Quantifying Server Memory Frequency Margin and Using It to Improve Performance in HPC Systems

Authors: Da Zhang[1], Gagandeep Panwar[1], Jagadish B.Kotra[2], Nathan DeBardeleben[3], Sean Blanchard[3], Xun Jian[1]

[1]Virginia Tech [2]AMD Research [3]Los Alamos National Laboratory

ISCA, 2021

Presented by Fiona Pichler

02.12.2021

Executive Summary

- Problem: DRAM manufacturers set memory frequency extra low to ensure reliability
  - This slows 99.999% of accesses down to benefit only 0.001% of accesses that need it
- Goal: Exploit memory frequency margin without loss of reliability for HPC systems
- Key Idea: Heterogeneously-accessed Dual Module Redundancy Hetero-DMR
  - Exploit HPC systems' abundant free memory to store copies of every data block
  - Operate the copies unreliably fast to speed up common case access → use the safely operated original blocks for recovery
- Evaluation Results: Real system and simulation analyses show
  - Reduction of job execution time by 15%
  - 1.4x turn around time speed up
  - 6% less energy per instruction

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Motivation

- Definition: **Frequency margin** is the gap between manufacturers specified frequency and the frequency at which memory still works correctly for most (>99.999%) accesses
- Manufacturers increase the reliability of their products by setting the frequency specification low
- There is no prior work on frequency margins

Scale of This Study

	DRAM type	# of modules	# of chips	Margin Studied
This Paper	DDR4 RDIMM	119	3006	frequency
Prior Work [60]	DDR3 SO-DIMM	96	768	latency
Prior Work [56]	DDR3 SO-DIMM	32	416	latency
Prior Work [47]	DDR3 SO-DIMM	30	240	latency
Prior Work [65]	LPDDR4	N/A	368	latency
Prior Work [62]	DDR3 SO-DIMM	34	248	latency
Prior Work [50]	DDR3 UDIMM	8	64	voltage

Study Results

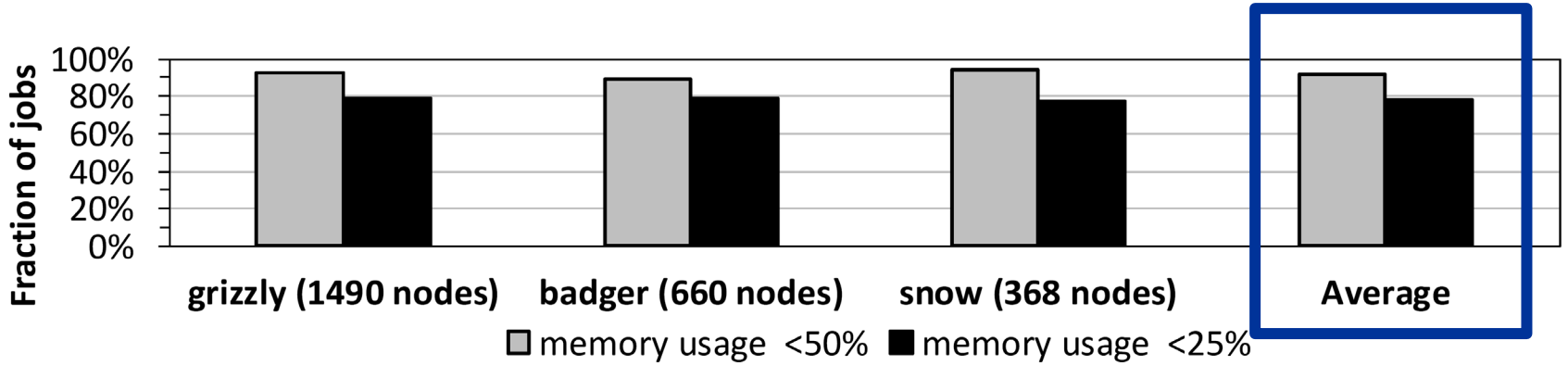
- Characterizing the memory frequency margin shows the potential of exploiting it
- Exploiting both memory frequency and latency margins provides 1.19x speedup on average
- Aging, #ranks/module, chip density and manufacturing date have little impact on frequency margin
- Exploiting latency has no effect on frequency margins

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Background

- Analysis of 3 billion memory measurements over 7 million machine-hours this paper reaches the same conclusion



