Coherence, Consistency, and Déjà vu: Memory Hierarchies in the Era of Specialization

Sarita Adve

University of Illinois at Urbana-Champaign

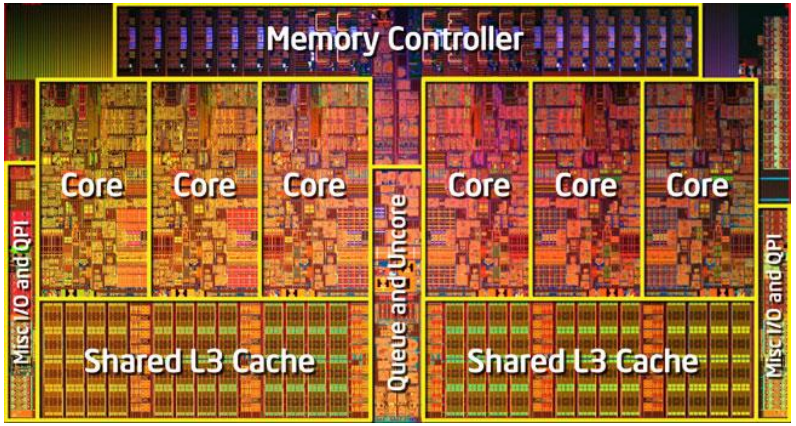
sadve@illinois.edu

w/ Johnathan Alsop, Rakesh Komuravelli, Matt Sinclair, Hyojin Sung and numerous colleagues and students over > 25 years

This work was supported in part by the NSF and by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

Silver Bullets for End of Moore's Law?

Parallelism



Specialization

BUT impact software, hardware, hardware-software interface

This talk: Memory hierarchy for heterogeneous parallel systems

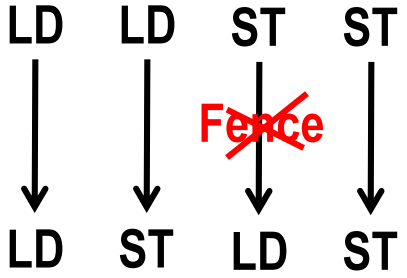
Global address space, coherence, consistency

But first …

My Story (1988 – 2016)

My Story (1988 – 2016)

- 1988 to 1989: What is a memory consistency model?
  - Simplest model: sequential consistency (SC) [Lamport79]
    - Memory operations execute one at a time in program order
    - Simple, but inefficient
  - Implementation/performance-centric view
    - Order in which memory operations execute
    - Different vendors w/ different models (orderings)
      - Alpha, Sun, x86, Itanium, IBM, AMD, HP, Cray, …
    - Many ambiguities due to complexity, by design(?), …



Memory model = What value can a read return?

HW/SW Interface: affects performance, programmability, portability

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
    - Distinguish data vs. synchronization (race)
    - Data (non-race) can be optimized
    - High performance for DRF programs

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model BUT racy programs need semantics

    $\Rightarrow$ **No out-of-thin-air values**

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model BUT racy programs need semantics

    $\Rightarrow$ No out-of-thin-air values $\Rightarrow$ DRF + big mess

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model + big mess
- 2005-08: C++ memory model [PLDI 2008]
  - DRF model

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model + big mess
- 2005-08: C++ memory model [PLDI 2008]
  - DRF model BUT need high performance; mismatched hardware

    $\Rightarrow$ **Relaxed atomics**

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-05: Java memory model [POPL05]
  - DRF model + big mess
- 2005-08: C++ memory model [PLDI 2008]
  - DRF model BUT need high performance; mismatched hardware

    ⇒ Relaxed atomics ⇒ DRF + big mess

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-08: Java, C++, … memory model [POPL05, PLDI08]
  - DRF model + big mess (but after 20 years, convergence at last)

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-08: Java, C++, … memory model [POPL05, PLDI08, **CACM10**]
  - DRF model + big mess (but after 20 years, convergence at last)

My Story (1988 – 2016)

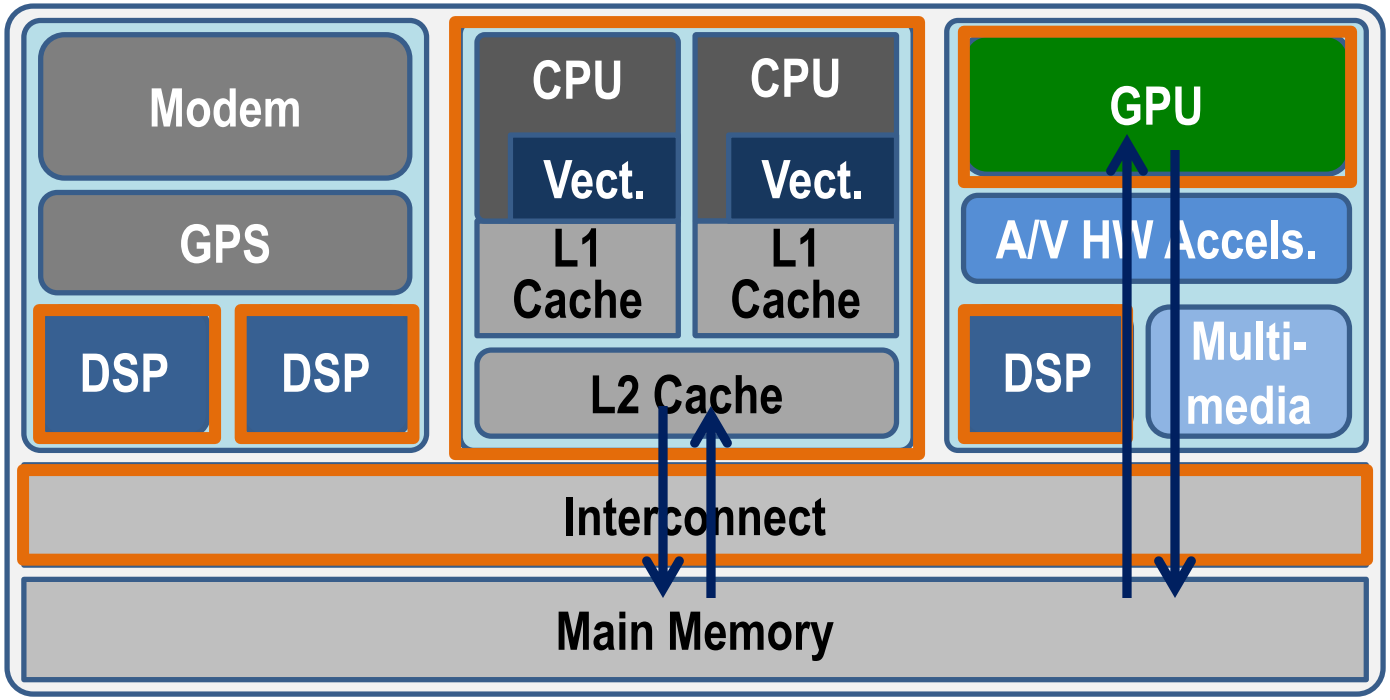
- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-08: Java, C++, … memory model [POPL05, PLDI08, CACM10]
  - DRF model + big mess (but after 20 years, convergence at last)
- 2008-14: Software-centric view for coherence: DeNovo protocol
  - More performance-, energy-, and complexity-efficient than MESI [PACT12, ASPLOS14, ASPLOS15]
    - Began with DPJ's disciplined parallelism [OOPSLA09, POPL11]
    - Identified fundamental, minimal coherence mechanisms
    - Loosened s/w constraints, but still minimal, efficient hardware

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-08: Java, C++, … memory model [POPL05, PLDI08, CACM10]
  - DRF model + big mess (but after 20 years, convergence at last)
- 2008-14: Software-centric view for coherence: DeNovo protocol
  - More performance-, energy-, and complexity-efficient than MESI [PACT12, ASPLOS14, ASPLOS15]
- 2014-: Déjà vu: Heterogeneous systems [ISCA15, Micro15]

Traditional Heterogeneous SoC Memory Hierarchies

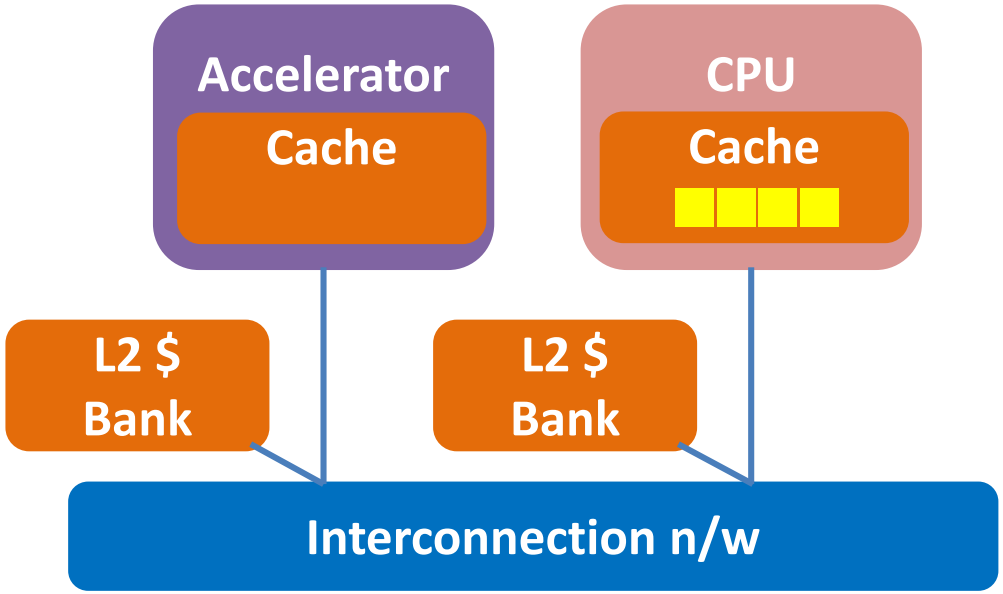
- **Loosely coupled memory hierarchies**
  - **Local memories don't communicate with each other**
  - **Unnecessary data movement**



A tightly coupled memory hierarchy is needed

Tightly Coupled SoC Memory Hierarchies

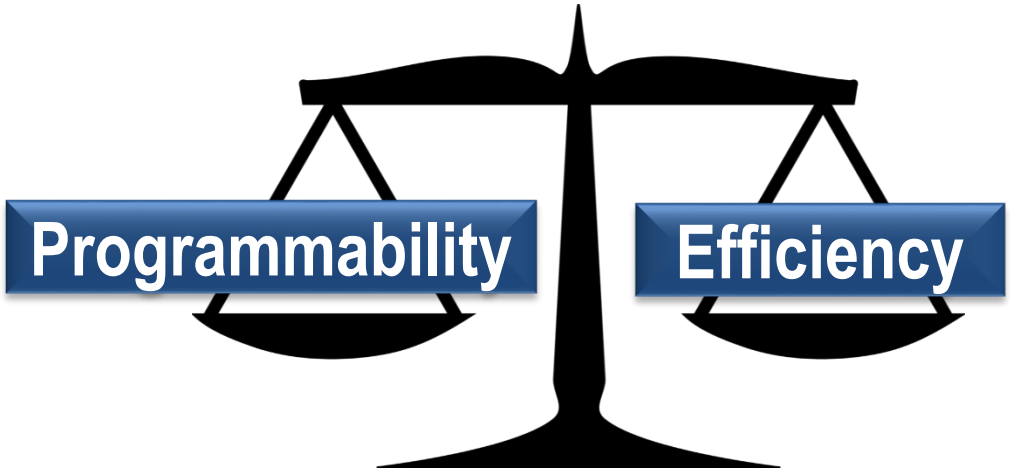
- **Tightly coupled memory hierarchies: unified address space**
  - **Entering mainstream, especially CPU-GPU**
  - **Accelerator can access CPU's data using same address**



