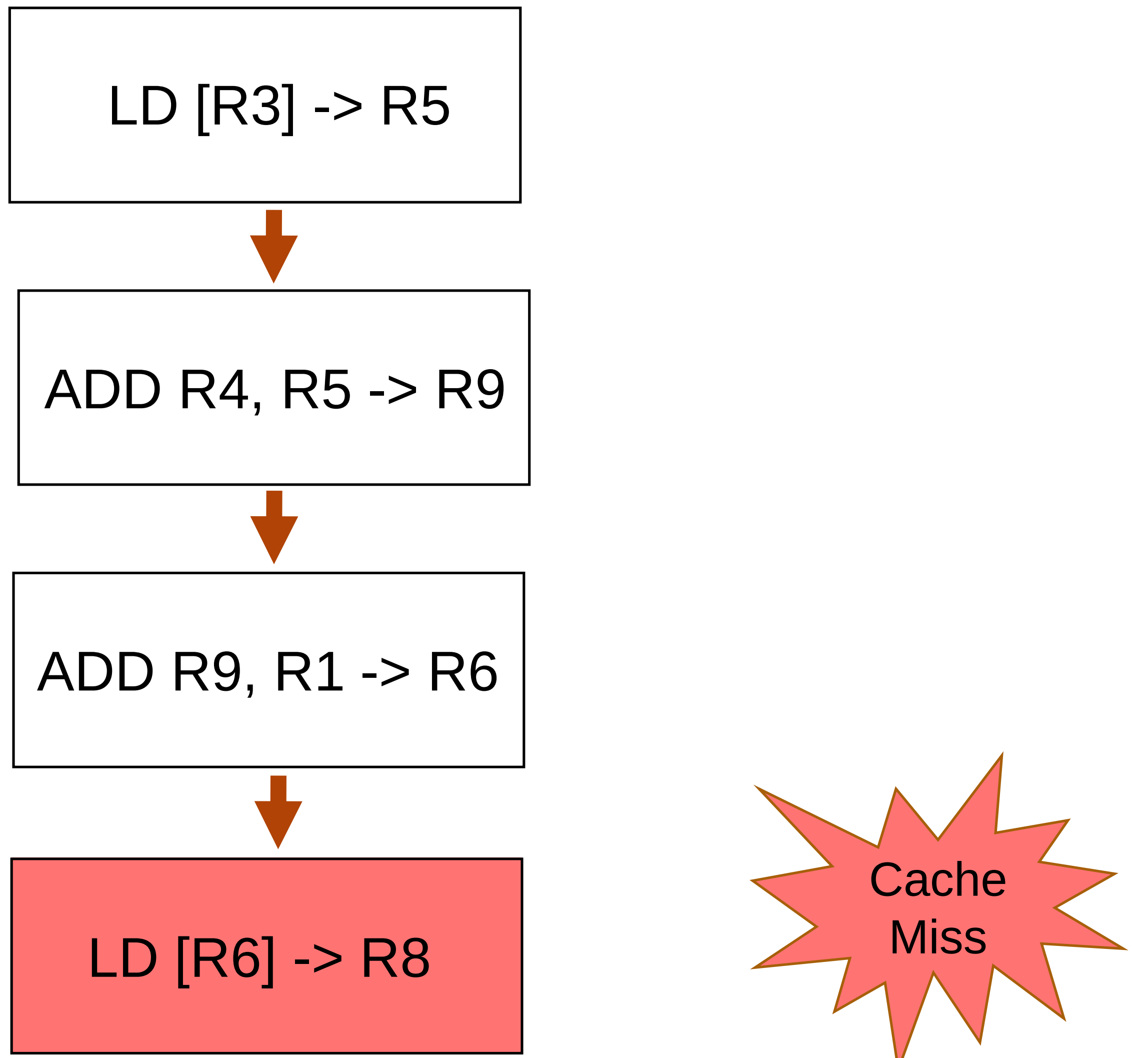
Runahead Buffer Overview

- Overview of Memory Dependence Chains
- Traditional Runahead Observations
- Runahead Buffer Proposal and Pipeline Modifications
- Runahead Buffer System Configuration and Evaluation
- Runahead Buffer Conclusions

