Asymmetry-aware execution placement on manycore chips

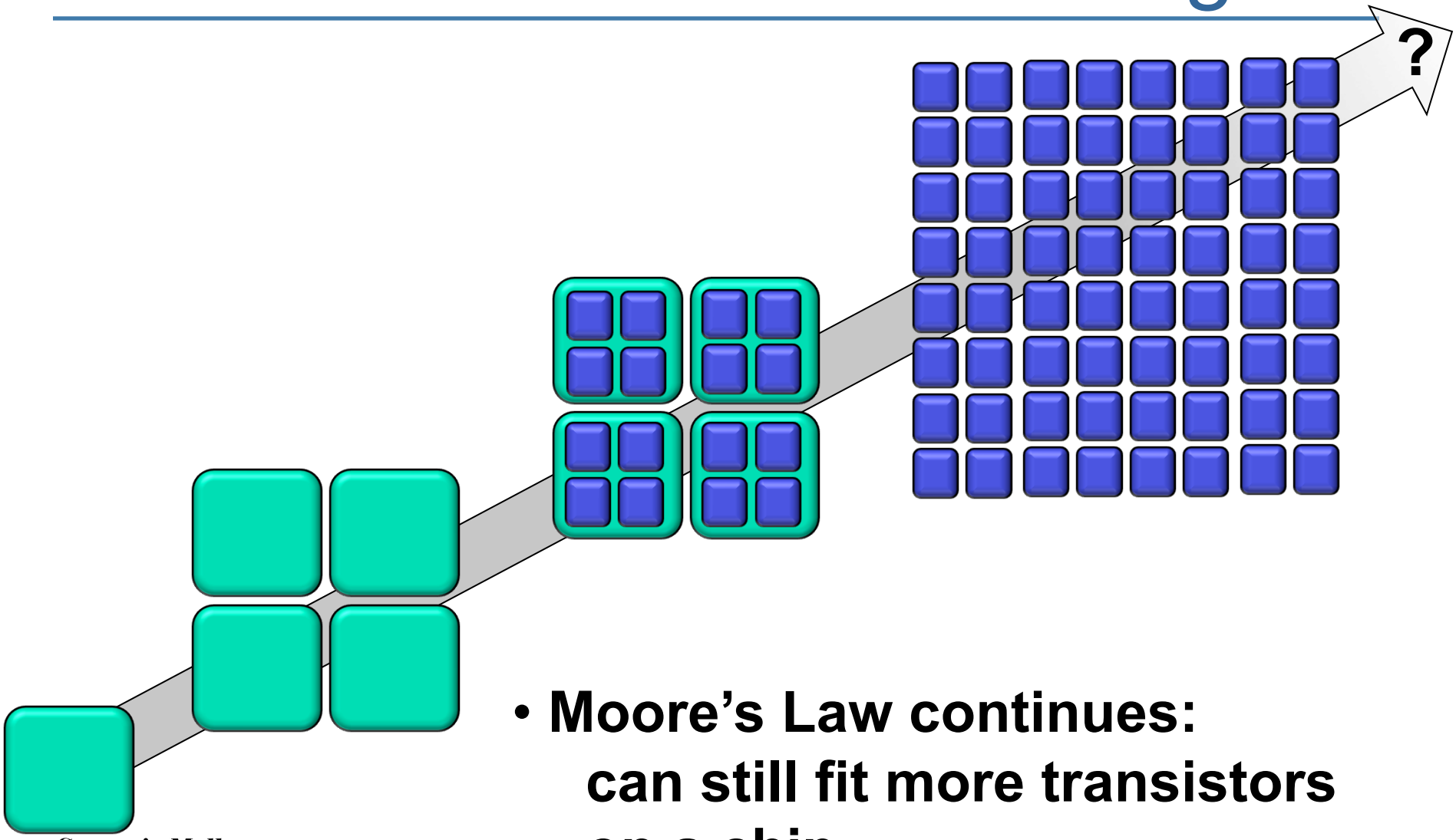
Alexey Tumanov

Joshua Wise, Onur Mutlu, Greg Ganger

CARNEGIE MELLON UNIVERSITY

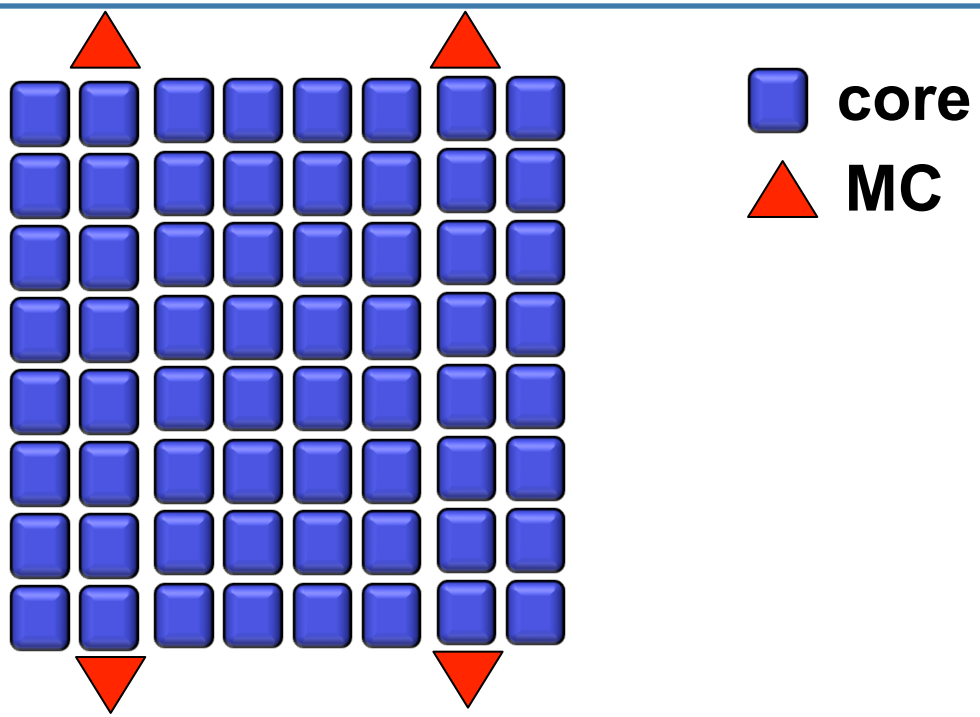
Carnegie Mellon
Parallel Data Laboratory

Introduction: Core Scaling



- **Moore's Law continues: can still fit more transistors on a chip**

Introduction: Memory Controllers

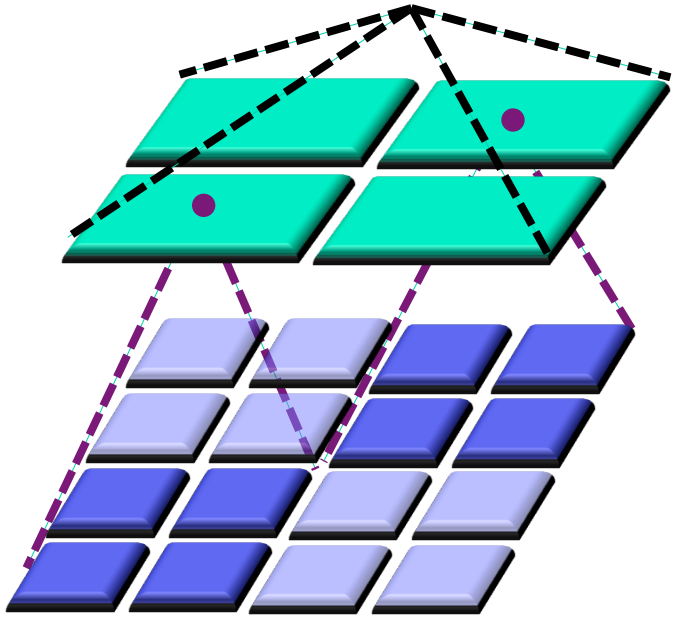


- Multi-core multi-MC chips gain prevalence
- But, now latency is NOT:
  - uniform memory access latency (UMA)
  - hierarchically non-uniform memory latency (NUMA)