Inefficient coherence and consistency

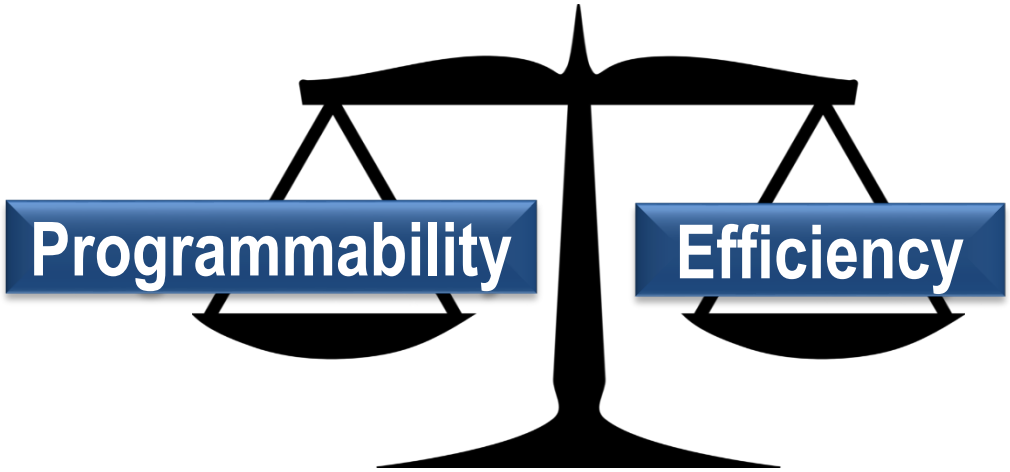
Specialized private memories still used for efficiency

Memory Hierarchies for Heterogeneous SoC



- **Efficient coherence (DeNovo), simple consistency (DRF)**
  [MICRO '15, Top Picks '16 Honorable Mention]
- **Better semantics for relaxed atomics and evaluation**
  [in review]
- **Integrate specialized memories in global address space**
  [ISCA '15, Top Picks '16 Honorable Mention]

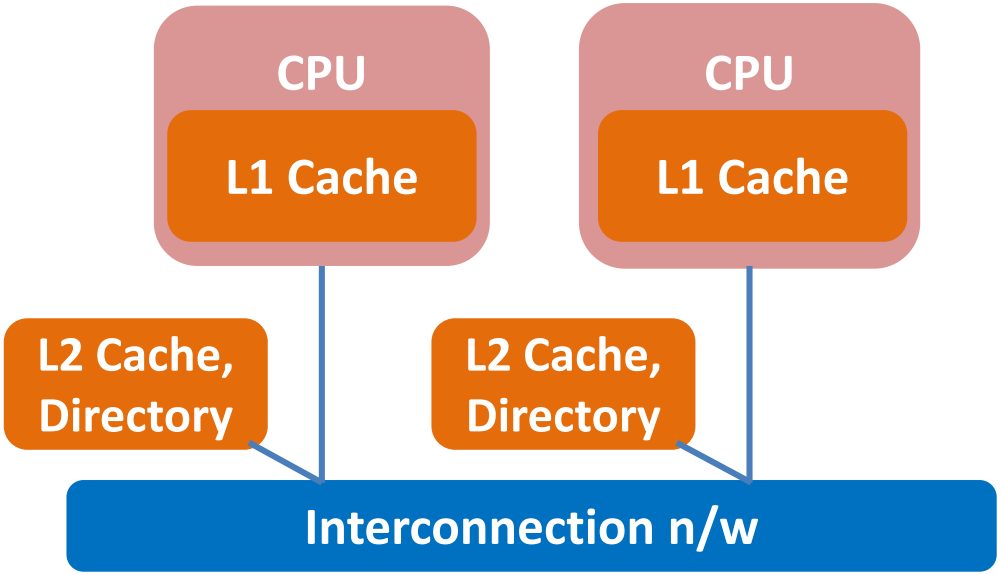
Memory Hierarchies for Heterogeneous SoC



- **Efficient coherence (DeNovo), simple consistency (DRF)**
  [MICRO '15, Top Picks '16 Honorable Mention]
- **Better semantics for relaxed atomics and evaluation**
  [in review]
- **Integrate specialized memories in global address space**
  [ISCA '15, Top Picks '16 Honorable Mention]

Focus: CPU-GPU systems with caches and scratchpads

CPU Coherence: MSI



- Single writer, multiple reader
  - On write miss, get ownership + invalidate all sharers
  - On read miss, add to sharer list

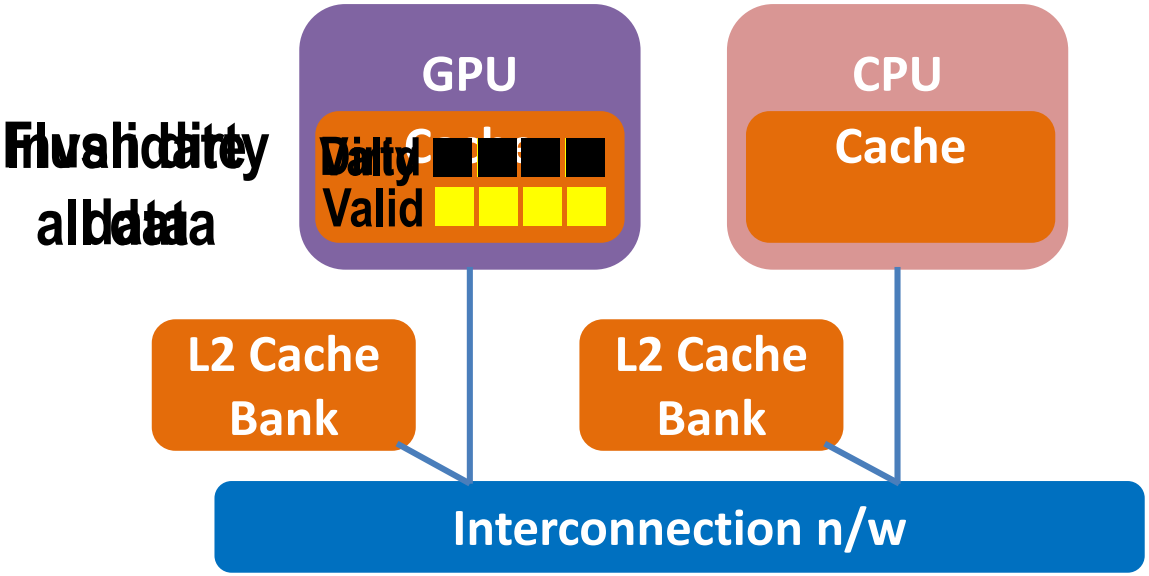
⇒ Directory to store sharer list

⇒ Many transient states

⇒ Excessive traffic, indirection

Complex + inefficient

GPU Coherence with DRF



- With data-race-free (DRF) memory model
  - No data races; synchs must be explicitly distinguished
  - At all synch points
    - Flush all dirty data: Unnecessary writethroughs
    - Invalidate all data: Can't reuse data across synch points
  - Synchronization accesses must go to last level cache (LLC)

Simple, but inefficient at synchronization

GPU Coherence with DRF

- With data-race-free (DRF) memory model
  - No data races; synchs must be explicitly distinguished
  - At all synch points
    - Flush all dirty data: Unnecessary writethroughs
    - Invalidate all data: Can't reuse data across synch points
  - Synchronization accesses must go to last level cache (LLC)

GPU Coherence with HRF

- With ~~data-~~race-free (~~DRF~~) memory model [ASPLOS '14] (heterogeneous; HRF)
  - No ~~data~~ races; synchs must be explicitly distinguished (heterogeneous and their scopes)
  - At all synch points (global)
    - Flush all dirty data: Unnecessary writethroughs
    - Invalidate all data: Can't reuse data across synch points
  - Synchronization accesses must go to last level cache (LLC) (Global)
  - No overhead for locally scoped synchs
- But higher programming complexity

Modern GPU Coherence & Consistency

	Consistency	*Coherence*
De Facto	Data-race-free (DRF) Simple	High overhead on synchs Inefficient
Recent	Heterogeneous-race-free (HRF) Scoped synchronization Complex	No overhead for local synchs Efficient for local synch

Do GPU models (HRF) need to be more complex than CPU models (DRF)?

NO! Not if coherence is done right!

DeNovo+DRF: Efficient AND simpler memory model

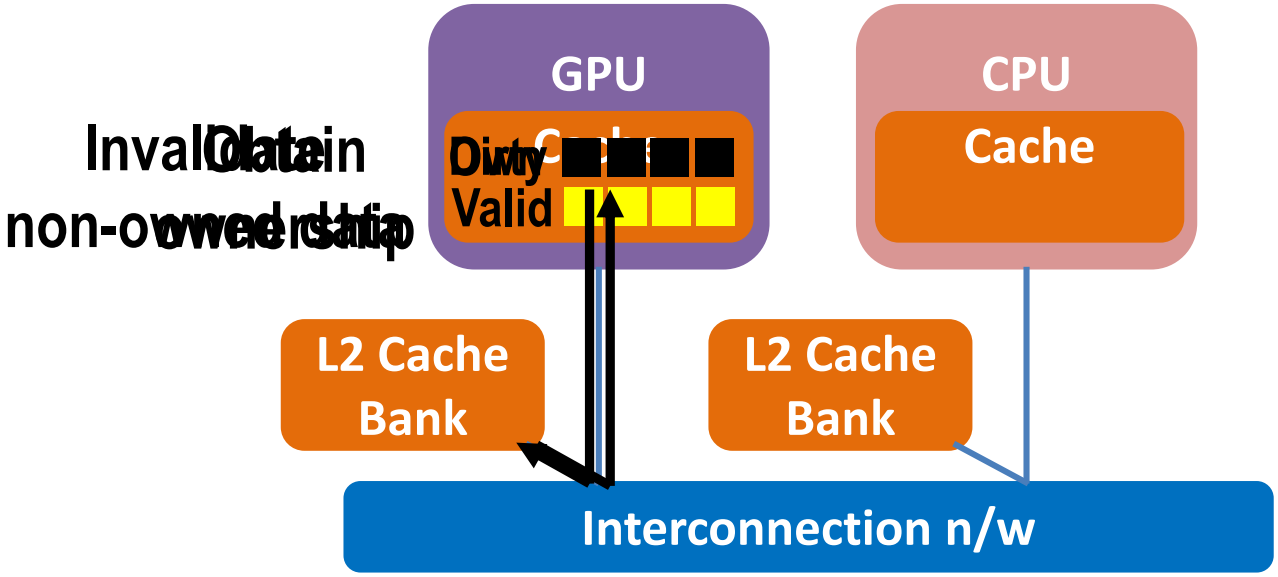
A Classification of Coherence Protocols

- Read hit: Don't return stale data
- Read miss: Find one up-to-date copy

		Invalidator	
		Writer	Reader
Track up-to-date copy	Ownership	MESI	DeNovo
	Writethrough		GPU

