Unless otherwise specified, all content throughout the presentation refers to X-Mem v2.2.3 as in the published paper.

X-Mem: A Cross-Platform and Extensible Memory Characterization Tool for the Cloud

Mark Gottscho[1,2]
Sriram Govindan[3]
Bikash Sharma[3]
Mohammed Shoaib[2]
Puneet Gupta[1]

[1]UCLA
Los Angeles, CA, USA

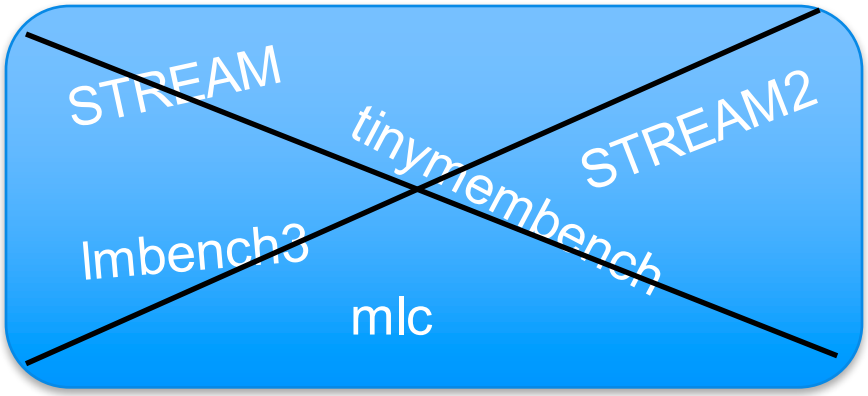
[2]Microsoft Research
Redmond, WA, USA

[3]Microsoft
Redmond, WA, USA

IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Uppsala, Sweden
April 19, 2016 – http://seas.ucla.edu/~gottscho

Motivation: Memory is Important in Cloud Computing

- *Cloud subscribers* want to maximize app. performance
- *Cloud providers* want to minimize CapEx/OpEx given SLAs
- Needs pressure memory hierarchy: characterization is critical
- Memory benchmarking tools don't meet key requirements
  - (A) Access pattern diversity
  - (B) Platform variability
  - (C) Metric flexibility
  - (D) Tool extensibility



Motivation: Memory is Important in Cloud Computing

- *Cloud subscribers* want to maximize app. performance
- *Cloud providers* want to minimize CapEx/OpEx given SLAs
- Needs pressure memory hierarchy: characterization is critical
- Memory benchmarking tools don't meet key requirements
  - (A) Access pattern diversity
  - (B) Platform variability
  - (C) Metric flexibility
  - (D) Tool extensibility

We propose X-Mem, a new tool!
Project homepage:
nanocad-lab.github.io/X-Mem
Source code:
github.com/Microsoft/X-Mem

X-Mem Feature
(A) Access Pattern Diversity

Degrees of Freedom

1.	Access granularity	32, 64, 128, 256, and 512-bit* chunk sizes
2.	Access types	Read or write
3.	Access patterns	Random, sequential and strided in $\pm 2^{0\text{-}4}$ chunks
4.	Parallelism	Multithreaded
5.	Page sizes	Large and normal
6.	Topologies	CPU and memory NUMA nodes, core affinity

**As of v2.4.1, April 2016*

(D) Extensibility: Developers can easily add specialized patterns through new benchmark kernel functions

X-Mem Feature
(B) Platform Variability

1.	OS Support	Windows, GNU/Linux
2.	Architectural support	x86, x86-64 with(out) AVX SIMD extensions, Xeon Phi*, ARMv7 with(out) NEON SIMD extensions, ARMv8

**As of v2.4.1, April 2016*

- Portable Python-based build system using SCons
- **(D) Extensibility**
  - OS and architecture ports are low effort
  - Apples-to-apples memory hierarchy comparisons

X-Mem Feature
(C) Metric Flexibility

- **Performance:** X-Mem measures real performance of the memory hierarchy *as could be seen by an application*
  - Distinct from performance counter or component-centric view
  - Aggregate throughput
  - Unloaded latency
  - Loaded latency
- **Power**
  - Simple software hooks for custom power measurement hardware
- **Statistics on each metric**
  - Mean, percentiles*, min/max*, etc. **As of v2.4.1, April 2016*
- **(D) Extensibility:** fault injection & reliability studies, data-aware power/performance bookkeeping for NVMs, etc.

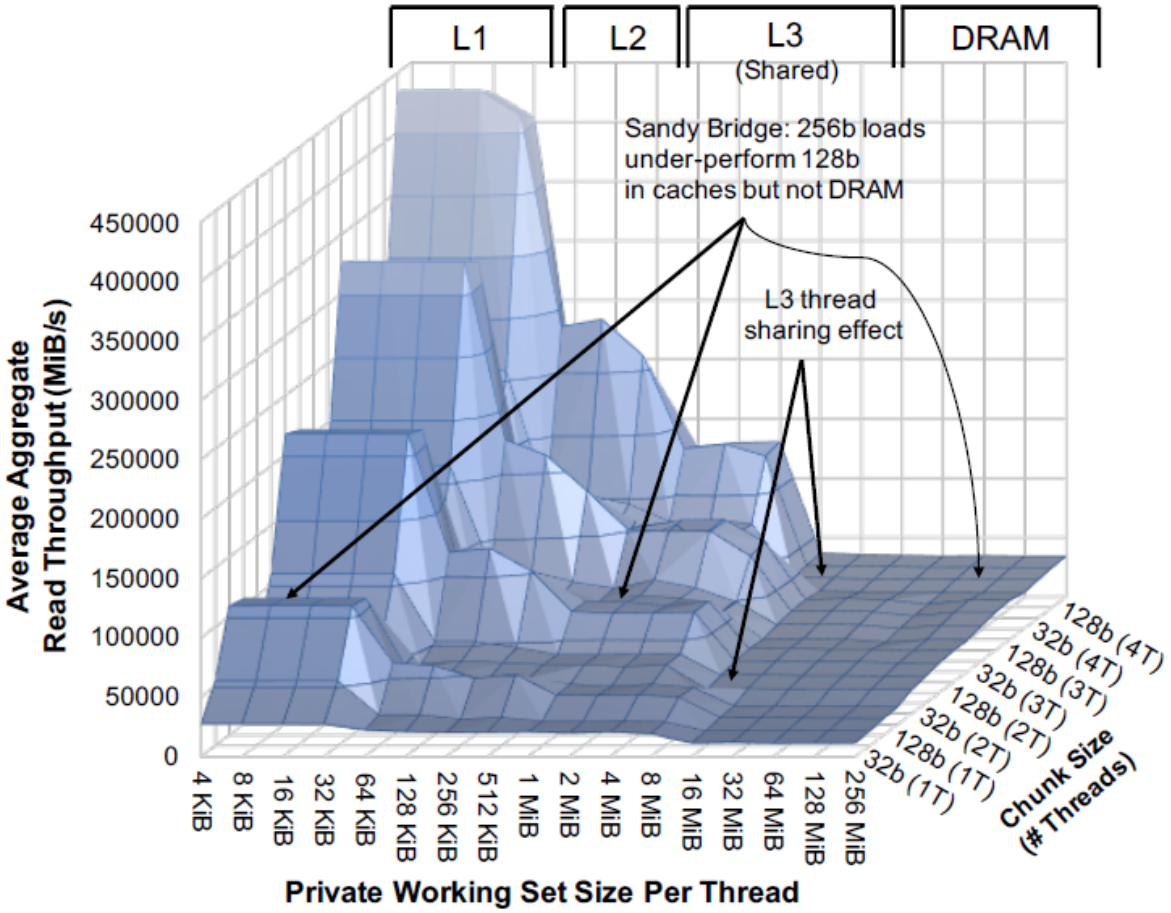
Case Study 1:
Characterization of the Memory Hierarchy for Cloud Subscribers

