Manycore: Will we learn from the past?

Tom Conte conte@gatech.edu

College of Computing

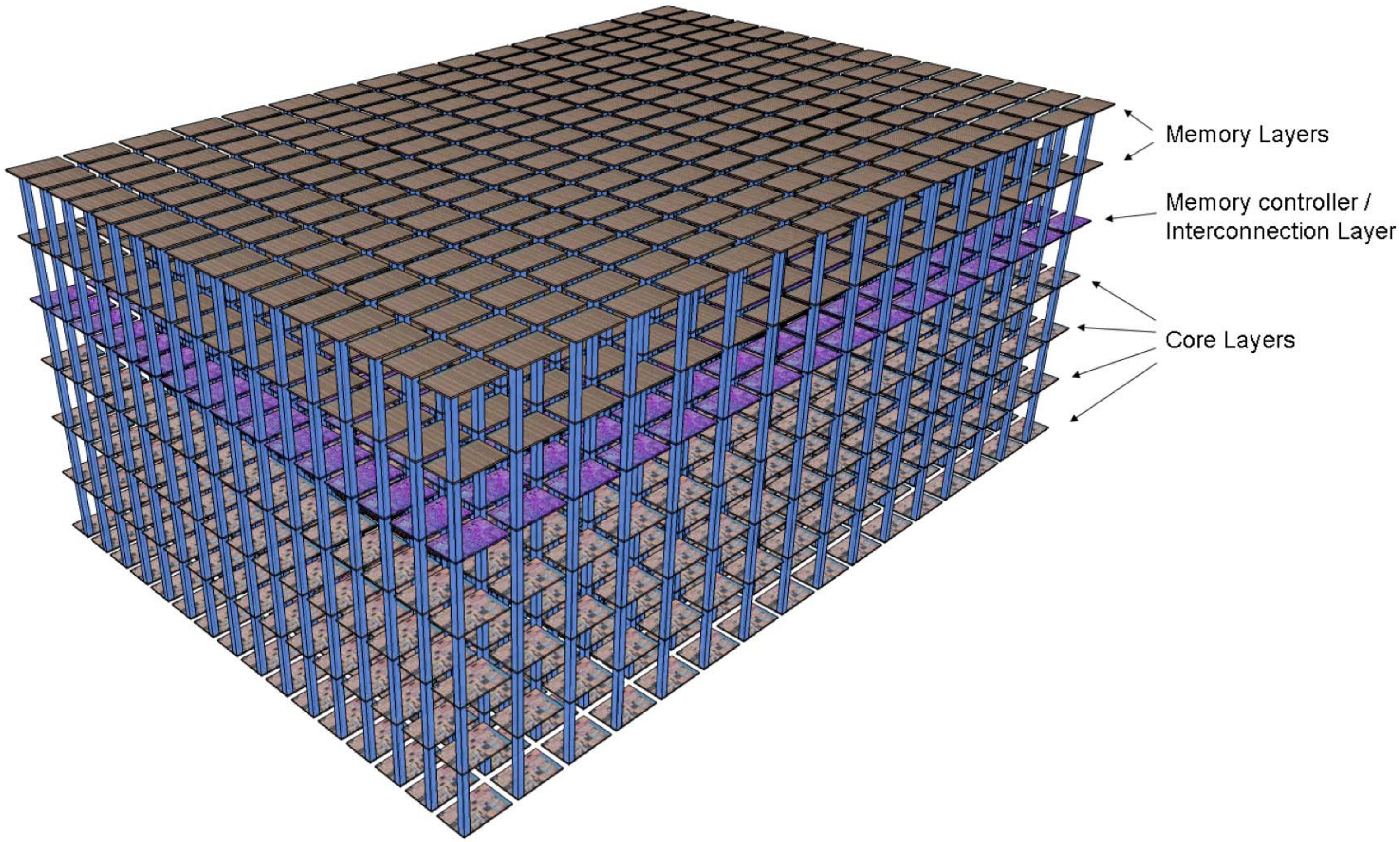
Georgia Institute of Technology

Georgia Tech College of Computing

Team

- Jesse Beu, Paul Bryan, Jason Poovey, Chad Rosier
- Faculty directly involved: Wayne Wolf, Sudha Yalamanchili, Milos Prvulovic, Santosh Pande, Nate Clark, Hyesoon Kim
- Other interested faculty…
  - Eric Rotenberg (NC State)
  - Hsien-Hsin Sean Lee (GT ECE)
  - Sung Kyu Lim (GT ECE)
  - Gabriel Loh (GT CS)

Veyron project

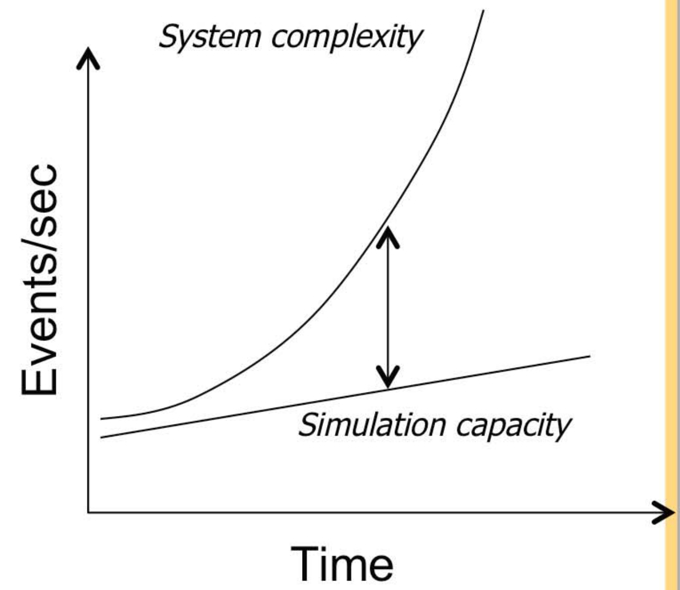


"1000 truly usable cores"

3D IC because when Moore's Law ends, go to S'mores Law

The State of Simulation

- System complexity is outpacing simulation capacity
- Cannot perform analysis at scale
  - 32 cores: barely, 1000 cores: *fahgettaboutit*
  - The problem will get worse, faster
- GT actively working on solutions in this space (*but not the topic of this talk…*)



Simulation Wall!

Multicore vs. manycore

Full out of order, 4-6 issue

BIG core, as large as power wall allows, good single-thread performance	BIG core, as large as power wall allows, good single-thread performance
BIG core, as large as power wall allows, good single-thread performance	BIG core, as large as power wall allows, good single-thread performance

Decent single thread performance, in-order, 2 issue

Smaller core	Smaller core	Smaller core	Smaller core
Smaller core	Smaller core	Smaller core	Smaller core
Smaller core	Smaller core	Smaller core	Smaller core
Smaller core	Smaller core	Smaller core	Smaller core

Multicore:
Optimized for throughput parallelism

Manycore:
Optimized for thread-level parallelism

Us vs. Them, again…

Famous us-vs-them in architecture wars

- Von Neumann vs. Dataflow
- CISC vs. RISC
- Superscalar vs. VLIW
- Shared memory vs. Message passing
- Manycore vs. Multicore?

- *There's a trend in history that we ignore at our own peril…*

Von Neumann vs. Data flow parallelism

- The fight:
  - Data flow extracts parallelism without the need for programmer-specified synchronization
    - but… only a subset of languages fit nicely into the model
  - Von Neumann continued sequential programming model using existing imperative languages
    - but… parallel programming notoriously complicated (“barrier everywhere” phenomenon)
- The winner:
  - Von Neumann: changing programmers harder than changing hardware

Georgia Tech College of Computing

RISC vs. CISC and then Superscalar vs. VLIW

- The fight:
  - RISC (vertical microcode exposed) and VLIW (horizontal microcode exposed) lead to simpler hardware
    - but… onus on the programmer and/or compiler
  - CISC (microcoded) and Superscalar (parallelism extracted in hardware) provide code compatibility
    - but… higher power, less parallelism extracted overall
- The winner:
  - RISC vs. CISC rendered irrelevant after P6 due to compatible installed base of x86 code
  - Superscalar won over VLIW
    - Last hold out of VLIW hitting code compatibility issues

Georgia Tech College of Computing

Shared memory vs. message passing

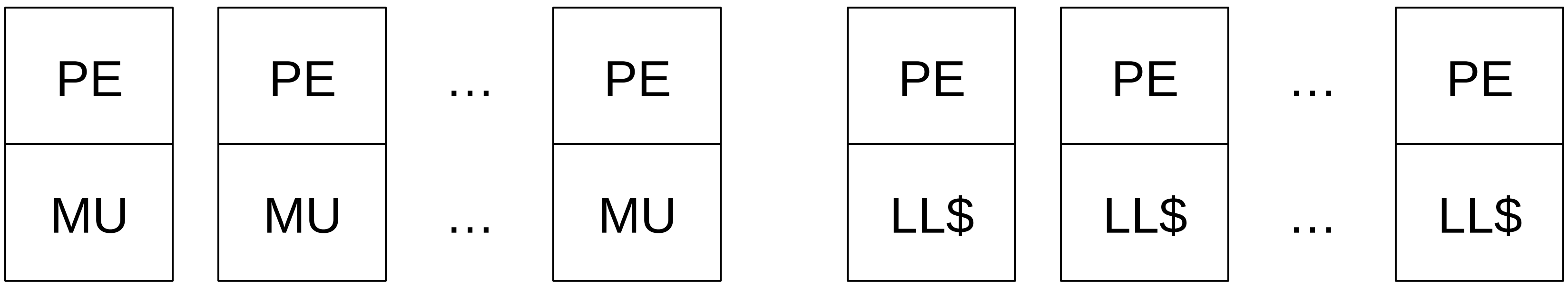
- The fight:
  - Message passing enabled much higher numbers of processing elements
    - but… programmers have to move data to the computation
  - Shared memory enabled easier programming models
    - but… memory coherence got complicated
    - and eventually foist it on the programmer again via weaker consistency models
- The winner:
  - Shared memory for all but CSE people

Georgia Tech College of Computing

What's the fight about for manycore?