- Reader-initiated invalidations
  - No invalidation or ack traffic, directories, transient states
- Obtaining ownership for written data
  - Reuse owned data across synchs (not flushed at synch points)

DeNovo Coherence with DRF



- With data-race-free (DRF) memory model
  - No data races; synchs must be explicitly distinguished
  - At all synch points
    - ~~Flush all dirty data~~ Obtain ownership for dirty data
    - Invalidate ~~all data~~ all non-owned data

    Can reuse owned data can be at L1
  - Synchronization accesses ~~must go to last level cache (LLC)~~
- 3% state overhead vs. GPU coherence + HRF

DeNovo Configurations Studied

- **DeNovo+DRF:**
  - Invalidate all non-owned data at synch points
- **DeNovo+RO+DRF:**
  - Avoids invalidating read-only data at synch points
- **DeNovo+HRF:**
  - Reuse valid data if synch is locally scoped

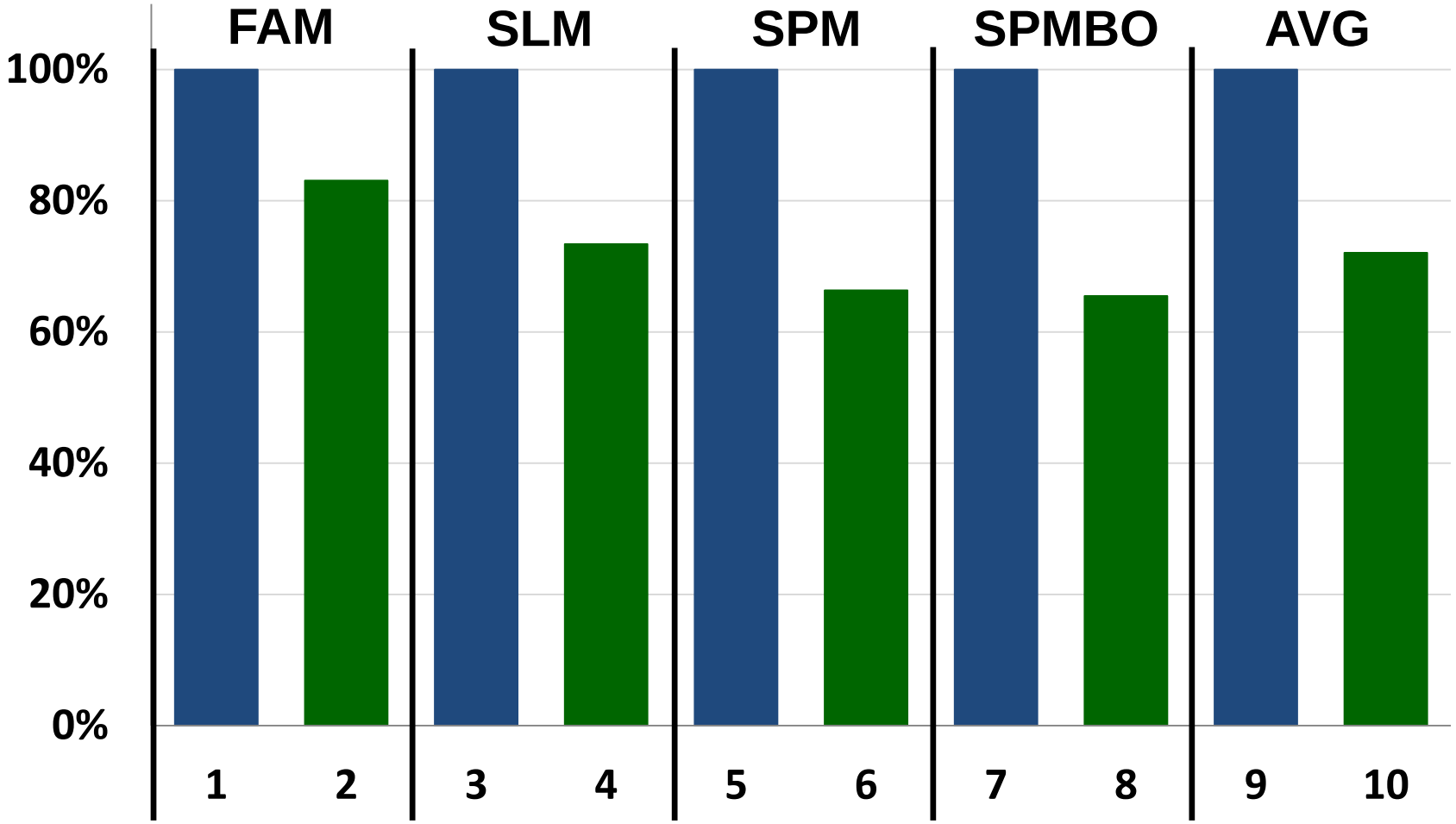
Coherence & Consistency Summary

Coherence + Consistency	Reuse Data		Do Synchs at L1
	Owned	Valid	
GPU + DRF (GD)	X	X	X
GPU + HRF (GH)	local	local	local
DeNovo + DRF (DD)	✓	X	✓
DeNovo-RO + DRF (DD+RO)	✓	read-only	✓
DeNovo + HRF (DH)	✓	local	✓

Evaluation Methodology

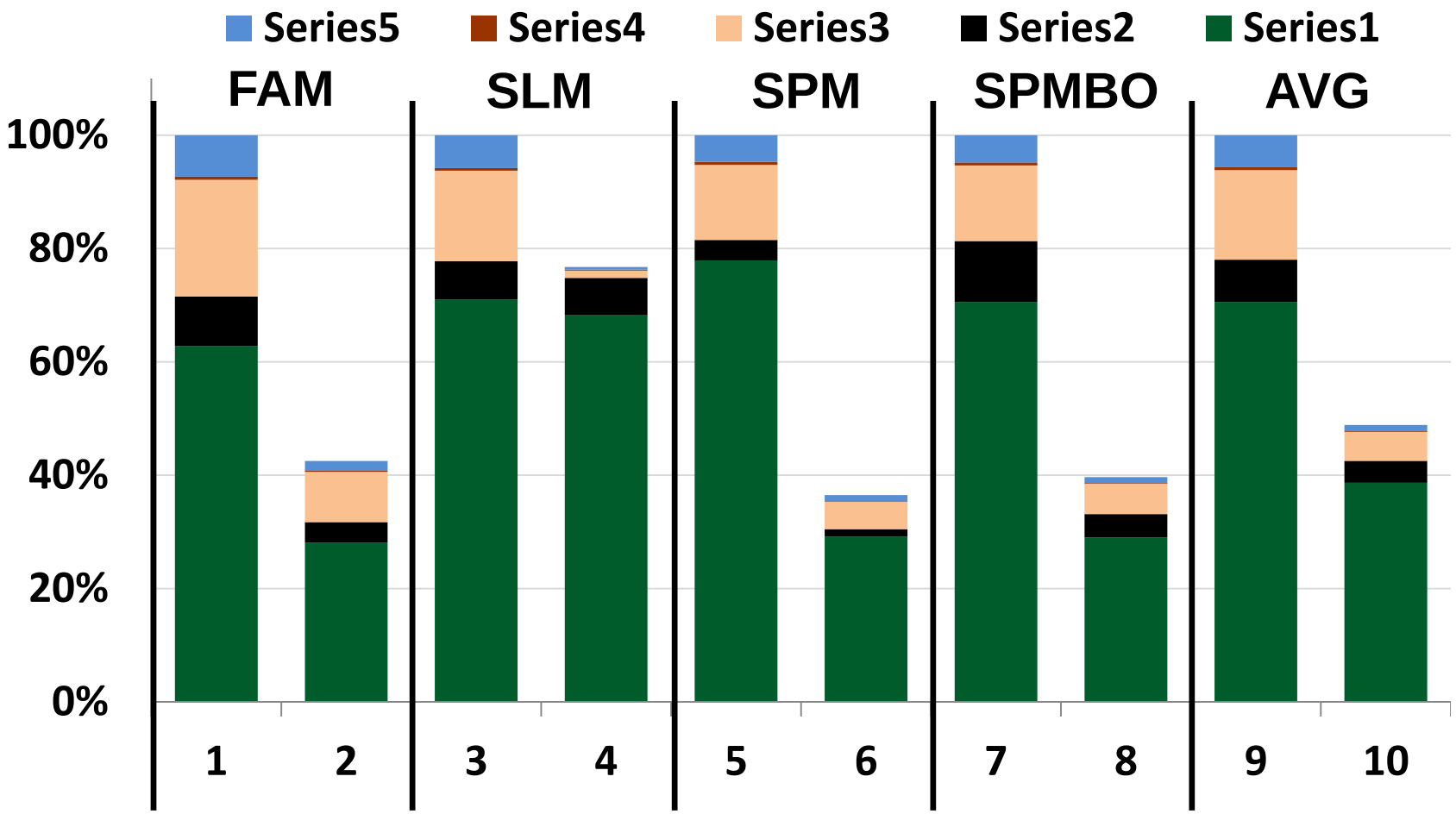
- 1 CPU core + 15 GPU compute units (CU)
  - Each node has private L1, scratchpad, tile of shared L2
- Simulation Environment
  - GEMS, Simics, Garnet, GPGPU-Sim, GPUWattch, McPAT
- Workloads
  - 10 apps from Rodinia, Parboil: no fine-grained synch
    - DeNovo and GPU coherence perform comparably
  - UC-Davis microbenchmarks + UTS from HRF paper:
    - Mutex, semaphore, barrier, work sharing
    - Shows potential for future apps
    - Created two versions of each: global, local/hybrid scoped synch