- Cloud subscribers should measure and leverage:
  - Cache micro-architecture
  - System-level memory management

- Understanding these enables improved application performance:
  - Workload partitioning among threads?
  - Working set size per thread?
  - Data access patterns?
  - When, where, and how to allocate memory?

Case Study 1:
Characterization of the Memory Hierarchy for Cloud Subscribers

Landscape of the memory hierarchy for the Desktop platform.

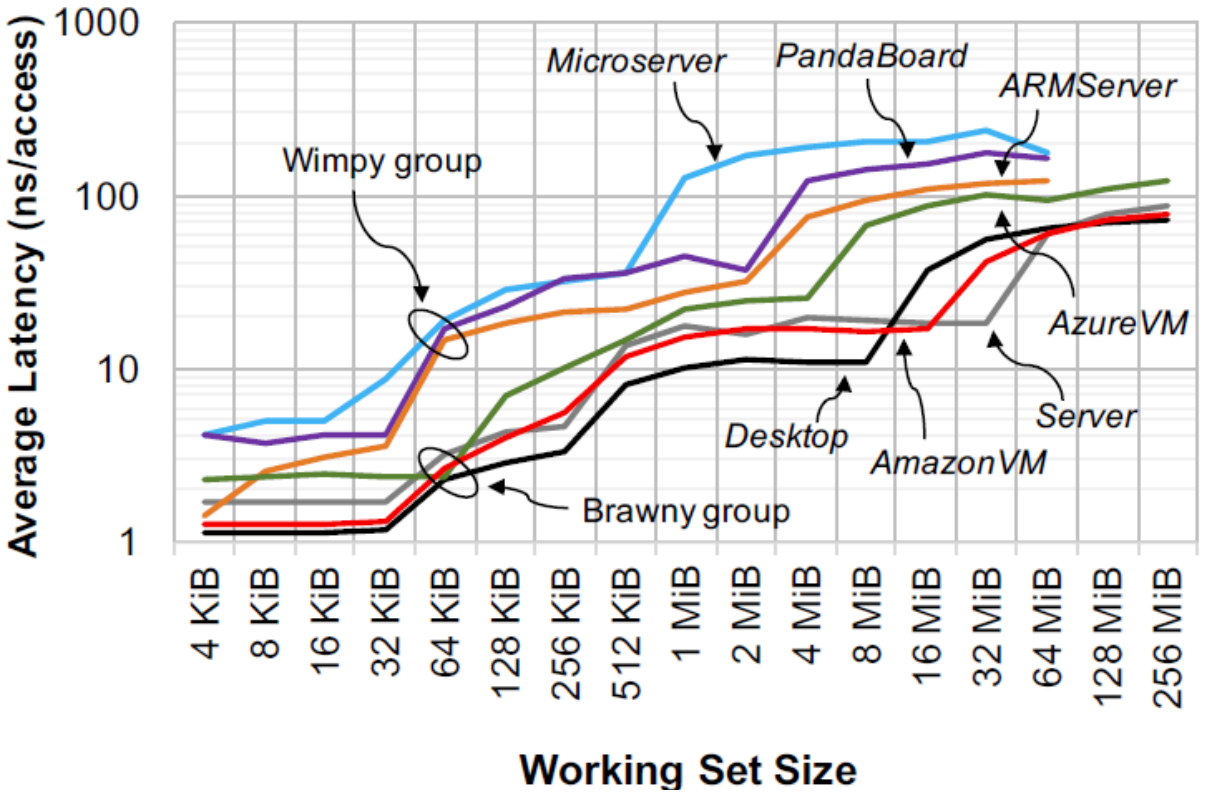


Case Study 2:
Cross-Platform Insights for Cloud Subscribers

- Cloud subscribers can use X-Mem to directly compare memory performance of very different platforms
  - x86 vs. ARM instruction set
  - Virtual vs. physical machines
  - Wimpy vs. brawny hardware
  - Apples-to-apples results from one tool
- This capability enables subscibers to:
  - Choose a target cloud platform that best suits workload characteristics

Case Study 2:
Cross-Platform Insights for Cloud Subscribers

Apples-to-apples comparison of cache hierarchy performance across seven different computing platforms