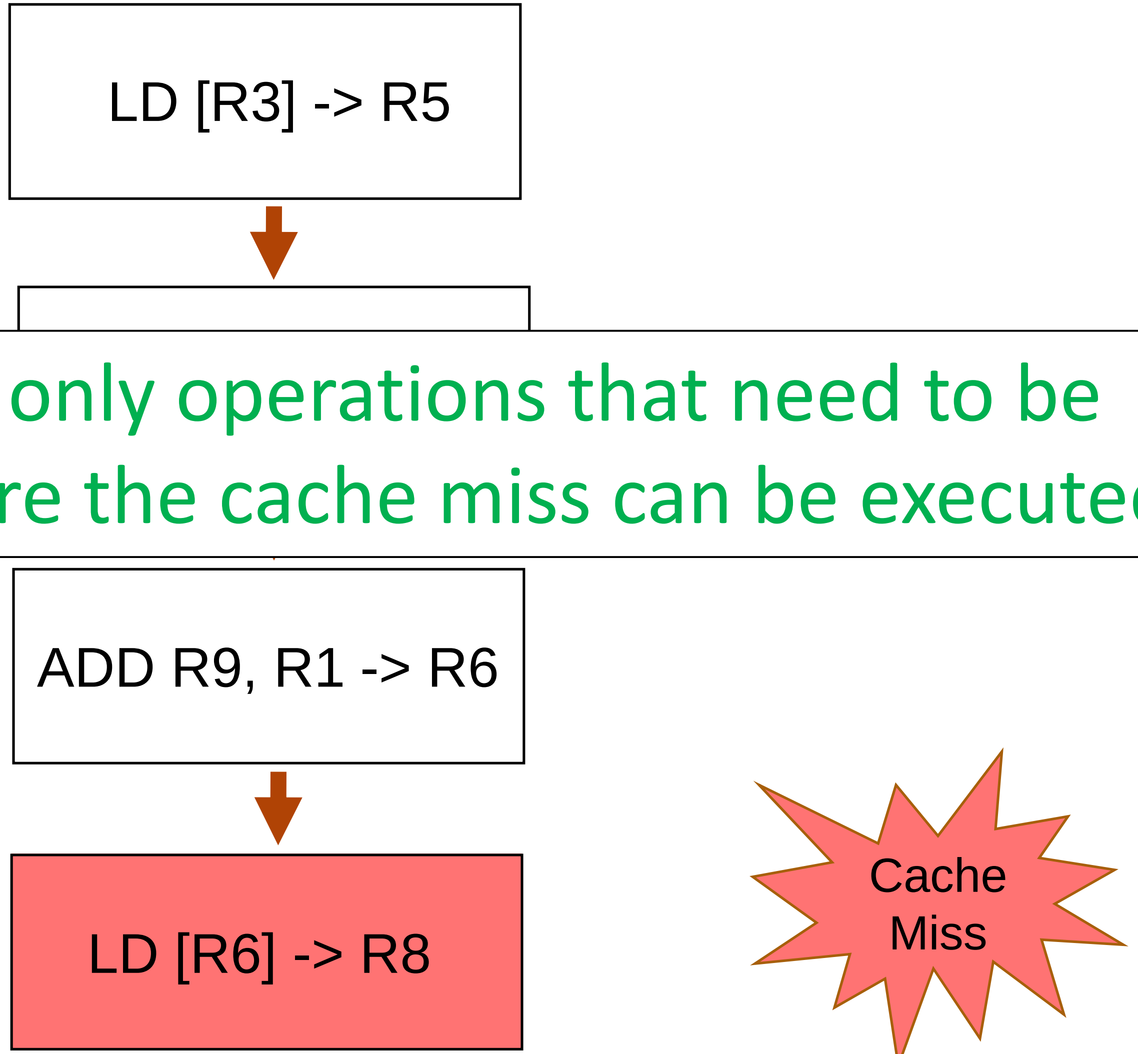
Background

- Every load has a chain of operations that must be completed to generate the address of the memory access

Example Dependence Chain



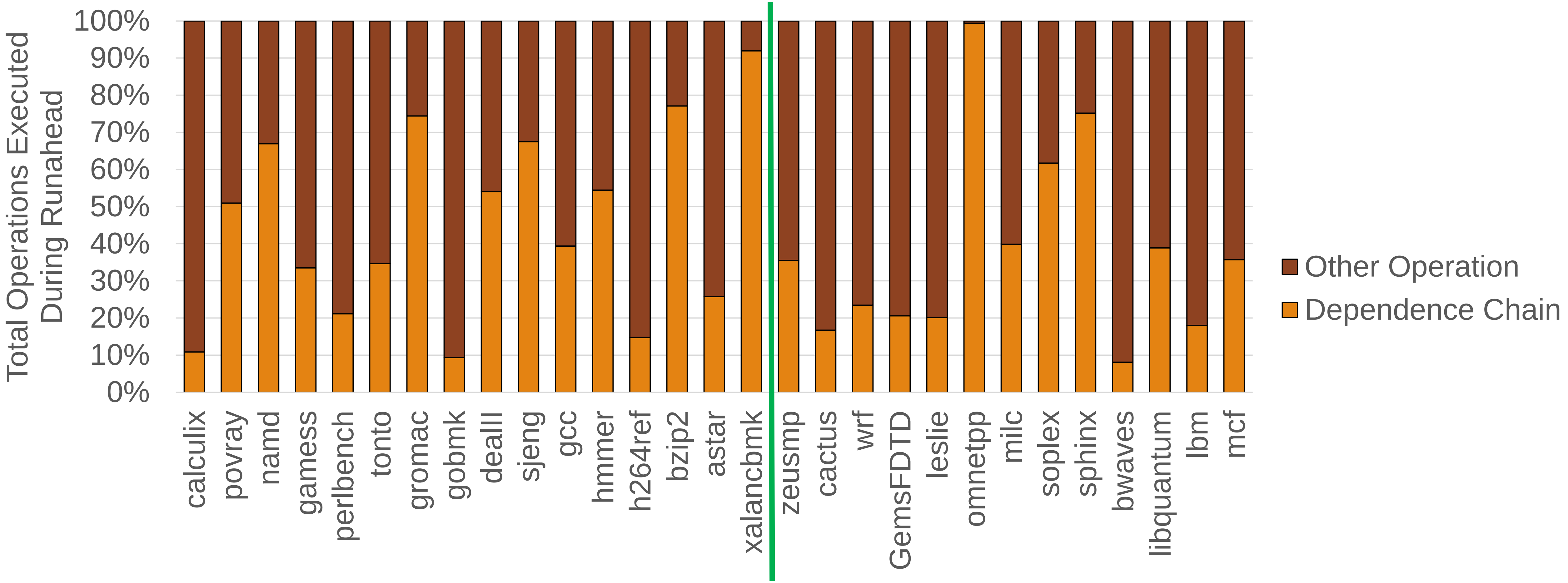
Example Dependence Chain



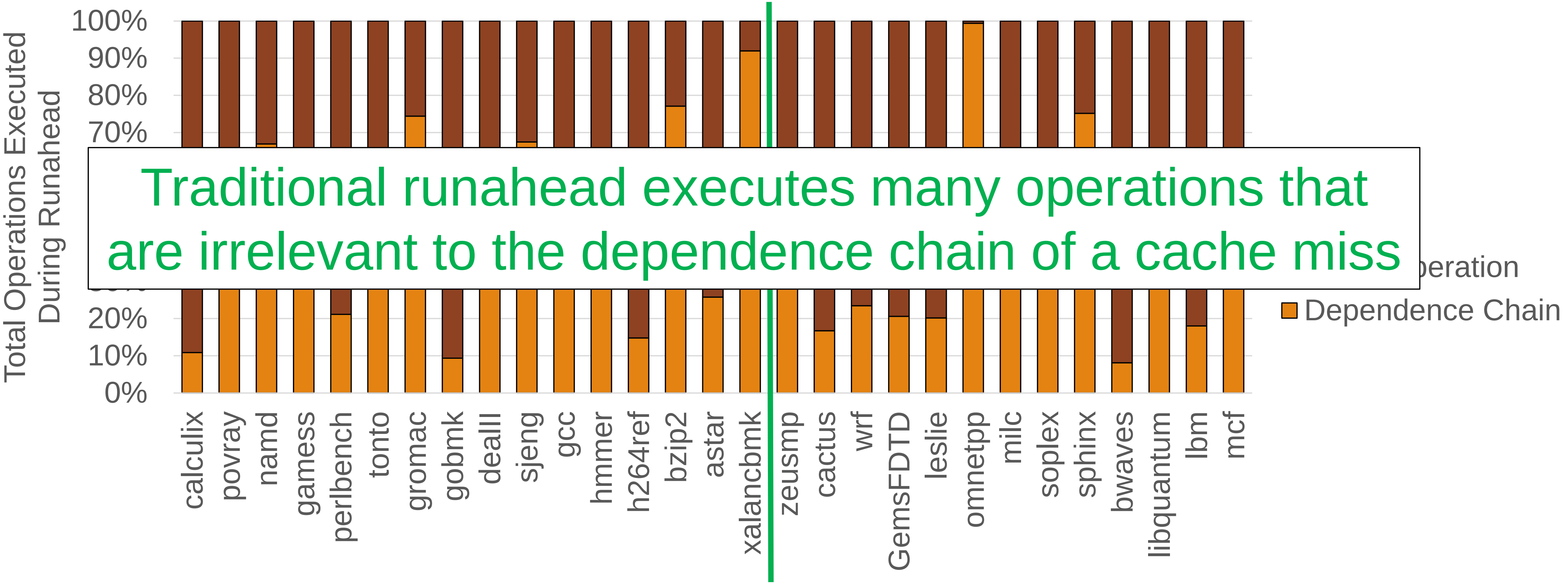
These are the only operations that need to be completed before the cache miss can be executed