NUMA

- NUMA: symmetrically non-uniform

crossbar

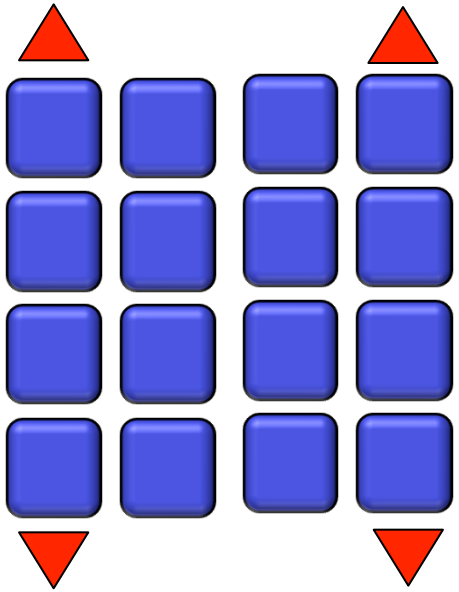


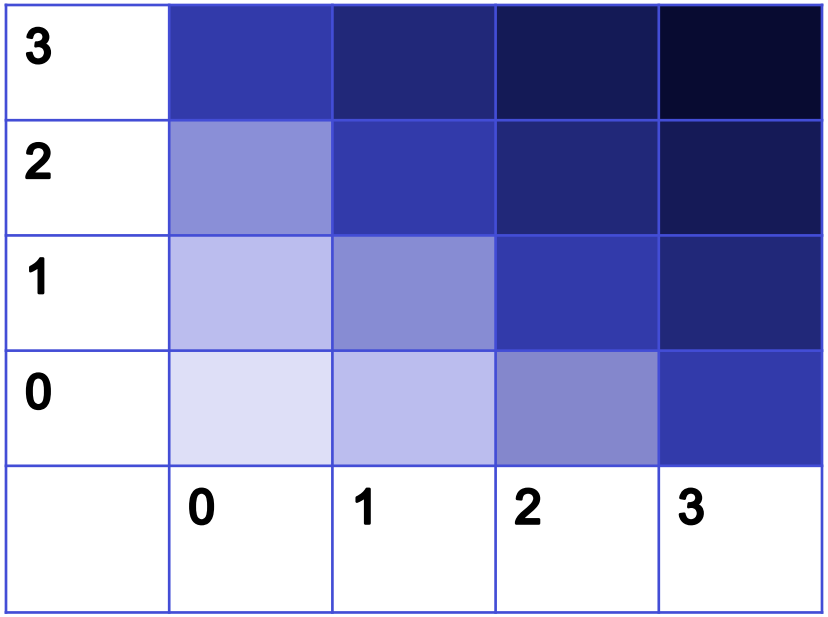
3	2x	2x	2x	2x
2	2x	2x	2x	2x
1	x	x	2x	2x
0	x	x	2x	2x
	0	1	2	3