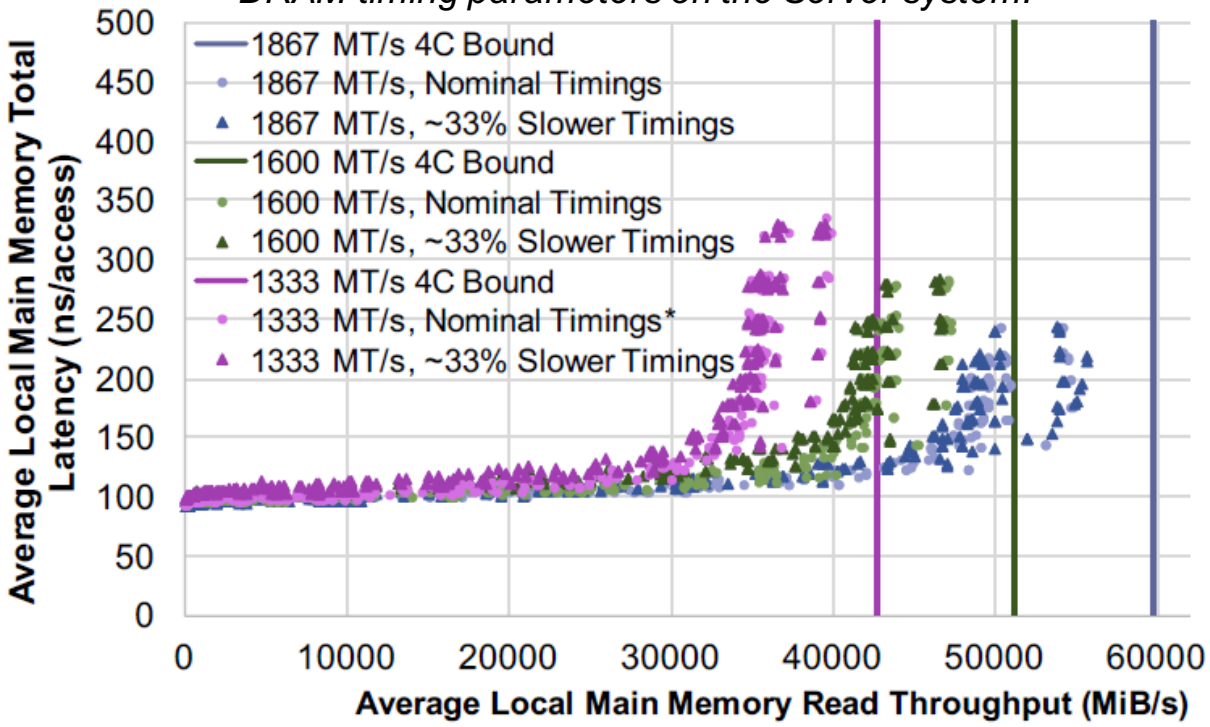
Case Study 3:
Impact of Tuning Platform Configurations for Cloud Providers

- Cloud providers can use X-Mem to evaluate the sensitivity of system-level performance and energy to memory configurations
  - Number of DRAM channels, DPC, RPD, channel frequency
  - DRAM timing parameters – variation-aware memory tuning? [Gottscho ESL'12, CODES+ISSS'12, TC'15, Chandrasekar DATE'14, Lee HPCA'15]
  - Analyze throughput, unloaded and loaded latency, different access patterns, etc.

- This capability enables providers to:
  - Optimally configure their platforms for different types of workloads
  - Maximize performance/$, minimize TCO, etc.

Case Study 3:
Impact of Tuning Platform Configurations for Cloud Providers

Sensitivity of system-level main memory performance to DRAM timing parameters on the Server system.



Summary

X-Mem is a flexible tool for characterizing memory systems

Surpasses capabilities of all prior tools

Several key features enable broad usability

(A) Access pattern diversity, (B) Platform variability, (C) Metric flexibility, (D) Tool extensibility

Case studies for cloud subscribers and providers

Showed how X-Mem can help optimize application perf., choose optimal platforms, and provision/configure HW for low cost

CONTACT:
http://seas.ucla.edu/~gottscho

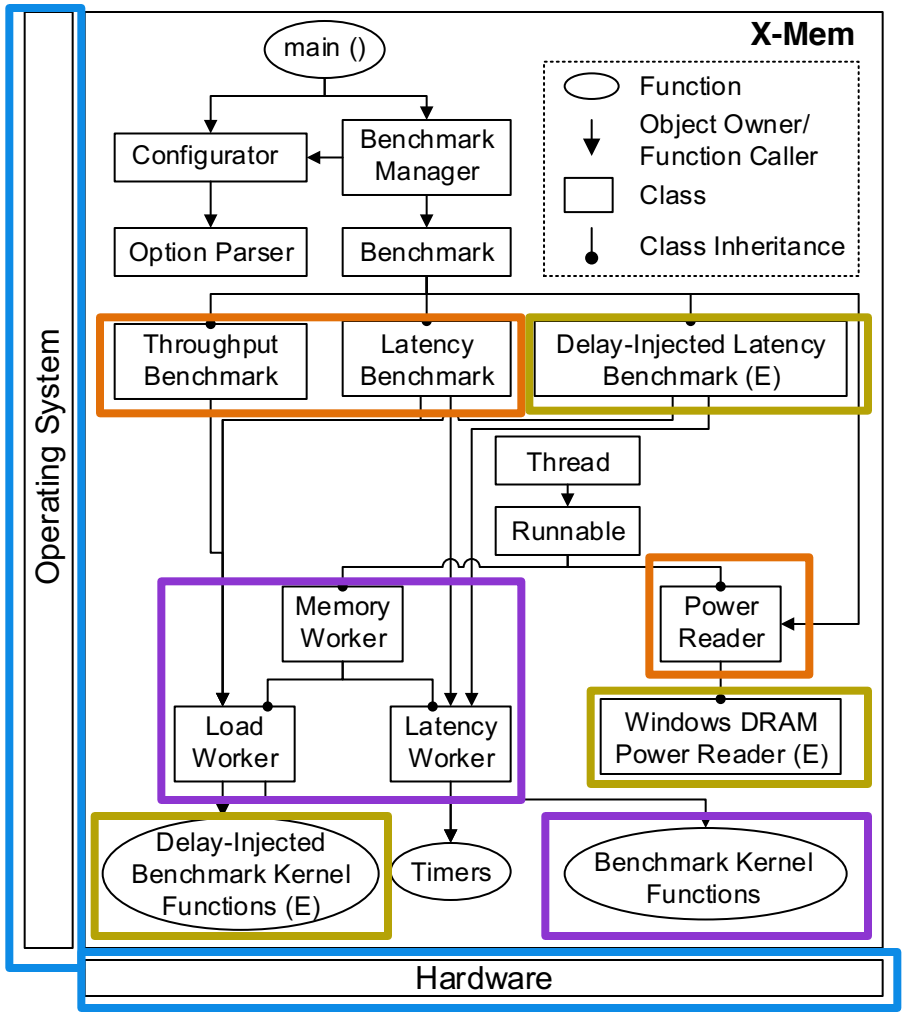
THANK YOU!