Global Synch – Execution Time



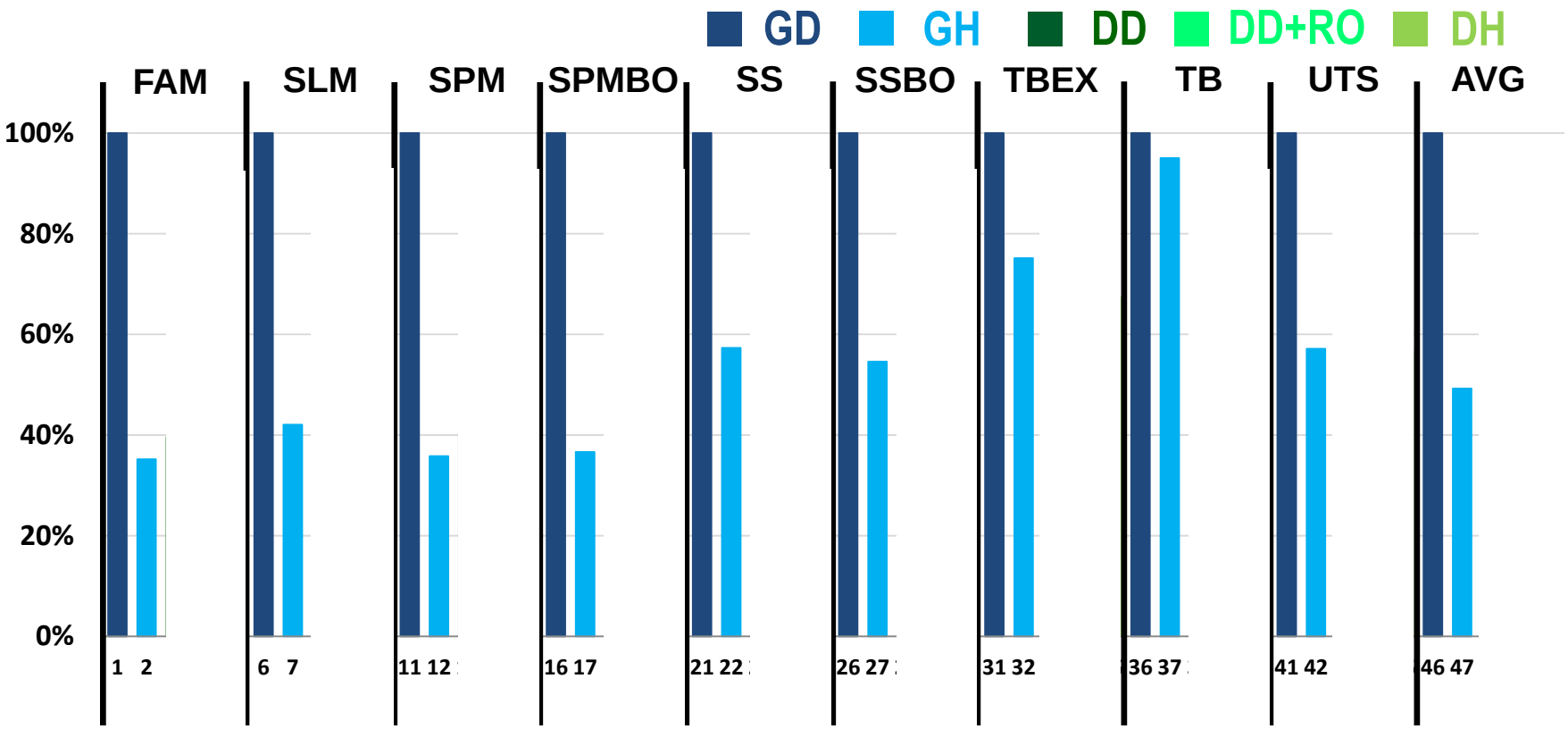
DeNovo has 28% lower execution time than GPU with global synch

Global Synch – Energy



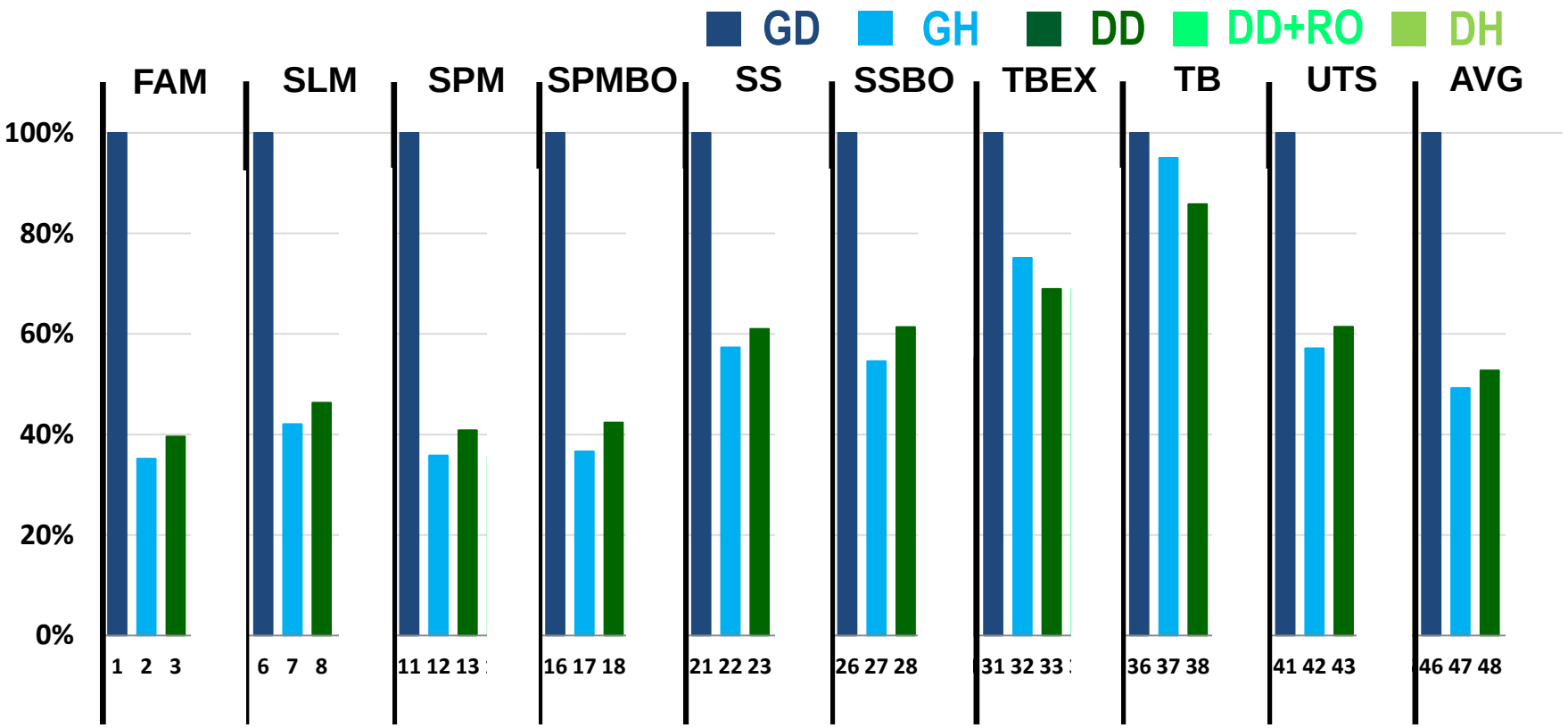
DeNovo has 51% lower energy than GPU with global synch

Local Synch – Execution Time



GPU+HRF is much better than GPU+DRF with local synch [ASPLOS '14]

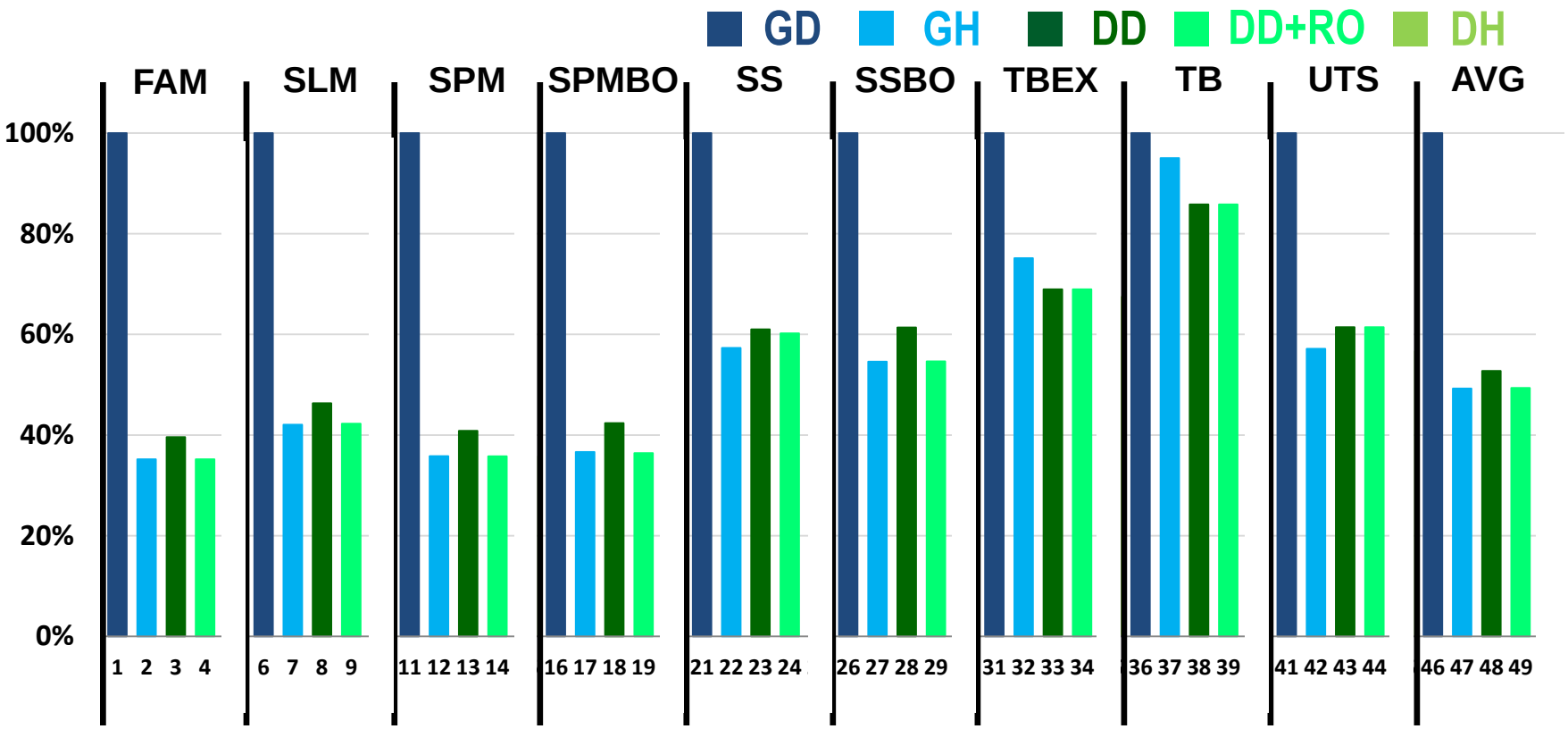
Local Synch – Execution Time



GPU+HRF is much better than GPU+DRF with local synch [ASPLOS '14]