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

Runahead Observations 1



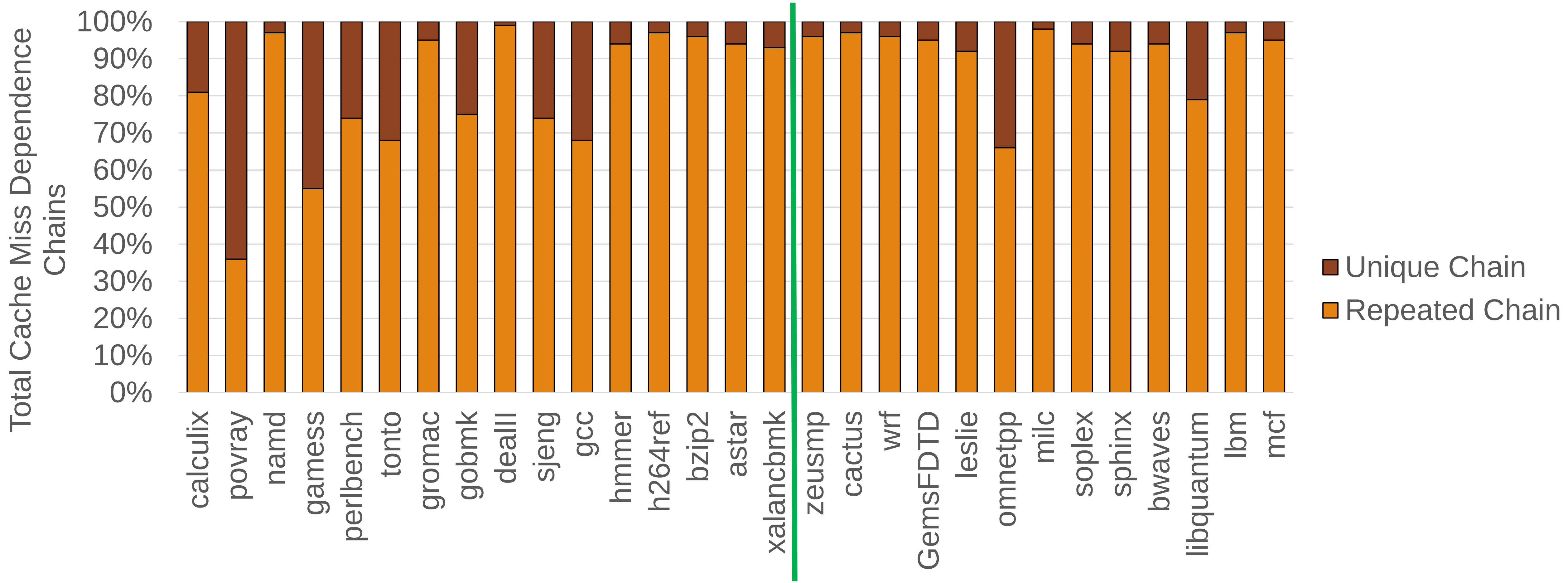
Runahead Observations 1



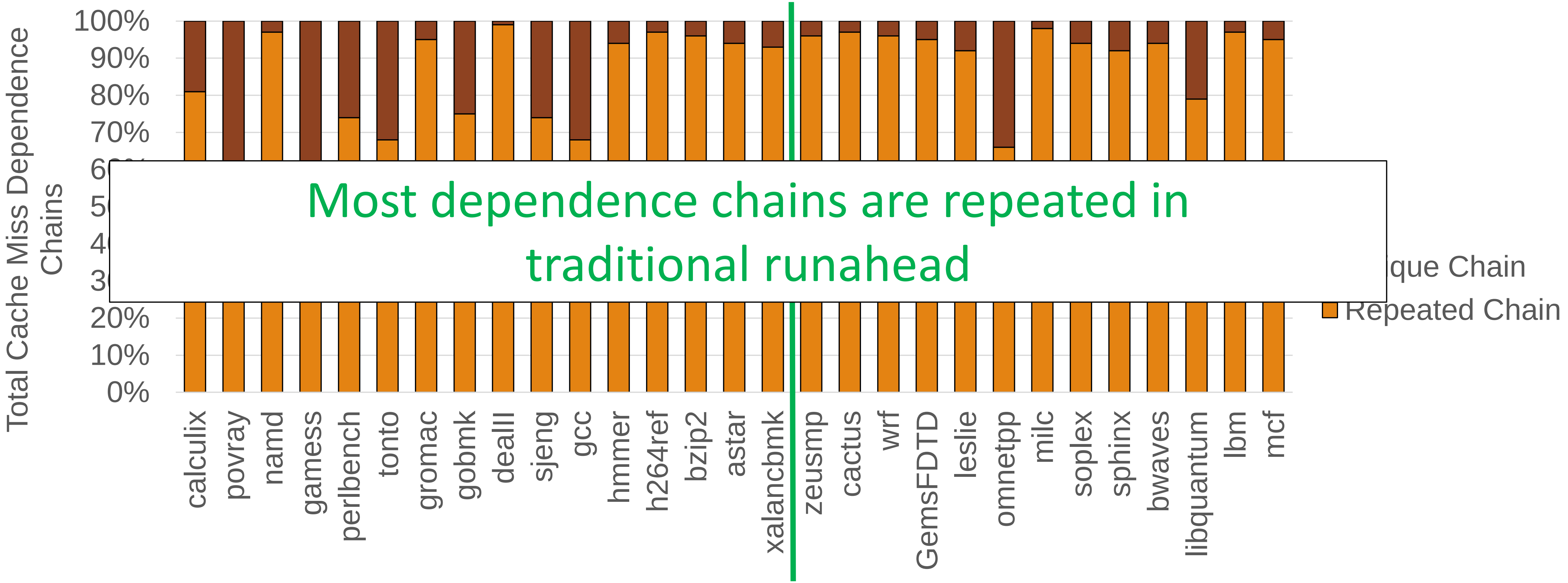
Traditional runahead executes many operations that are irrelevant to the dependence chain of a cache miss