Project homepage:
nanocad-lab.github.io/X-Mem
Source code:
github.com/Microsoft/X-Mem

BACKUP SLIDES

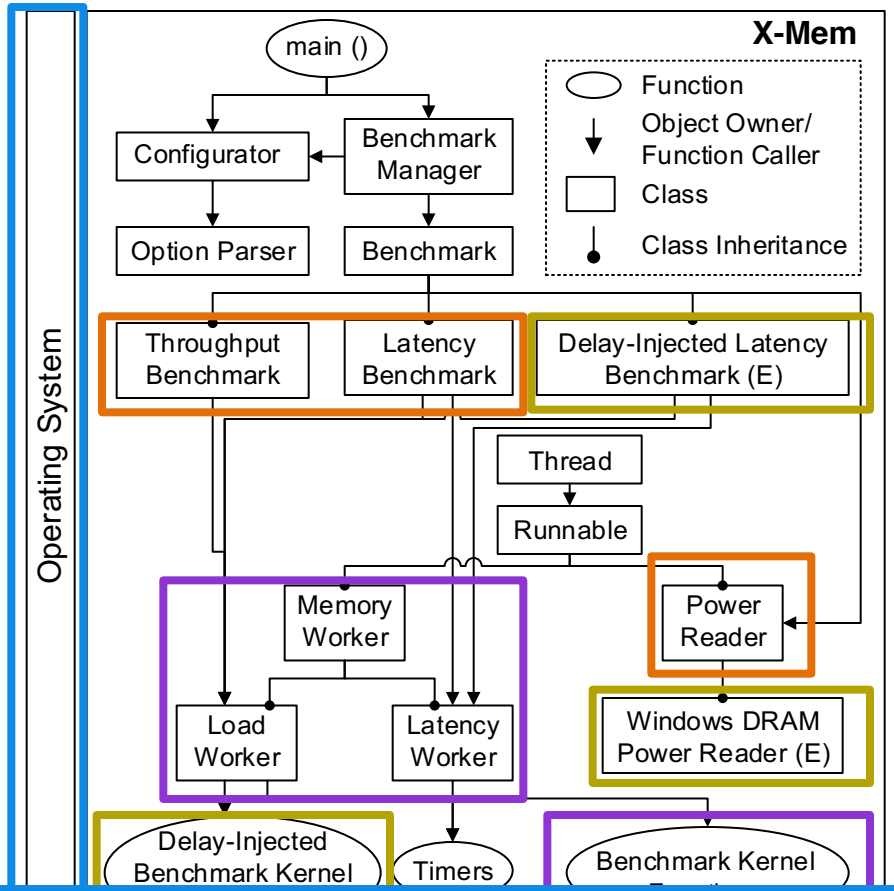
X-Mem: A New Memory Characterization Tool

- Object-oriented C++
- Caches through DRAM
- (A) Access pattern diversity
- (B) Platform variability
- (C) Metric flexibility
- (D) Tool extensibility
- Open-source
- User-friendly CLI & documentation



X-Mem: A New Memory Characterization Tool

- Object-oriented C++
- Caches through DRAM
- (A) Access pattern diversity
- (B) Platform variability
- (C) Metric flexibility
- (D) Tool extensibility
- Open-source
- User-friendly CLI & documentation

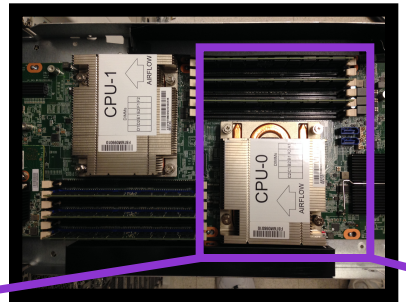


Latest SW & documentation available @ nanocad-lab.github.io/X-Mem

Tool Comparison

Tool	Thru-put	Lat.	Loaded Lat.	Multi-Thrd.	NUMA	Lrg. Pages	Power	Cache & Mem.	Native Linux	Native Win.	x86	x86-64	ARM	Vector Inst.	Open Src.	Lang.	(A) Acc. Patt. Divers.	(B) Platf. Var.	(C) Metric Flex.	(D) Tool Extens.
STREAM v5.10 [13]	✓			❍				❍	✓		✓	❍	❍		✓	C, FORTRAN				
STREAM2 v0.1 [14]	✓			❍				❍	✓		✓	❍	❍		✓	FORTRAN				
lmbench3 [15]	✓	✓		✓				❍	✓		✓	❍	❍		✓	C				
TinyMemBench v0.3.9 [16]	✓	✓				✓		✓	✓		✓	❍	❍	✓	✓	C				
mlc v2.3 [17]	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓								
X-Mem v2.2.3 [18], [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	C++	✓	✓	✓	✓

Main Memory System



NUMA Node 0

CPU Core
L1I$
L1D$
L2$
Shared L3$
Memory Controller(s)
Main Memory

DIMM (Socketed)
DRAM Chip
8b
Bank
(Independent, Lockstep within rank)
Array
Rows
Row Buffer
Columns
Location
Data (8 1T1C Bitcells)
Memory Controller
64b DATA (+8b ECC parity)
Channels
Rank (Lockstep group of DRAMs)
Channel: CMD, ADDR, CLK, RANK_SEL
(Shared among all ranks and DRAMs in the channel)

Breakdown of Main Memory

X-Mem (v2.2.3) Command-Line Interface

Option	Args.	Description
-a or --all		Use all benchmark kernels.
-c or --chunk_size†	32, 64*, 128, 256	Memory access width for load kernels.
-e or --extension†	0, 1, ...	Run extended mode with given index.
-f or --output_file	FILE.csv	Dump benchmark results to a CSV file.
-h or --help		Print X-Mem usage and exit.
-i or --base_test_index	0*, 1, ...	Base of enumeration for benchmarks.
-j or --num_worker_threads	1*, 2, ...	Total number of threads to use.
-l or --latency*		Run (un)loaded latency benchmarks.
-n or --iterations	1*, 2, ...	Number of iterations per benchmark.
-r or --random_access		Use random-access load kernels.
-s or --sequential_access		Use seq. or strided-access load kernels.
-t or --throughput*		Run throughput benchmarks.
-u or --ignore_numa		Only test NUMA node 0.
-v or --verbose		Enable verbose standard output.
-w or --working_set_size	4*, 8, ...	Per-thread array size in KiB.
-L or --large_pages		Use OS-defined large pages.
-R or --reads*		Use read accesses for load kernels.
-W or --writes*		Use write accesses for load kernels.
-S or --stride_size†	±1*, 2, 4, 8, 16	Stride length in multiples of chunk size. Only applies to load kernels with -s.