- Programmers dictate our architectures
- Shared memory is *easier to program*
  - Limited shared memory (regions, clusters) is MPI under a different name
- Oft heard claim: Manycore is limited to "friendly" applications, not general purpose
  - Mainly because grew out of GPUs, but note that "General purpose" is always a moving target
- Ah, but 1000 coherent cores are hard to do in hardware…

DSM vs. Multi/Manycore cache coherence



DSM	Manycore
NUMA: Static address => MU map	NUCA or LLS: Static address => LL$ map
COMA: Dynamic => MU map	COCA or LLP: Dynamic => LL$ map

Georgia Tech College of Computing

NUCA vs. COCA

- NUCA (Non-uniform cache architecture)
  - No duplication of a line: only stored in one place
  - Bad: that ‘place’ may be the wrong place, sharing prohibitive
  - Soln: Move data between NUCA banks
- COCA (Conventionally organized cache architecture)
  - Good: lines near where they’re needed
  - Bad: Duplicates a line, LL$’s need to be kept coherent
  - Soln: (Directory) coherence

Directory-shared (DS) vs Directory-private (DP)

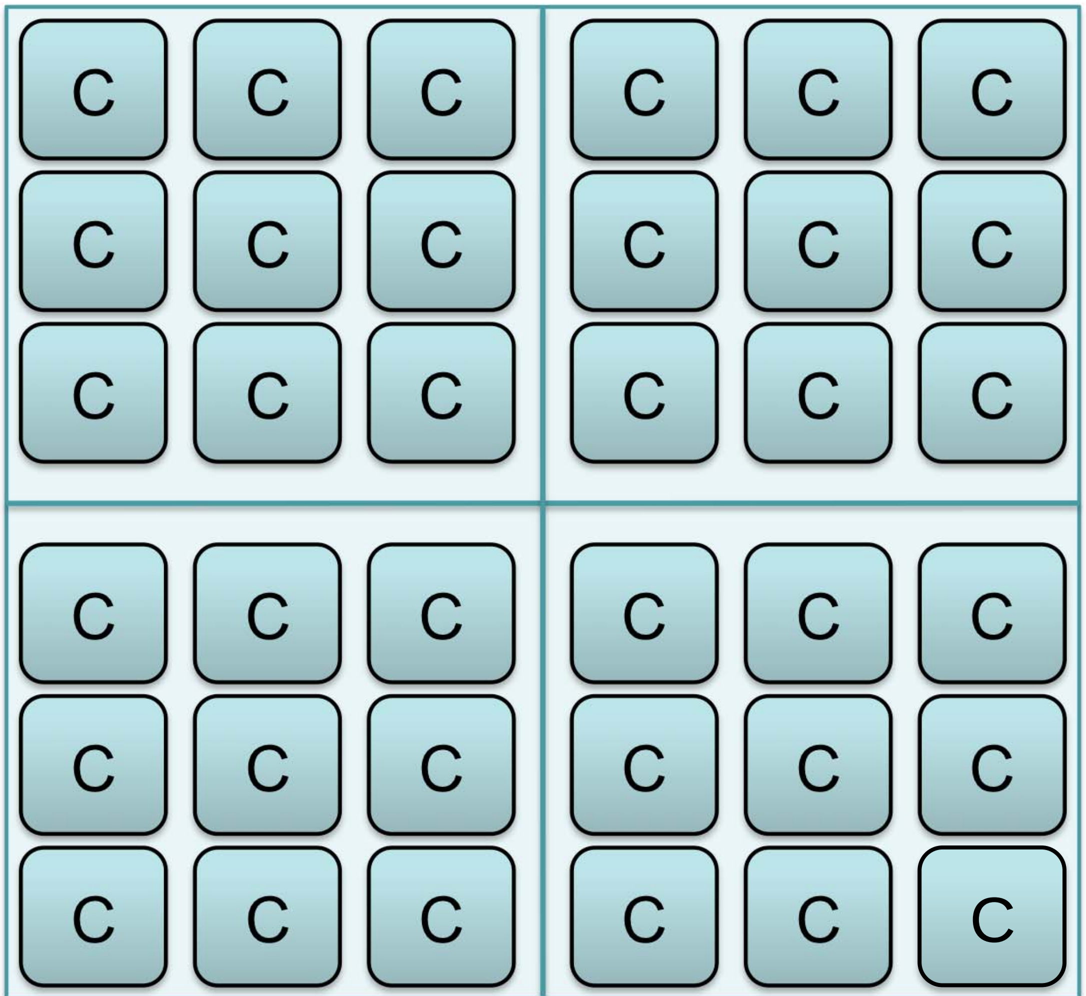
DS:
Home for dir of fixed range of addrs

DP:
Home for dir <u>cache</u> of larger (off-chip) dir

DS vs DP (cont)

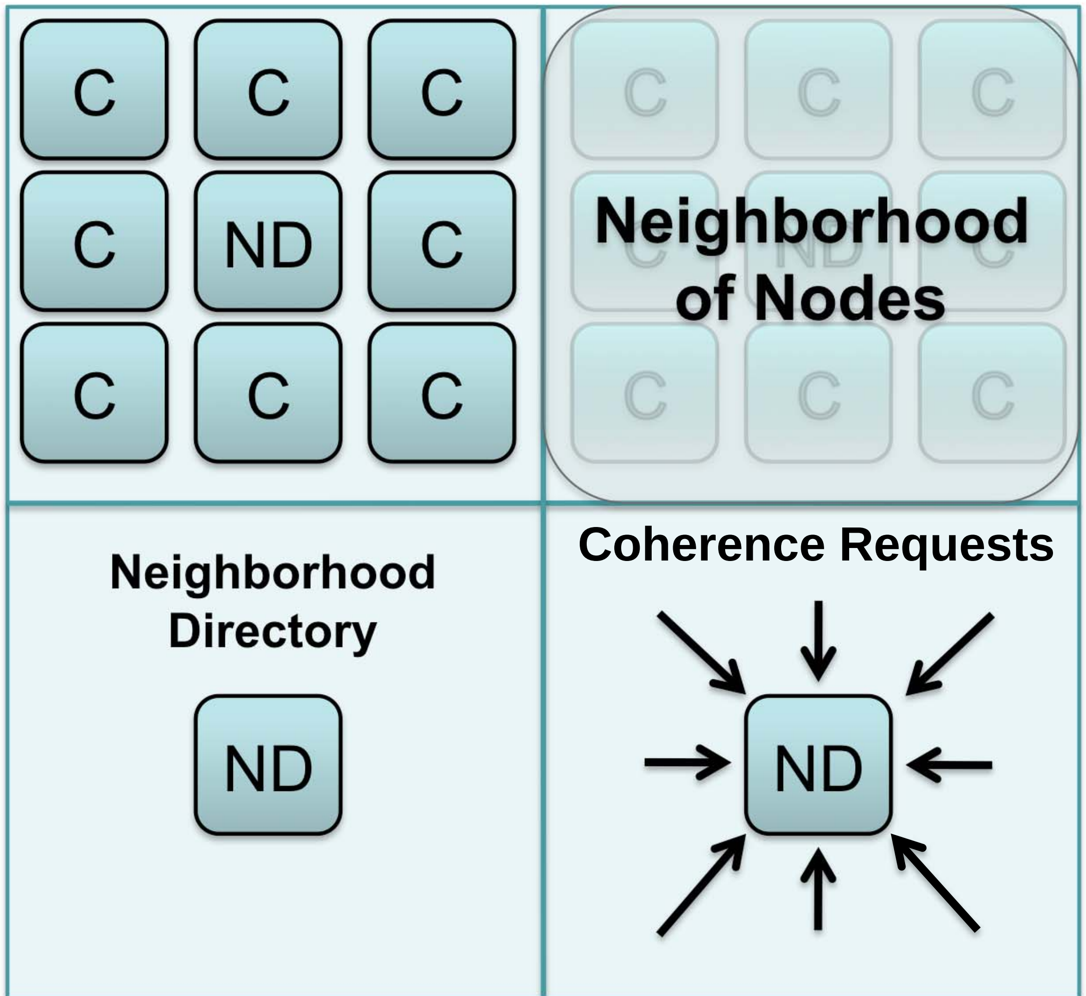
- DS is the NUCA of directory structures
  - Good: No duplication of a dir entry: only stored in one place, easy to find an entry
  - Bad: that ‘place’ may be the wrong place, lots of network traffic
  - Soln: Move dir entries– CAN’T
- DP is the COCA of directory structures
  - Good: dir entries near where they’re needed
  - Bad: Duplicates a dir entry, directories need to be kept coherent, large amount of area/space, no go-to “home” for a first time dir-miss
  - Soln: coherence – <u>of directories</u>
- Or a hybrid…