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

Runahead Observations 2

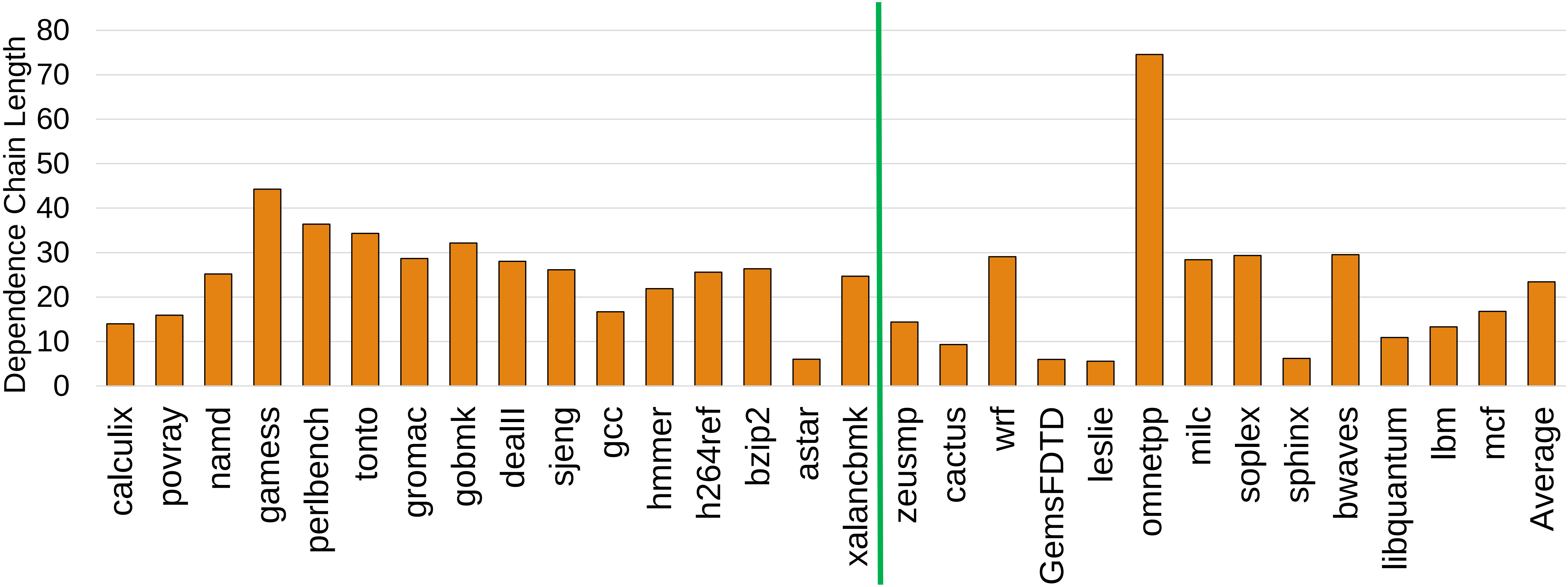


Runahead Observations 2

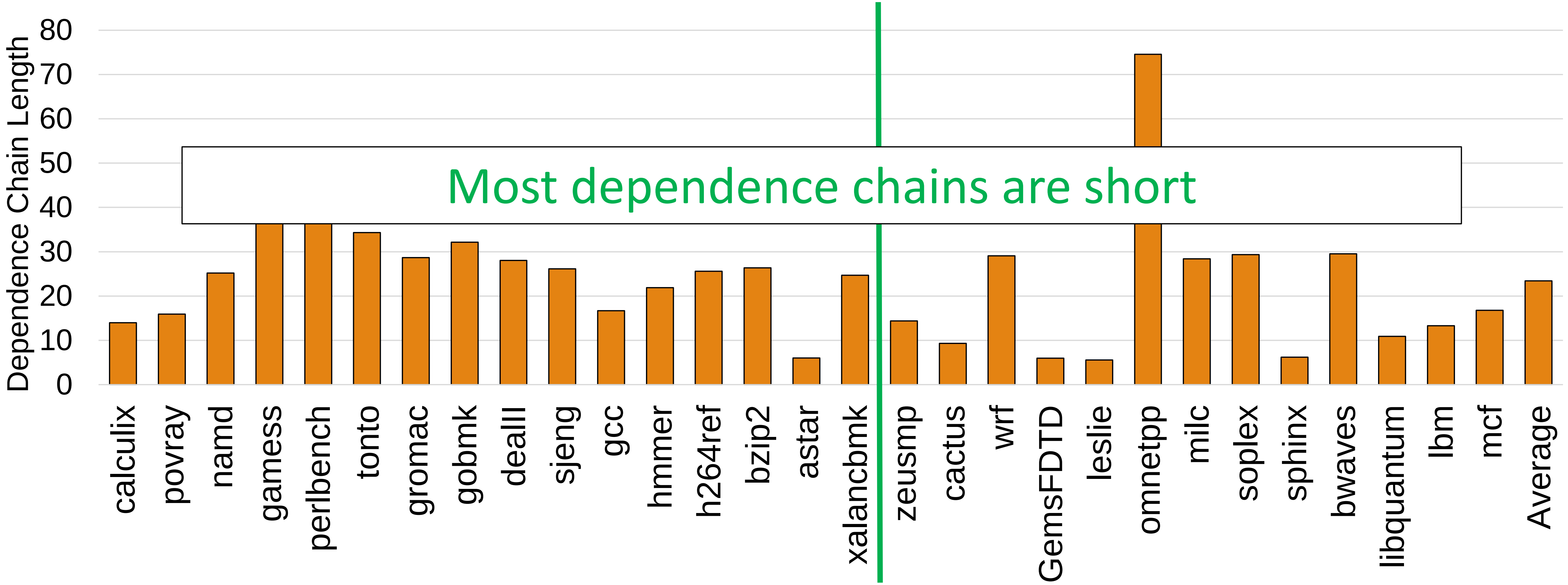


Most dependence chains are repeated in traditional runahead

Runahead Observations 3



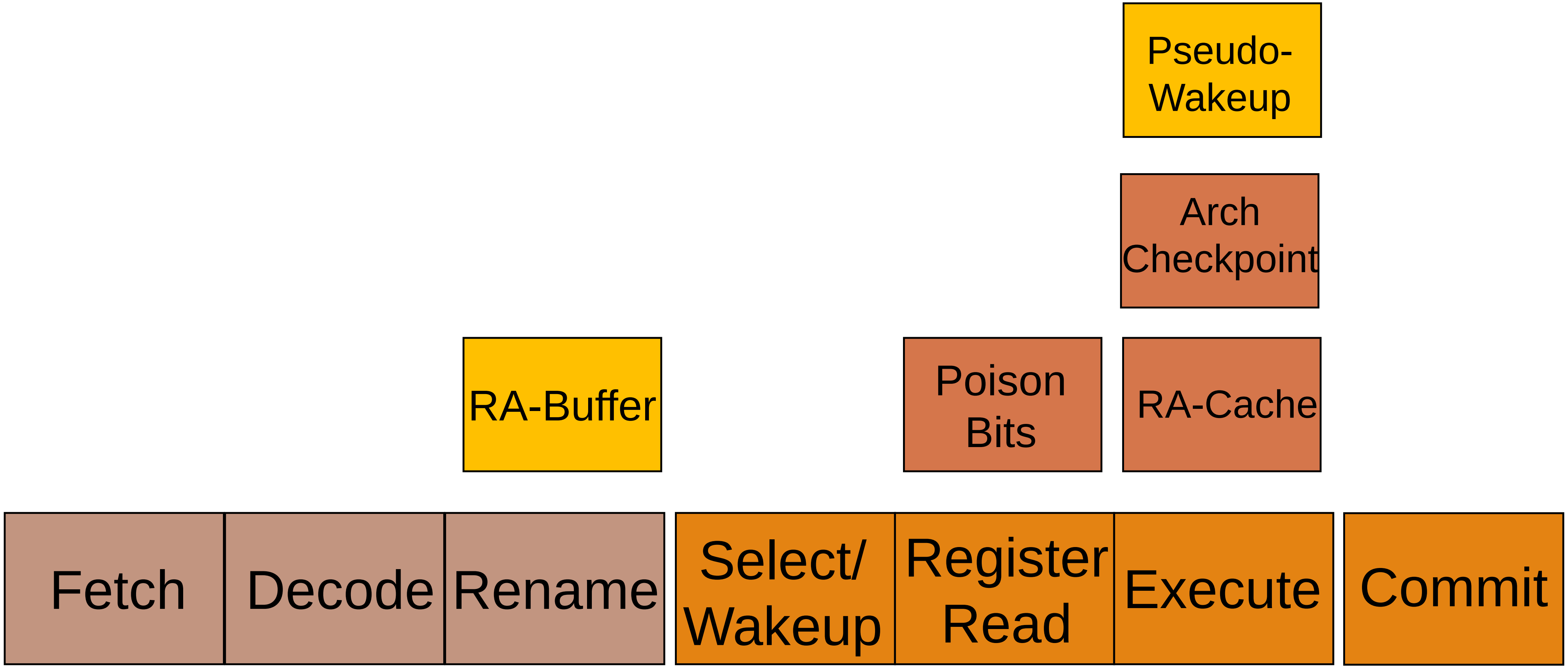
Runahead Observations 3



Runahead Buffer

- At a full window stall, dynamically identify the dependence chain to use during runahead from the reorder buffer
- Once the chain is identified, we place it in a runahead buffer
- The front-end is then clock-gated and the runahead buffer directly feeds decoded micro-ops into the back-end for runahead execution

Runahead Buffer Pipeline Modifications



