Chasing Away RAts: Semantics and Evaluation for Relaxed Atomics on Heterogeneous Systems

Matthew D. Sinclair[*], Johnathan Alsop[^], Sarita V. Adve[+]

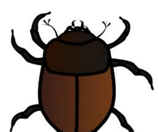
* University of Wisconsin-Madison
^ AMD Research
+ University of Illinois @ Urbana-Champaign
sinclair@cs.wisc.edu, hetero@cs.illinois.edu

Sponsors: NSF and Center for Future Architectures Research (C-FAR) Research center (co-sponsored by SRC and DARPA)

"Everyone (thinks they) can ~~cook~~" use relaxed atomics (RAts)

Correctness ~~Health code~~ violations:

Incorrect usage  No formal definition  Not portable

 Hard to debug  Out-of-thin-air values 

No Formal Specification for Relaxed Atomics

C++17 "specification" for relaxed atomics

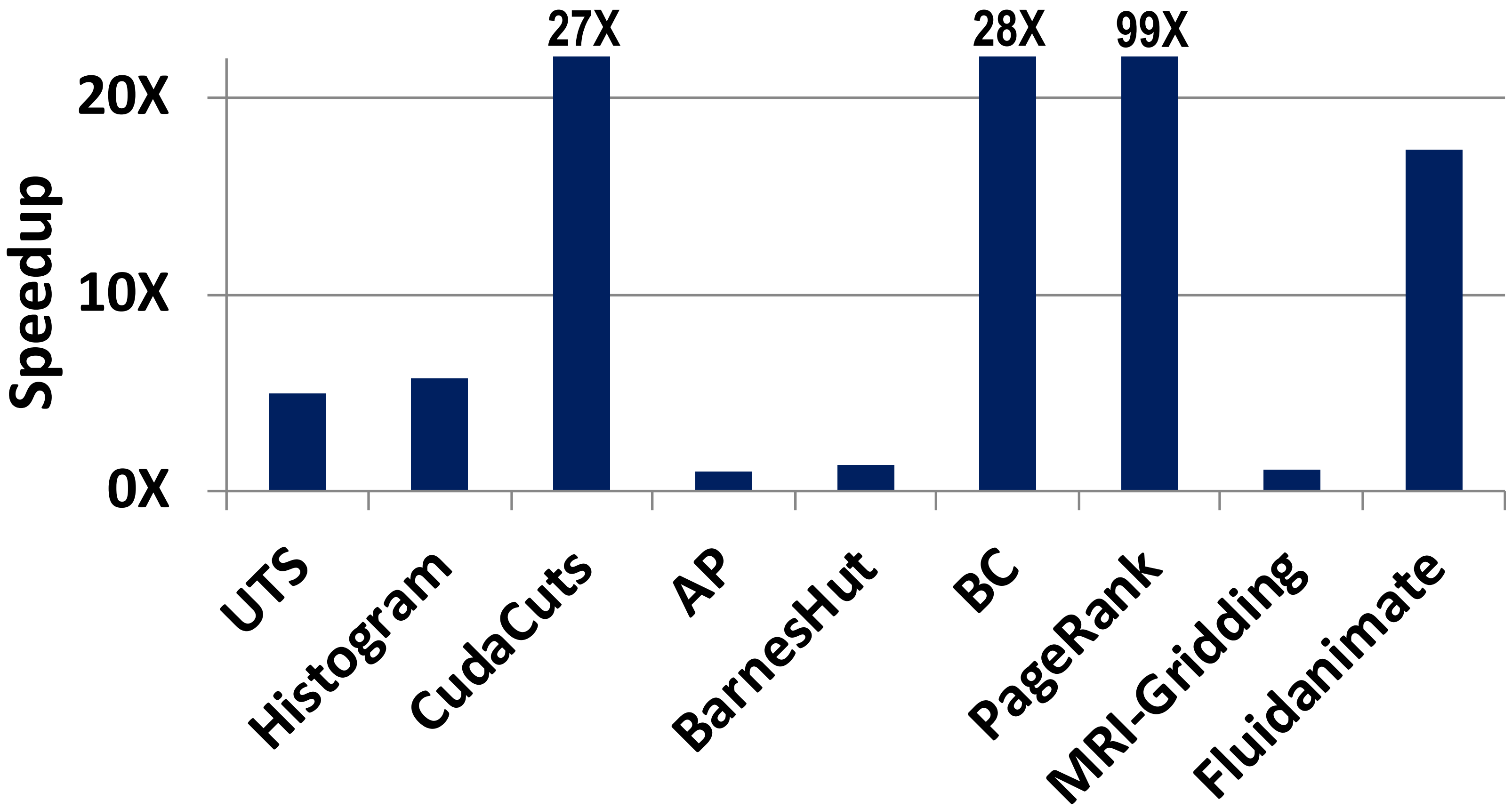
- Races that don't order other accesses
- Implementations should ensure no “out-of-thin-air” values are computed that circularly depend on their own computation

“C++ (relaxed) atomics were the **worst idea ever**. I just spent days (and days) trying to get something to work. … My example **only has 2 addresses and 4 accesses, it shouldn’t be this hard**. Can you help?”