DS-DP hybrid



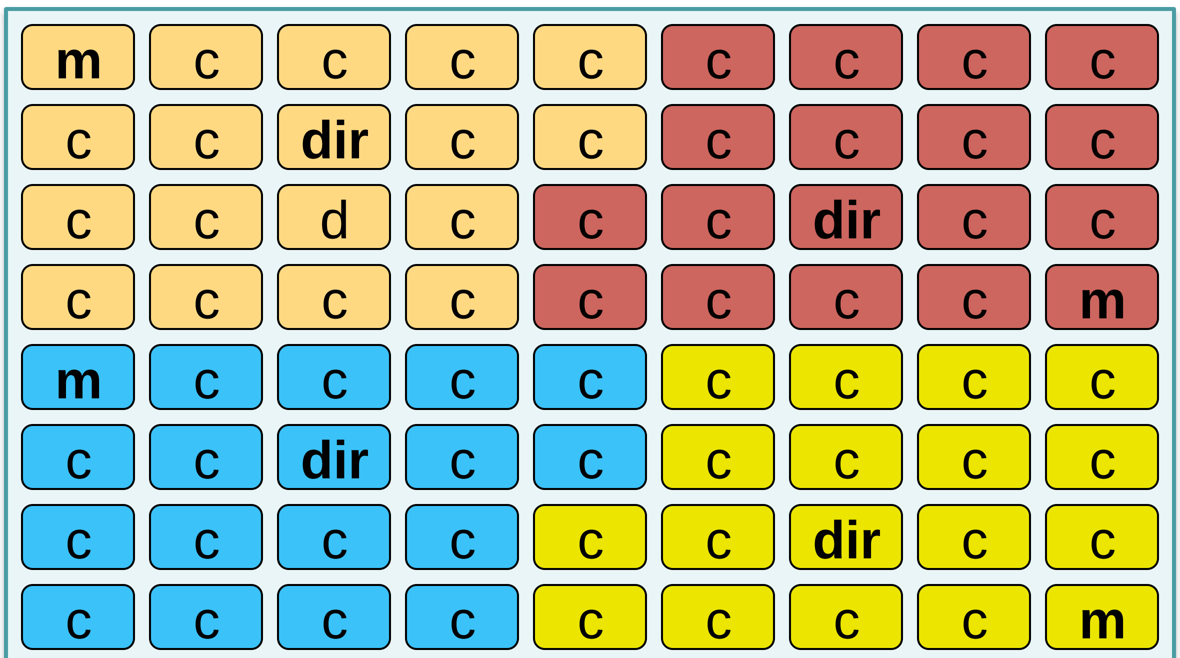
DS-DP

Each ND assigned a range of addrs for first time miss



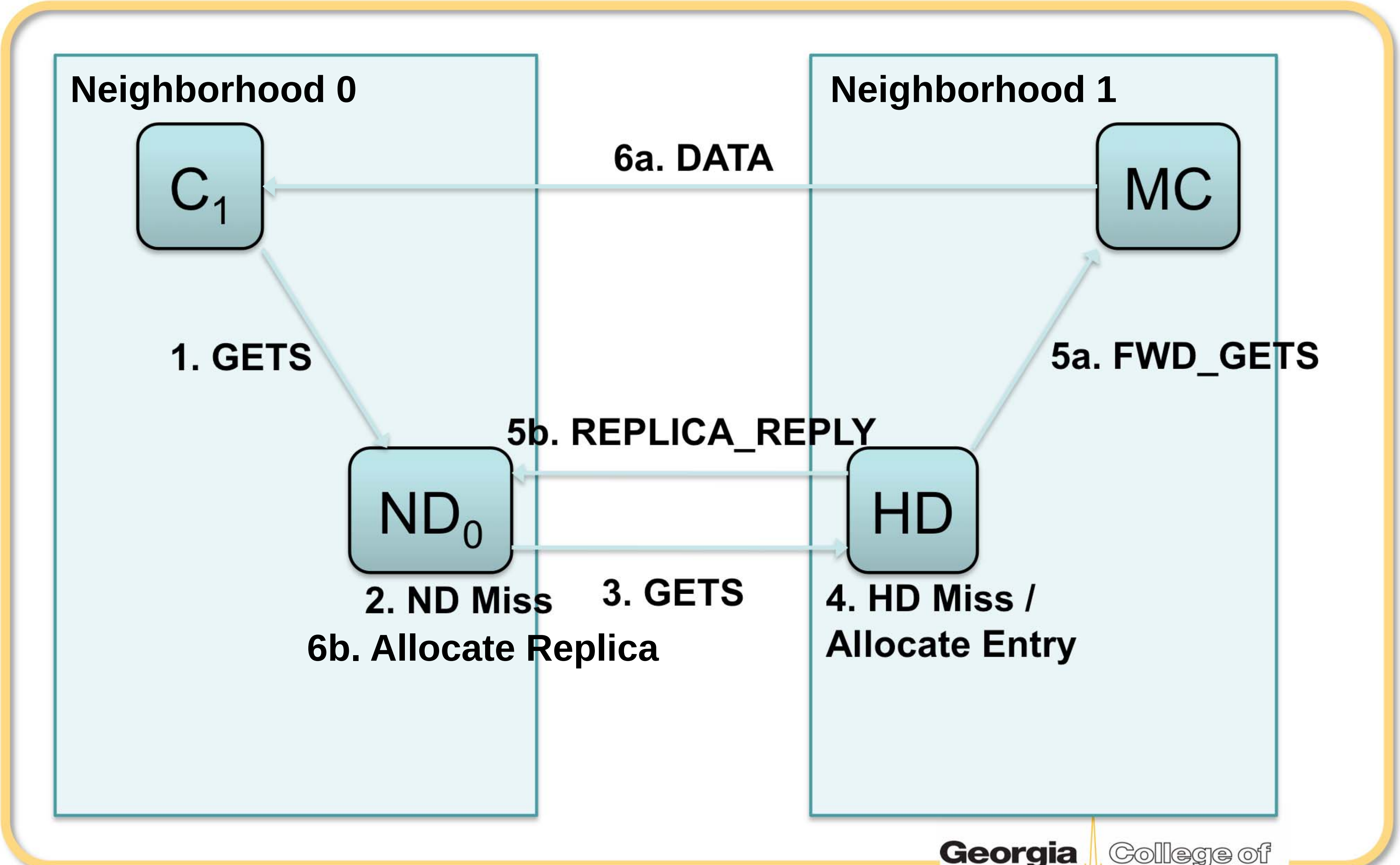
DS-DP 64-Core Tessellation

- 4 Neighborhoods grouped by color
- Per Neighborhood: 1 neighborhood directory (dir)16 cores (c), and 1 memory controller (m)