Runahead Buffer Chain Generation

Cycle: 6

Source Register Search List: P9, P4, P5

0xA	LD [P15] -> P2	LD [R0] -> R2
0xD	LD [P3] -> P5	LD [R3] -> R5
0xE	ADD P4, P5 -> P9	ADD R4, R5 -> R7
0x7	ADD P9, P1 -> P6	ADD R7, R1 -> R6
0x8	MOV P6 -> P7	MOV R6 -> R0
0xA	LD [P7] -> P8	LD [R0] -> R2

Runahead Buffer Optimizations

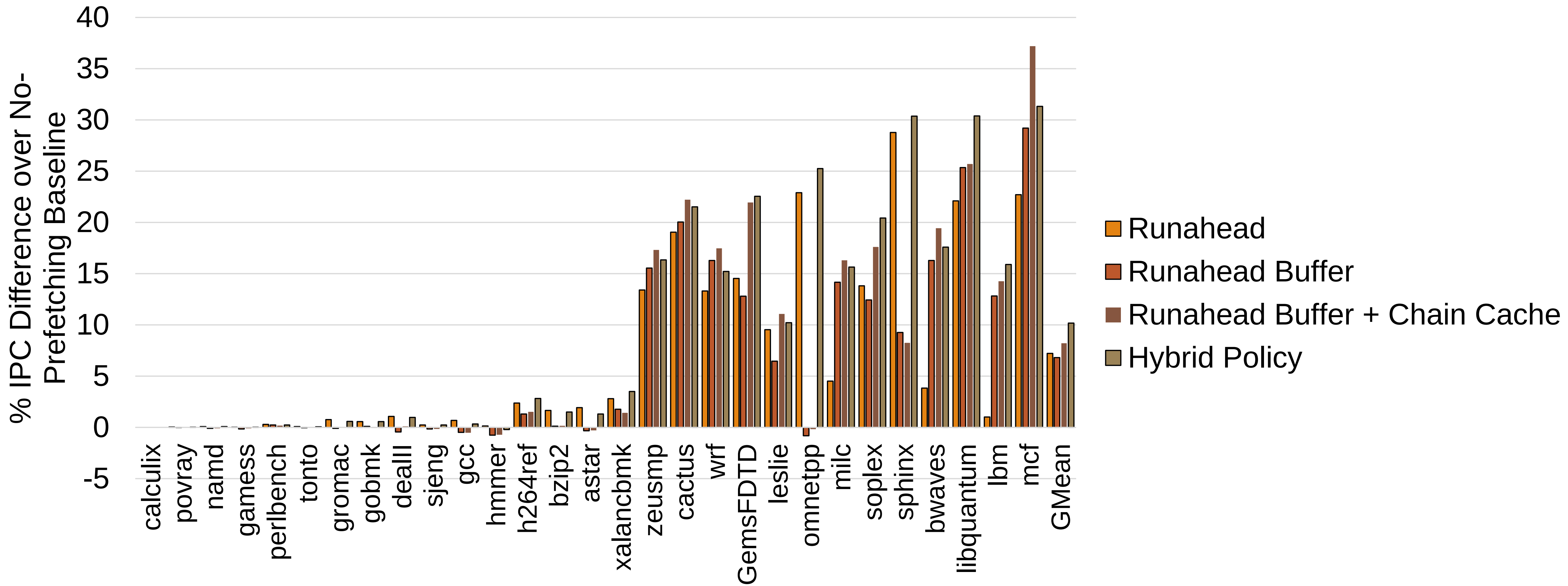
- A small dependence chain cache (2-entries) improves performance
- Hybrid Policy
  - The core begins traditional runahead execution instead of using the runahead buffer if:
    - An operation with the same PC as the operation that is blocking the ROB is not found in the ROB
    - The generated dependence chain is too long (more than 32 operations)