bottom left MC access from all cores

ANUMA

- ANUMA: asymmetrically non-uniform





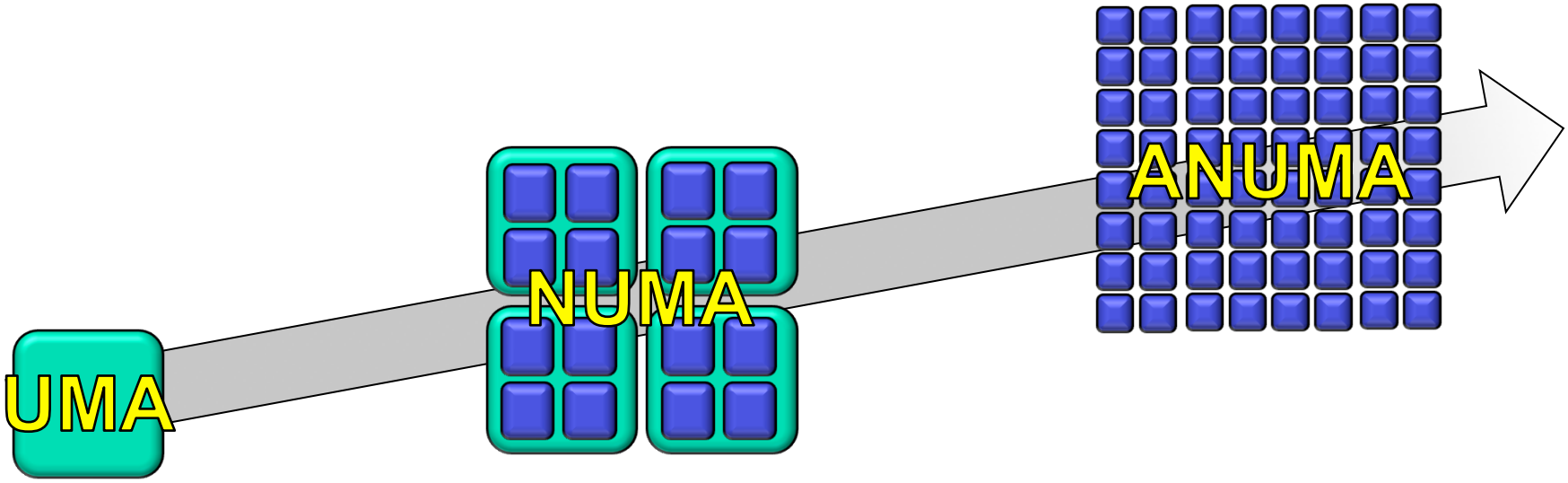
core

MC

bottom left MC access from all cores

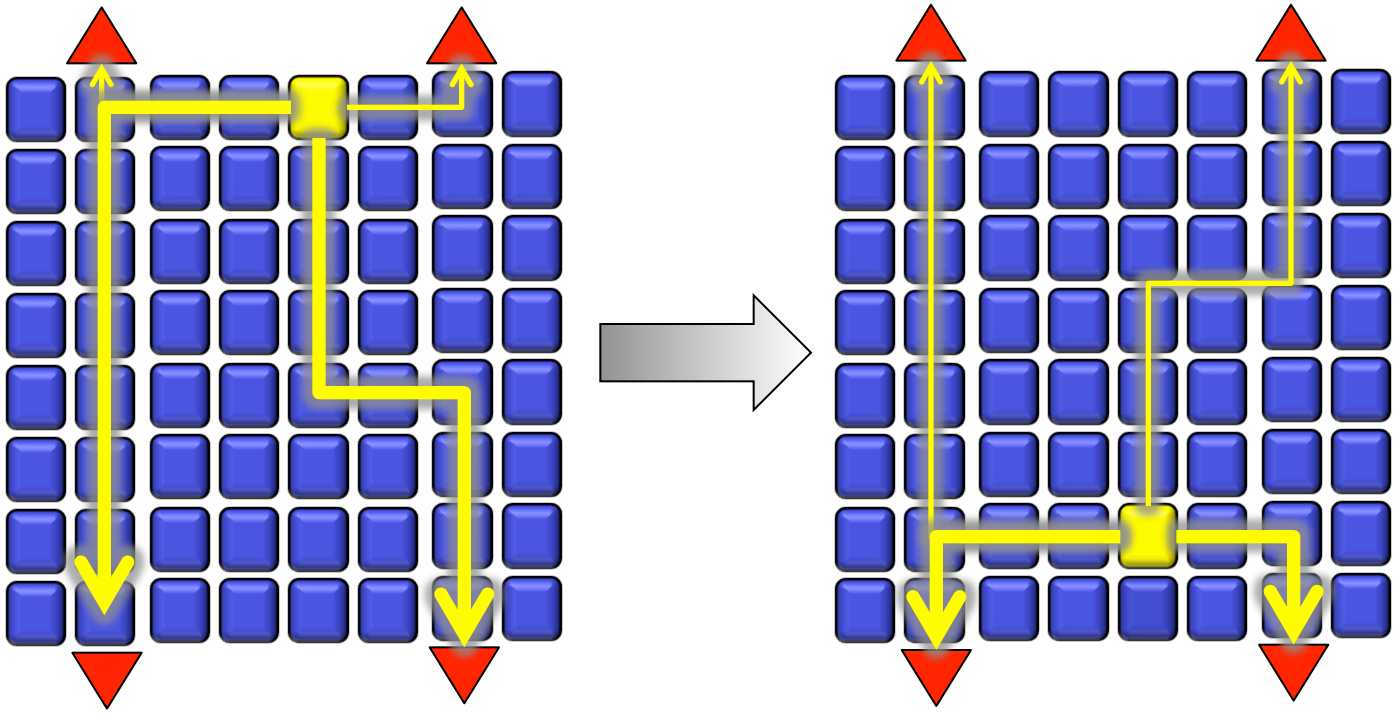
NUMA vs. ANUMA

- Latency & bandwidth asymmetries in multi-MC manycore NoC systems are different
  - [*numa*] sharp division into local/remote that dwarfs other intra-node asymmetries
  - [*anuma*] gradual continuum of access latencies



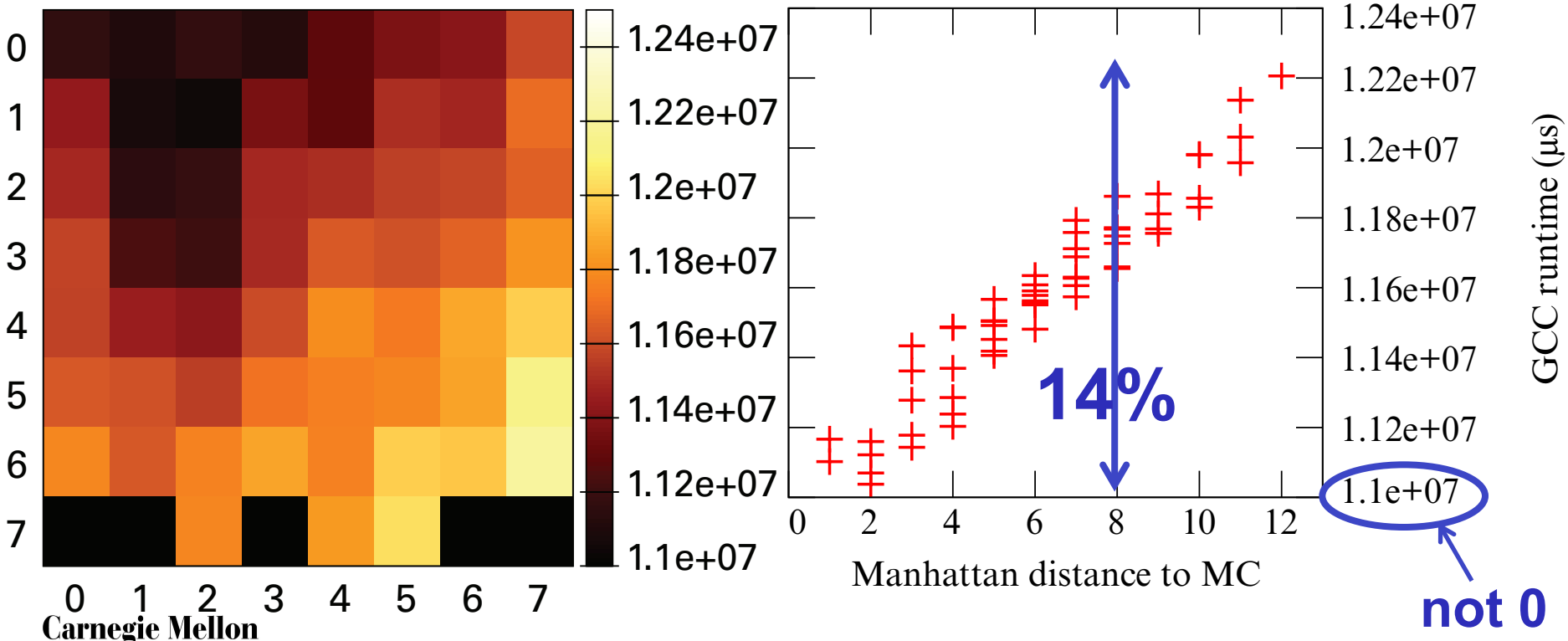
Problem Statement

- MC access asymmetries can result in suboptimal placement w.r.t. MC access patterns



Motivating Experiments

- "Bare metal" latency differential ~ 14%
  - does it matter?
- GCC: 1 thread, 1 MC, 1 core at a time, tile linux



Carnegie Mellon
Parallel Data Laboratory

Motivation

- Latency differential gets much worse
  - 14% figure is NOT an upper bound
- Major factor: contention on the NoC
  - NoC node and link contention slows down packets
  - amplifying effect on “bare metal” asymmetries
- Examples:
  - up to 100% latency differential in a simulator

Exploring Solution Space: NUMA

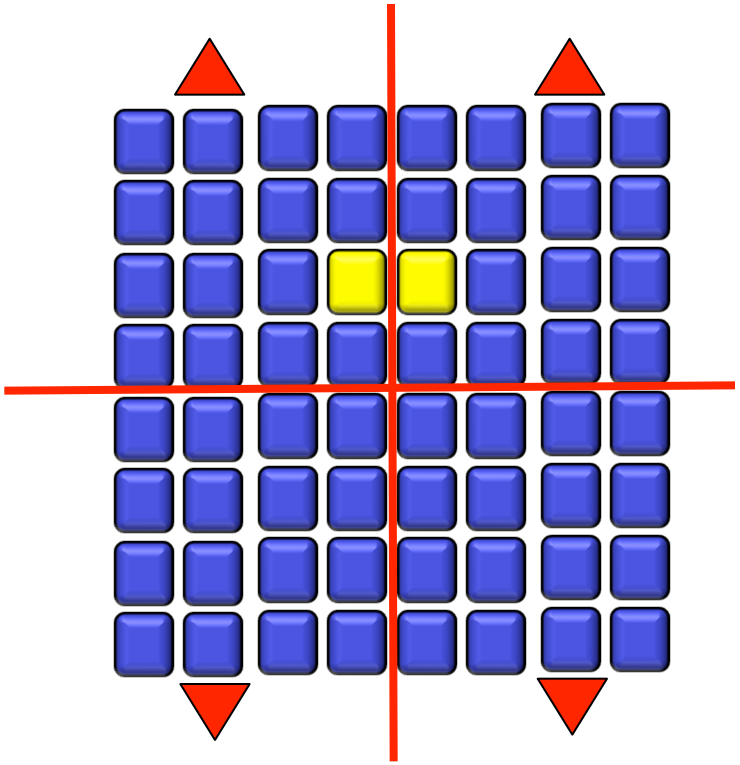
- Designate grid quadrants into NUMA zones

(+) a well-explored approach

(+) kernel code readily available

(-) lacks flexibility

(-) contention avoidance



Solution Space: data placement

- OS allocates physical frames in a way that:
  - balances the NoC contention
  - optimizes some objective function
  - maximizes throughput

(+) proactive

(-) no crystal ball: which pages will be hot?