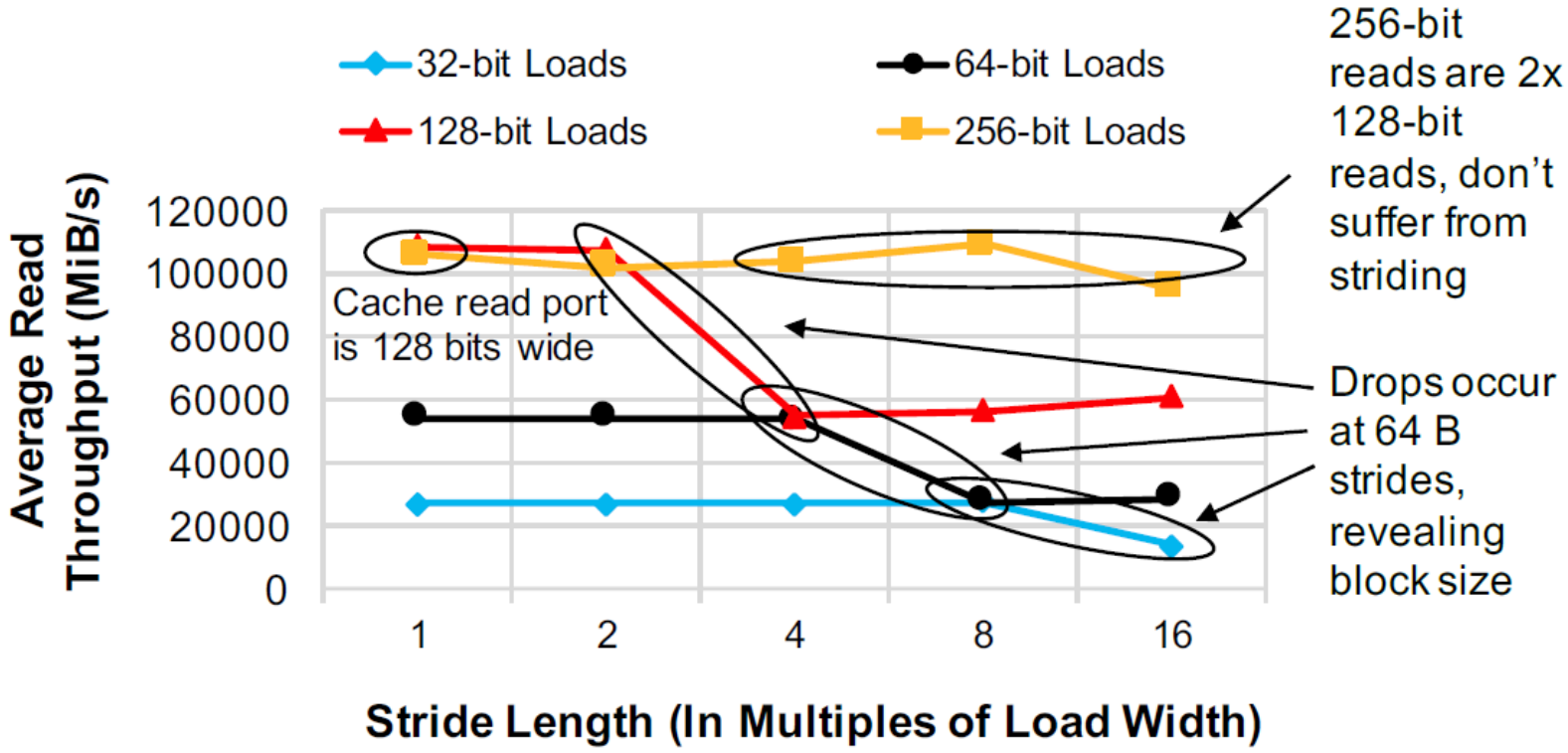
Experimental Platform Details

System Name	ISA	CPU	No. Cores	CPU Freq.	L1$	L2$	L3$	$ Blk.	Process	OS	NUMA	ECC
Desktop	x86-64 w/ AVX	Intel Core i7-3820 (Sandy Bridge-E)	4	3.6 GHz*, 1.2 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 10 MiB, 20-way	64 B	32 nm	Linux		
Server	x86-64 w/ AVX2	Dual Intel Xeon E5-2600 v3 series (Haswell-EP)	12 per CPU	2.4 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 30 MiB, 20-way	64 B	22 nm	Win.	✓	✓
Microserver	x86-64	Intel Atom S1240 (Centerton)	2	1.6 GHz	split, private, 24 KiB 6-way data, 32 KiB 8-way inst.	private, 512 KiB, 8-way	-	64 B	32nm	Win.		✓
PandaBoard (ES)	ARMv7-A w/ NEON	TI OMAP 4460 (ARM Cortex-A9)	2	1.2 GHz	split, private, 32 KiB, 4-way	shared, 1 MiB	-	32 B	45 nm	Linux		
AzureVM	x86-64	AMD Opteron 4171 HE	4	2.1 GHz	split, private, 64 KiB, 2-way	private, 512 KiB, 16-way	shared, 6 MiB, 48-way	64 B	45 nm	Linux		✓
AmazonVM	x86-64 w/ AVX2	Intel Xeon E5-2666 v3 (Haswell-EP)	4	2.9 GHz	split, private, 32 KiB, 8-way	private, 256 KiB, 8-way	shared, 25 MiB, 20-way	64 B	22 nm	Linux		✓
ARMServer	ARMv7-A	Marvell Armada 370 (ARM Cortex-A9)	4	1.2 GHz	split, private, 32 KiB, 4/8-way (I/D)	private, 256 KiB, 4-way	-	32 B	?	Linux		?