System Configuration

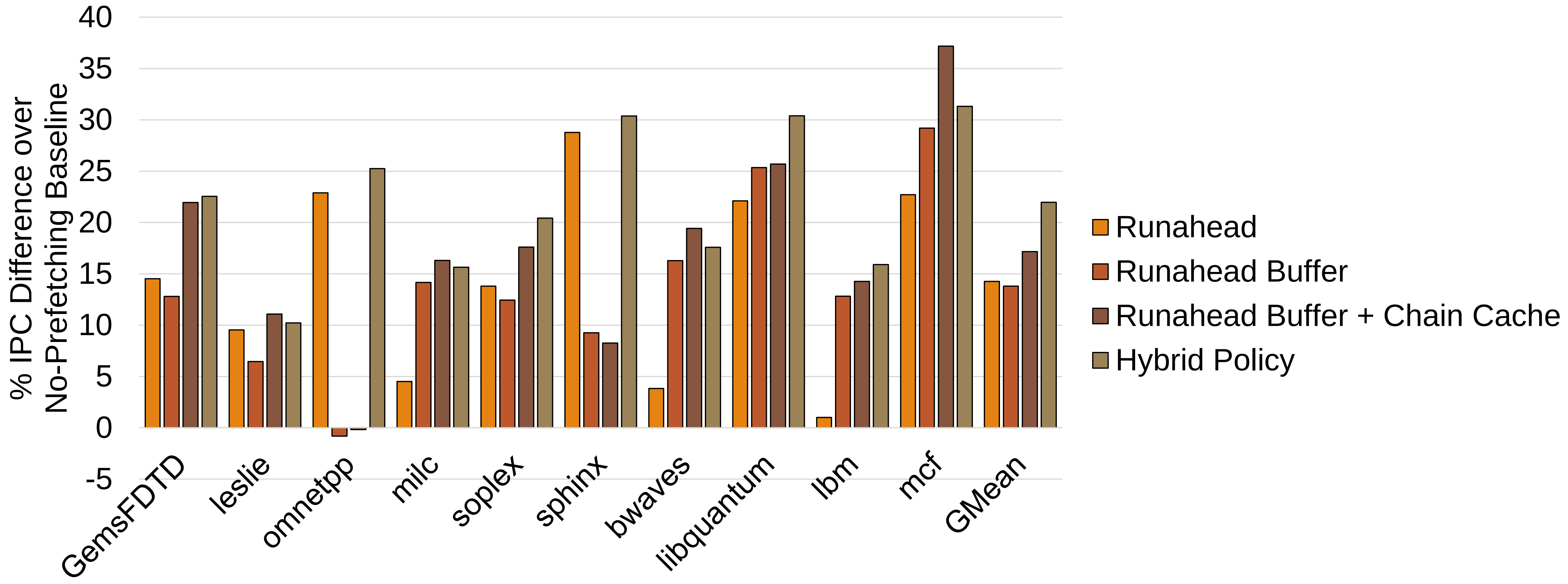
- Single Core
  - 4-wide Issue
  - 192 Entry Reorder Buffer
- Runahead Buffer
  - 32 Entry
  - Runahead Buffer Chain Cache: 2-Entries
- Caches
  - 32 KB L1 I/D-Cache, 3-Cycle
  - 1MB Last Level Cache, 18-Cycle
- Stream Prefetcher
- Non-Uniform Access Latency DRAM System

- 5 Configurations
  - Traditional Runahead
  - Runahead Buffer
  - Runahead Buffer + Chain Cache
  - Hybrid Policy
  - Traditional Runahead + Energy Optimizations