- Email from employee at major research lab

Formal specification for relaxed atomics is a longstanding problem

Why Use Relaxed Atomics?



- But generally use simple, SW-based coherence
  - Cost of staying away from relaxed atomics too high!

Our Approach

- Previous work
  - Goal: formal semantics for all possible relaxed atomics uses
  - Unsuccessful despite ~15 years of effort
- Insight: analyze how real codes use relaxed atomics
  - What are common uses of relaxed atomics?
  - Why do they work?
  - Can we formalize semantics for them?

Contributions [ISCA '17]

- Identified common uses of relaxed atomics
  - Work queues, event counters, ref counters, seqlocks, …
- Data-race-free-relaxed (DRFrlx) memory model:
  - Sequentially consistent (SC) centric semantics + efficiency
- Evaluated benefits of using relaxed atomics
  - Up to 53% less cycles (33% avg), 40% less energy (20% avg)

Everyone can safely use RAts

Outline

- Motivation
- Background
- Data-race-free-relaxed
- Results
- Conclusion

Atomics Background

- **Default: Data-race-free-0 (DRF0) [ISCA '90]**
  - Identify all races as synchronization accesses (C++: atomics)

```
// each thread
for i = 0:n
    ...
    ADD R4, A[i], R1   synch (atomic)
    ADD R5, B[i], R1   synch (atomic)
    ...
```

  - All atomics order data accesses
  - Atomics order other atomics

⇒Ensures SC semantics if no data races

Atomics Background (Cont.)

- Default: Data-race-free-0 (DRF0) [ISCA ‘90]
  – All atomics order data accesses
  – Atomics order other atomics
  ⇒Ensures SC semantics if no data races
- Data-race-free-1 (DRF1): unpaired atomics [TPDS ‘93]
  + Unpaired atomics do not order data accesses
  – Atomics order other atomics
  ⇒Ensures SC semantics if no data races
- Relaxed atomics [PLDI ‘08]
  + Do not order data or other atomics
  ⇒But can violate SC and no formal specification

Outline

- Motivation
- Background
- Data-race-free-relaxed
- Results
- Conclusion

Identifying Relaxed Atomic Use Cases

- **Our Approach**
  - What are common uses of relaxed atomics?
  - Why do they work?
  - Can we formalize semantics for them?
- **Contacted vendors, developers, and researchers**