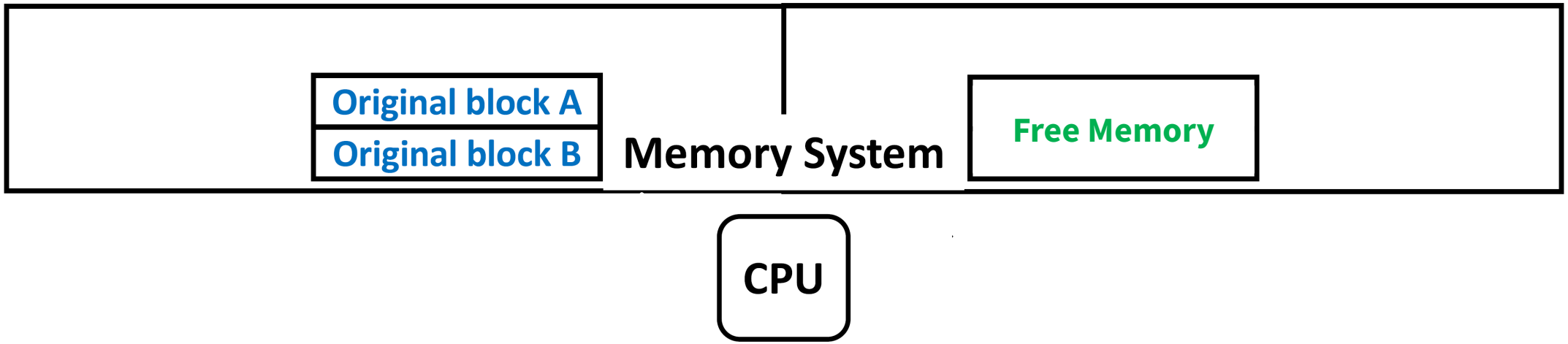
HPC systems have abundant free memory

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Key Idea

- Heterogeneously-accessed Dual Module Redundancy, Hetero-DMR
- Exploit memory frequency without loss of reliability
- Copy all data, so we have a set of original and a set of copied data in memory
- Exploit memory frequency only on copies
  - In case of an error we still have the untouched original



Keeping the Original Data Safe (1/2)

Write mode:

1. Save copy in same channel on the same location in different ranks to keep overhead low
2. Operate safely (normal frequency) for all data on writes

- Writes make only 15% of all memory accesses
- Lowering frequency when switching from read to write increases latency by 100x
  - Switch 100x less from read to write -> increase write batch size by 100x

Keeping the Original Data Safe (2/2)

Read mode:

- Only read from Copies, except for error correction

General:

- Set original blocks to self-refresh
- No CPU can overclock self-refresh mode

Error Detection

- Use existing ECC, but only for detection
- ECC encode/decode lies in CPU -> we don’t need to make changes to memory
- Use Bamboo-ECC, an especially reliable and adaptive ECC technique
  - detect all up to 8 byte errors
- 8B+ errors can’t always be detected
- Use a threshold for the number of errors after which frequency is not exploited anymore to keep the probability of 8B+ errors low

Error Correction

1. Slow memory access down and reliably read the original
   - Only happens for < 0.001% of all accesses
2. Speed up memory access again

Memory Frequency Variability

Channel level

- Different modules can have different margins
- Choose channel with highest frequency to exploit the margin

Node level

- Different channels in a Node have different margins
- Node-level frequency margin = Lowest channel-level frequency margin

System level

- Margin aware job schedulers to not waste potential

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Implementation

- More than 1/2 free memory -> Hetero-DMR replicates every block and operates fast on copies
- Less than 1/2 free memory -> Hetero-DMR operates at normal frequency
- To increase the write batch size by 100x, add 128KB 64-way victim writeback cache per channel between LLC and channel's write buffer
- Memory modules with permanent but ECC correctable faults are only used to store originals
- Margins are profiled at boot time and periodically re-profiled
  - for this a mechanism from another paper is used

Longeterm Effects of Hetero-DMR

Hetero-DMR should not increase aging

- No increased operation-voltage
- No increased DIMM temperature
- DRAM-cells have practically infinite endurance

- This was just argued, not tested

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Methodology (1/2)