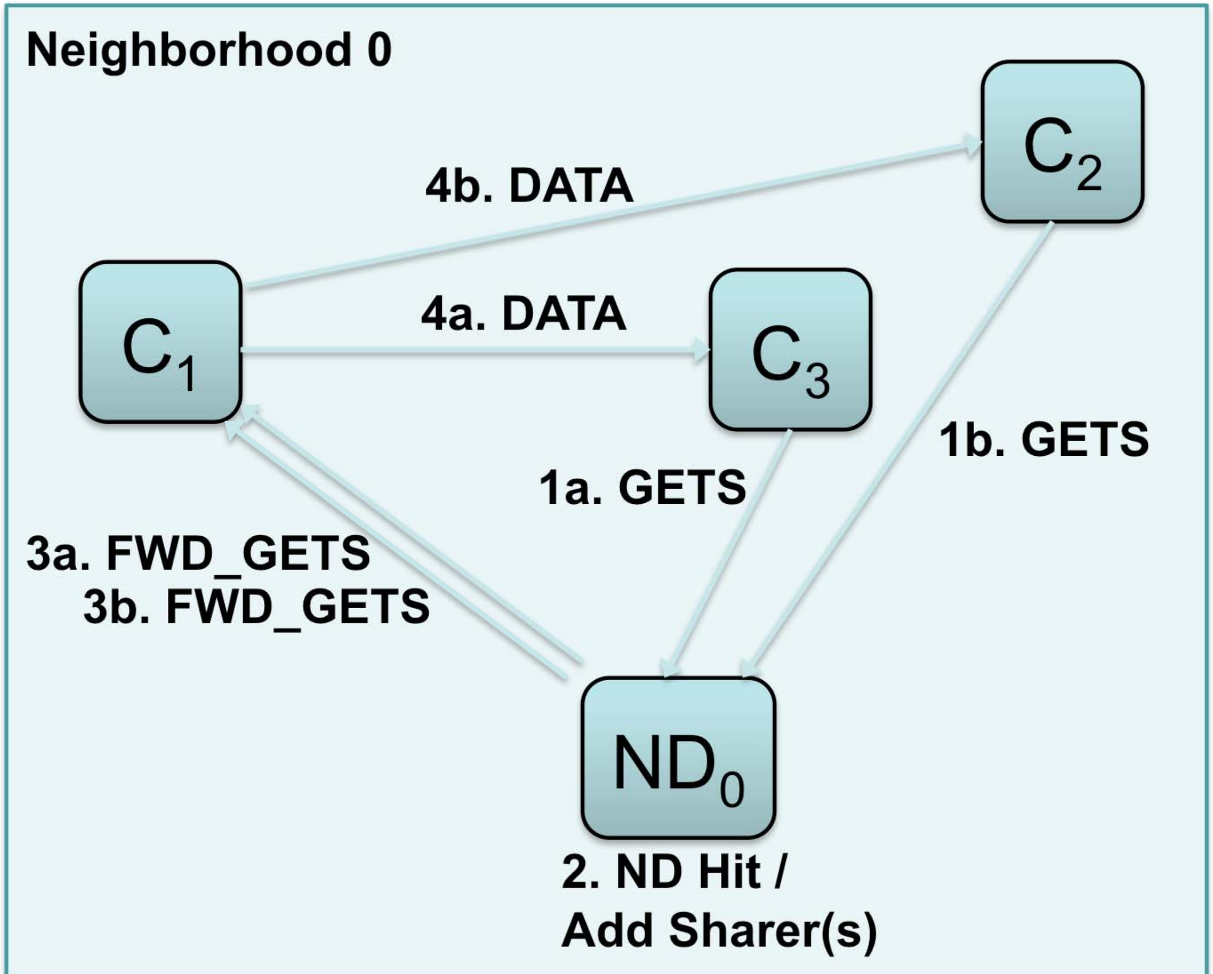
Example I: First Time Miss

C – Core
ND – Neighborhood Dir
HD – Home Dir (DS)
MC – Memory Controller

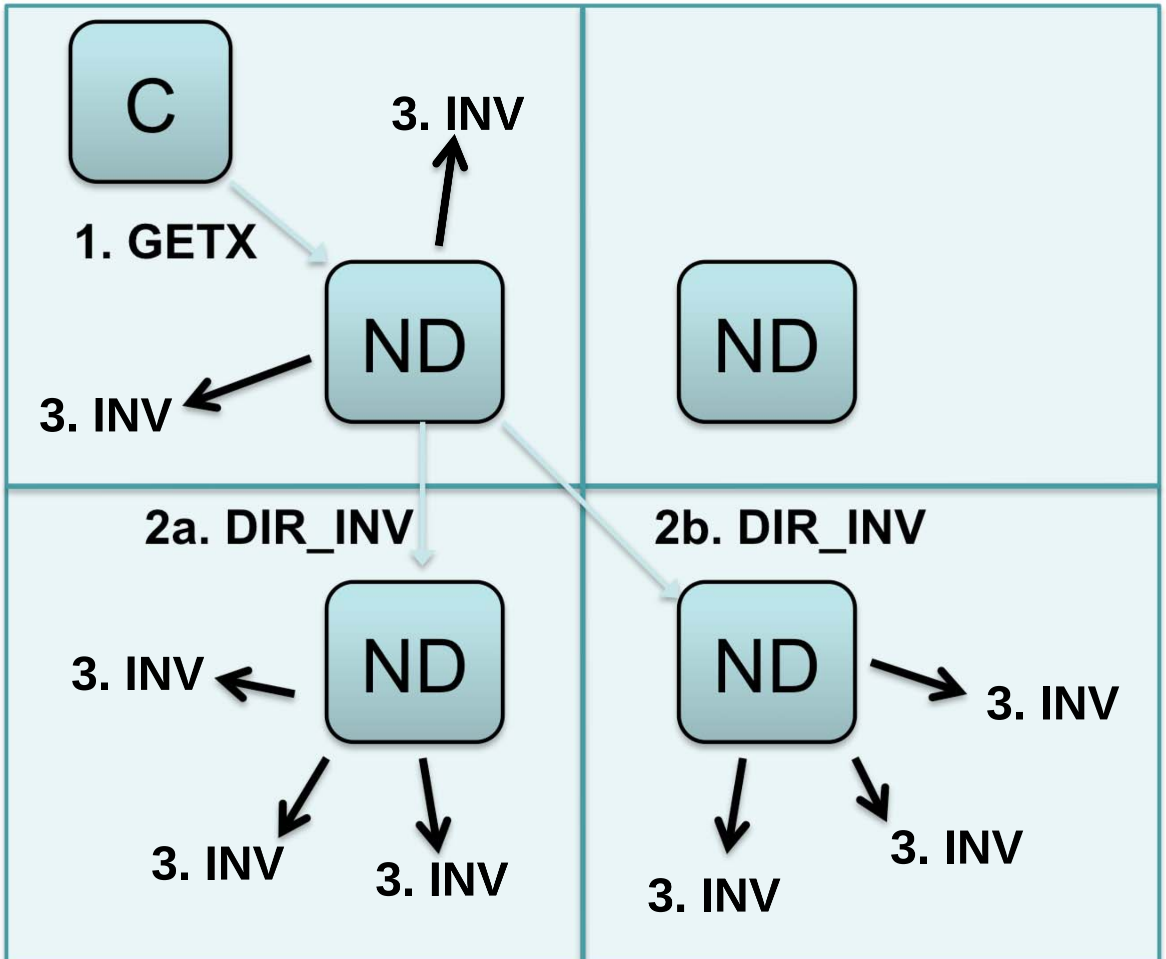


Georgia Tech College of Computing

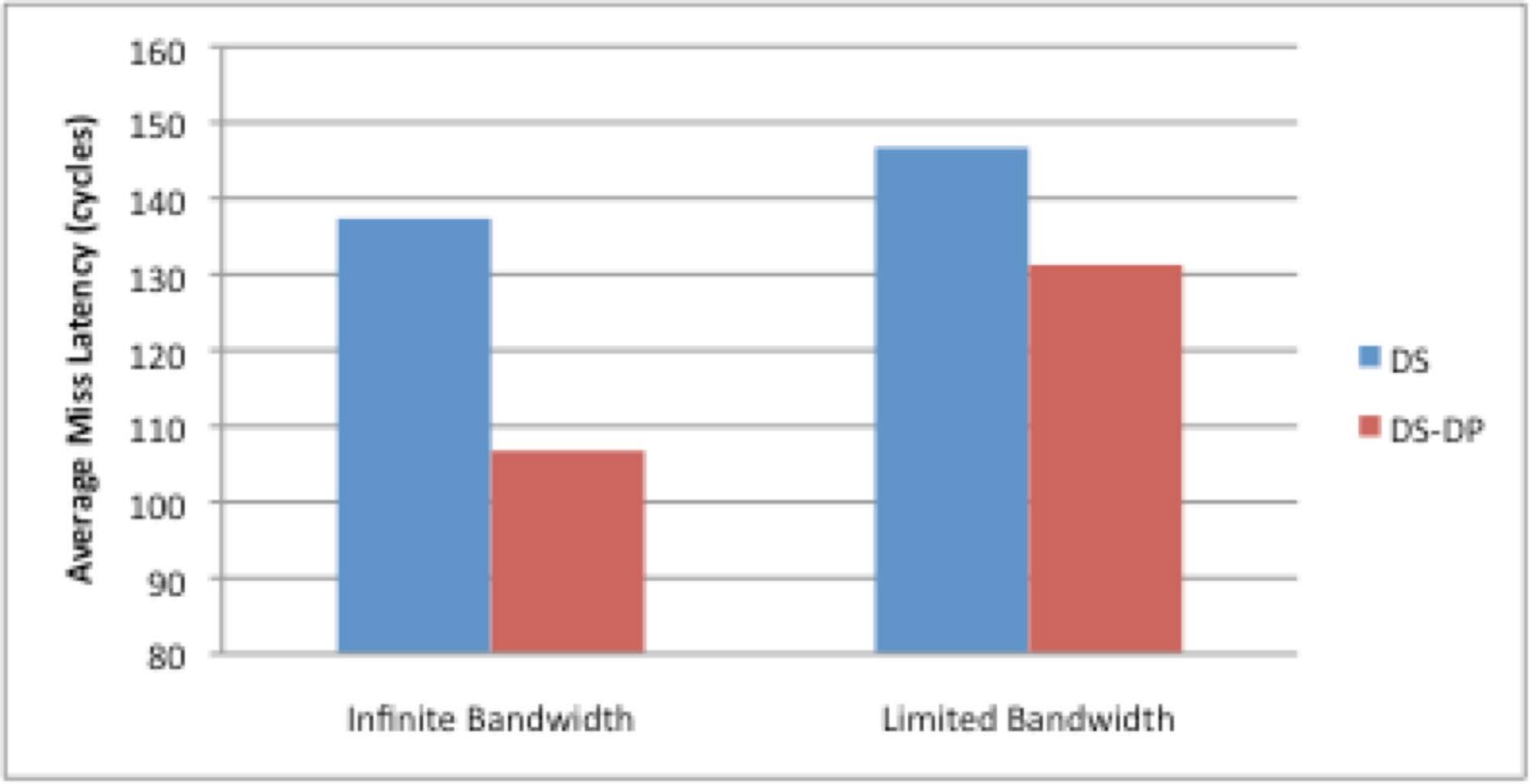
Example 2: Neighborhood Dir Hit



Example 3: GET eXclusive invalidation

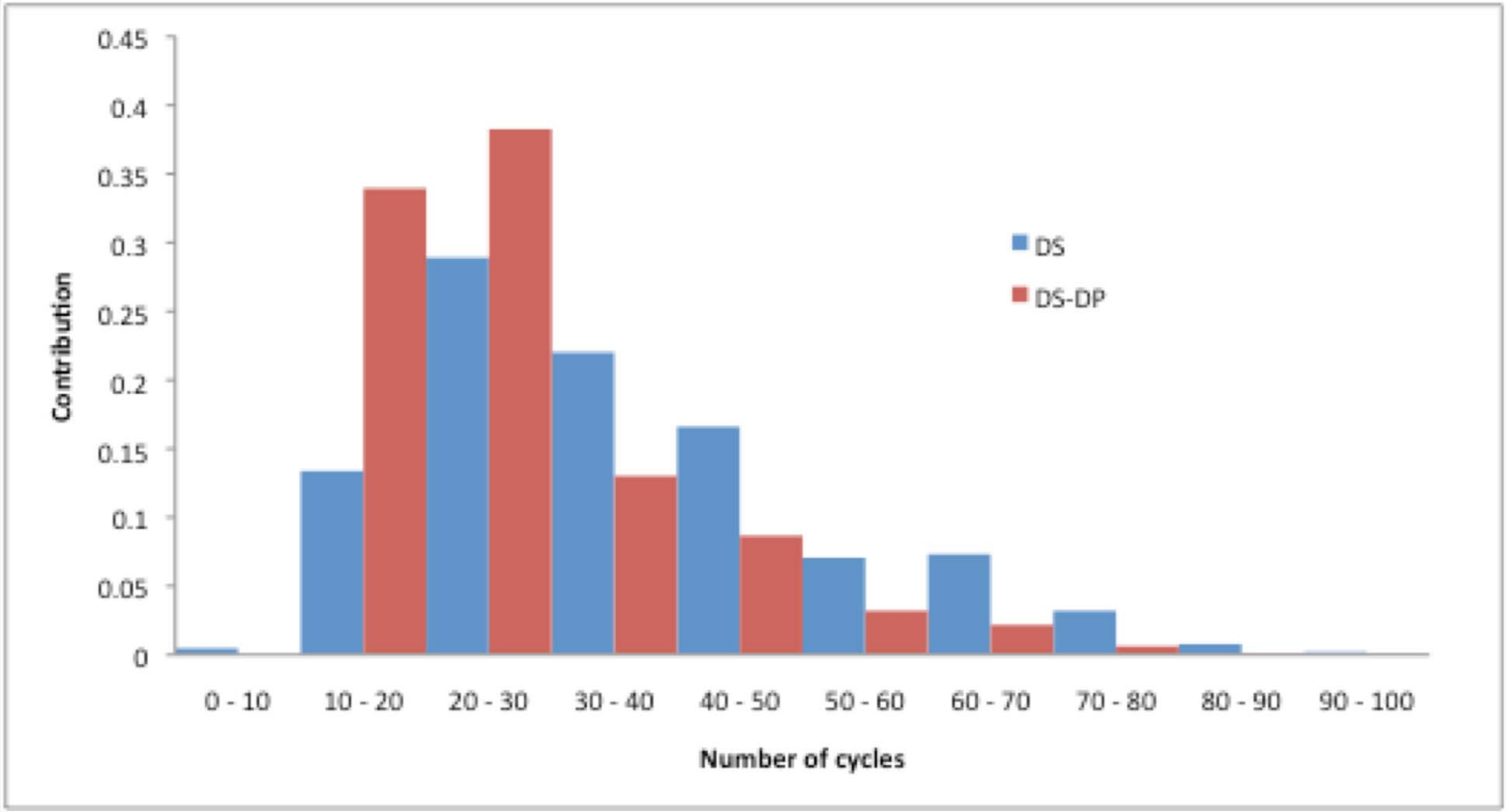


Average miss latency



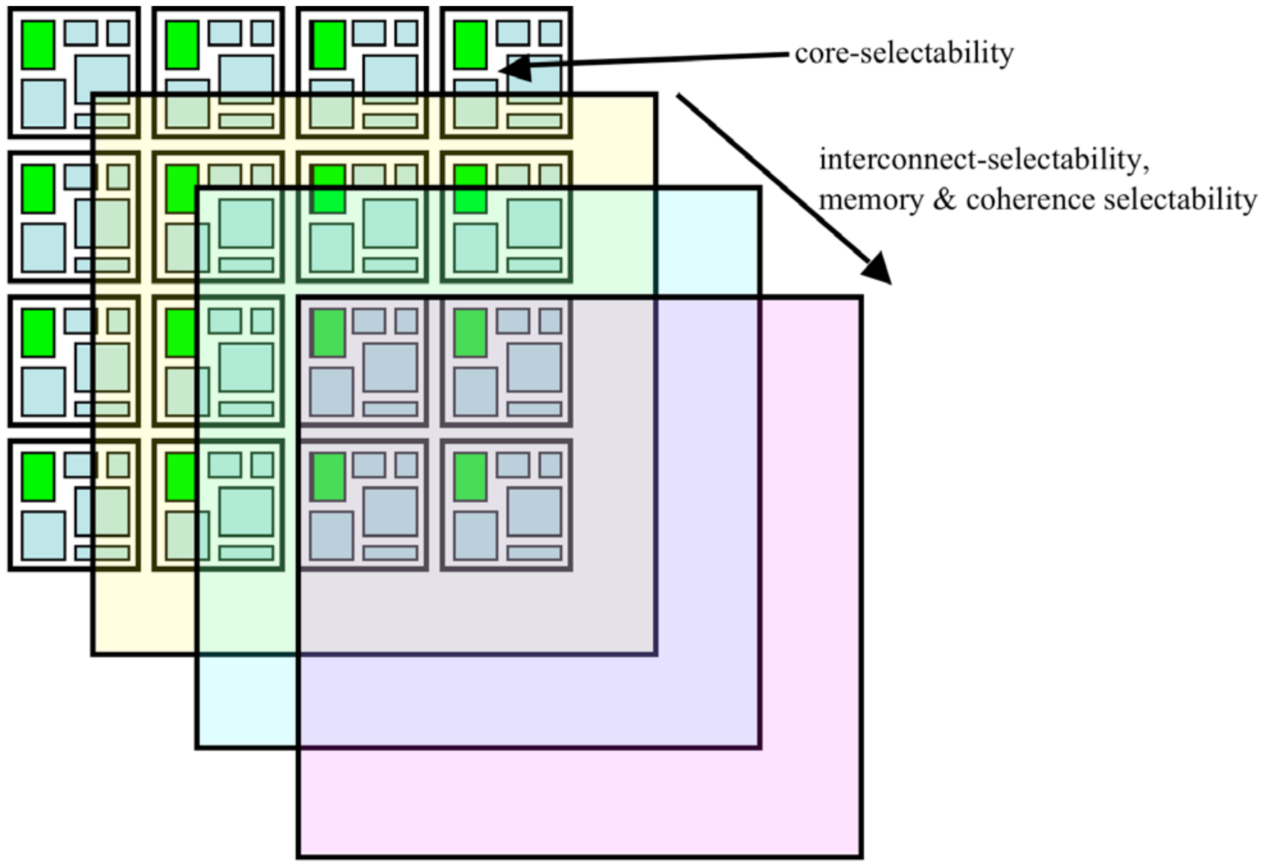
Georgia Tech College of Computing

Distribution of message latencies



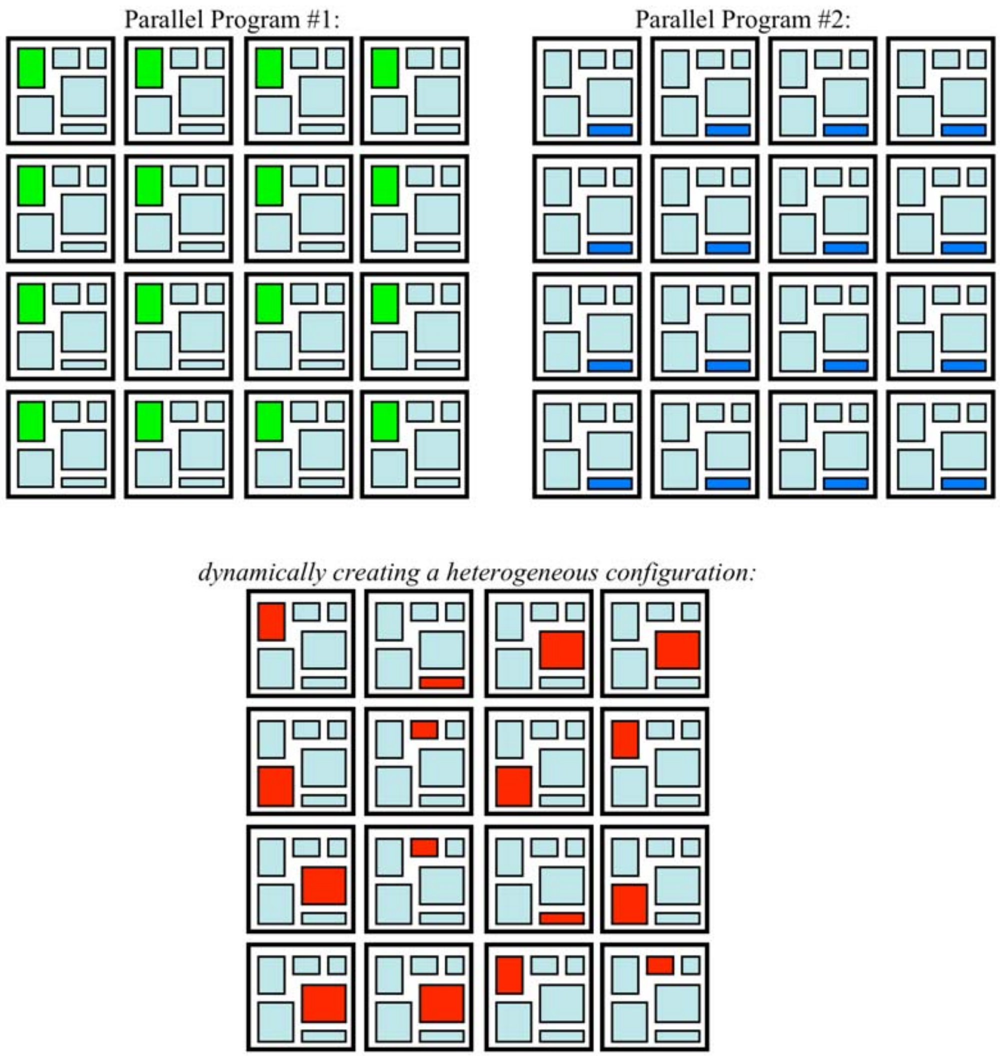
Georgia Tech College of Computing

Veyron heterogeneity



Georgia Tech College of Computing

Core selectability



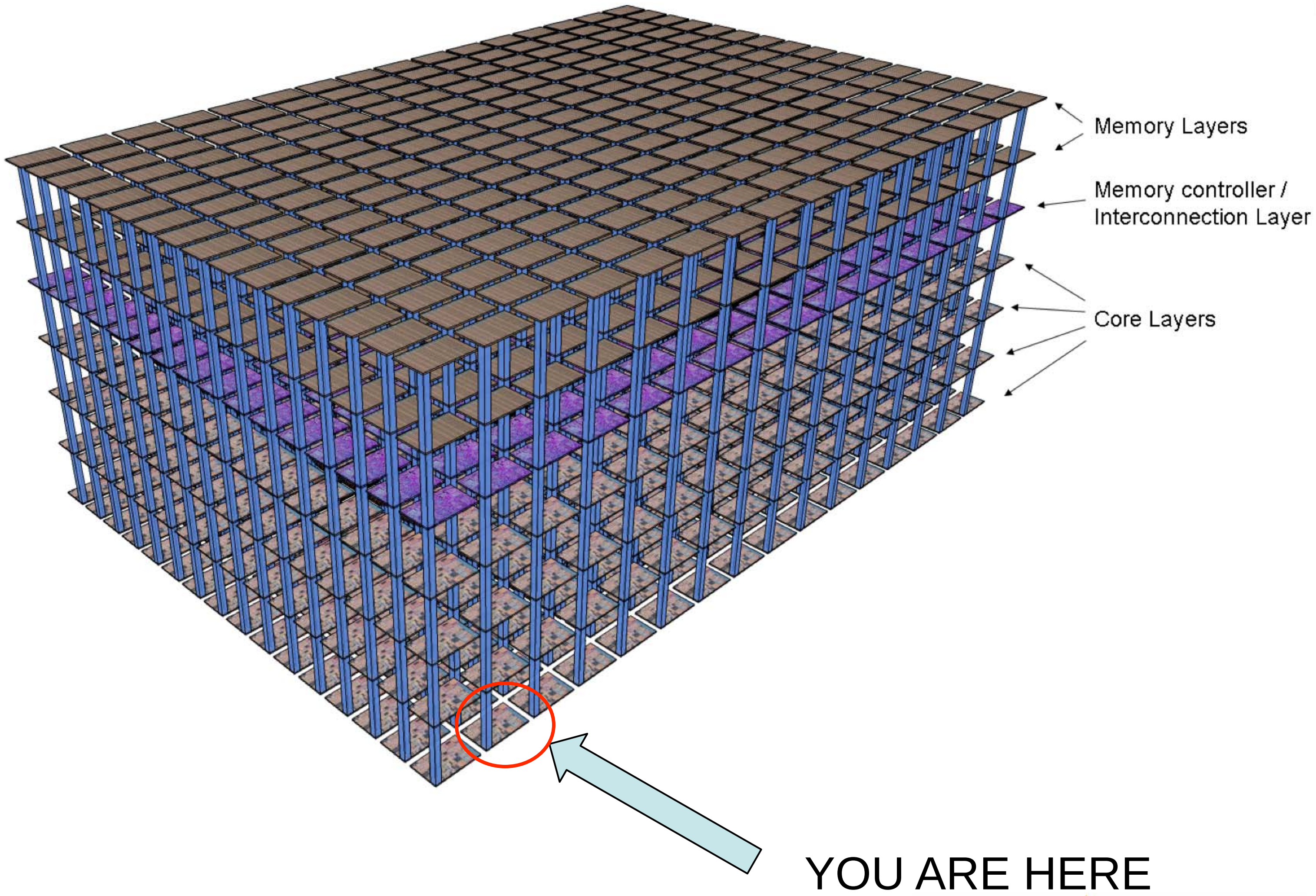
Tech College of Computing

Why?

- "Powerful" facts
  - Custom design is vastly lower power than general purpose
  - General purpose is more … general purpose
- Select between customized, application-specific design layers in the 3D stack
- Old idea, but prior approaches to this failed because:
  - Multiple packages, one per custom design
  - Cross chip => pin crossing power burn
  - Cross chip => data in wrong place

Georgia Tech College of Computing

Veyron: The cores



Georgia Tech College of Computing

Feature wish list for the cores

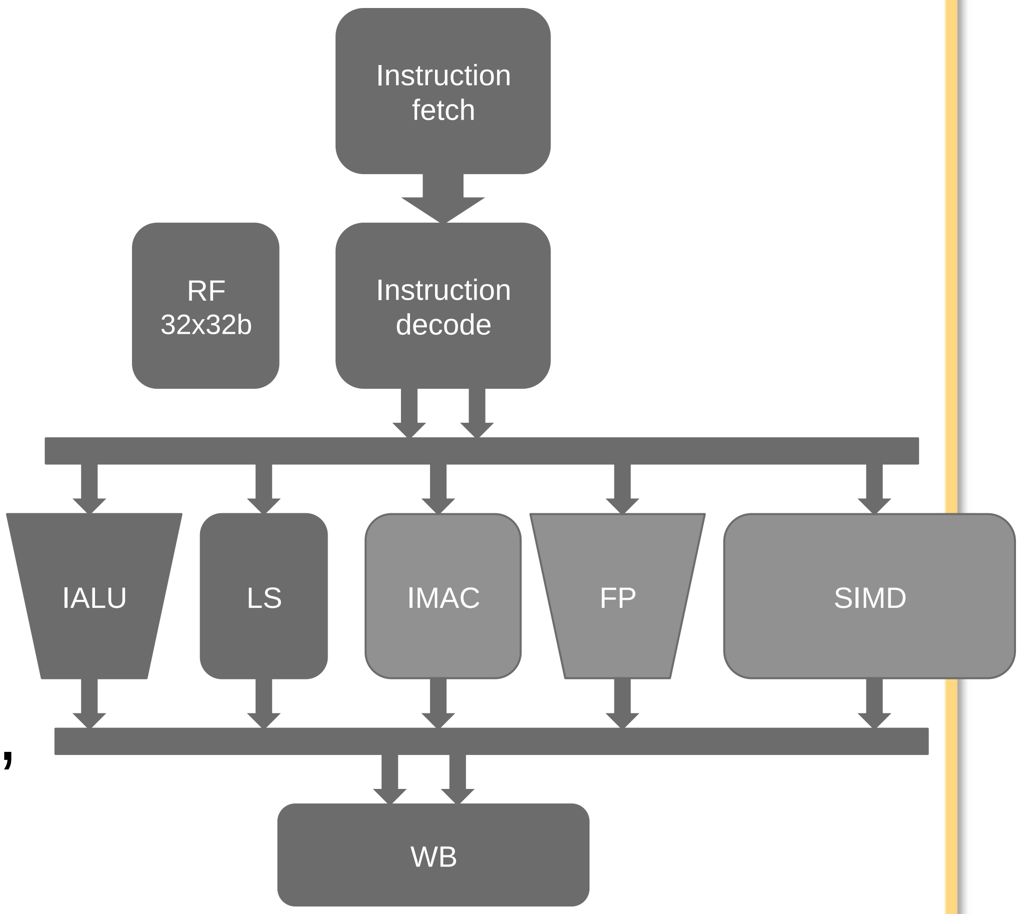
- Scalable issue widths
- Function unit selectability
- ILP rich
- Small
- Synthesizeable …*by graduate students*
- Plug-compatible FUs
- ISA compatible across family
- Ability to use open source compilers, debuggers, libraries
- Low power features (turn units on/off programmatically)

CLAW

- Clustered Length-Adaptive Word:
  - Clusterable in-order processor
  - Originally designed for low-power embedded, effort started in 2004 by Balaji Iyer, funded by Qualcomm, NSF, Redhat
  - ISA is a clustered VLIW extension of OpenRISC
- Not a "paper design"
  - Synthesizeable Verilog
    - 0.15mm^2 in 45nm
  - Complete compiler tool chain: GNU tools, uLibC, GCC 4.1, (including Treegion scheduler, Haifa vectorizer)