Work Queues

Seqlocks

Flags

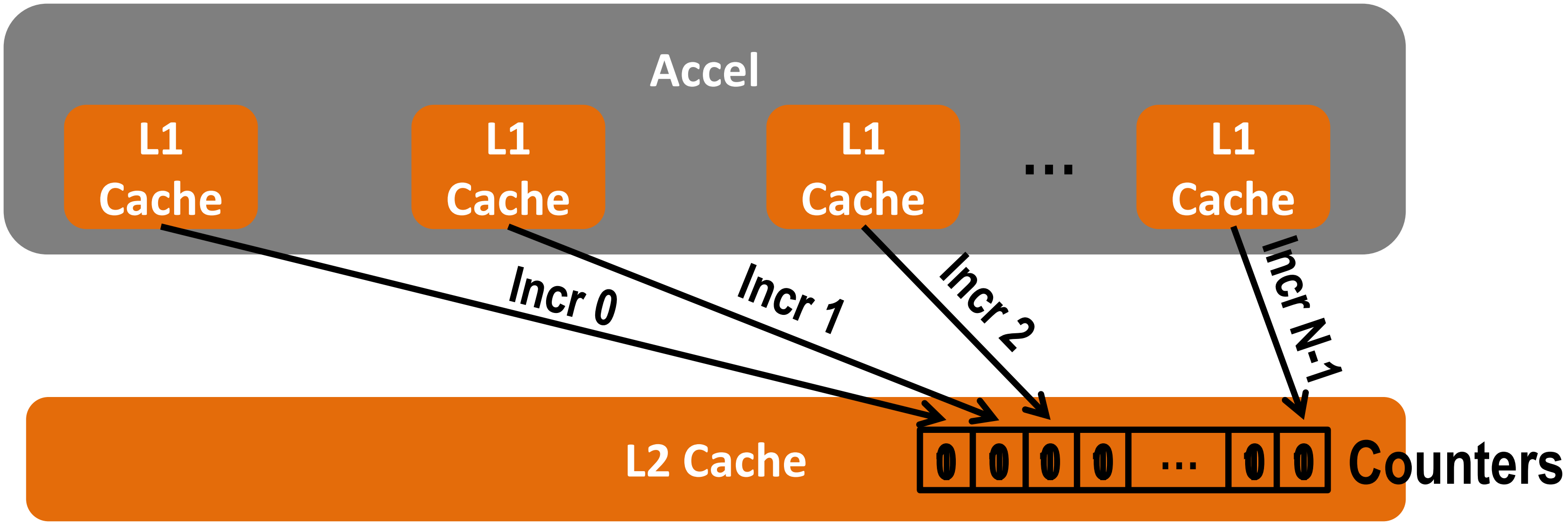
Ref Counters

Event Counters

Split Counters

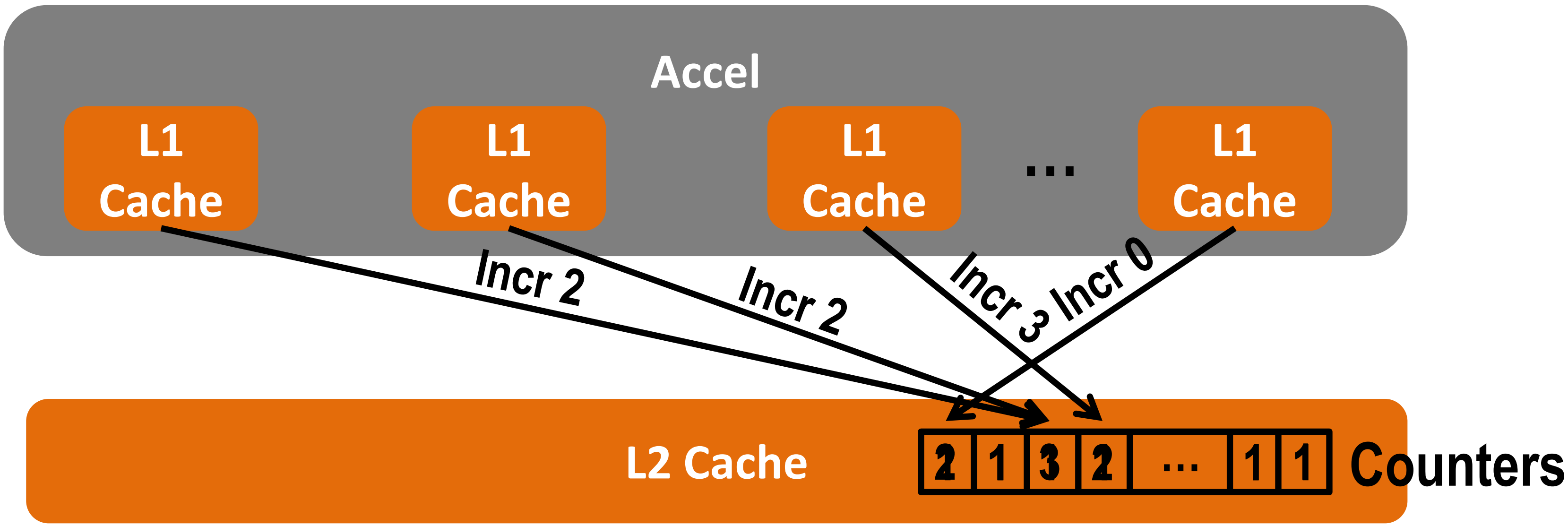
How do relaxed atomics work in Event Counters?

Event Counter



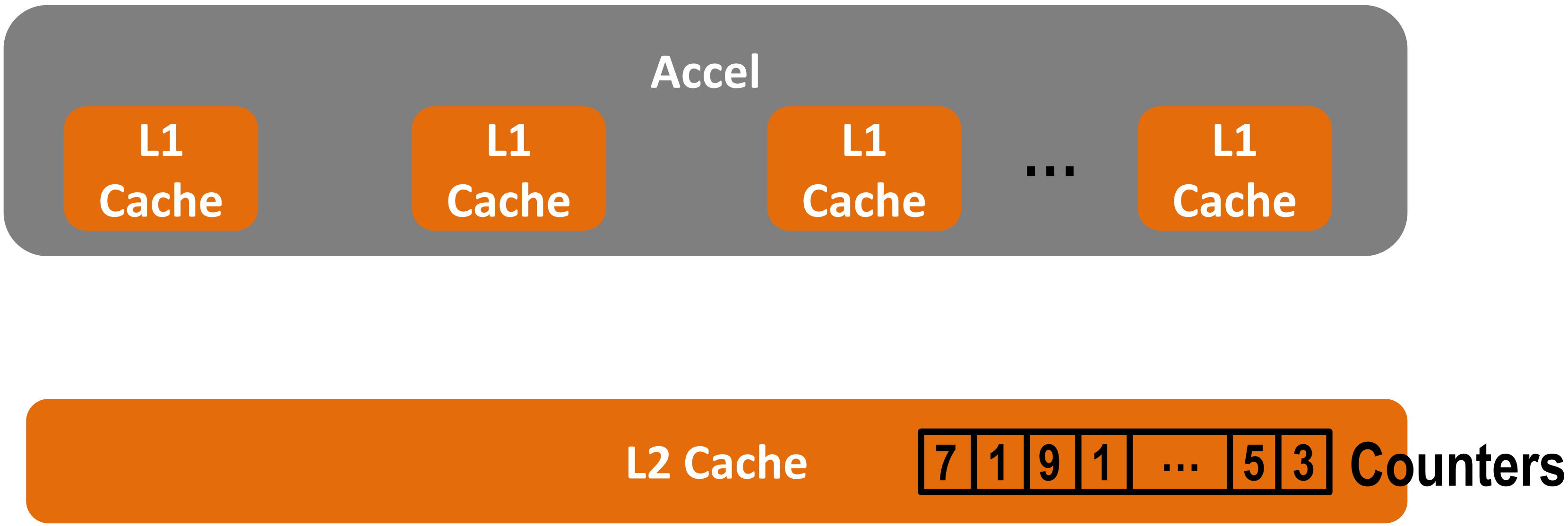
- Threads concurrently update counters
  - Read part of a data array, updates its counter

Event Counter (Cont.)