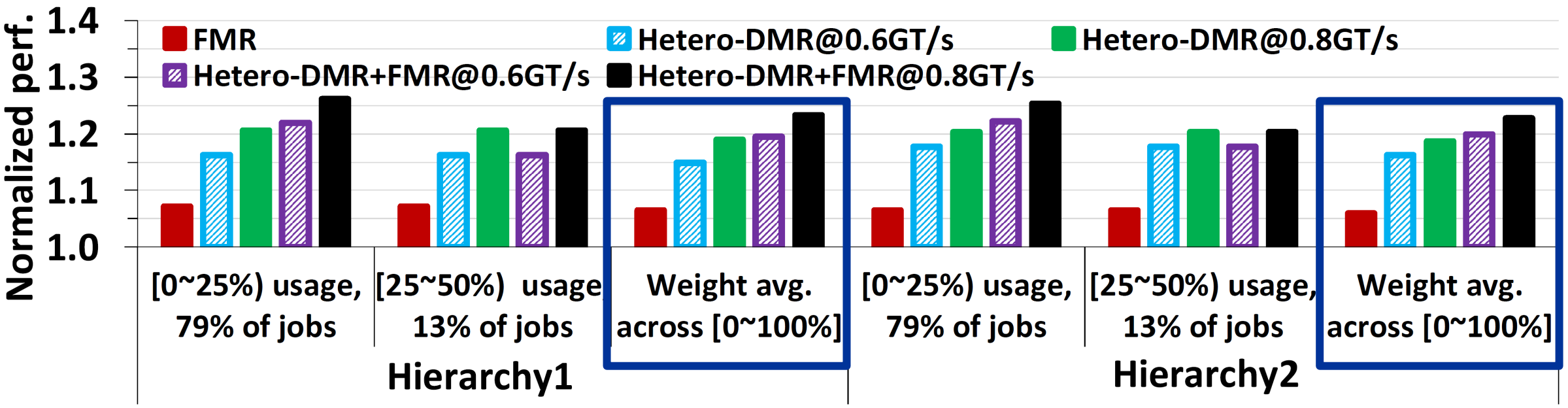
- Simulation of a single-node system with Hetero-DMR with Gem5
  - Ramulator as the memory subsystem
  - Simulate the CPU used for the frequency margin tests
  - 2 Memory hierarchies

	Memory Hierarchy1	**Memory Hierarchy2**
L2$+L3$ per core	4.5MB / core	2.375MB / core
Cores	8 cores	16 cores
Memory Channels	1 channel, 2modules/channel, 2ranks/module	4 channels, 2modules/channel, 2ranks/module

Methodology (2/2)

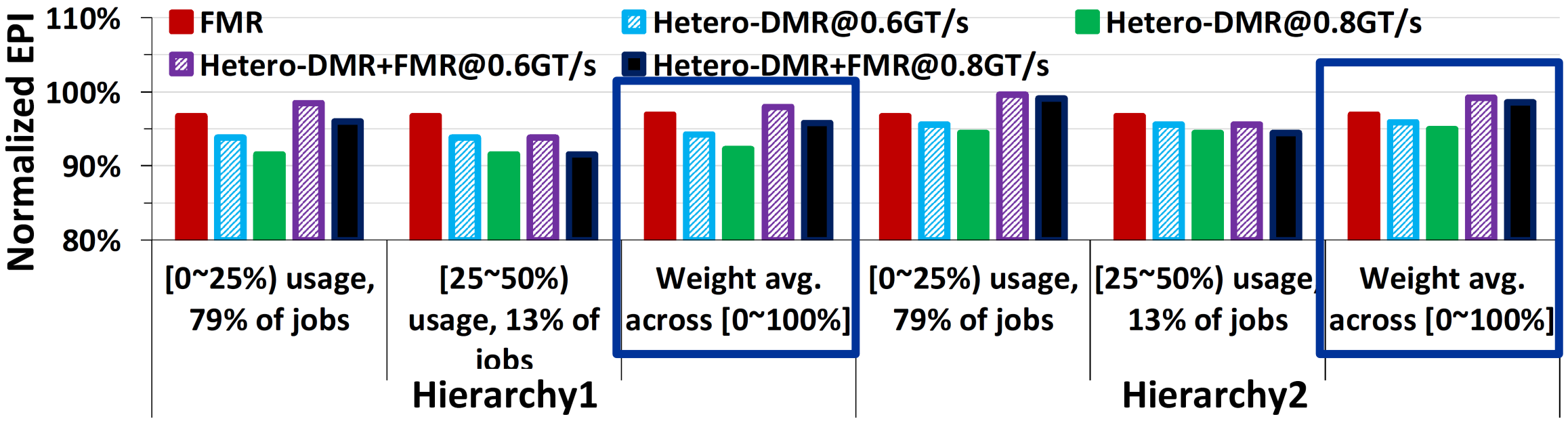
- Hetero–DMR: with 0.8GT/s and 0.6GT/s node-level frequency margins
- Hetero-DMR +FMR:
  - FMR: Free-memory-aware Memory Replication → copy memory and access the one currently in the faster state
  - when memory utilization is <25% make two copies and apply Hetero-DMR for the with FMR found faster copy for memory
  - when memory utilization is >25% operate as only Hetero-DMR without any FMR influence
- The Results are normalized to the Commercial Baseline, which means operating without Hetero-DMR or FMR