System Name	Config. Name	Memory Type	No. Channels	DPC	RPD	DIMM Capacity	Chan. MT/s	nCAS - clk (tCAS - ns)	nRCD - clk (tRCD - ns)	nRP - clk (tRP - ns)	nRAS - clk (tRAS - ns)
*Desktop**	*1333 MT/s, Nominal Timings 4C*	DDR3 U	4	2	2	2 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	11 (16.5 ns)	24 (36.0 ns)
Desktop	*1333 MT/s, ≈33% Slower Timings 4C*	DDR3 U	4	2	2	2 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	15 (22.5 ns)	32 (48.0 ns)
Desktop	*800 MT/s, Nominal Timings 4C*	DDR3 U	4	2	2	2 GiB	800	7 (17.5 ns)	7 (17.5 ns)	8 (20.0 ns)	16 (40.0 ns)
Desktop	*800 MT/s, ≈33% Slower Timings 4C*	DDR3 U	4	2	2	2 GiB	800	10 (25.0 ns)	10 (25.0 ns)	11 (27.5 ns)	22 (55.0 ns)
Desktop	*1333 MT/s, Nominal Timings 1C*	DDR3 U	1	2	2	2 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	11 (16.5 ns)	24 (36.0 ns)
Desktop	*1333 MT/s, ≈33% Slower Timings 1C*	DDR3 U	1	2	2	2 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	15 (22.5 ns)	32 (48.0 ns)
Desktop	*800 MT/s, Nominal Timings 1C*	DDR3 U	1	2	2	2 GiB	800	7 (17.5 ns)	7 (17.5 ns)	8 (20.0 ns)	16 (40.0 ns)
Desktop	*800 MT/s, ≈33% Slower Timings 1C*	DDR3 U	1	2	2	2 GiB	800	10 (25.0 ns)	10 (25.0 ns)	11 (27.5 ns)	22 (55.0 ns)
*Server**	*1333 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1333	9 (13.5 ns)	9 (13.5 ns)	9 (13.5 ns)	24 (36.0 ns)
Server	*1333 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1333	12 (18.0 ns)	12 (18.0 ns)	12 (18.0 ns)	32 (48.0 ns)
Server	*1600 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1600	11 (13.75 ns)	11 (13.75 ns)	11 (13.75 ns)	29 (36.25 ns)
Server	*1600 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1600	15 (18.75 ns)	15 (18.75 ns)	15 (18.75 ns)	38 (47.5 ns)
Server	*1867 MT/s, Nominal Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1867	13 (13.92 ns)	13 (13.92 ns)	13 (13.92 ns)	34 (36.42 ns)
Server	*1867 MT/s, ≈33% Slower Timings*	DDR3 R	4 per CPU	1	2	16 GiB	1867	18 (19.28 ns)	18 (19.28 ns)	18 (19.28 ns)	46 (49.27 ns)

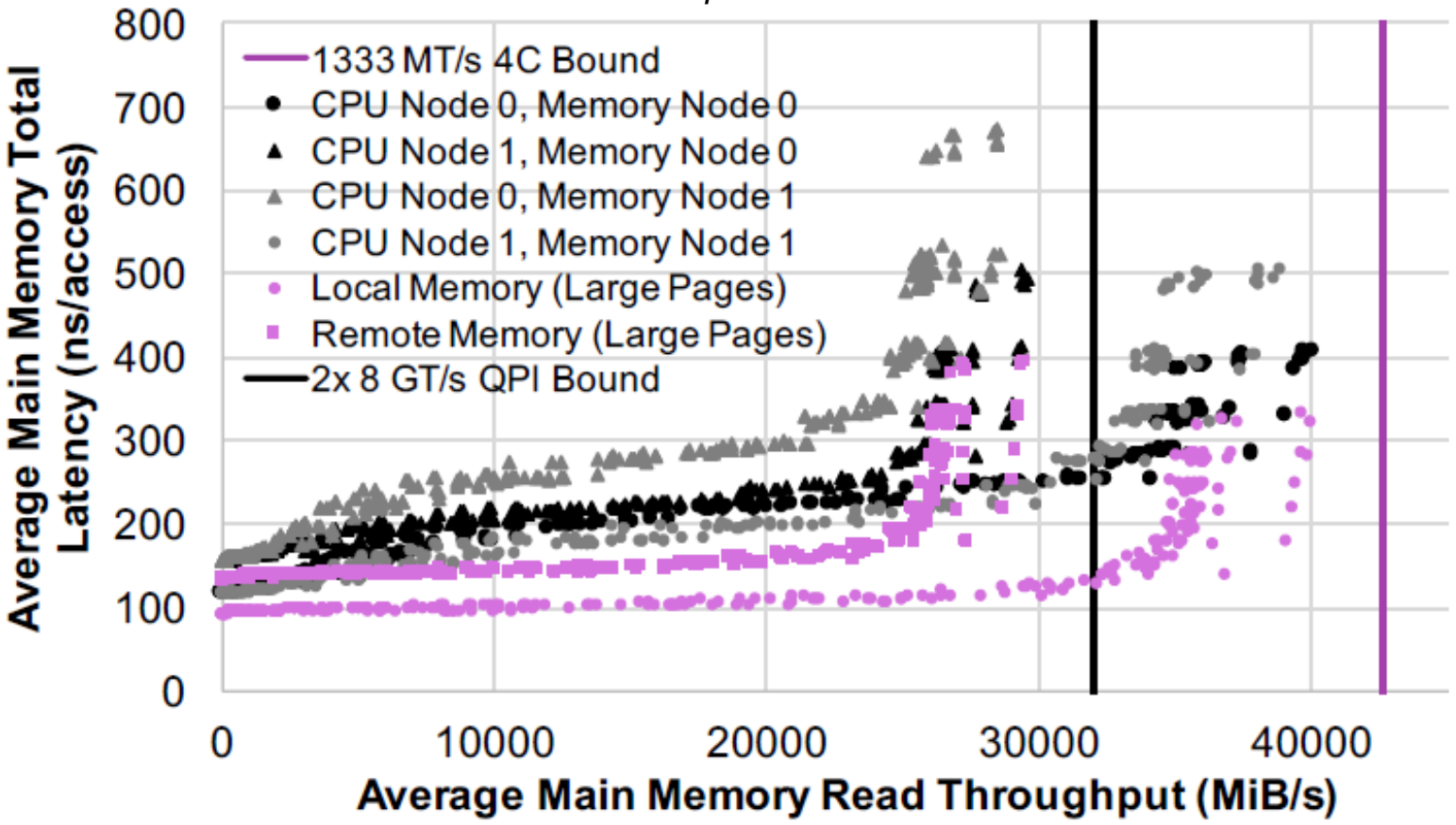
Case Study 1 (cont.):
Characterization of the Memory Hierarchy for Cloud Subscribers

Single-thread strided L1D read throughput for the Desktop platform.