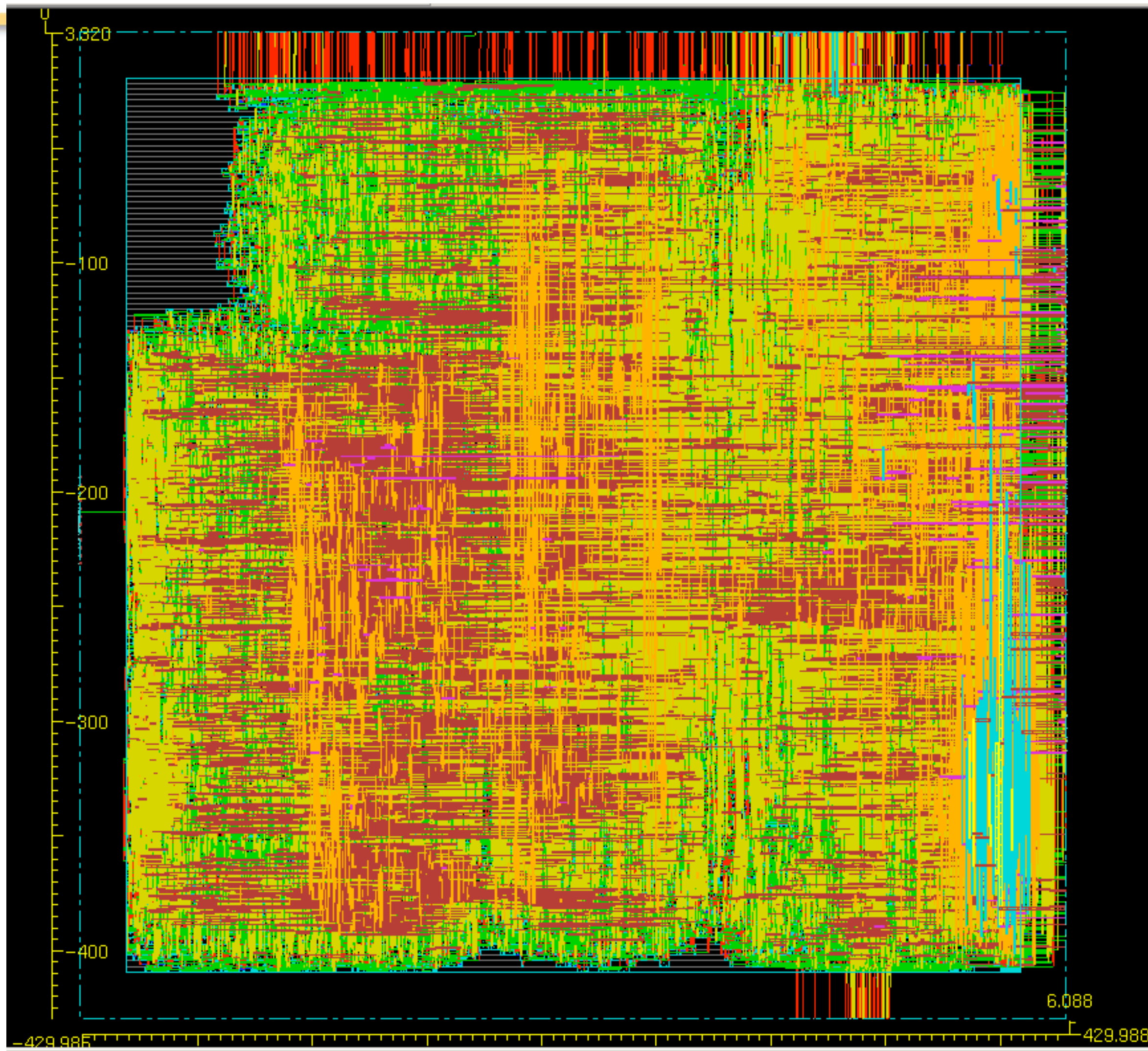
CLAW architecture

- 2 operands/cluste
- Scalable to multiple clusters: 2, 4, 6 issue
- 32-entry RFs
- 2 ALUs, Load/Store
- Plug in IMAC, FP, SIMD
- Five+ stage pipeline: IF, ID, RR, EX1, (EX2), WB
- Multiple hardware threads



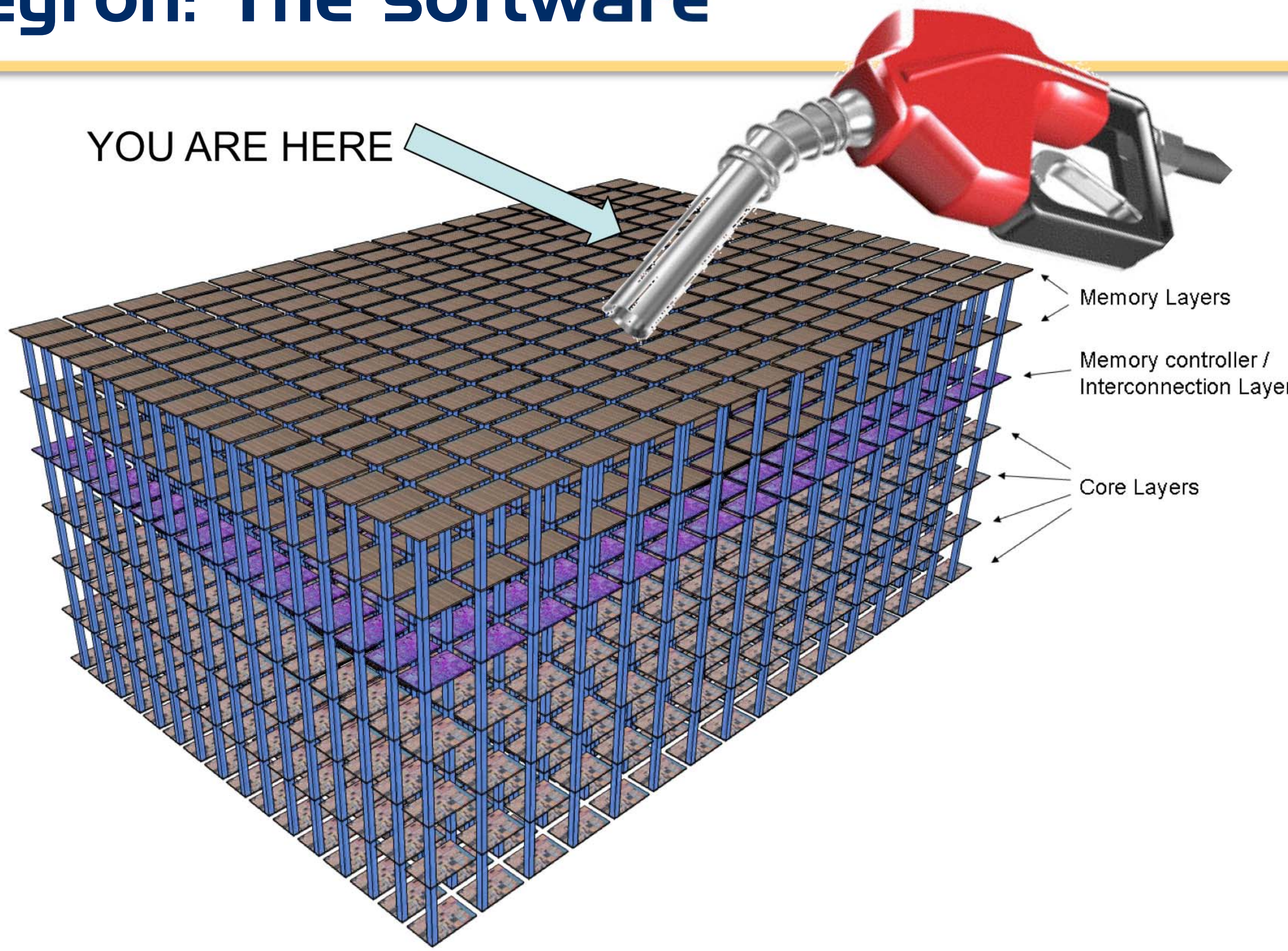
Georgia Tech College of Computing

Place and route of CLAW in 45nm



Georgia Tech College of Computing

Veyron: The software



Georgia Tech College of Computing

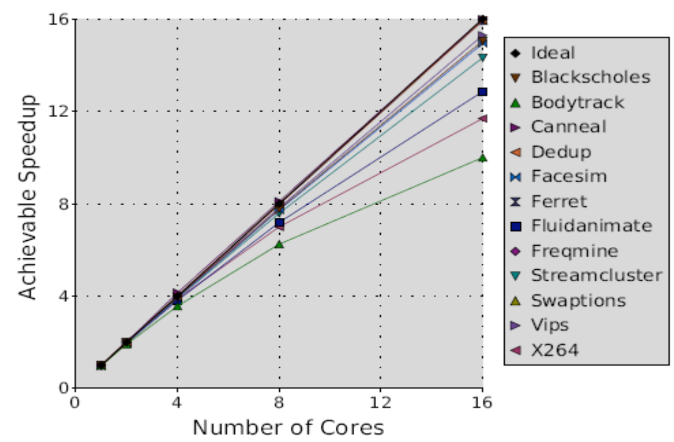
Benchmarking Manycore

- Throughput benchmarking for multiple processors
  - Good:
    - Easy to use all processors
  - Bad:
    - “representative” of future applications?
    - system measurement issues
- Multi-threaded programming models for design comparison
    - e.g. Splash-2, PARSEC
  - Good:
    - Easier measurement techniques
  - Bad:
    - Harder to effectively use all processors

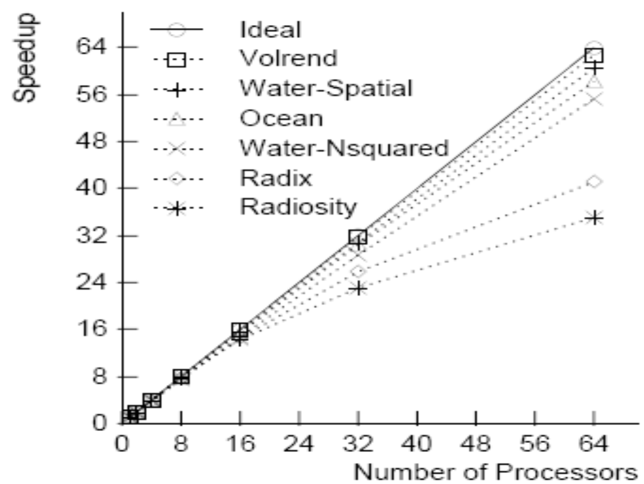
Georgia Tech College of Computing

Previous Scalability Assumptions

- PARSEC:
  - Measure inherent concurrency based on executed instructions in parallel and serial code sections
  - Delays on contended locks and load imbalance are neglected
  - CMP$im used to model a CMP cache hierarchy (application level only)



- Splash-2:
  - Measure actual concurrency on an abstract machine
  - Every instruction completes in 1 cycle