Hetero-DMR Simulation Speedup



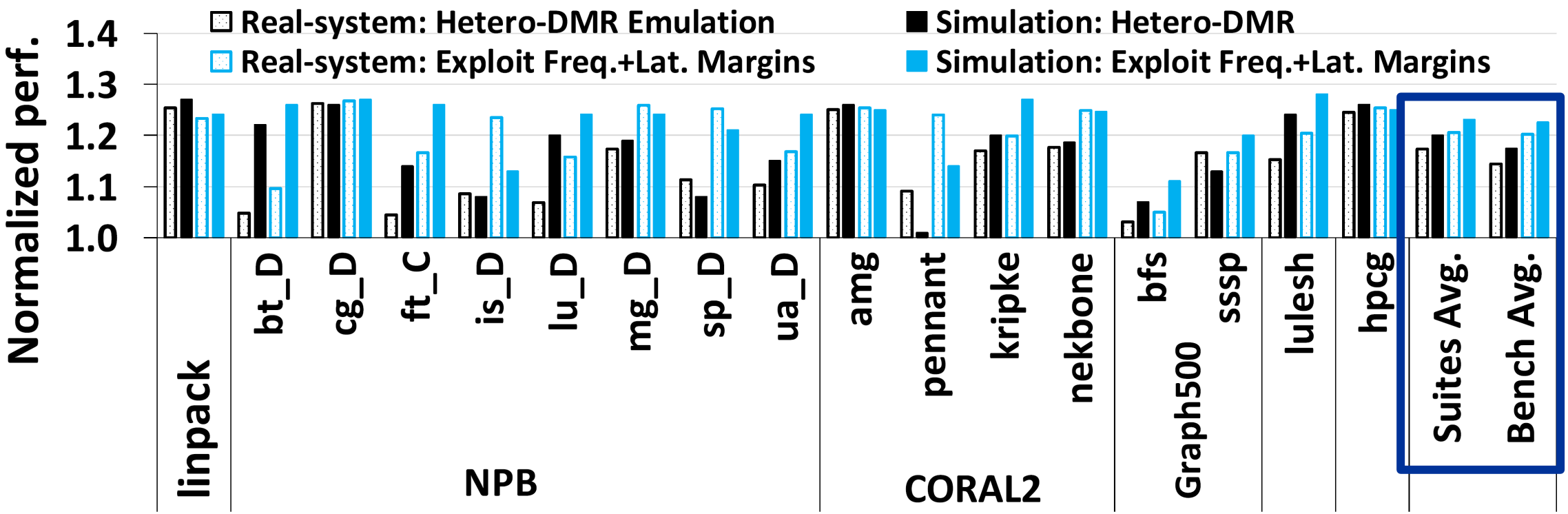
- There's almost no difference between the 2 hierarchies
- We get the best performance from Hetero-DMR + FMR

Hetero-DMR Simulation Energy per Instruction



Hetero-DMR improves Energy Per Instruction by improving performance

Hetero-DMR on Real System vs. Simulation



Simulating Hetero-DMR performance is very similar to real-system Hetero-DMR performance

Results

- Commodity RDIMMs can operate on average 27% faster without errors for 99.999%+ of memory accesses
- Hetero-DMR reduces job execution time by 15% on average
- This means 1.17x average speedup
- 1.4x turnaround-time-level speedup
- 6% improved EPI on average
- **With Hetero-DMR a System is faster while using less energy**

Overview

- Motivation
- Background
- Key Idea
- Implementation
- Results
- Conclusion
- Strengths and Weaknesses
- Discussion

Conclusion

- Problem: DRAM manufacturers set memory frequency extra low to ensure reliability
  - This slows 99.999% of accesses down to benefit only 0.001% of accesses that need it
- Goal: Exploit memory frequency margin without loss of reliability for HPC systems
- Key Idea: Heterogeneously-accessed Dual Module Redundancy, Hetero-DMR
  - Exploit HPC systems' abundant free memory to store copies of every data block
  - Operate the copies unreliably fast to speed up common case access → use the safely operated original blocks for recovery
- Evaluation Results: Real system and simulation analyses show
  - Reduction of job execution time by 15%
  - 1.4x turn around time speed up
  - 6% less energy per instruction

Questions?

Strengths

- First study on memory frequency margin
- First study on memory margins for servers
- Large scale study
- Future hardware considered
- Faster & less energy consumption

Weaknesses

- Needs free memory
- Extra cache needed for Hetero-DMR
- No long-term study
- Weak CPU for study
- Cloud memory specs are not public
- They emphasize being the first to do a study on memory frequency margin too much

Discussion

- Can we use Hetero-DMR for general systems?
- Add extra memory?
- Use this 2 sets of data idea to exploit other margins? Voltage?
- Is 1.17x speed up worth it?

Thank you