(-) once allocated, can't be easily undone

  - inter-MC page migration worsens NoC contention

Solution Space: thread placement

- Move execution closer to MCs used

(-) waits for the problem to occur before acting

(-) cache thrashing

(+) better-informed decisions are possible

  - MC usage statistics collection

Solution Space: hybrid

- Combine execution and data placement
  - allocate frames, then adjust execution placement
  - place execution, then adjust data placement

(+) Corrective action possible

(-) Complexity

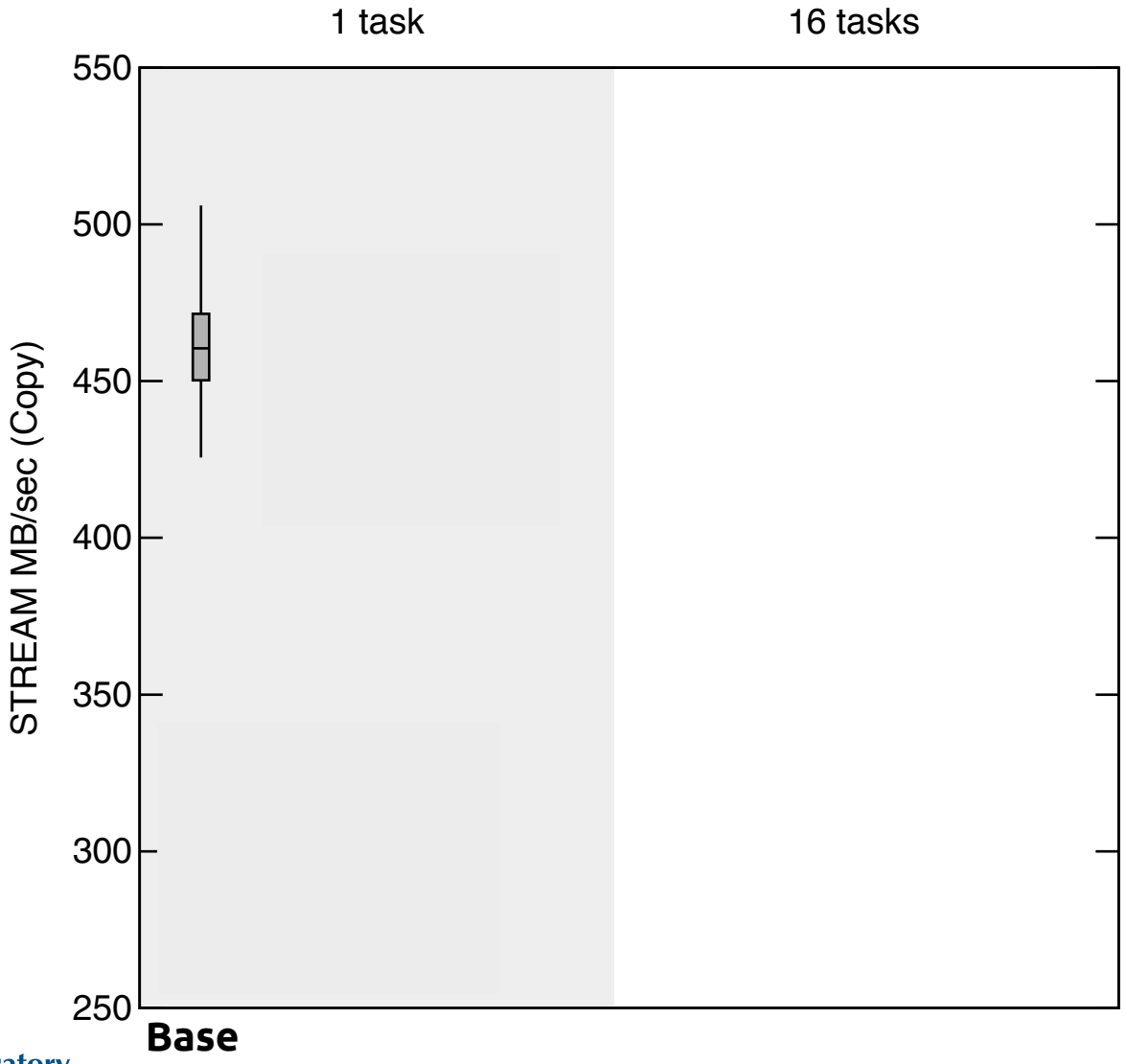
MC Usage Stats Collection

- Interface:
  - procfs control handle to start/stop collection
  - procfs data file for stats data extraction
    - pages accessed per MC per sampling cycle
- Implementation
  - instrumented VM to page fault
  - each page fault is attributed to corresponding MC_i
  - very inefficient, but does work
  - adjustable sampling frequency (20Hz)
    - trades off overhead vs. accuracy