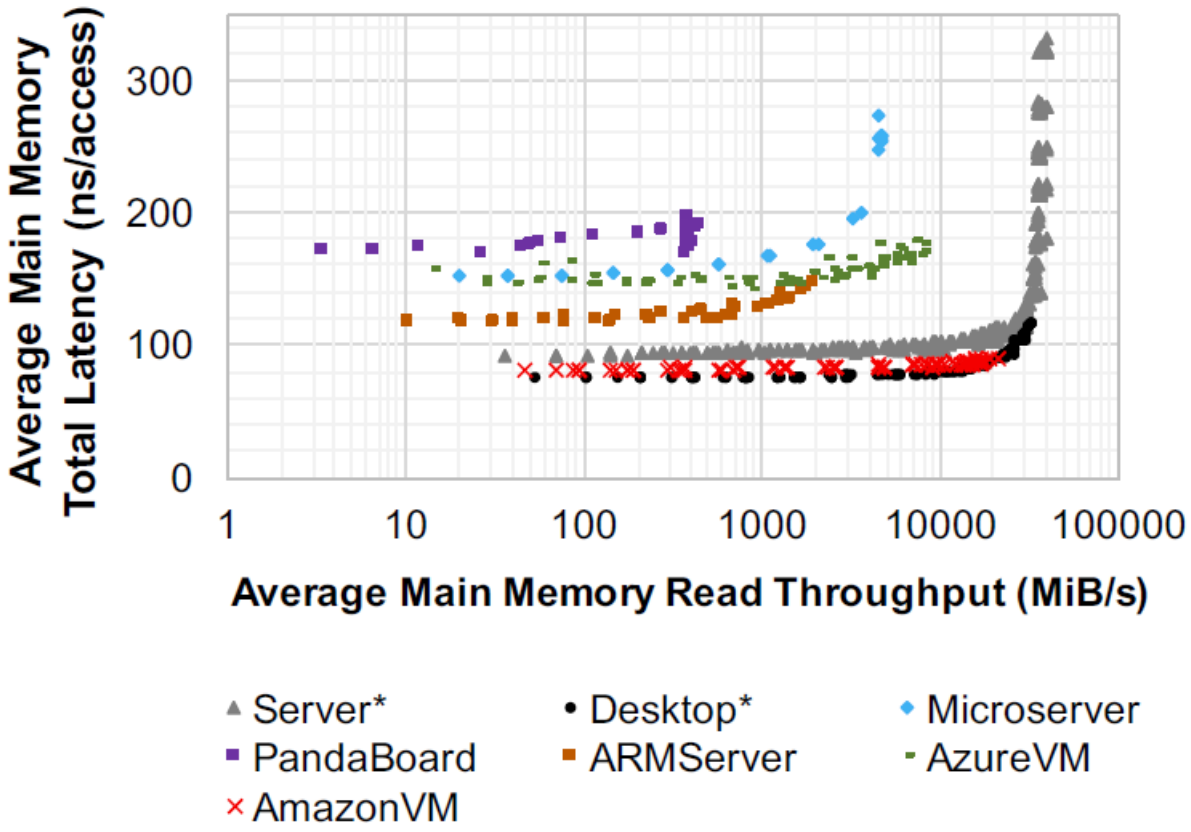
Case Study 1 (cont.):
Characterization of the Memory Hierarchy for Cloud Subscribers

Relationship between NUMA and page size on the Server platform.



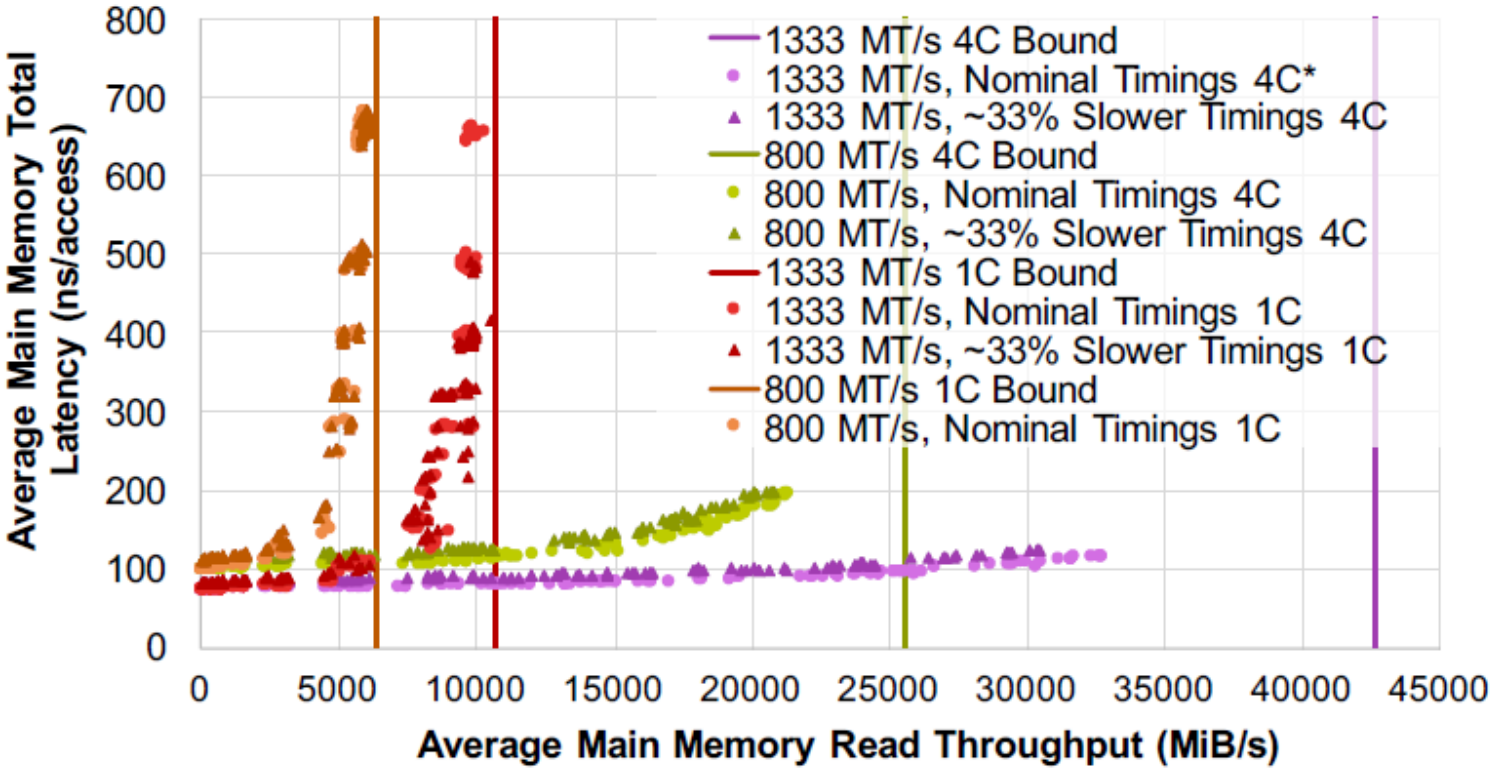
Case Study 2 (cont.):
Cross-Platform Insights for Cloud Subscribers