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

Event Counter (Cont.)



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

Commutative increments: order does not affect final result

How to formalize?

Incorporating Commutativity Into DRFrlx

- New relaxed atomic category: commutative
- Formalism:
  - Accesses are commutative
  - Intermediate values must not be observed

$\Rightarrow$ Final result is always SC

What about the other use cases?

Incorporating Other Use Cases Into DRFrlx

Work Queues
Seqlocks
Flags
Ref Counters
Split Counters

Use Case	Category	Semantics
Work Queues Flags	Unpaired Non-Ordering	SC
Event Counters Seqlocks	Commutative Speculative	Final result always SC
Ref Counters Split Counters	Quantum	SC-centric: non-SC parts isolated

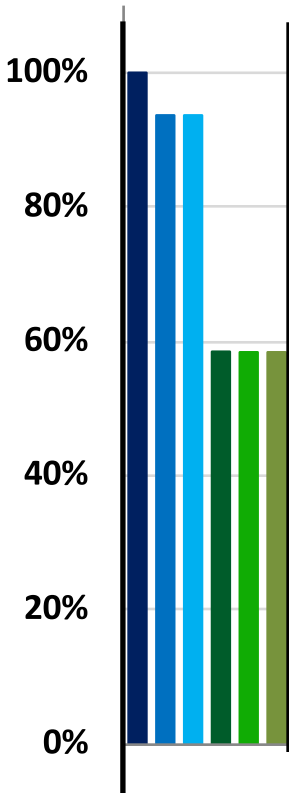
Outline

- Motivation
- Background
- Data-race-free-relaxed
- Results
- Conclusion

Evaluation Methodology

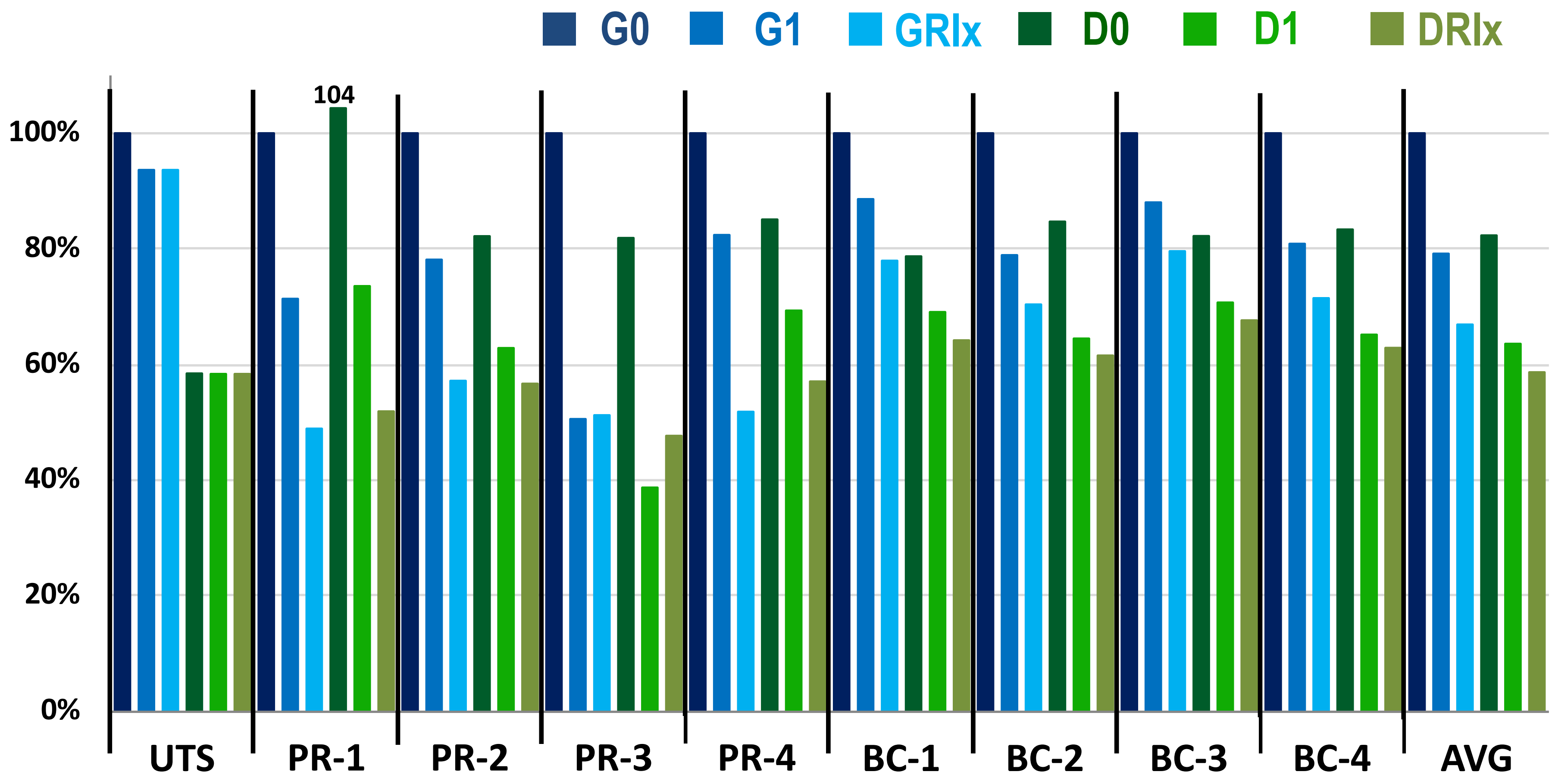
- 1 CPU core + 15 GPU compute units (CU)
  - Each node has private L1, scratchpad, tile of shared L2
- Simulation Environment
  - GEMS, Simics, Garnet, GPGPU-Sim, GPUWattch, McPAT
- Study DRF0, DRF1, DRFrlx w/ GPU & DeNovo coherence
- Workloads
  - Microbenchmarks for each use case
    - Relaxed atomics help a little (Avg: 10% cycles, 5% energy)
  - Benchmarks with biggest RAts speedups on discrete GPU
    - UTS, PageRank (PR), Betweeness Centrality (BC)