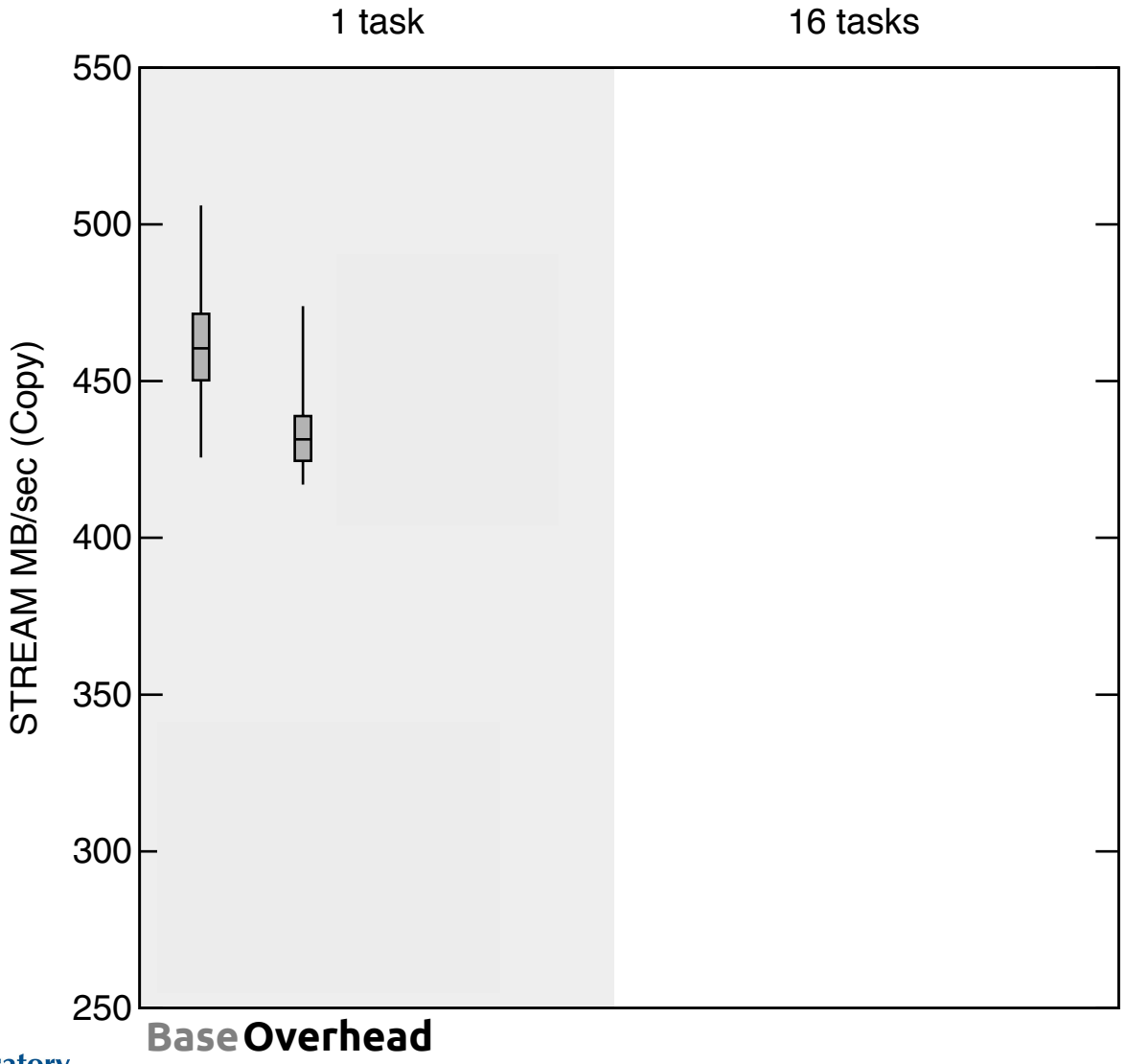
Placement Algorithm

- Place threads nearest to MCs they use
  - greedily based on memory intensity
- Weighted geometric placement algorithm
  - calculate thread's desired coordinates as a function of
    - MC coordinates
    - thread's access count vector (L1 normal)
- Placement collisions resolved
  - find second choices in direction of preferred MCs