Apples-to-apples comparison of main memory performance across seven different computing platforms



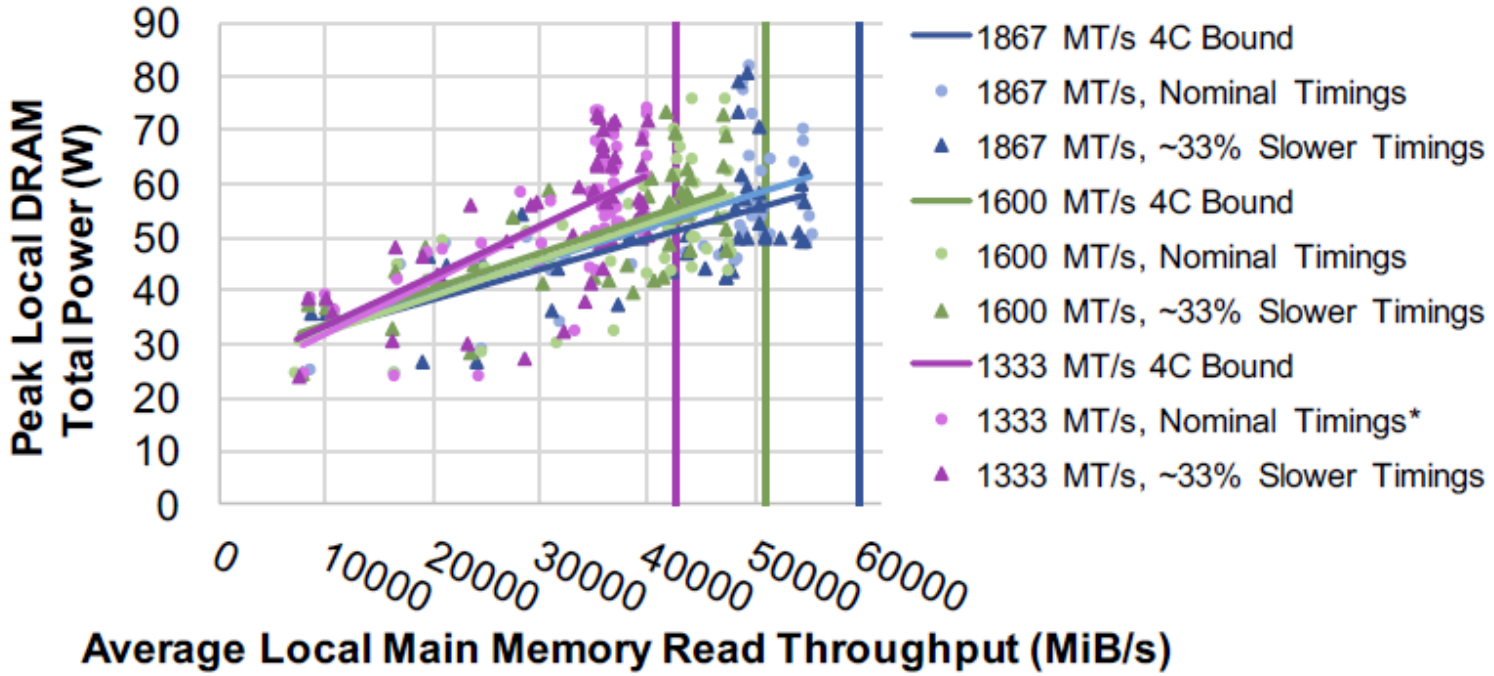
Case Study 3 (cont.):
Impact of Tuning Platform Configurations for Cloud Providers

Sensitivity of system-level main memory performance to DRAM timing parameters on the Desktop system @ 3.6 GHz.



Case Study 3 (cont.):
Impact of Tuning Platform Configurations for Cloud Providers

Impact of memory configuration on DRAM power in the Server platform.



Case Study 3 (cont.)

Impact of Tuning Platform Configurations for Cloud Providers

Remote access: Up to 45% slower

channels: no impact

Mem. Channel Frequency → / Platforms ↓ Timings →	*1867 MT/s* *Nom.*	*1867 MT/s* *≈ 33% Slow*	*1600 MT/s* *Nom.*	*1600 MT/s* *≈ 33% Slow*	*1333 MT/s* *Nom.*	*1333 MT/s* *≈ 33% Slow*	*800 MT/s* *Nom.*	*800 MT/s* *≈ 33% Slow*
***Server* (NUMA Local, Lrg. Pgs.)**	91.43	91.54	91.66	95.74	91.99*	97.61	-	-
***Server* (NUMA Remote, Lrg. Pgs.)**	126.51	128.54	129.62	139.25	133.59*	141.69	-	-
***Desktop 4C* @ 3.6 GHz**	-	-	-	-	73.33*	81.91	97.21	110.89
***Desktop 1C* @ 3.6 GHz**	-	-	-	-	72.38	80.94	97.36	109.56
***Desktop 4C* @ 1.2 GHz**	-	-	-	-	109.65	118.25	131.86	145.76
***Desktop 1C* @ 1.2 GHz**	-	-	-	-	108.44	117.09	131.85	144.46

Figure: Sensitivity of unloaded latency (ns/access) w.r.t. CPU & DDR3 frequency, DRAM timing, # DDR3 channels

CPU underclocked 3X: 50% higher DRAM lat.

DRAM timings 33% slower → up to 12% slower overall

Benchmarks are memory BW starved; *relative* impact of DRAM timings is LESS w/ more threads

Benchmark	Config.	1T	2T	3T	4T
canneal	*1333 MT/s 4C**	9.74%	9.02%	8.83%	8.89%
canneal	*800 MT/s 1C*	9.90%	9.29%	8.38%	7.83%
streamcluster	*1333 MT/s 4C**	11.14%	11.53%	11.82%	12.24%
streamcluster	*800 MT/s 1C*	8.10%	5.93%	2.63%	1.24%

Memory has enough BW; benchmarks appear latency-bound

Figure: Impact of 33% slower DRAM timings on memory-intensive PARSEC benchmarks with w.r.t # threads

Takeaway: Don't bother optimizing DRAM latency until bandwidth problem is solved!
→ Depends on relative balance of CPU/mem → Inconsistent with prior work [Meza DSN'15]