Relaxed Atomics Applications – Execution Time



- G0 = GPU coherence + DRF0
- G1 = GPU coherence + DRF1
- GRlx = GPU coherence + DRFrlx
- D0 = DeNovo coherence + DRF0
- D1 = DeNovo coherence + DRF1
- DRlx = DeNovo coherence + DRFrlx

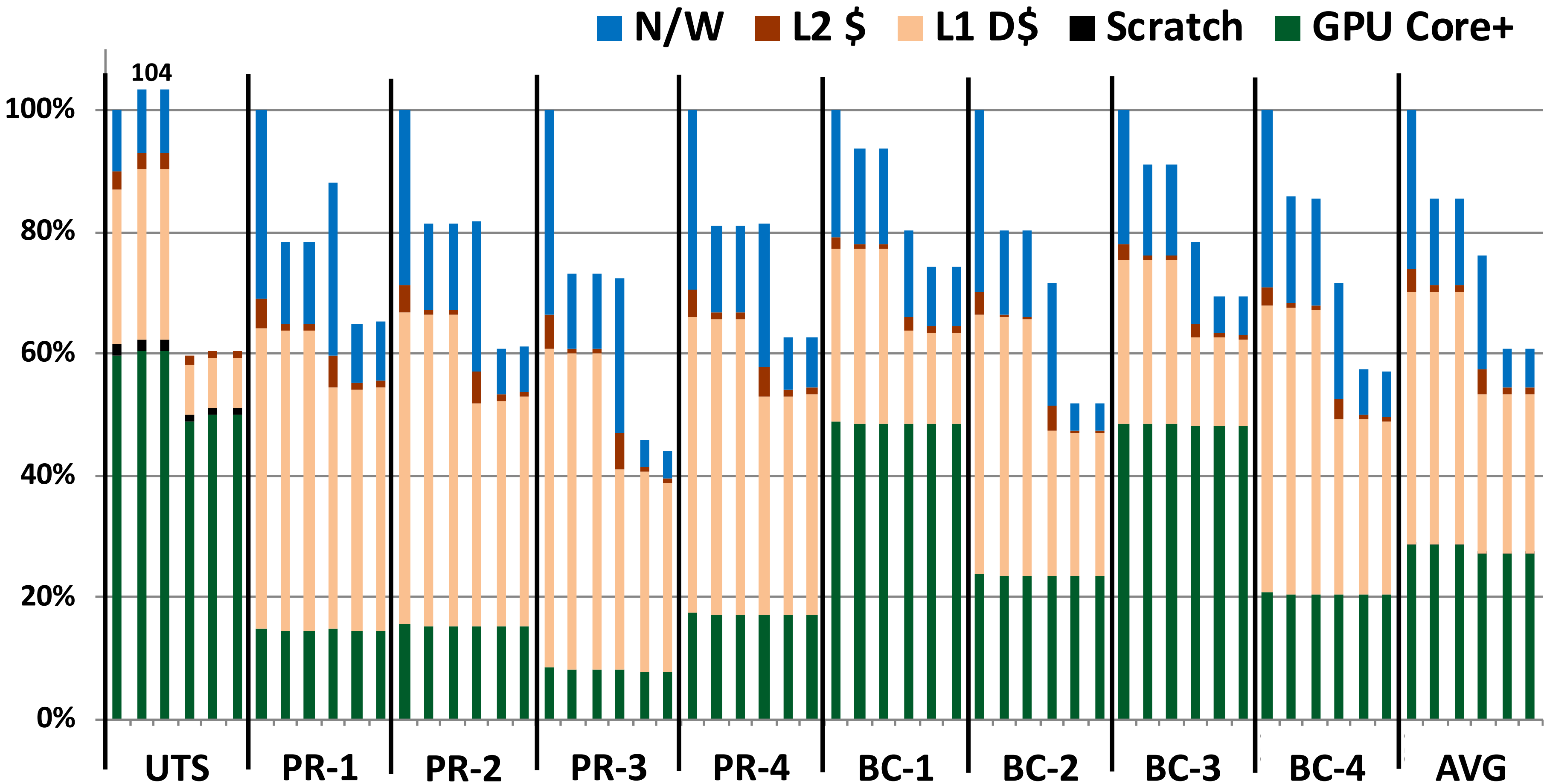
Relaxed Atomics Applications – Execution Time