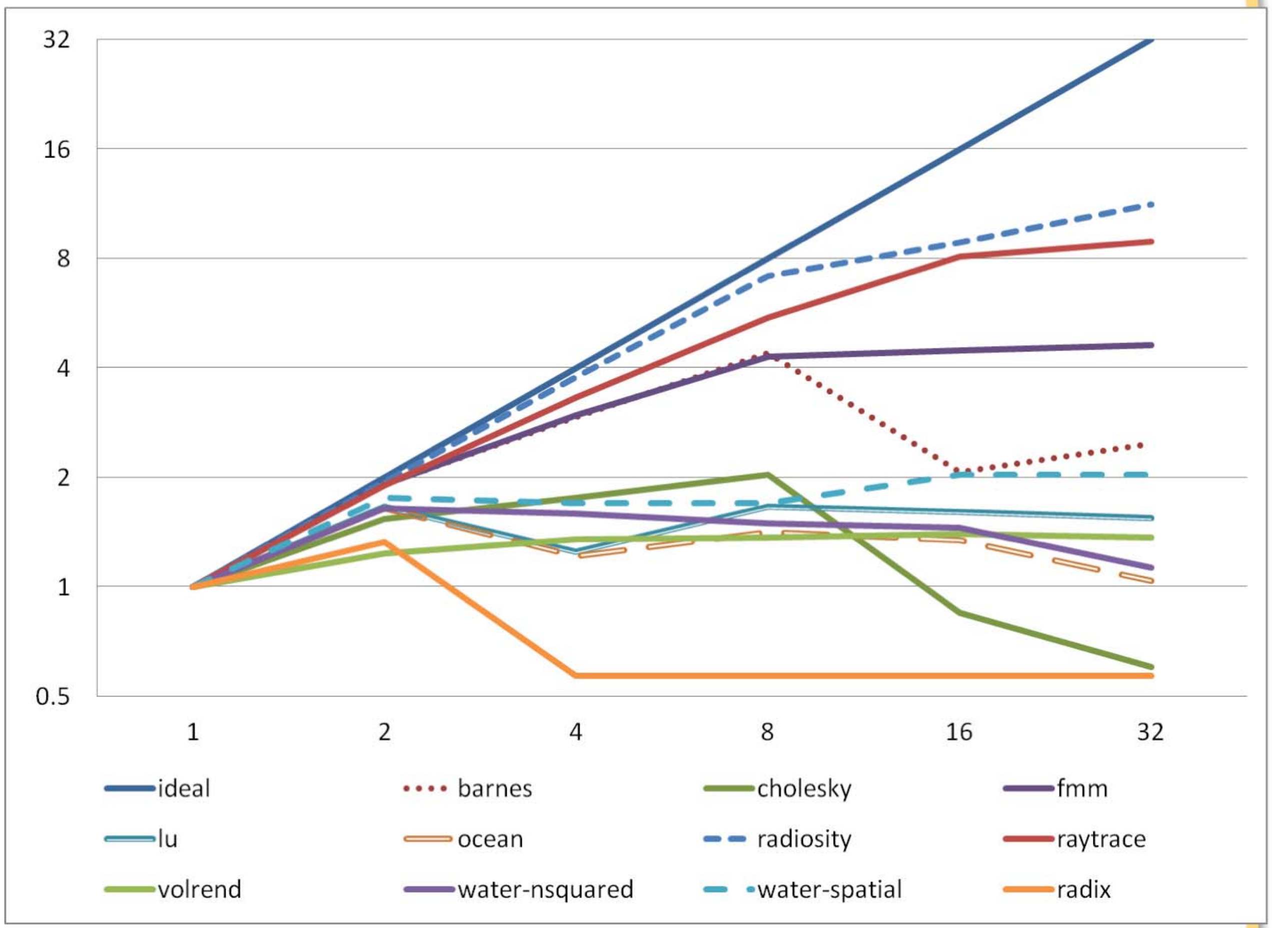
- Both are interested in the inherent program characteristics rather than performance

Georgia Tech College of Computing

Splash-2 Scalability with OS

Average speedup	
2	1.718
4	2.188
8	3.105
16	3.213
32	3.501

Speedup times derived from wall clock time

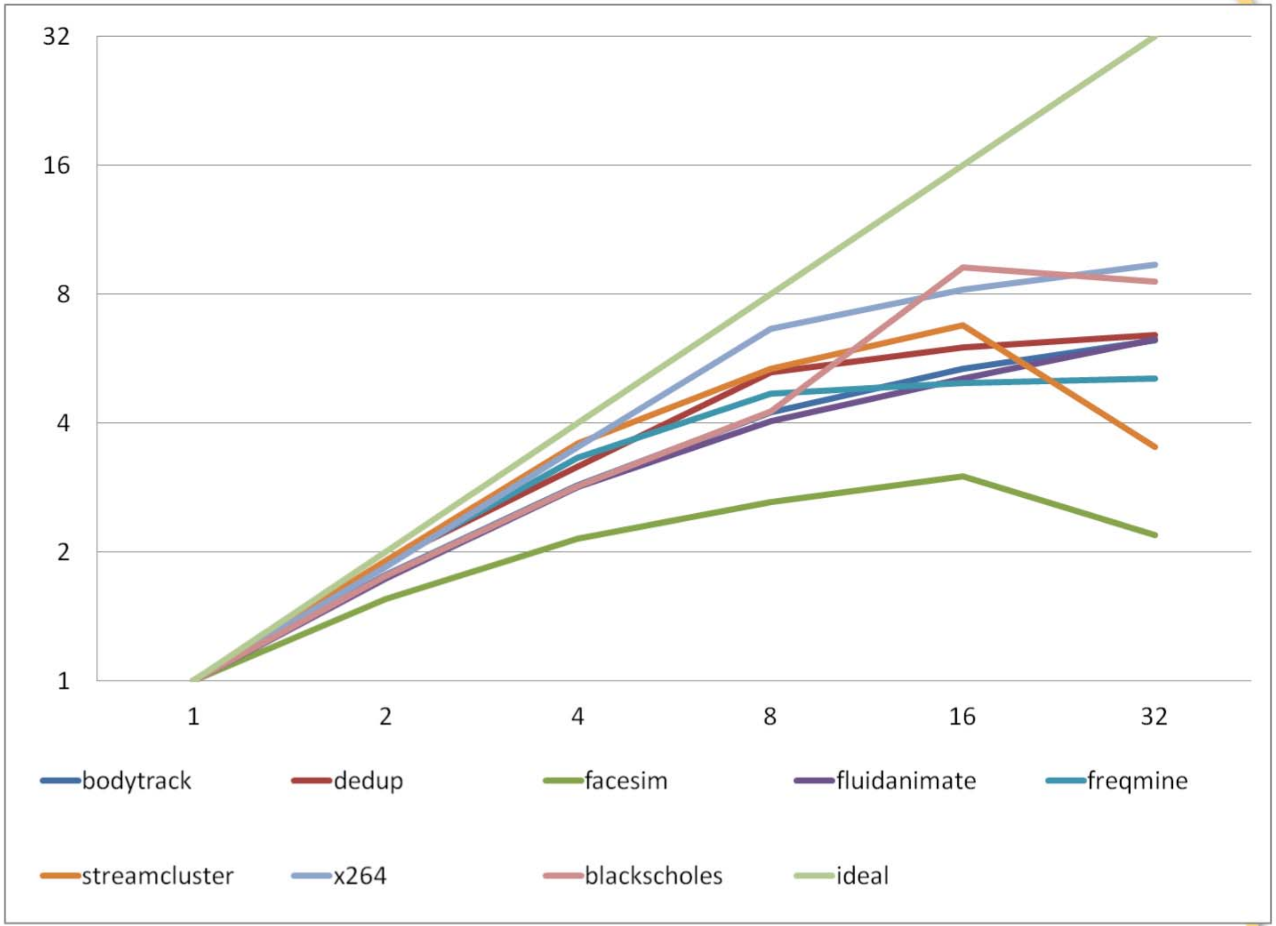


Threads vs. Speedup

Georgia Tech College of Computing

PARSEC Scalability with OS

Average speedup	
2	1.797
4	3.064
8	4.694
16	5.630
32	5.585



Threads vs. Speedup

Performance Comparison

- Average scalability compared against theoretical projections

PARSEC

Processors	Projected Speedup	Measured Speedup
2	1.831	1.797
4	3.183	3.064
8	5.162	4.694
16	7.678	5.63
32	10.36	5.585

Splash-2

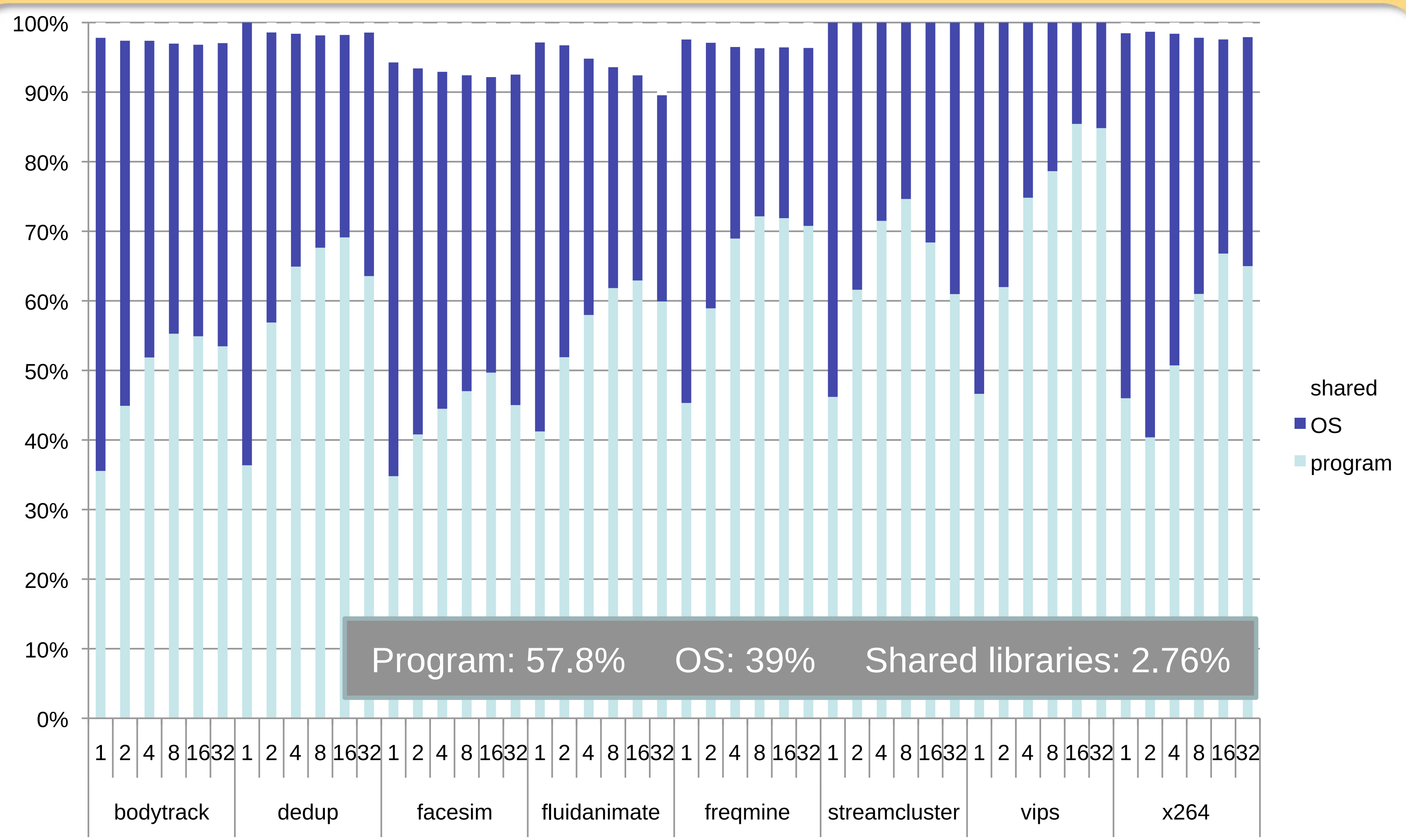
Processors	Projected Speedup	Measured Speedup
2	1.625	1.683
4	2.552	2.041
8	3.834	2.875
16	5.466	2.973
32	7.318	3.234