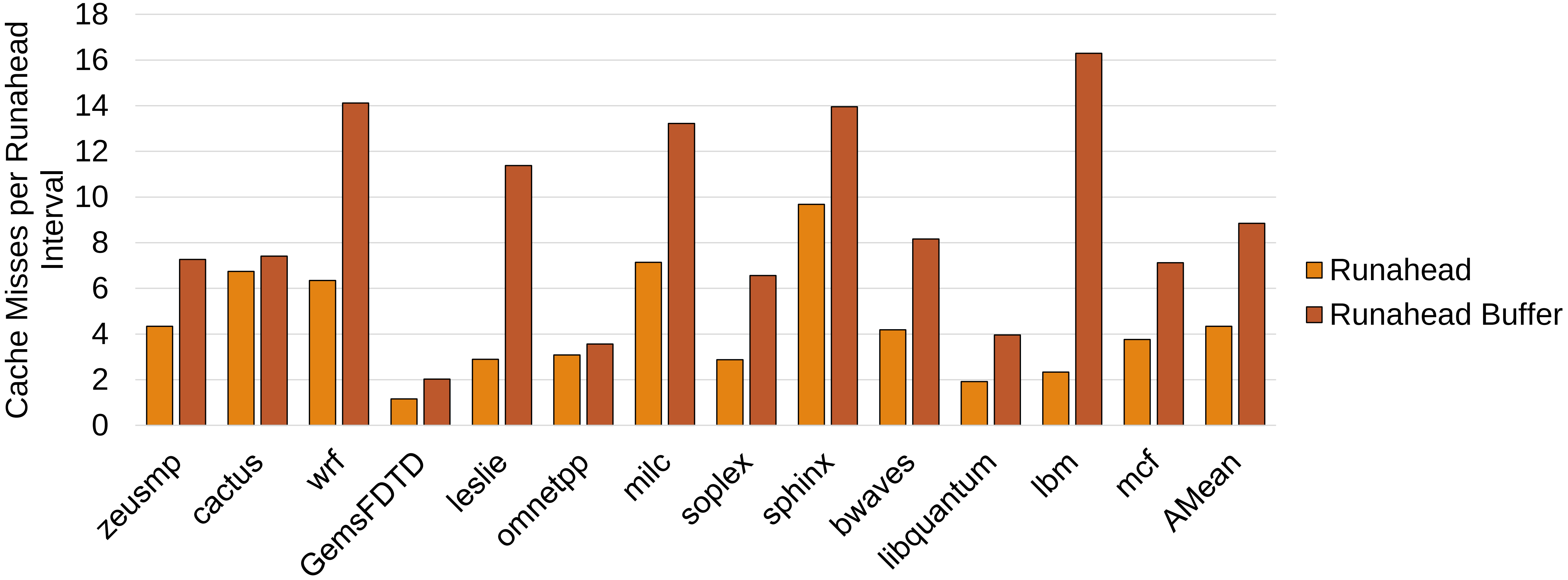
Runahead Buffer Performance



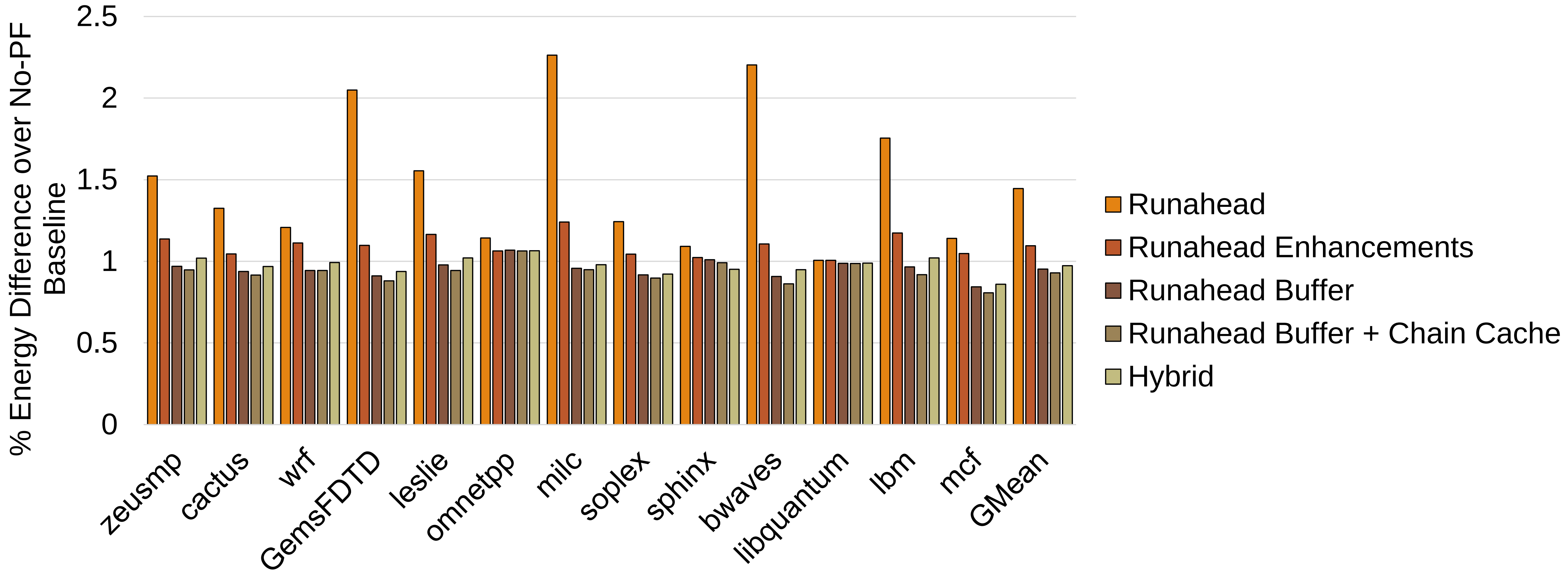
Runahead Buffer Performance



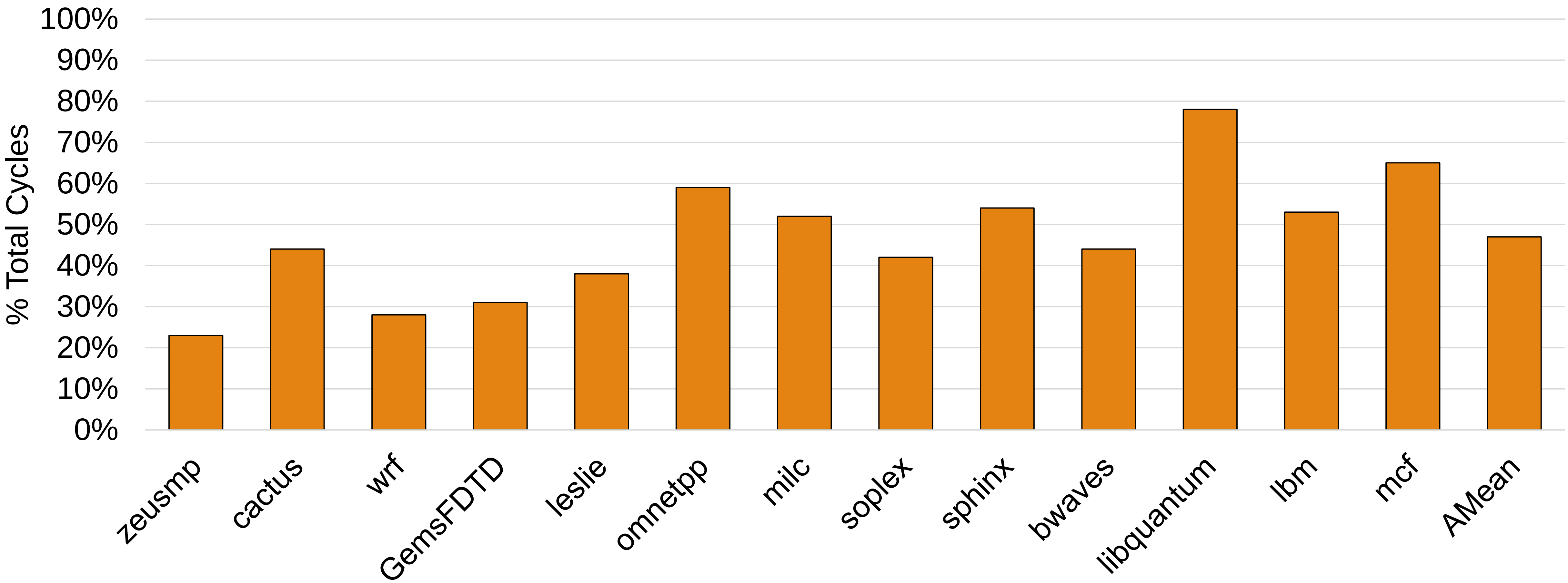
Runahead Buffer MLP



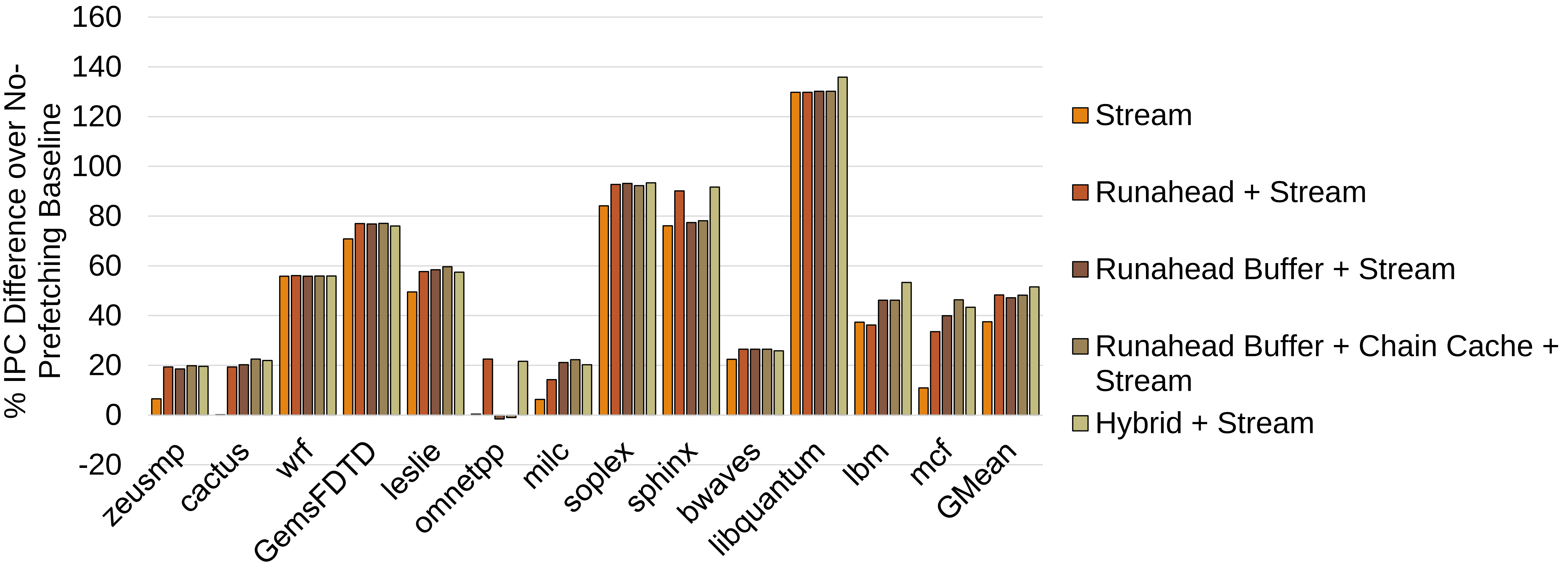
Energy Analysis



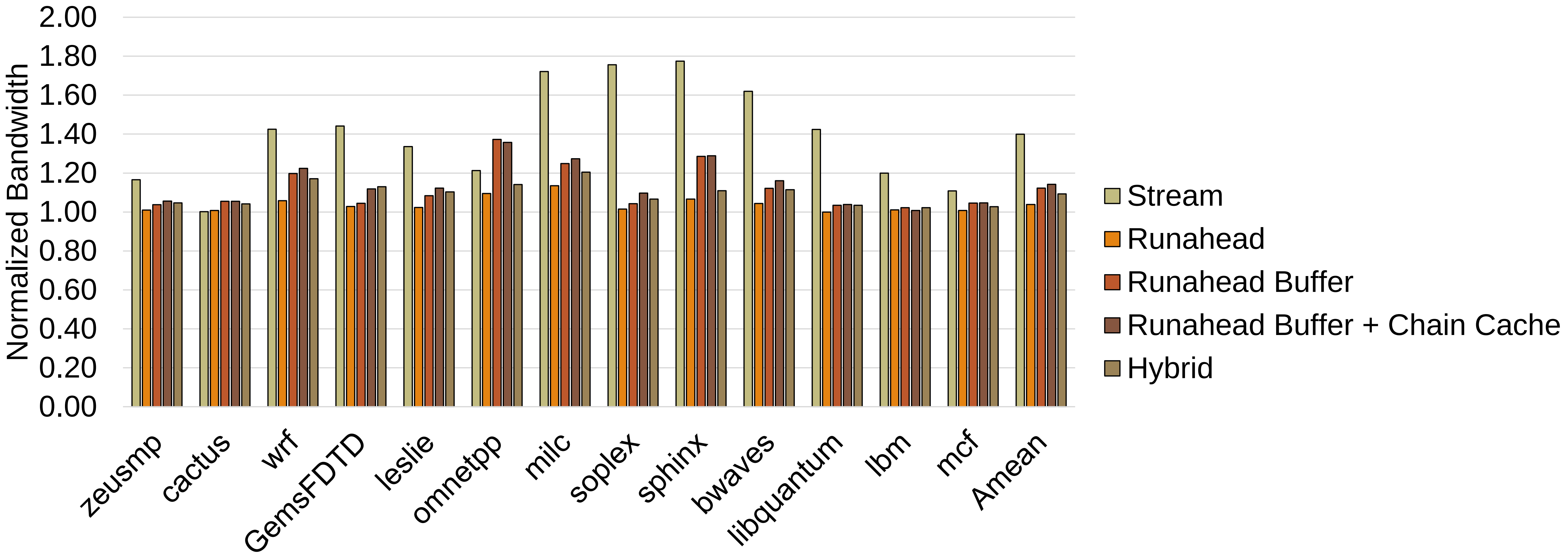
Stall Cycles in Runahead Buffer Mode



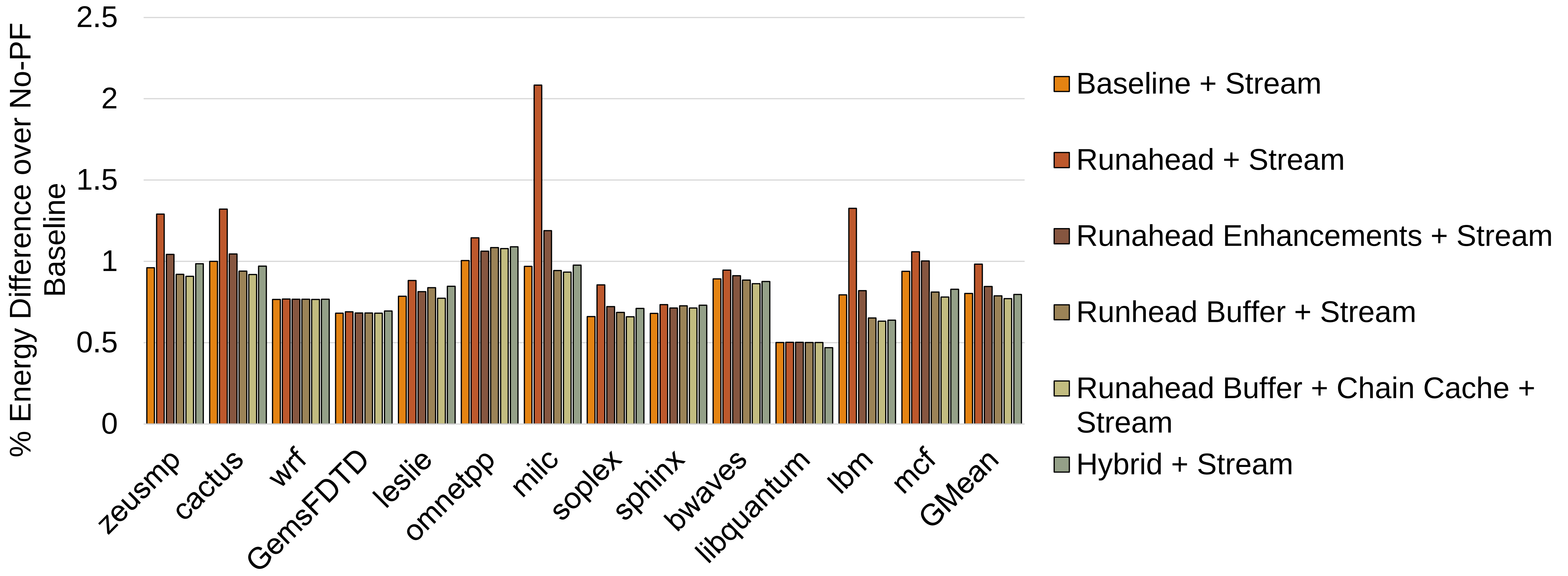
Stream Prefetching



Bandwidth Consumption



Energy Analysis



Runahead Buffer Conclusions

- Many of the operations that are executed in traditional runahead execution are unnecessary to generate cache misses
- The runahead buffer uses filtered dependence chains that only contain the operations required for a cache miss
- These chains are generally short
- This chain is read into a buffer and speculatively executed as if they were in a loop when the core would be otherwise idle

Runahead Buffer Conclusions

- The runahead buffer enables the front-end to be idle for 47% of the total execution cycles of the medium and high memory intensity SPEC CPU2006 benchmarks
- The runahead buffer generates over twice as much MLP on average as traditional runahead execution
- The runahead buffer results in a 17.2% performance increase and 6.7% decrease in energy consumption over a system with no-prefetching. Traditional runahead execution results in a 12.3% performance increase and 9.5% energy increase

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

Filtered Runahead Execution with a Runahead Buffer

Milad Hashemi
Yale N. Patt
December 8, 2015