Relaxed atomics reduce cycles up to ~50%

DeNovo increases reuse over GPU: 10% avg. for DRFrlx

Relaxed Atomics Applications – Energy



Energy similar to execution time trends

DeNovo's reuse reduces energy over GPU: 29% avg. for DRFrlx

Conclusion

- Cost of avoiding relaxed atomics too high
- Difficult to use correctly: no formal specification
- Insight: Analyze how real codes use relaxed atomics

DRFrlx: SC-centric semantics + efficiency

Everyone can safely use RAts

BACKUP

Consistency is Complex

"If you think you understand quantum computers, it's because you don't. Quantum computing is actually *harder* than memory consistency models."

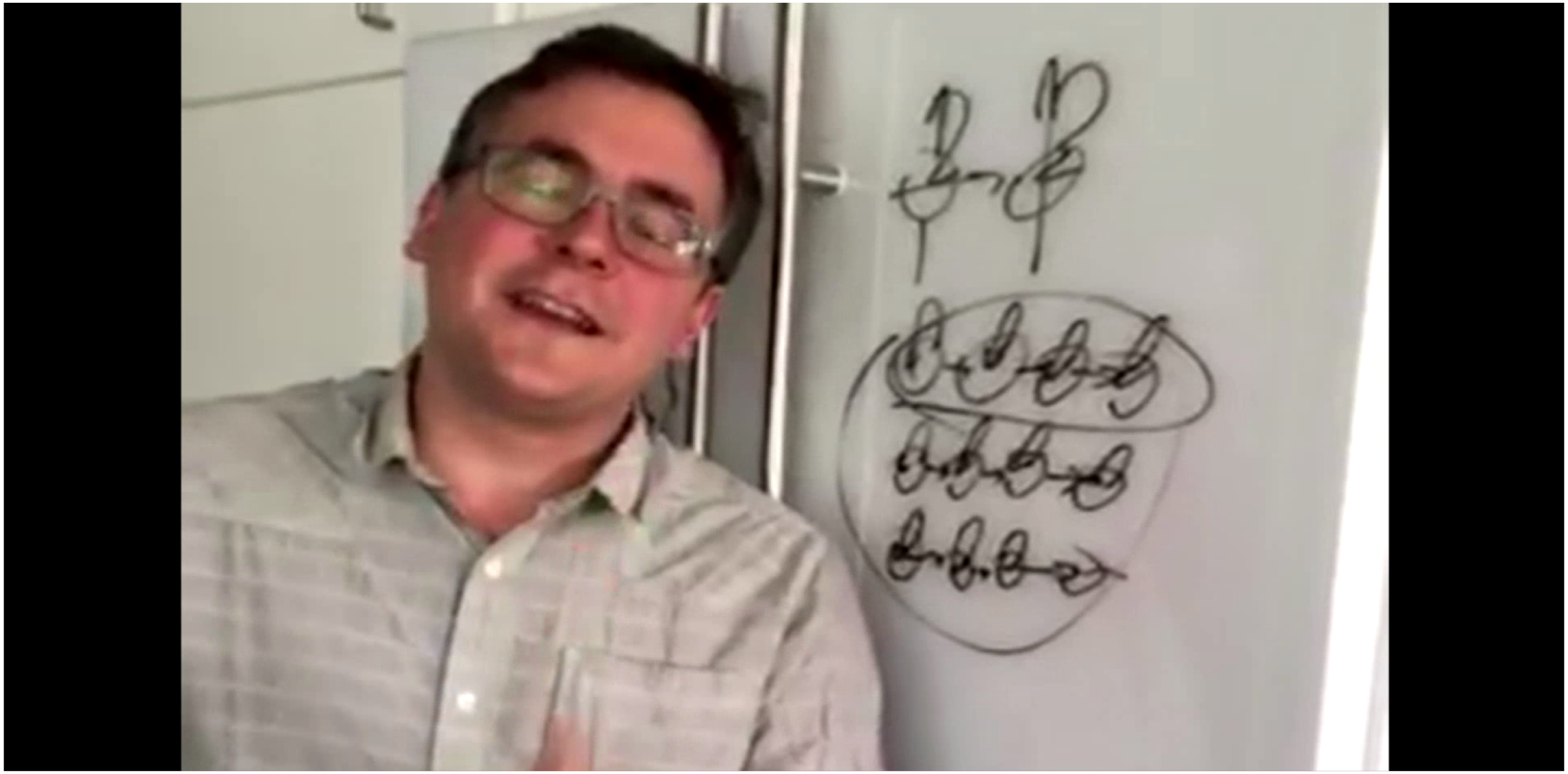
- Luis Ceze, video in ISCA '16 Keynote