"Scalability"

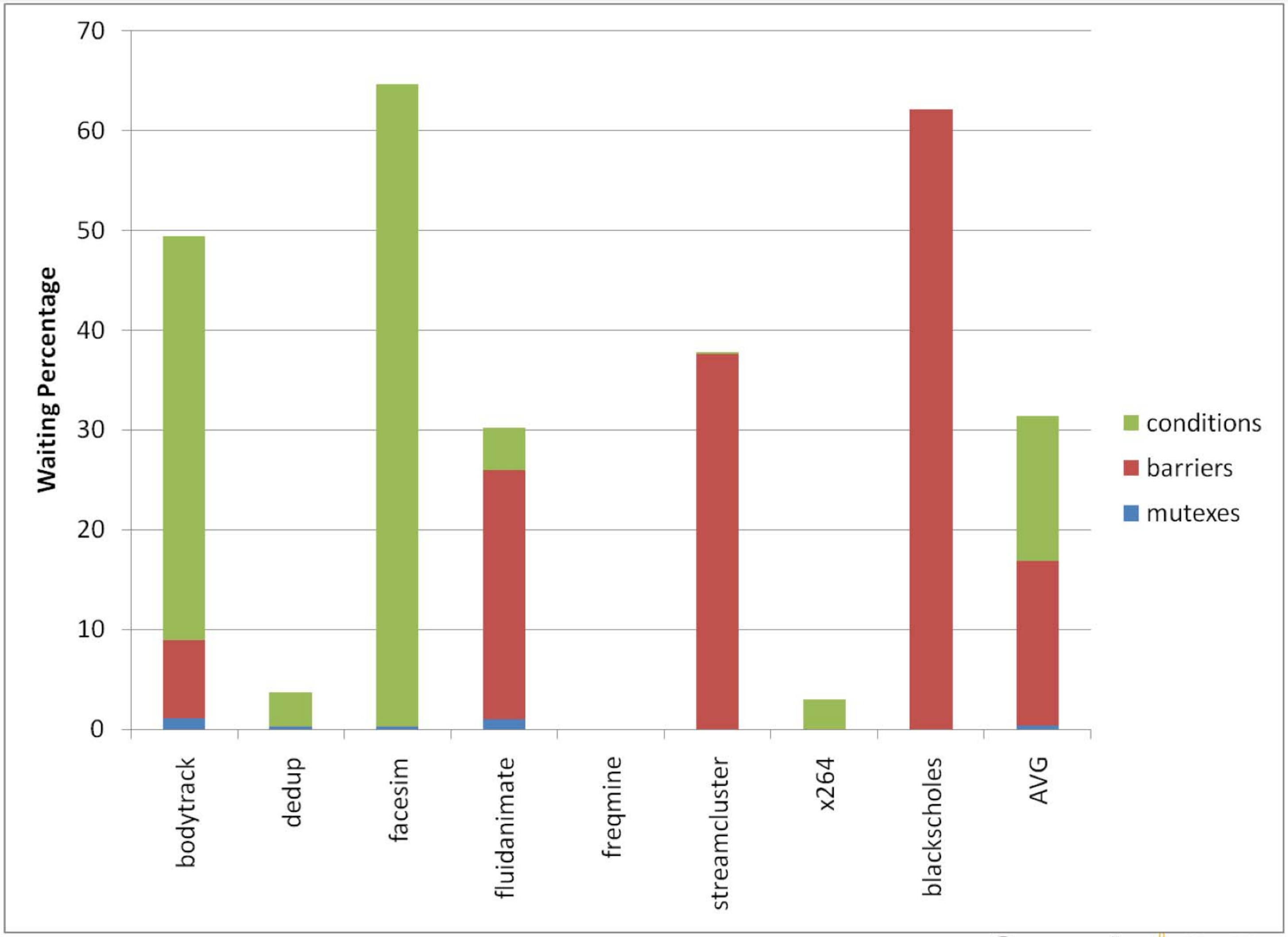
- Scalability saturation and even degradation was observed…why?

- Potential reasons:
  - Microarchitectural efficiency
  - Inherent workload parallelism
  - Initialization code dominates
  - Synchronization efficiency
  - OS scheduling / context switch overhead
  - OS accounting / memory management
  - Shared library behavior

PARSEC Deconstruction

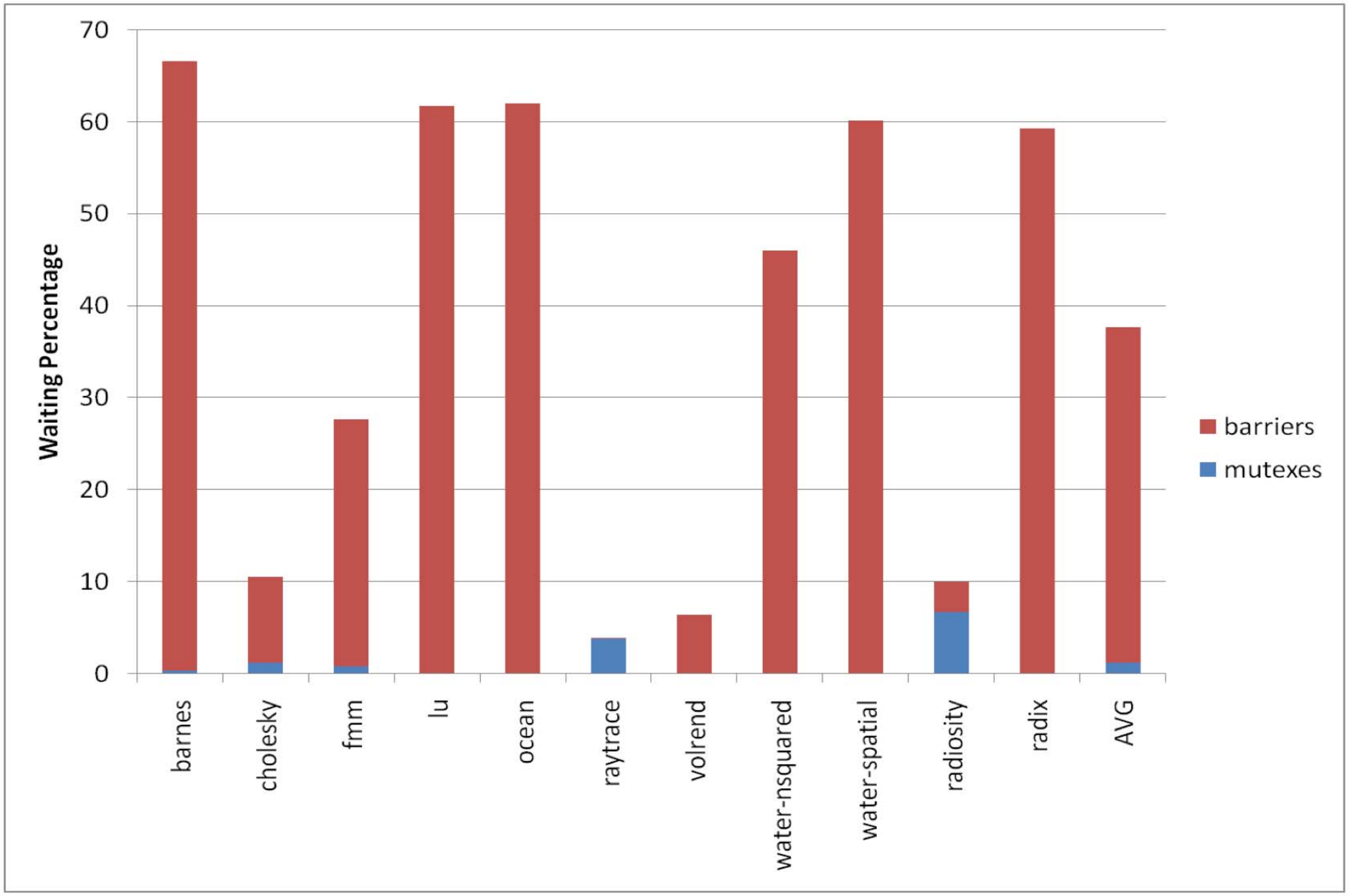


PARSEC Synchronization



Georgia Tech College of Computing

Splash-2 Synchronization



Georgia Tech College of Computing

"Parallel" benchmarks

- These benchmarks do not scale to 1000 cores!
- Synchronization is the main limiter of scalability
  - Barriers and condition variables major contributors
  - Mutexes often uncontested, but will change
- As the core counts increase, every fractional percentage of overhead will be relevant to scalability evaluation
- Synchronization
  - Mutexes will be increasingly important at 1000 cores
  - OS interaction (< 3%) will matter (big red arrow)

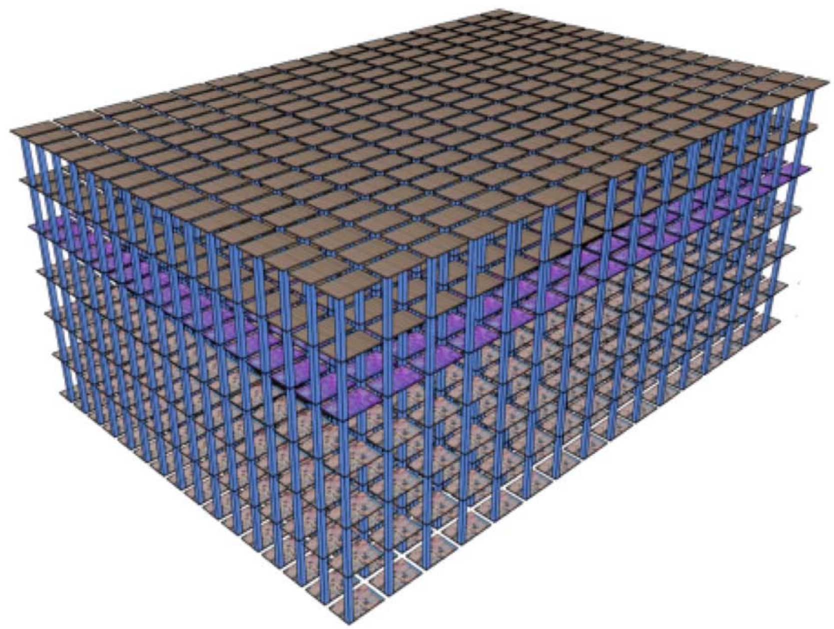
Now We Have a Benchmark Catch 22

- Architecture is benchmark-driven
- The benchmarks we have are not scalable to 1000 cores
  - Barriers and condition variables major contributors
  - Mutexes often uncontested, but will change
  - OS interaction (< 3%) will matter (big red arrow)
- Solutions?
  - Expand the applications to new spaces
  - Stop treating benchmarks as "black boxes" – architects must become *computational scientists* as well

Georgia Tech College of Computing

Summary: Georgia Tech Veyron project

- Design, simulate and (hopefully) construct a 1000 core *general purpose* manycore
- What's new:
  - Programmers matter
  - 3D tech
  - Coherent 1000 nodes in hardware
  - COCA
  - Heterogeneous cores with a common ISA for low power



END

QUESTIONS?