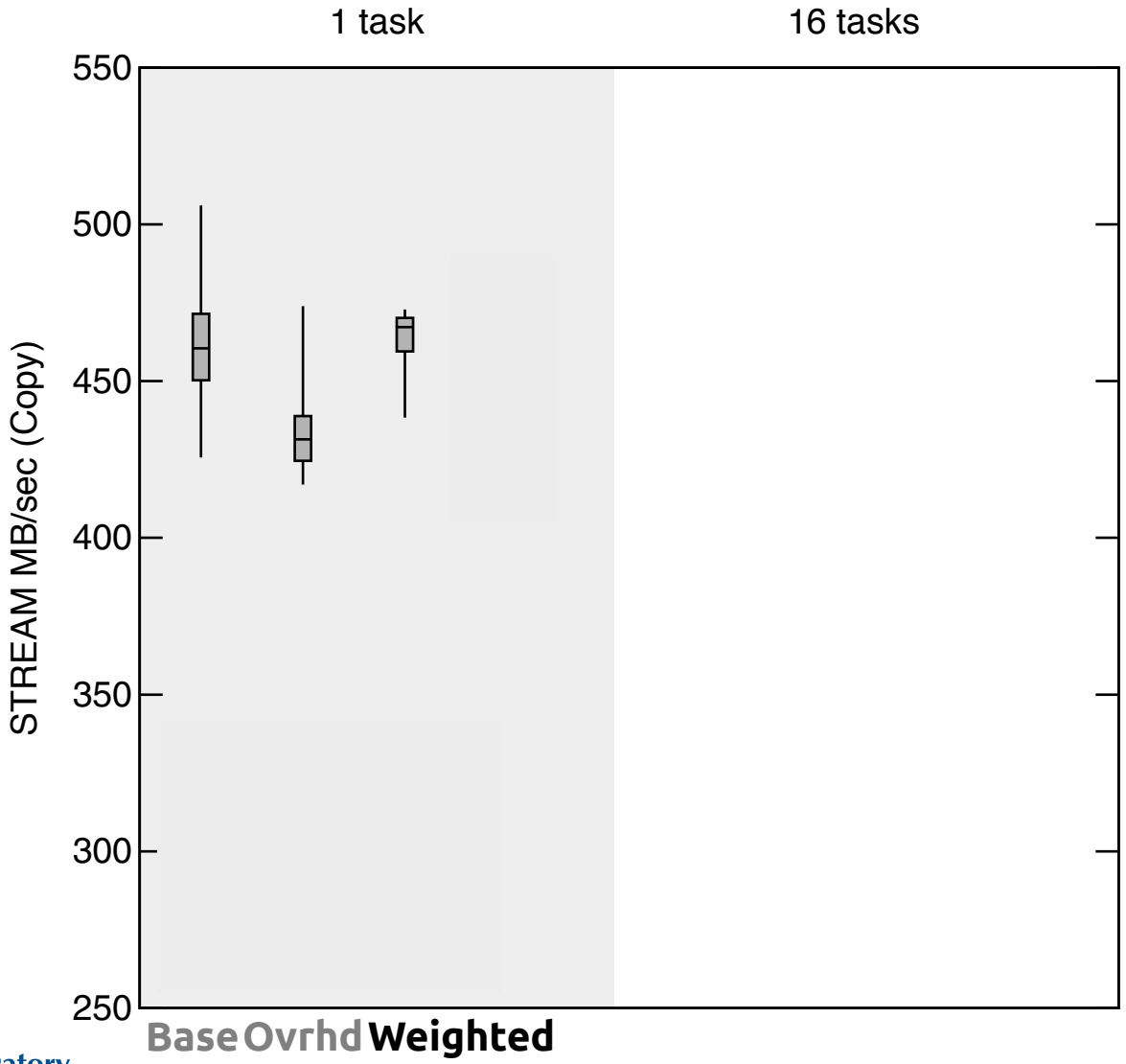
Bandwidth Performance Results



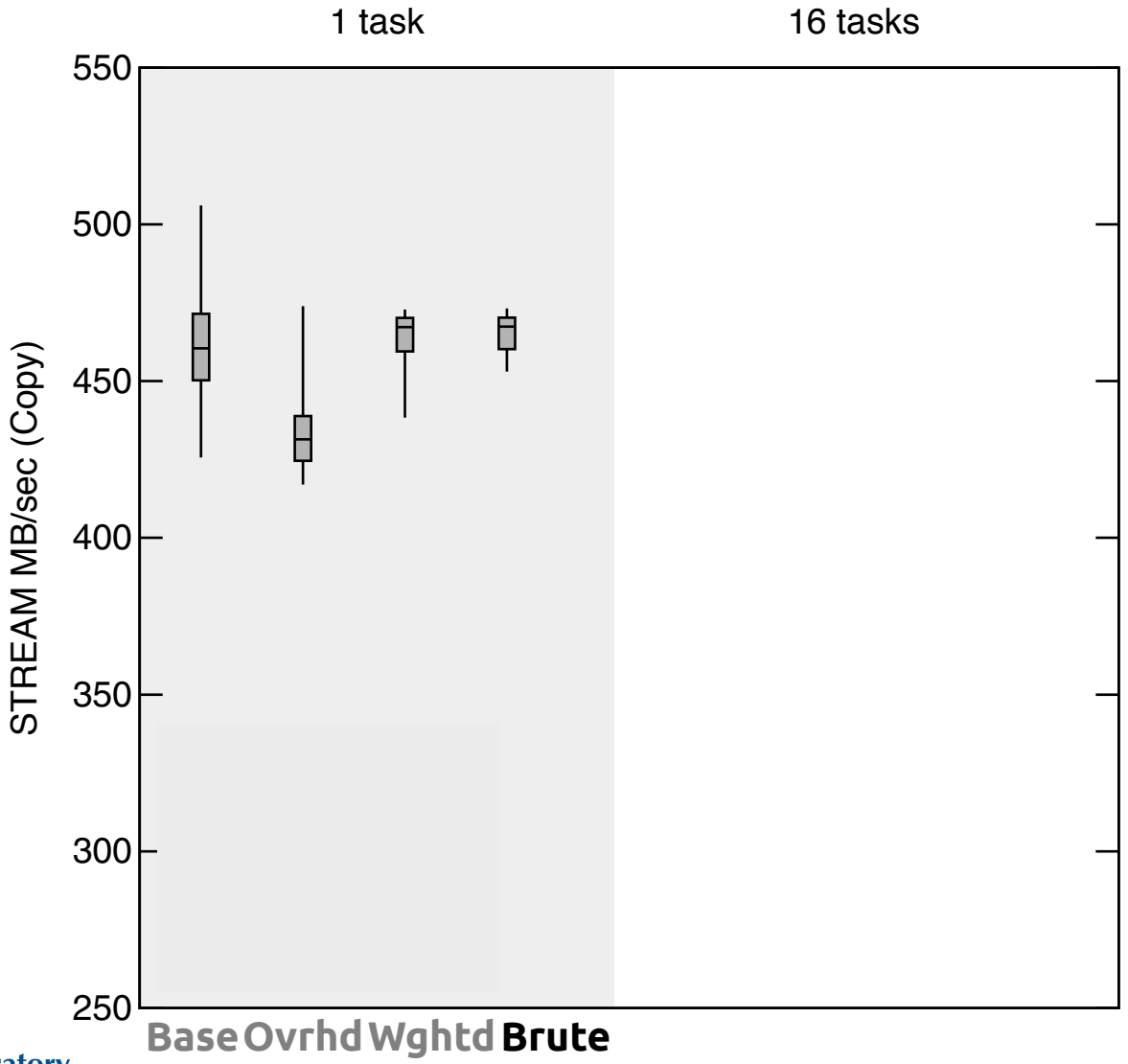
Bandwidth Performance Results



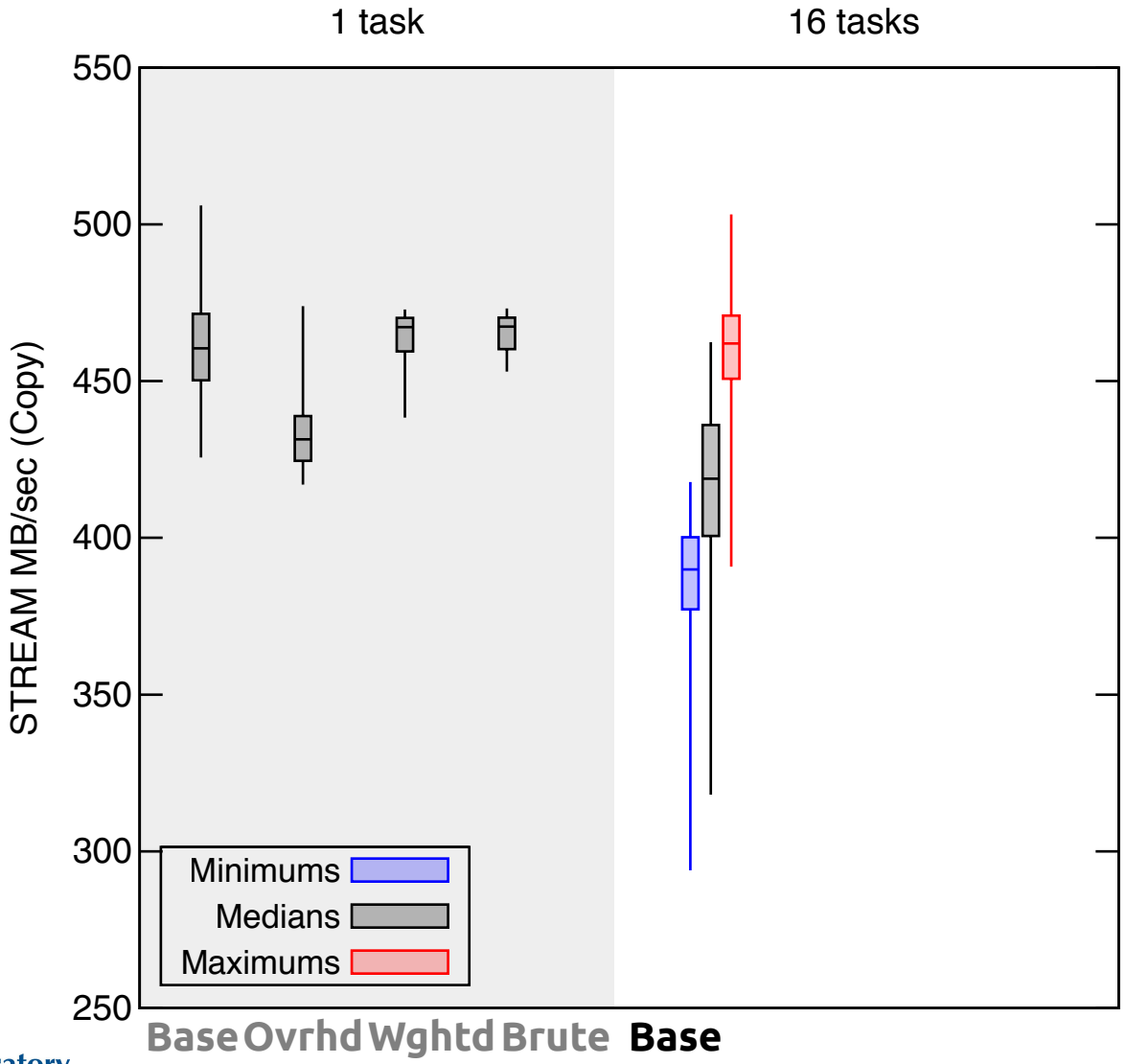
Bandwidth Performance Results



Bandwidth Performance Results

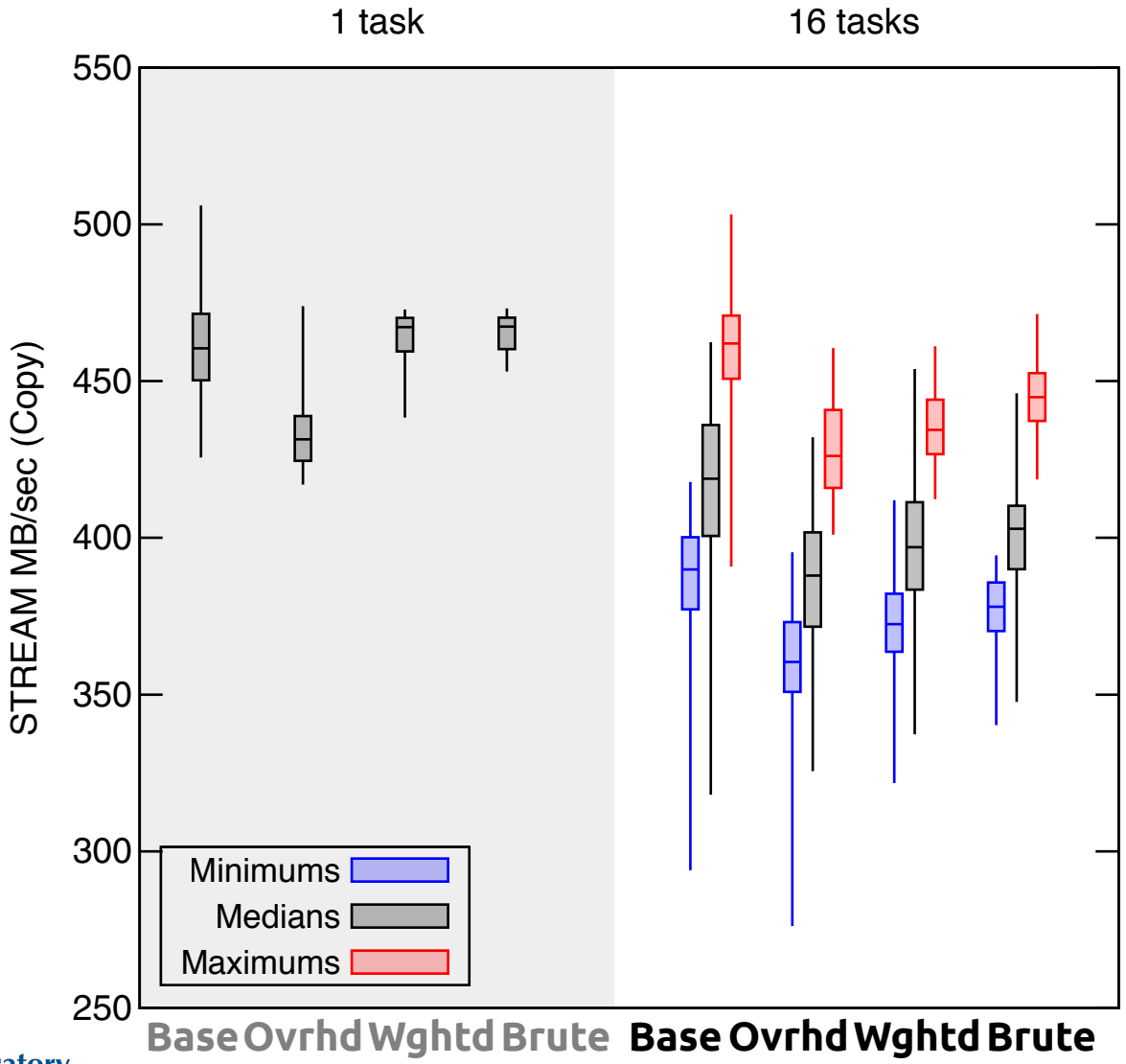


Bandwidth Performance Results

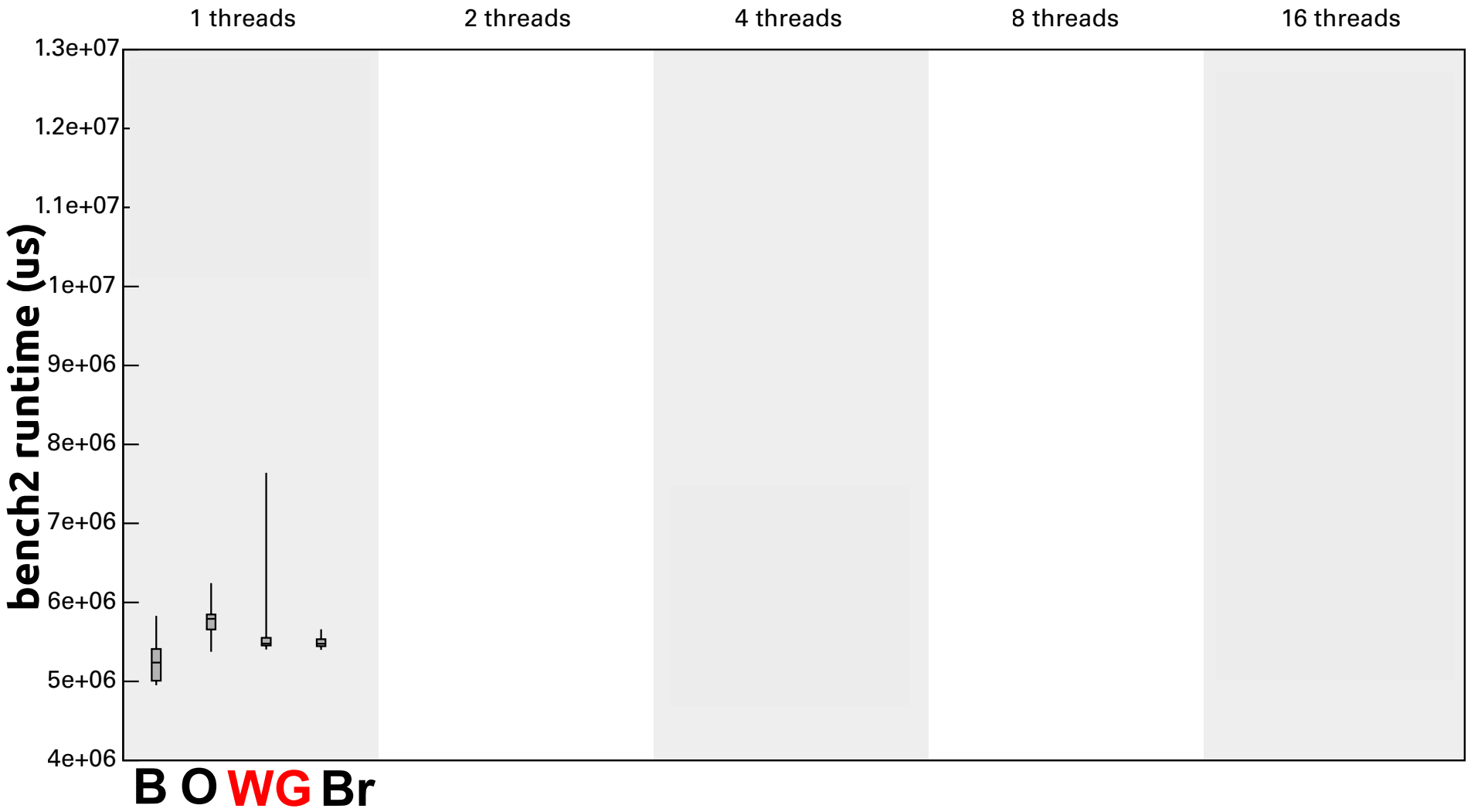


Carnegie Mellon
Parallel Data Laboratory

Bandwidth Performance Results



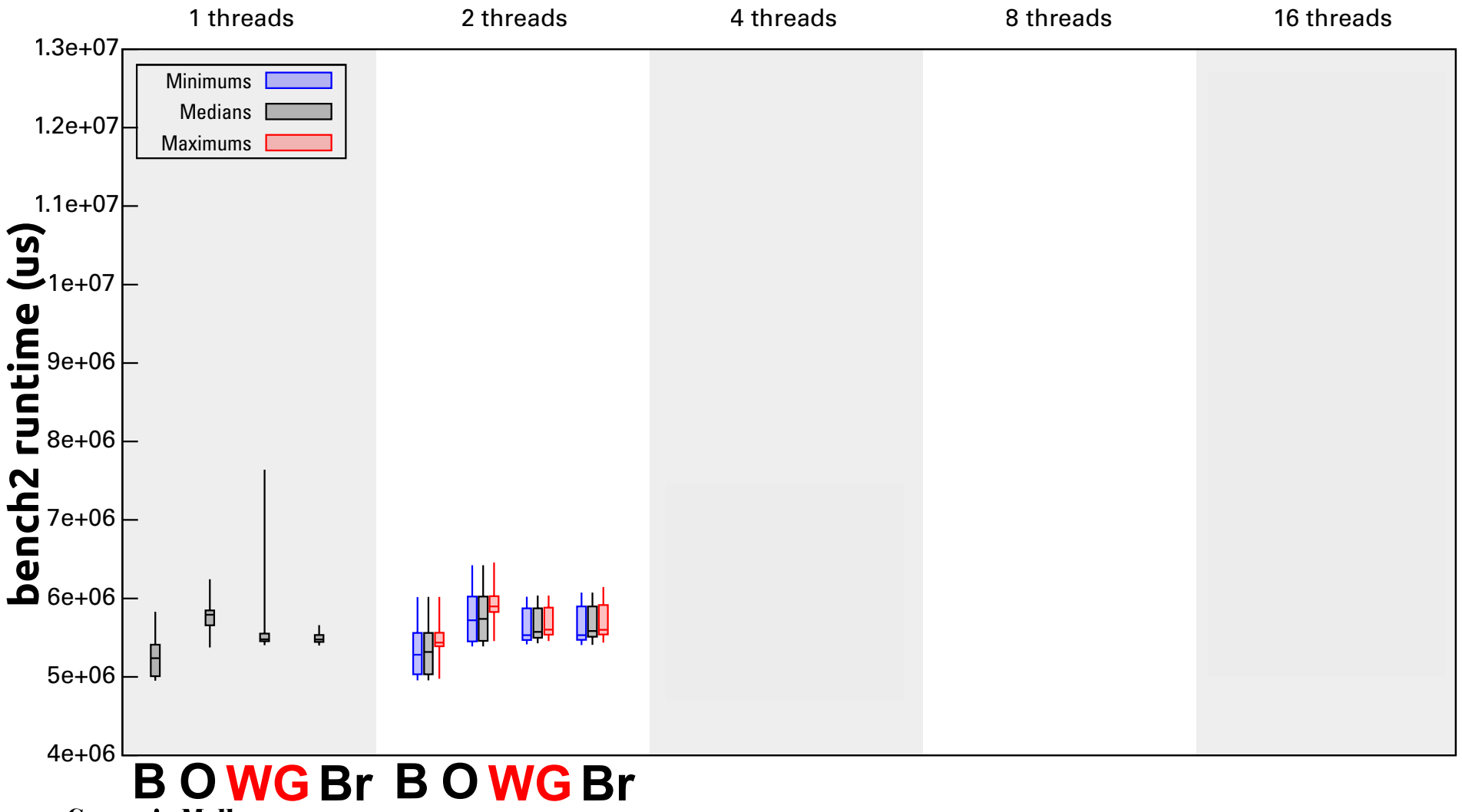
Runtime Performance Results



B O WG Br

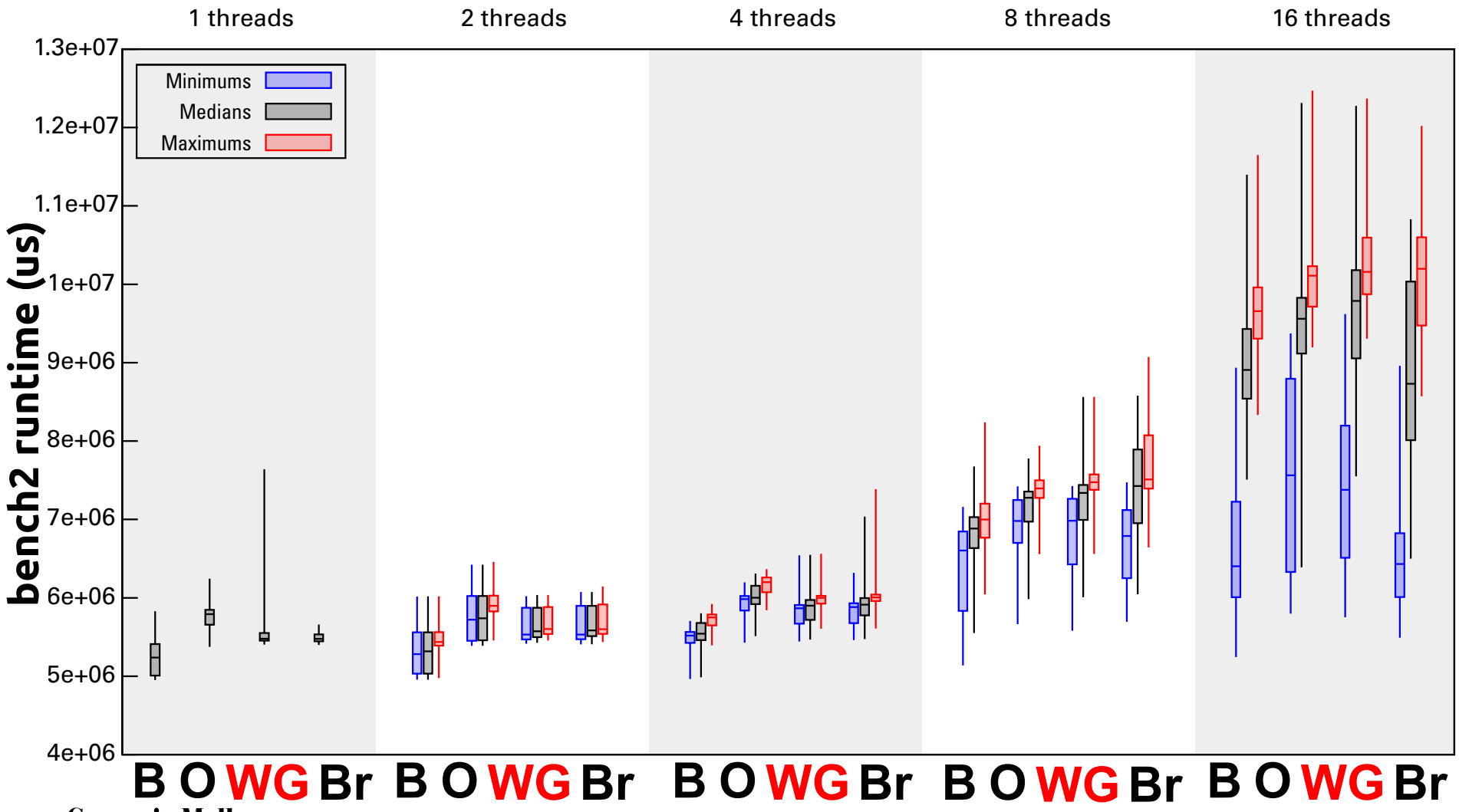
Carnegie Mellon
Parallel Data Laboratory

Runtime Performance Results



Carnegie Mellon
Parallel Data Laboratory

Runtime Performance Results



Takeaways

- Promising, but stat collection overhead is high
- Call for per-core MC stats counters
  - in hardware
- Works despite high cost of migration

Future Work

- More experimentation to understand benefits
  - as a function of application properties
  - heterogeneous workloads
- ANUMCA
  - exploring how these same ideas apply to cache
  - cache access is asymmetrically non-uniform too
- Optimizing the application-level goal
  - SLAs, multi-threaded app. performance goals

Conclusion

- Promising, but stat collection overhead is high
- Call for per-core MC stats counters
- Works despite high cost of migration

- Questions?

References

[1] Alexey Tumanov, Joshua Wise, Onur Mutlu, Greg Ganger. "Asymmetry-Aware Execution Placement on Manycore Chips. In Proc. of SFMA'13, EuroSys 2013, Czech Republic.

[2] Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi. "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems". In Proc. of HPCA'2013.

[3] Tilera Corporation, "The Processor Architecture Overview for the TILEPro Series", Feb 2013. [pdf]