DeNovo+DRF comparable to GPU+HRF, but simpler consistency model

Local Synch – Execution Time

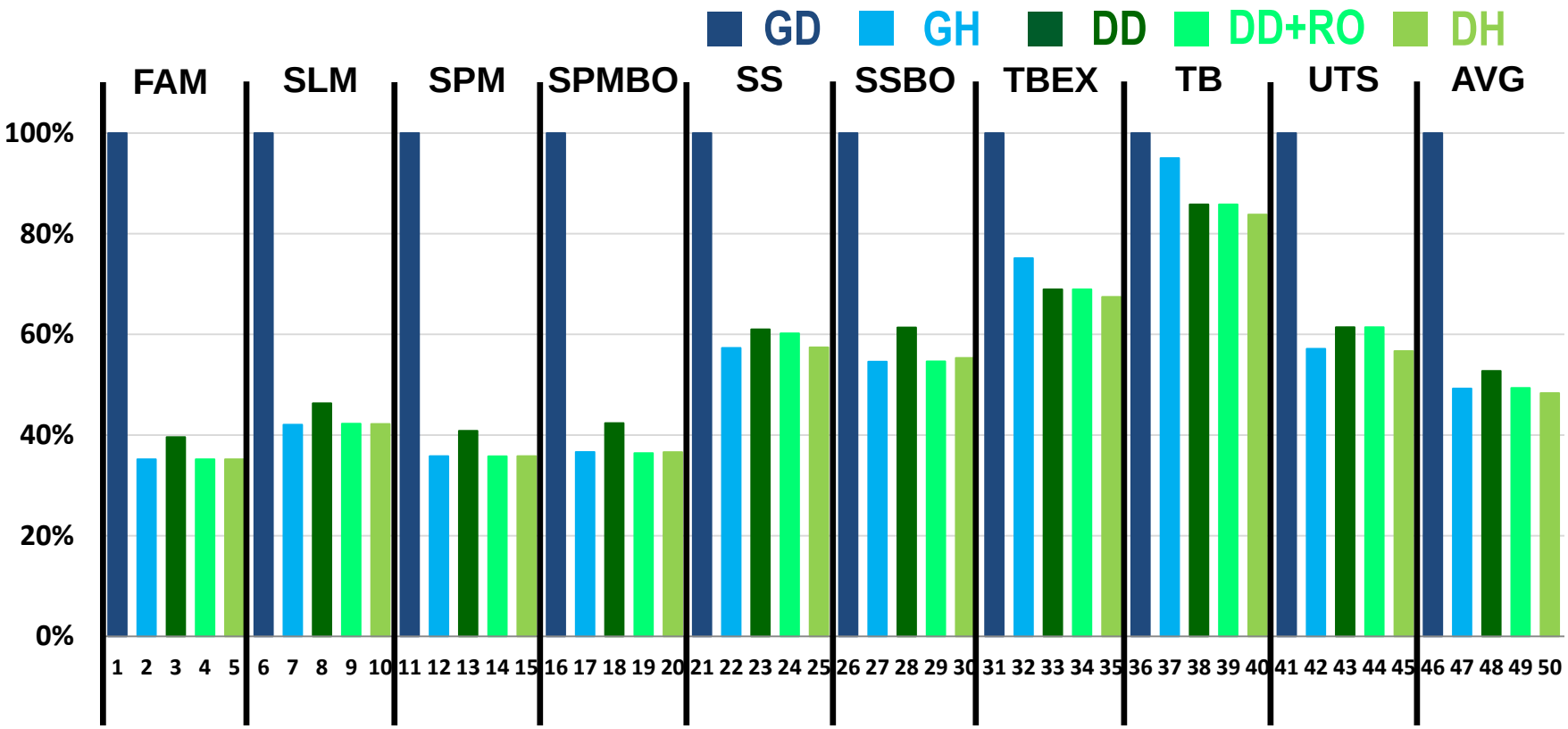


GPU+HRF is much better than GPU+DRF with local synch [ASPLOS '14]

DeNovo+DRF comparable to GPU+HRF, but simpler consistency model

DeNovo-RO+DRF reduces gap by not invalidating read-only data

Local Synch – Execution Time



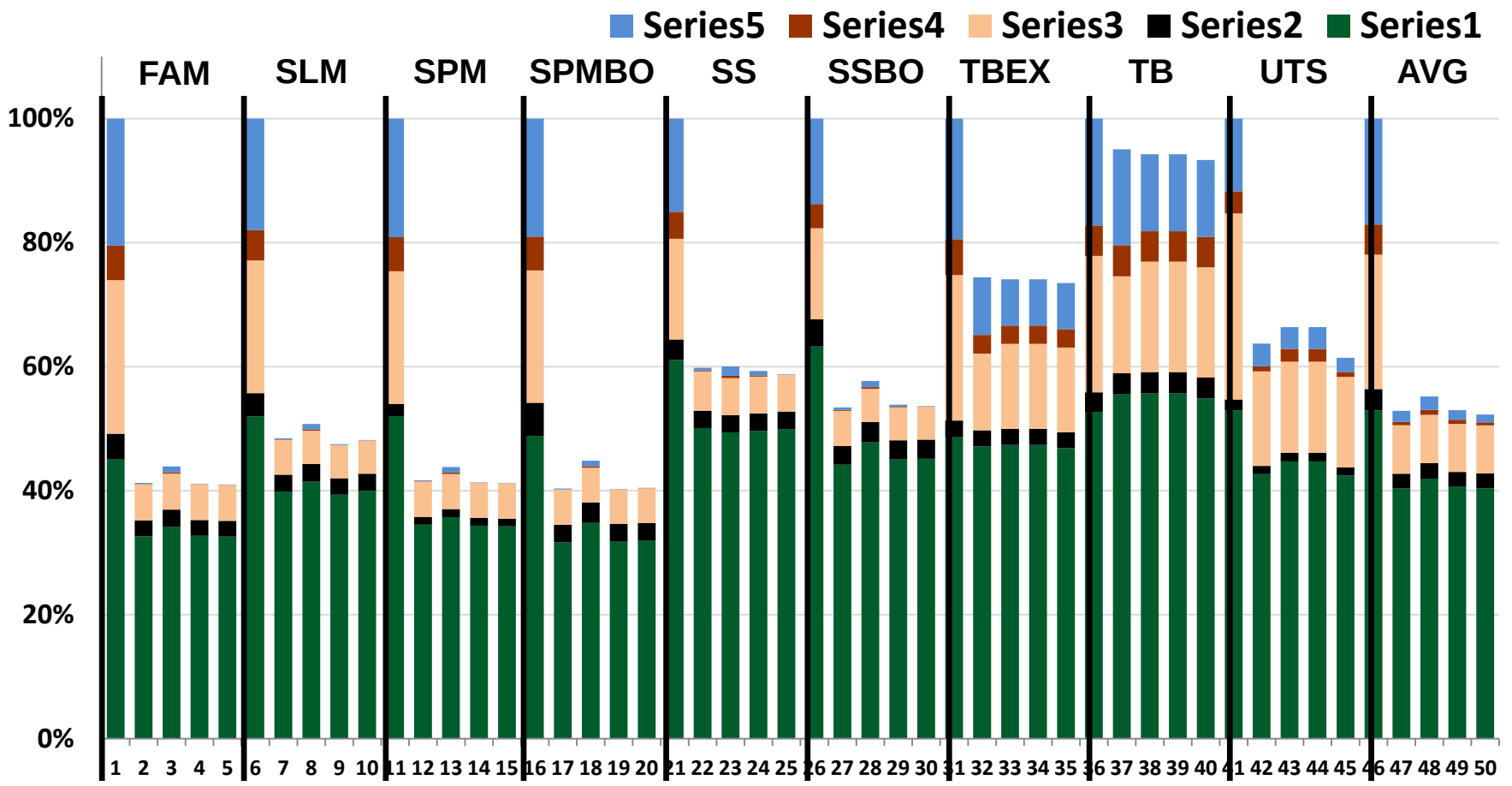
GPU+HRF is much better than GPU+DRF with local synch [ASPLOS '14]

DeNovo+DRF comparable to GPU+HRF, but simpler consistency model

DeNovo-RO+DRF reduces gap by not invalidating read-only data

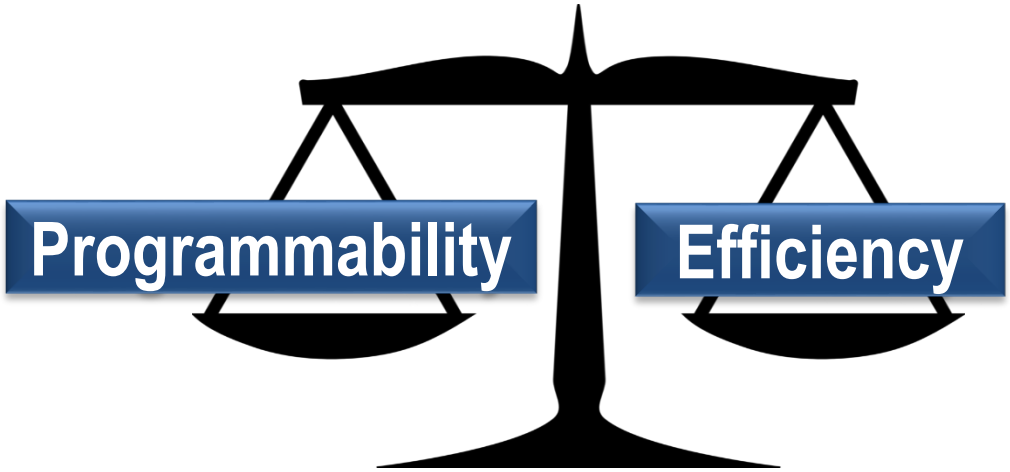
DeNovo+HRF is best, if consistency complexity acceptable

Local Synch – Energy



Energy trends similar to execution time

Memory Hierarchies for Heterogeneous SoC



- Efficient coherence (DeNovo), simple consistency (DRF)
  [MICRO '15, Top Picks '16 Honorable Mention]
- Better semantics for relaxed atomics and evaluation
  [in review]
- Integrate specialized memories in global address space
  [ISCA '15, Top Picks '16 Honorable Mention]

Background: Atomic Ordering Constraints

Consistency	Allowed Reorderings			Retains SC semantics?
	Data Data	Synch Data	Synch Synch	
DRF0	✓	Data - Acq, Rel - Data	X	✓

- Racy synchronizations implemented with atomics
- DRF0: simple and efficient
  - All atomics assumed to order data accesses
  - Atomics can't be reordered with atomics
  - Ensures SC semantics

Background: Atomic Ordering Constraints

Consistency	Allowed Reorderings			Retains SC semantics?
	Data Data	Synch Data	Synch Synch	
DRF0	✓	Data - Acq, Rel - Data	X	✓
DRF1	✓	DRF0 + unpaired	X	✓

- DRF1 uses more SW info to identify unpaired atomics
  - Unpaired atomics do not order any data accesses
  - Unpaired avoid invalidations/flushes for heterogeneous systems
  - Ensures SC semantics

Background: Atomic Ordering Constraints

Consistency	Allowed Reorderings			Retains SC semantics?
	Data Data	Synch Data	Synch Synch	
DRF0	✓	Data - Acq, Rel - Data	X	✓
DRF1	✓	DRF0 + unpaired	X	✓
DRF1 + relaxed atomics	✓	DRF1 + relaxed	relaxed	X

- Use more SW info to identify relaxed atomics
  - Reordered, overlapped with all other memory accesses
  - BUT violates SC: very hard to formalize and reason about

Relaxed Atomics

- **CPUs: clear directions to programmers to avoid**
  - **Only expert programmers of performance critical code use**
  - **C, C++, Java: still no acceptable semantics!**

- **Heterogeneous Systems**
  - **OpenCL, HSA, HRF: adopted similar consistency to DRF0**
  - **But generally use simple, SW-based coherence**
  - **Cost of staying away from relaxed atomics too high!**
  - **DeNovo helps, but relaxed atomics still beneficial**

Can we utilize additional SW info to give SC even with relaxed atomics?

How Are Relaxed Atomics Used?