Memory consistency: gold standard for complexity

Relaxed atomics add even more complexity

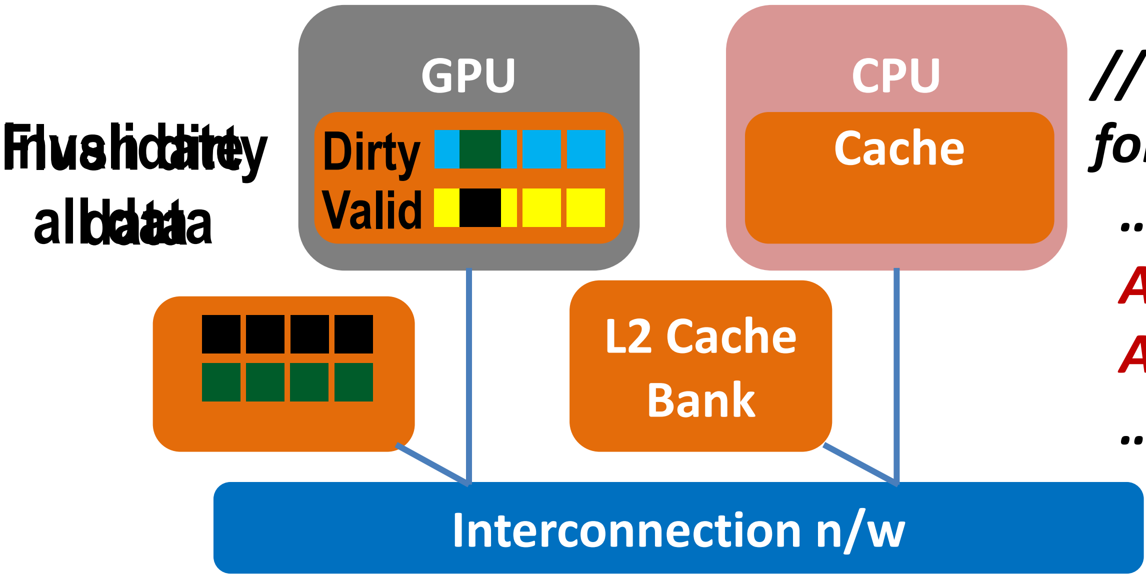
Consistency is Complex

How hard are consistency models?



Memory consistency: gold standard for complexity

Atomics in Data-Race-Free-0 (DRF0)



// each thread
for i = start:end
...
ATOM.ADD R4, A[i], R1
ATOM.ADD R5, B[i], R1
...

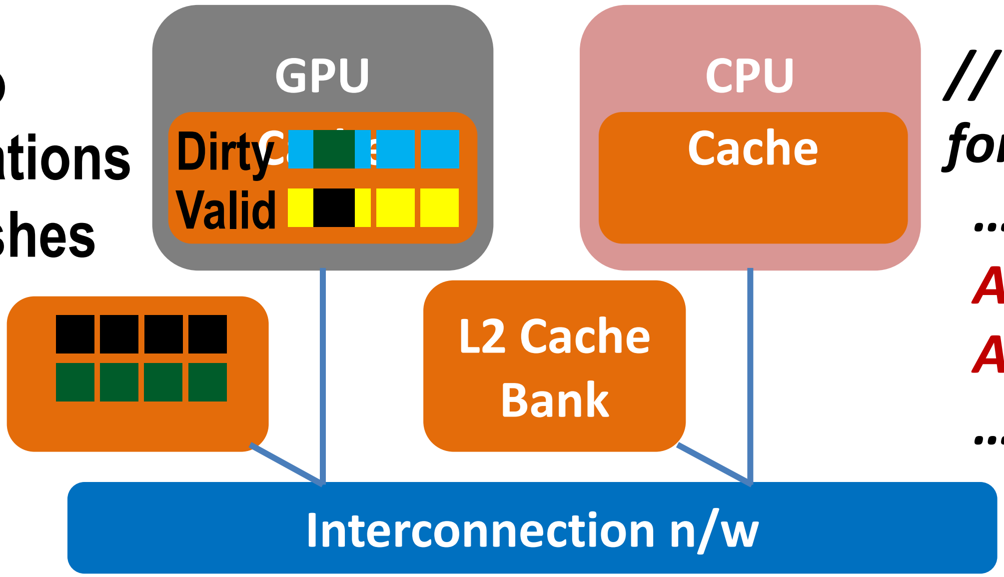
- **Default: DRF0 [ISCA '90]**
  - All atomics order data accesses
  - Atomics order other atomics

⇒Ensures SC semantics

Precludes data reuse and overlapping atomics

Atomics in Data-Race-Free-1 (DRF1)

No invalidations or flushes



// each thread
for i = start:end
...
ATOM.ADD R4, A[i], R1 *UnP*
ATOM.ADD R5, B[i], R1 *UnP*
...