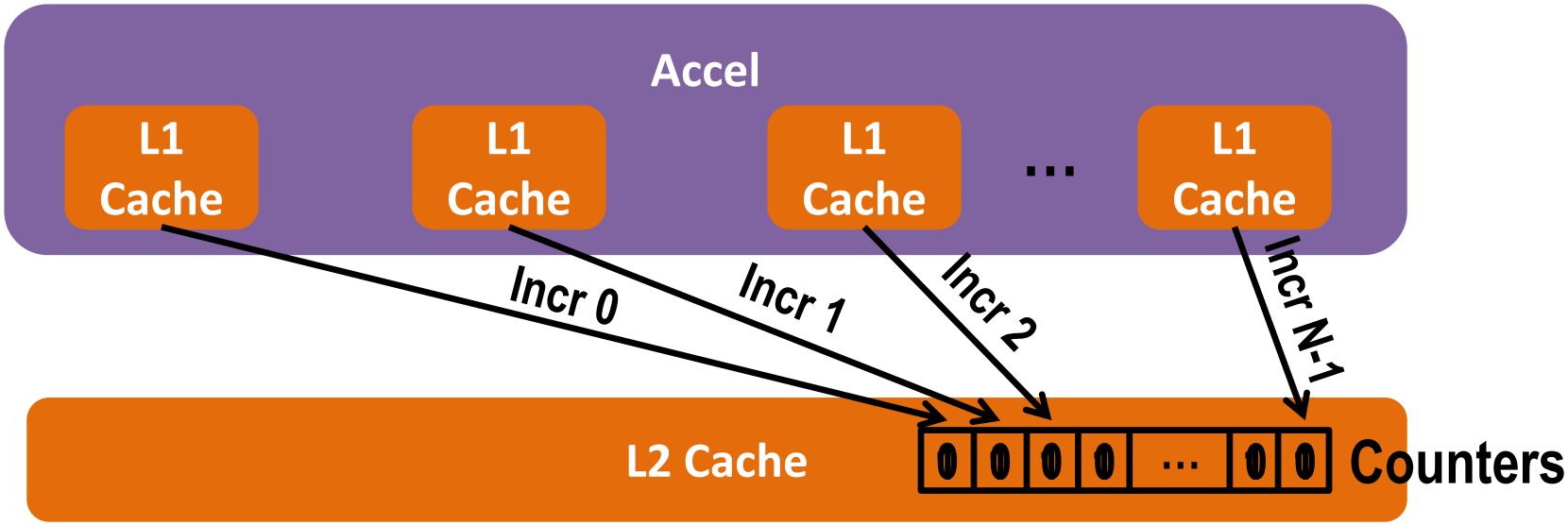
- **Examined how relaxed atomics are used**
  - **Collected examples from developers**
  - **Categorized which relaxations are beneficial**
  - **Extended DRF0/1 to allow varying levels of atomic relaxation**

- **Contributions:**
  1. **DRF2: preserves SC-centric semantics**
  2. **Evaluated benefit of using relaxed atomics**
     - **Usually small benefit (≤ 6% better perf for microbenchmarks)**
     - **Gains sometimes significant (up to 60% better perf for PR)**

Relaxed Atomic Use Cases

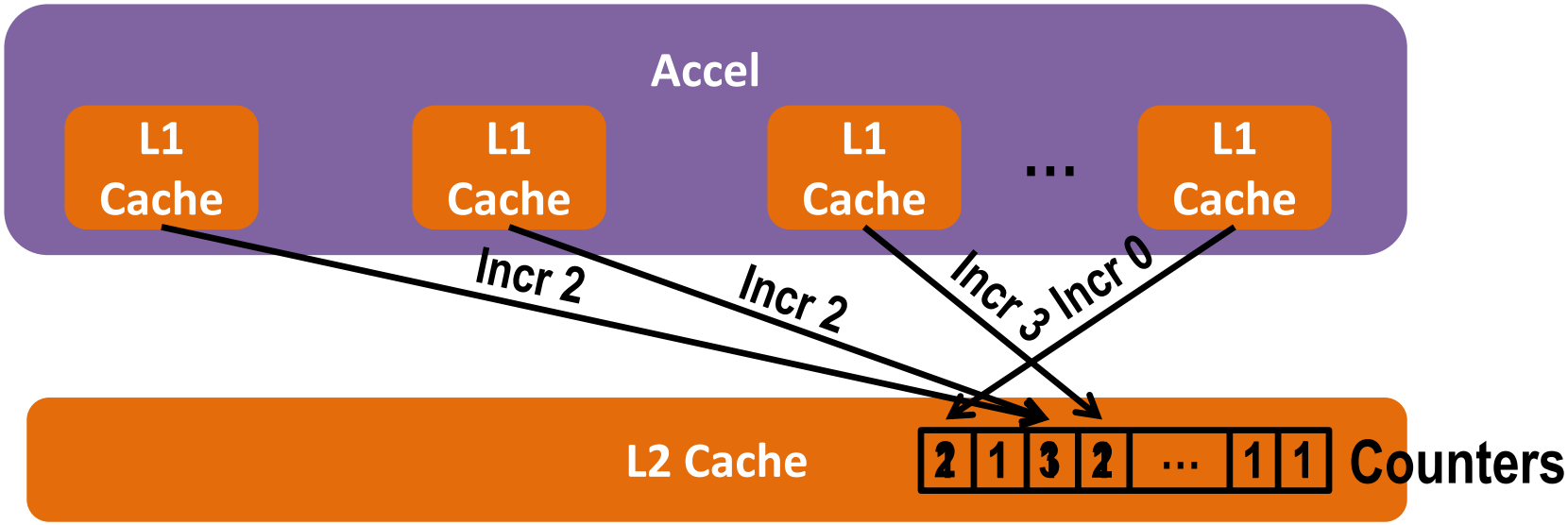
- How existing apps use relaxed atomics:
  1. Unpaired
  2. Commutative
  3. Non-Ordering
  4. Quantum

Commutative – Event Counter



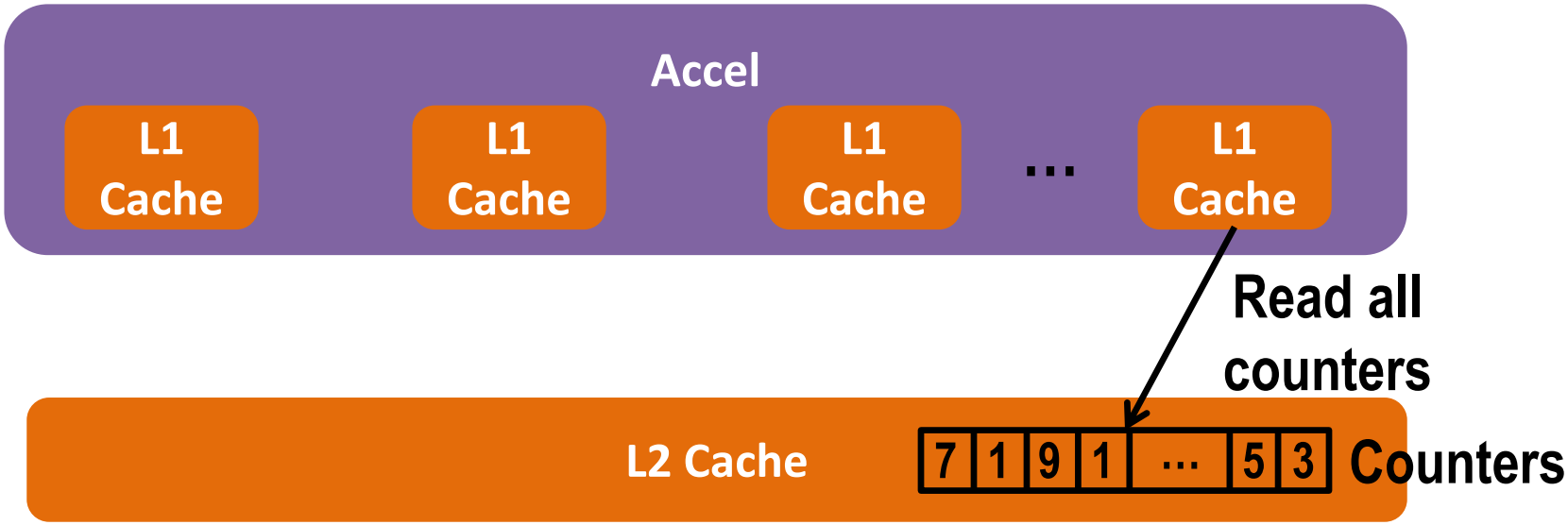
1. Threads concurrently update counters
   - Read part of a data array, updated its counter

Commutative – Event Counter (Cont.)



1. Threads concurrently update counters
   - Read part of a data array, updated its counter
   - Increments race, so have to use atomics

Commutative – Event Counter (Cont.)



1. Threads concurrently update counters
   - Read part of a data array, updated its counter
   - Increments race, so have to use atomics
2. Once all threads done, one thread reads all counters

Commutative – Event Counter (Cont.)

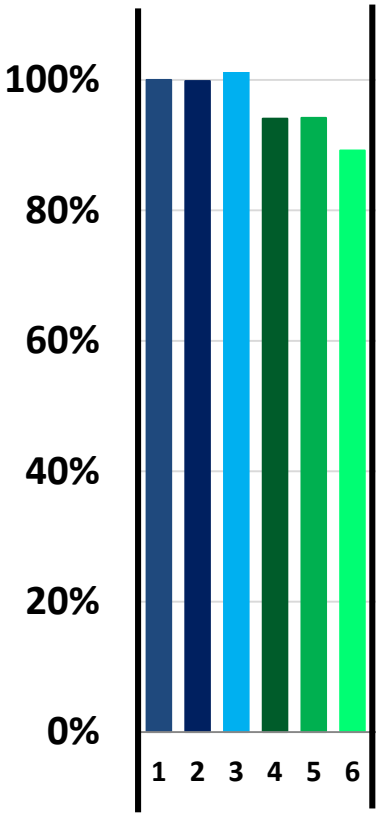
- DRF0 and DRF1 ensure SC semantics
  - DRF0 overly restrictive: increments do not order data
  - DRF1: little benefit because no reuse in data
- Relaxed atomics:
  - Reorder, overlap atomics from same thread
  - Commutative increments – result is same regardless of order
- DRF2
  - Distinguish commutative: intermediate values not observable
  - Define commutative races
  - Program is DRF2 if DRF1 and no commutative races
  - DRF2 systems give efficiency and SC to DRF2 programs

Evaluation Methodology

- **Similar simulation environment**
  - **Extended to compare DRF0, DRF1, and DRF2**
  - **Do not compare to HRF because few apps use scopes**

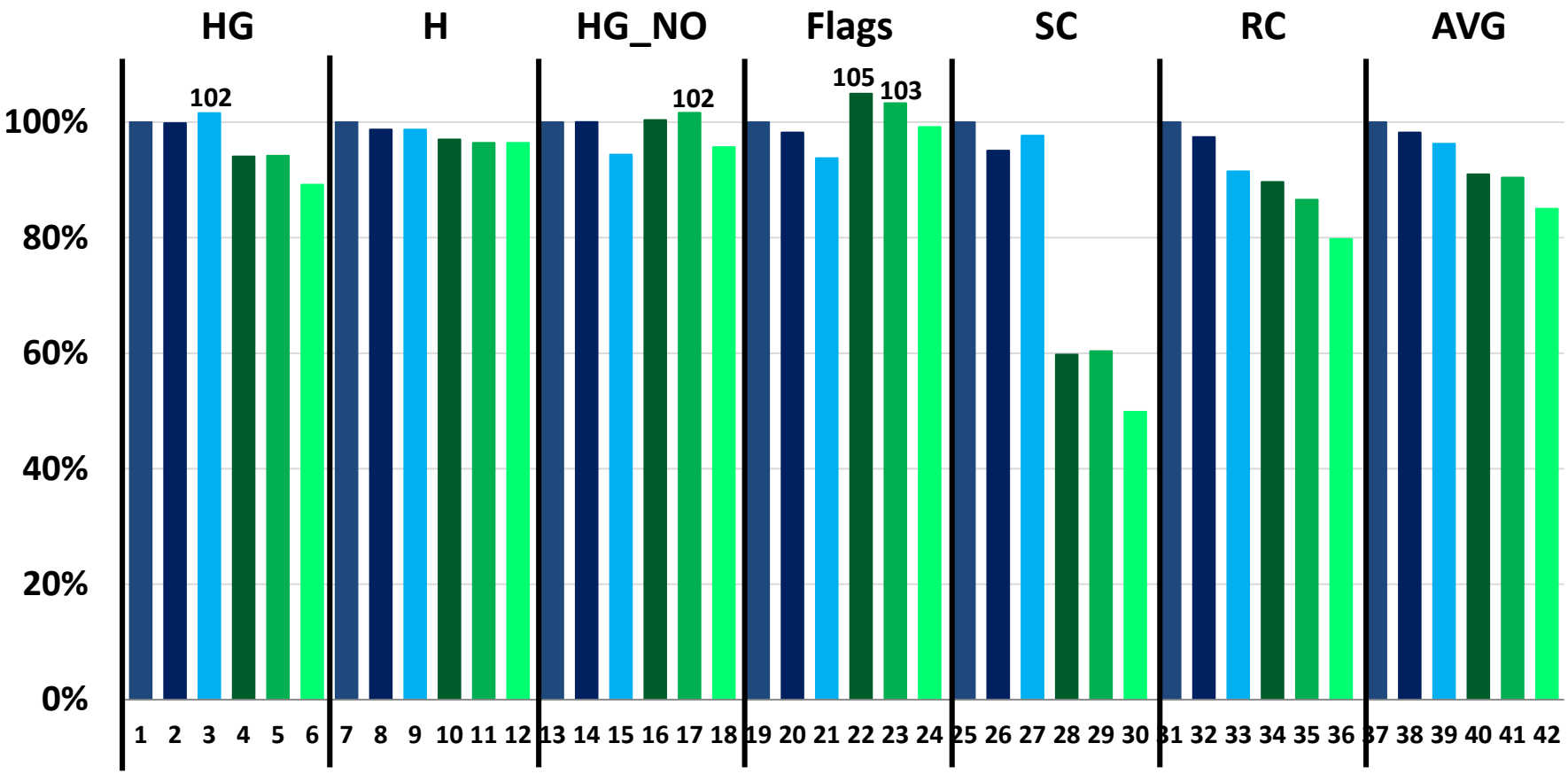
- **Workloads**
  - **Microbenchmarks**
    - **Traditional use cases for relaxed atomics**
    - **Stress memory system (high contention)**
  - **All benchmarks from major suites with ≥ 2% global atomics**
    - **UTS, PageRank (PR), Betweeness Centrality (BC)**
    - **Show 4 representative graphs for PR and BC**

Relaxed Atomic Microbenchmarks – Execution Time



GD0 = GPU coherence with DRF0
GD1 = GPU coherence with DRF1
GD2 = GPU coherence with DRF2
DD0 = DeNovo coherence with DRF0
DD1 = DeNovo coherence with DRF1
DD2 = DeNovo coherence with DRF2

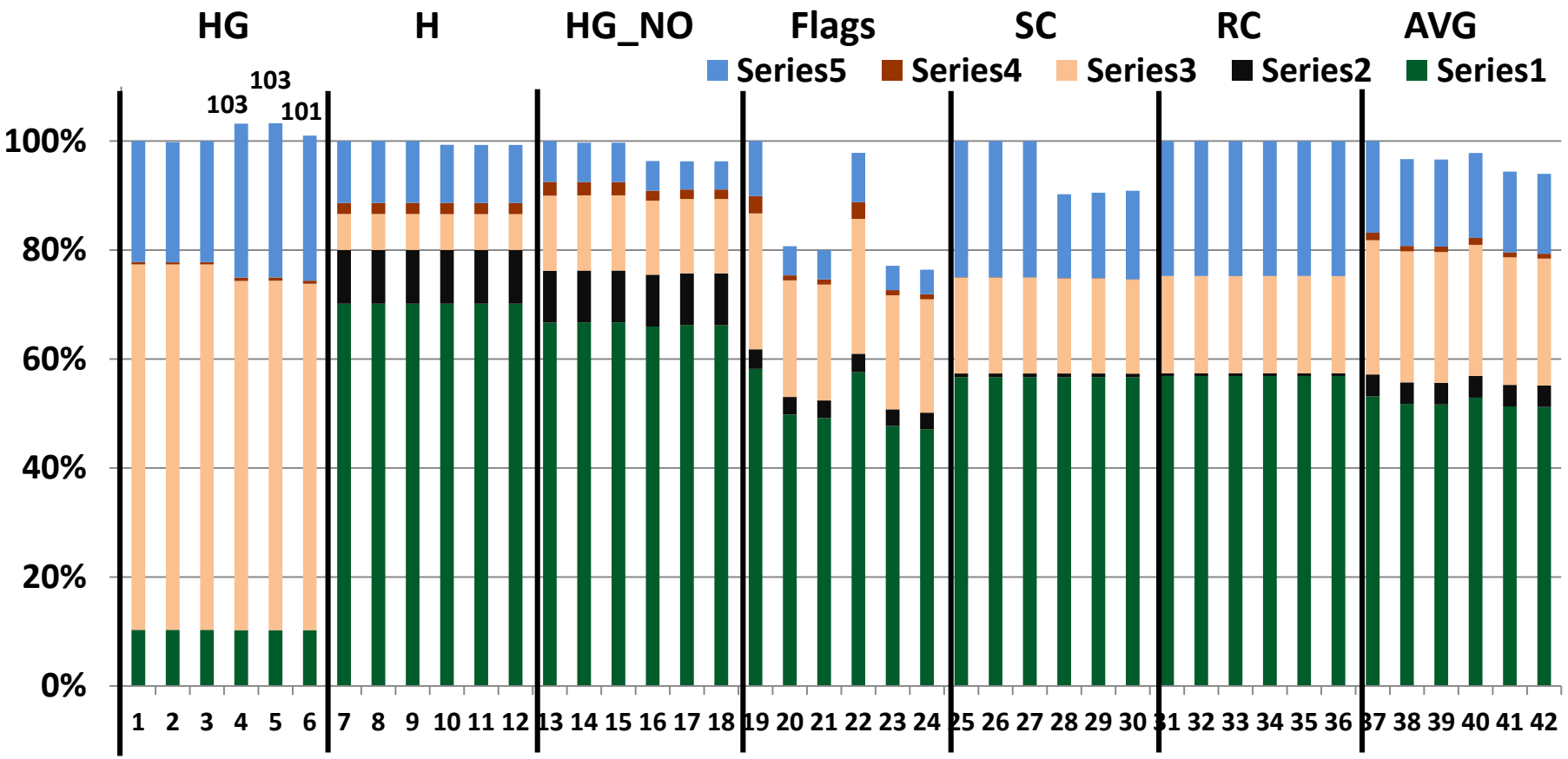
Relaxed Atomic Microbenchmarks – Execution Time



DRF1 and DRF2 do not significantly affect performance (≤ 6% on average)

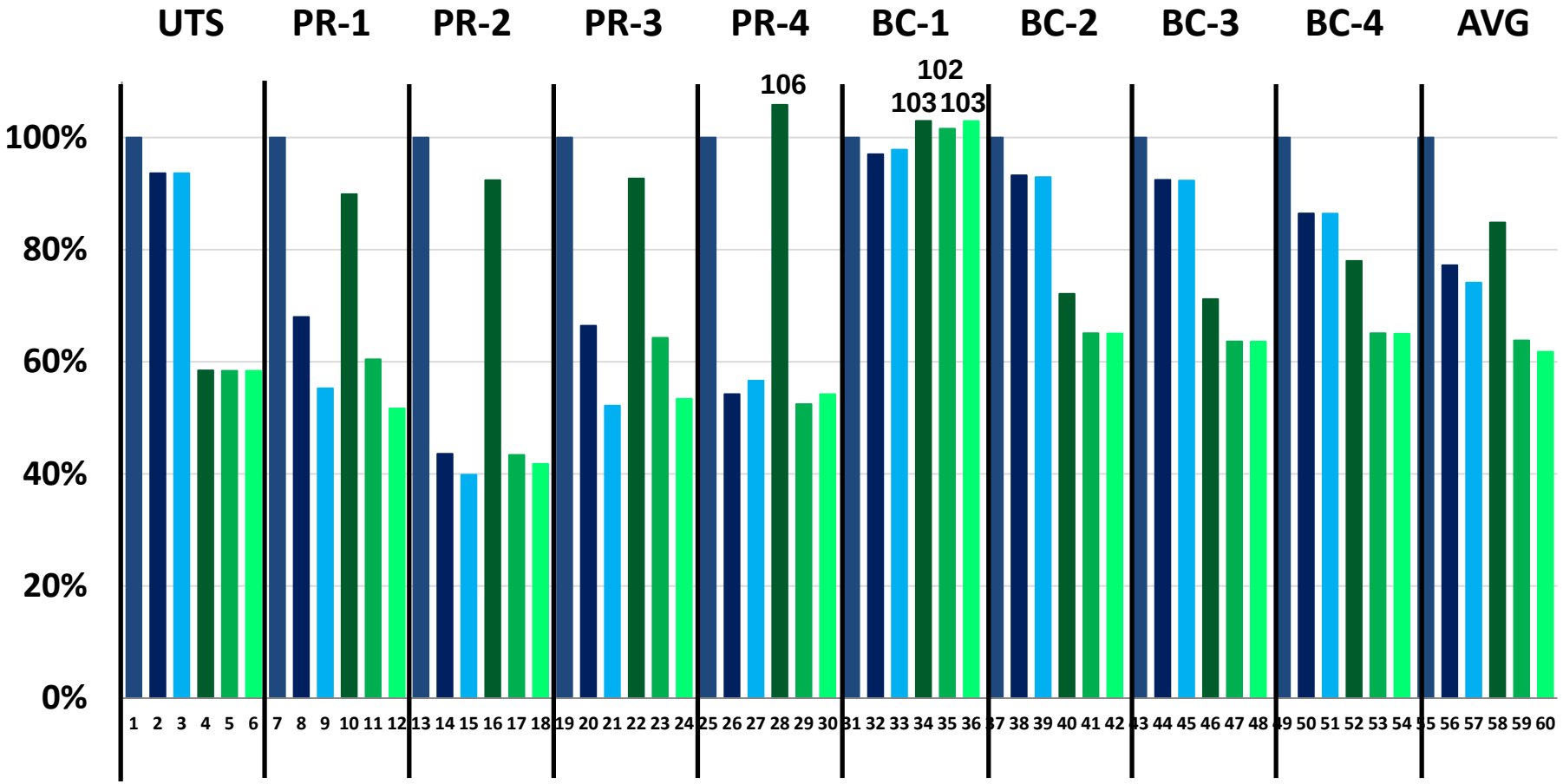
DeNovo exploits synch reuse, outperforms GPU (DRF2: 11% avg)

Relaxed Atomic Microbenchmarks – Energy



Energy trends similar to execution time

Relaxed Atomic Apps – Execution Time

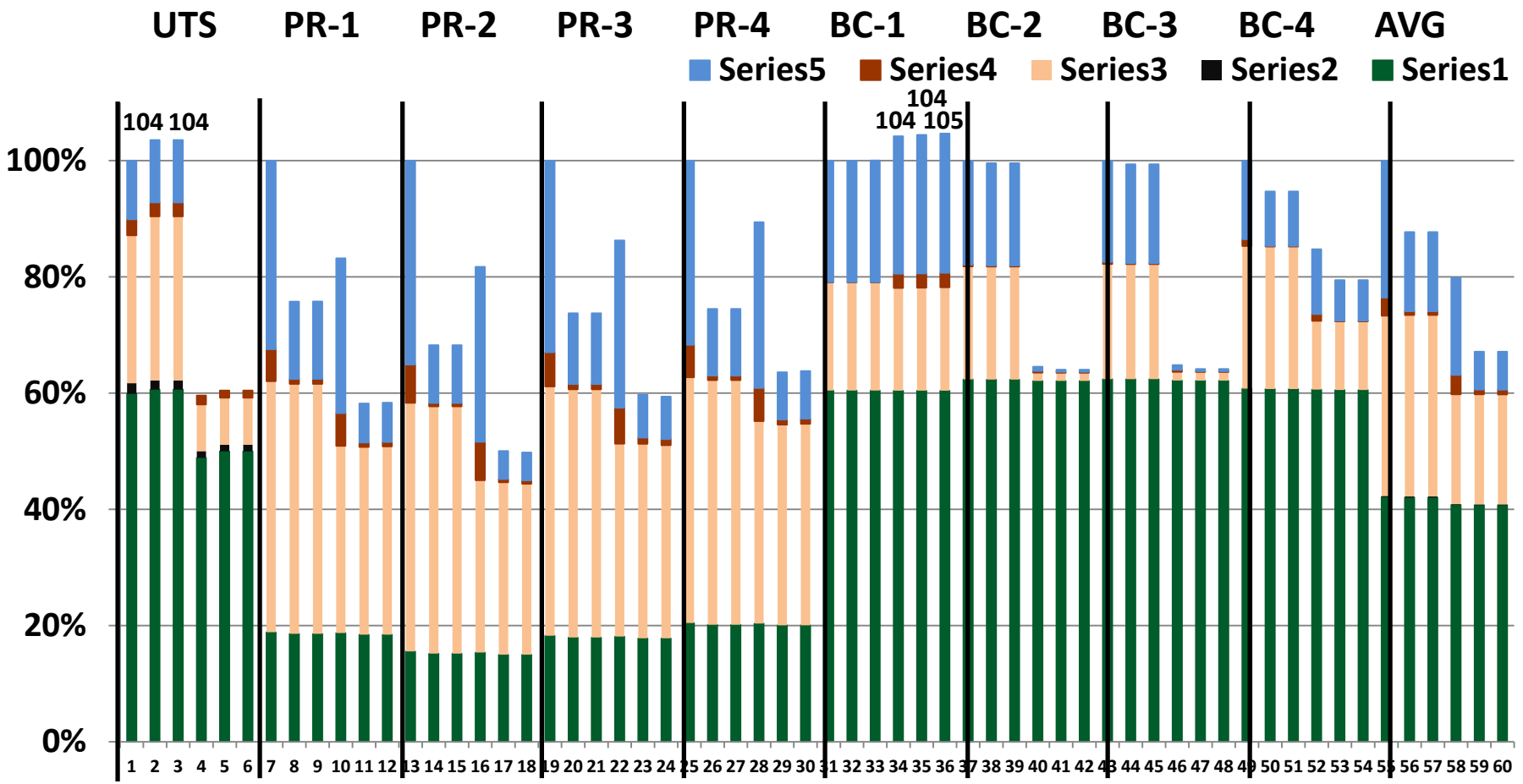


Weakening consistency model helps a lot for PageRank (up to 60% for GPU)

DRF1 avoids costly synchronization overhead (23% average improvement)

DRF2 overlaps atomics (up to 21% better than DRF1)

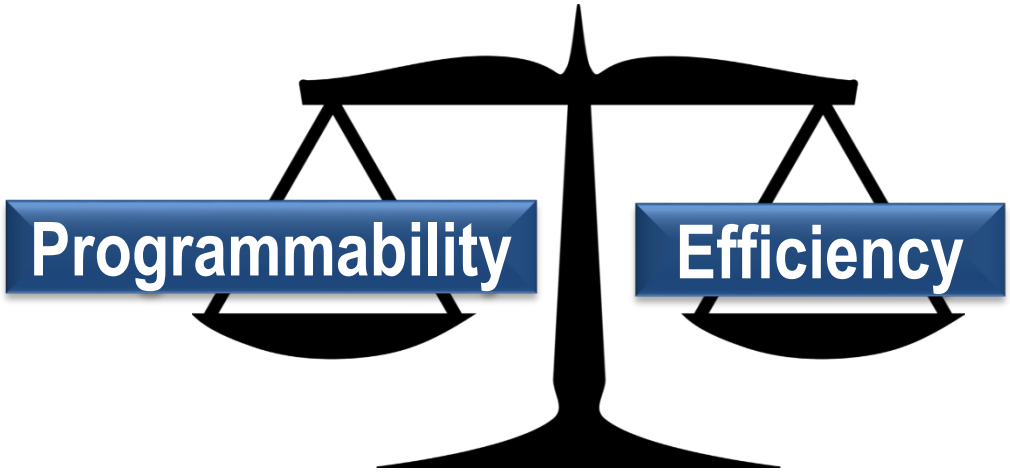
Relaxed Atomic Apps – Energy



Energy somewhat similar to execution time trends

DRF2: DeNovo's data and synch reuse reduces energy (23% avg vs GPU)

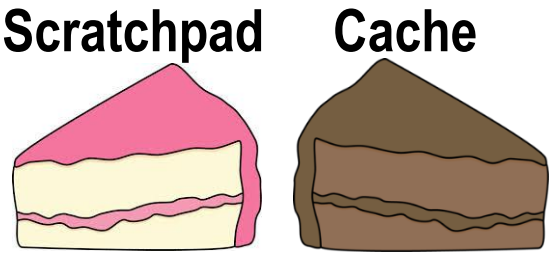
Memory Hierarchies for Heterogeneous SoC



- Efficient coherence (DeNovo), simple consistency (DRF)
  [MICRO '15, Top Picks '16 Honorable Mention]
- Better semantics for relaxed atomics and evaluation
  [in review]
- Integrate specialized memories in global address space
  [ISCA '15, Top Picks '16 Honorable Mention]

Specialized Memories for Efficiency

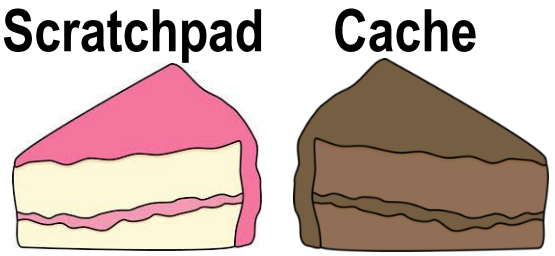
- Heterogeneous SoCs use specialized memories for energy
- E.g., scratchpads, FIFOs, stream buffers, …



	Scratchpad	Cache
Directly addressed: no tags/TLB/conflicts	✓	X
Compact storage: no holes in cache lines	✓	X

Specialized Memories for Energy

- Heterogeneous SoCs use specialized memories for energy
- E.g., scratchpads, FIFOs, stream buffers, …

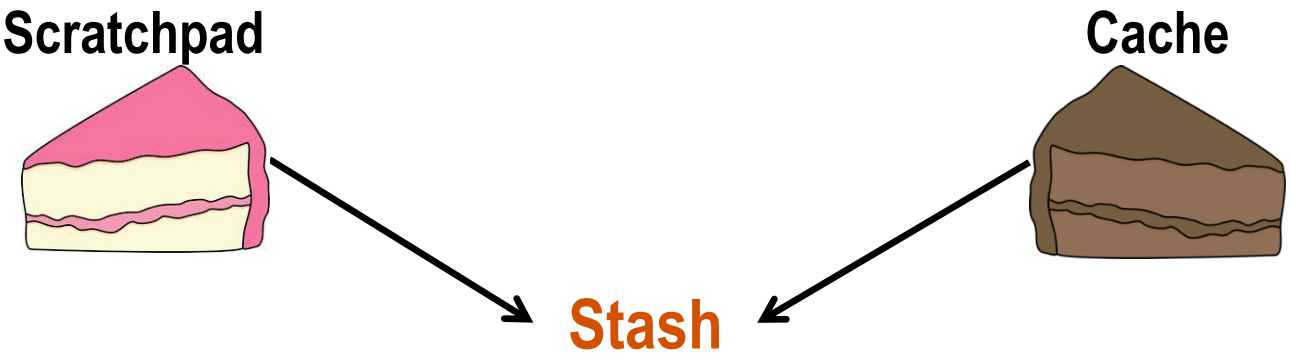


	Scratchpad	Cache
Directly addressed: no tags/TLB/conflicts	✓	X
Compact storage: no holes in cache lines	✓	X
Global address space: implicit data movement	X	✓
Coherent: reuse, lazy writebacks	X	✓

Can specialized memories be globally addressable, coherent?

Can we have our scratchpad and cache it too?

Can We Have Our Scratchpad and Cache it Too?



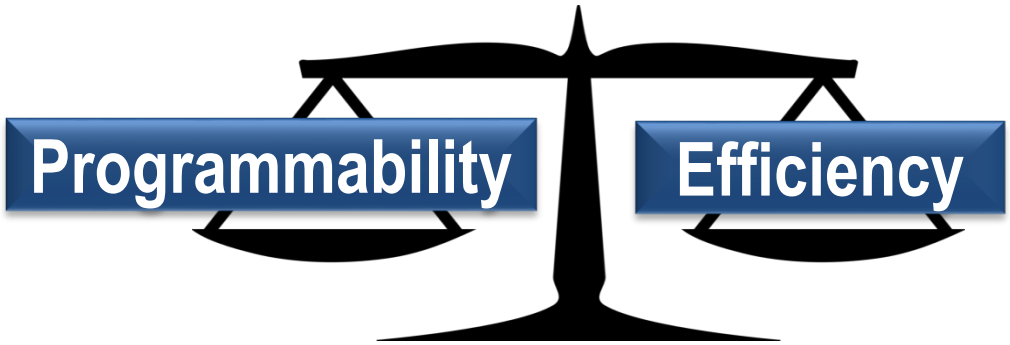
+ Directly addressable
+ Compact storage

+ Global address space
+ Coherent

- Make specialized memories globally addressable, coherent
  - Efficient address mapping
  - Efficient coherence protocol
- Focus: CPU-GPU systems with scratchpads and caches
  - Up to 31% less execution time, 51% less energy

Conclusion

- Tension between programmability and efficiency
  - Coherence: performs poorly for emerging apps
  - Consistency: complicated, relaxed atomics worsen
  - Specialized memories: not visible in global address space



- Insight: adjust coherence and consistency complexity
  - Efficient coherence [MICRO '15, TP '16 HM]
  - DRF consistency model [MICRO '15, TP '16 HM, in submission]
  - Specialized mems in global addr space [ISCA '15, TP '16 HM]
  - Future: optimize DeNovo; integrate more specialized mems; interface

My Story (1988 – 2016)

- 1988 to 1989: What is a memory model?
  - What value can a read return?
- 1990s: Software-centric view: Data-race-free (DRF) model [ISCA90, …]
  - Sequential consistency for data-race-free programs
- 2000-08: Java, C++, … memory model [POPL05, PLDI08, CACM10]
  - DRF model + big mess (but after 20 years, convergence at last)
- 2008-14: Software-centric view for coherence: DeNovo protocol
  - More performance-, energy-, and complexity-efficient than MESI [PACT12, ASPLOS14, ASPLOS15]
- 2014-16: Déjà vu: Heterogeneous systems [ISCA15, Micro15]
  - Coherence, consistency, global addressability