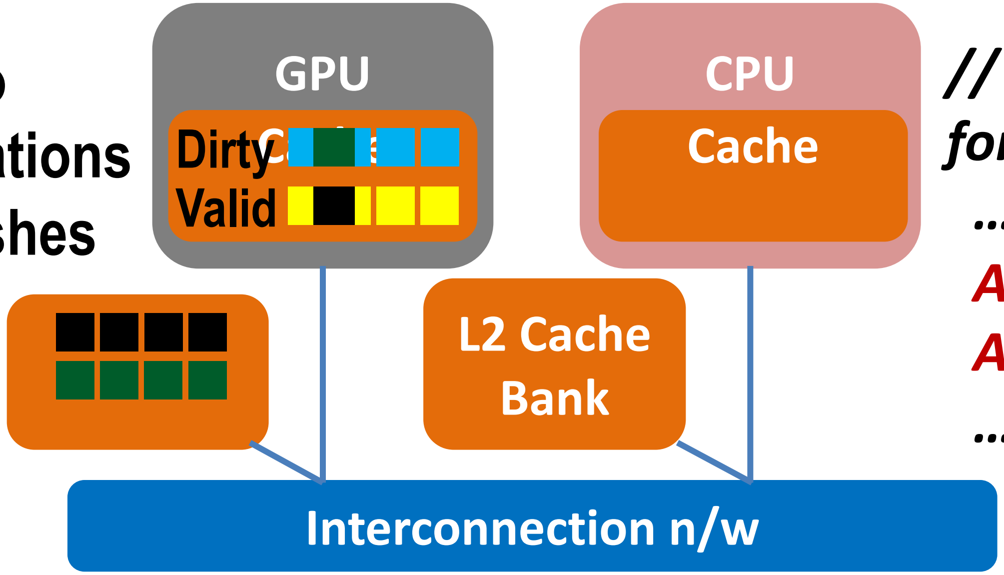
- **Unpaired atomics do not order any data accesses**
  - \+ Avoids invalidations and flushes
  - – Atomics order other atomics

⇒**Ensures SC semantics**

Can reuse data but cannot overlap atomics

Relaxed Atomics

No invalidations or flushes

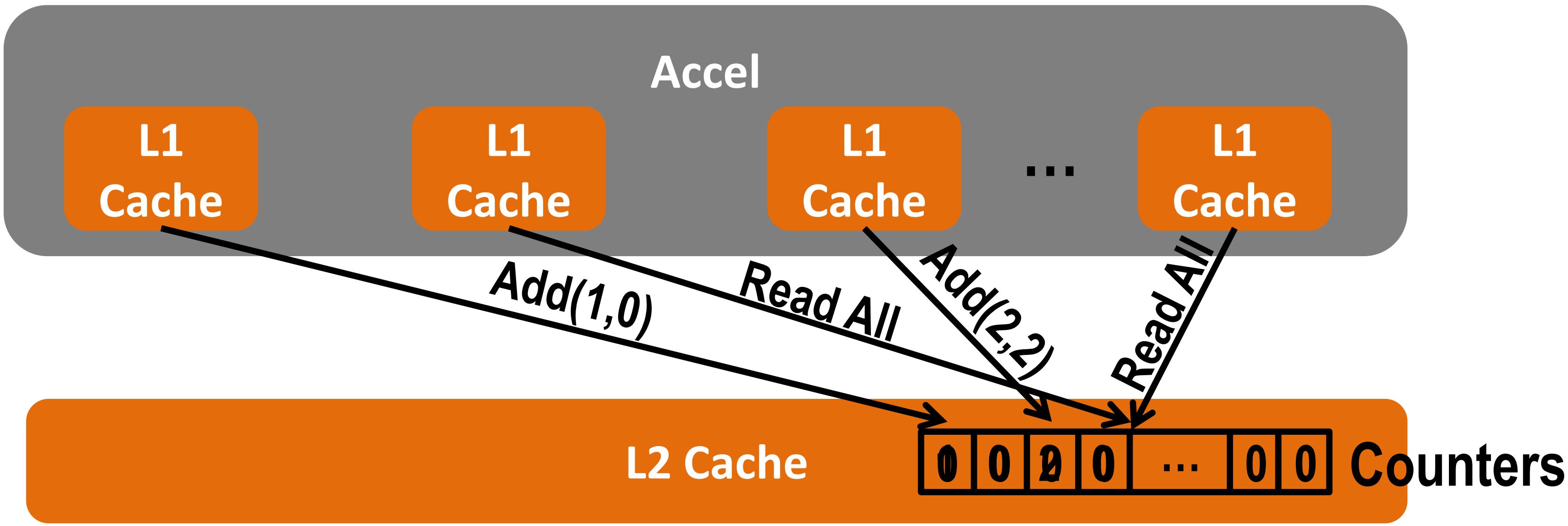



```
// each thread
for i = start:end
  ...
  ATOM.ADD R4, A[i], R1 RLX
  ATOM.ADD R5, B[i], R1 RLX
  ...
```

- **Relaxed atomics do not order data or other atomics**
  + Reorder, overlap with all other memory accesses

But can violate SC and no formal specification

Split Counter

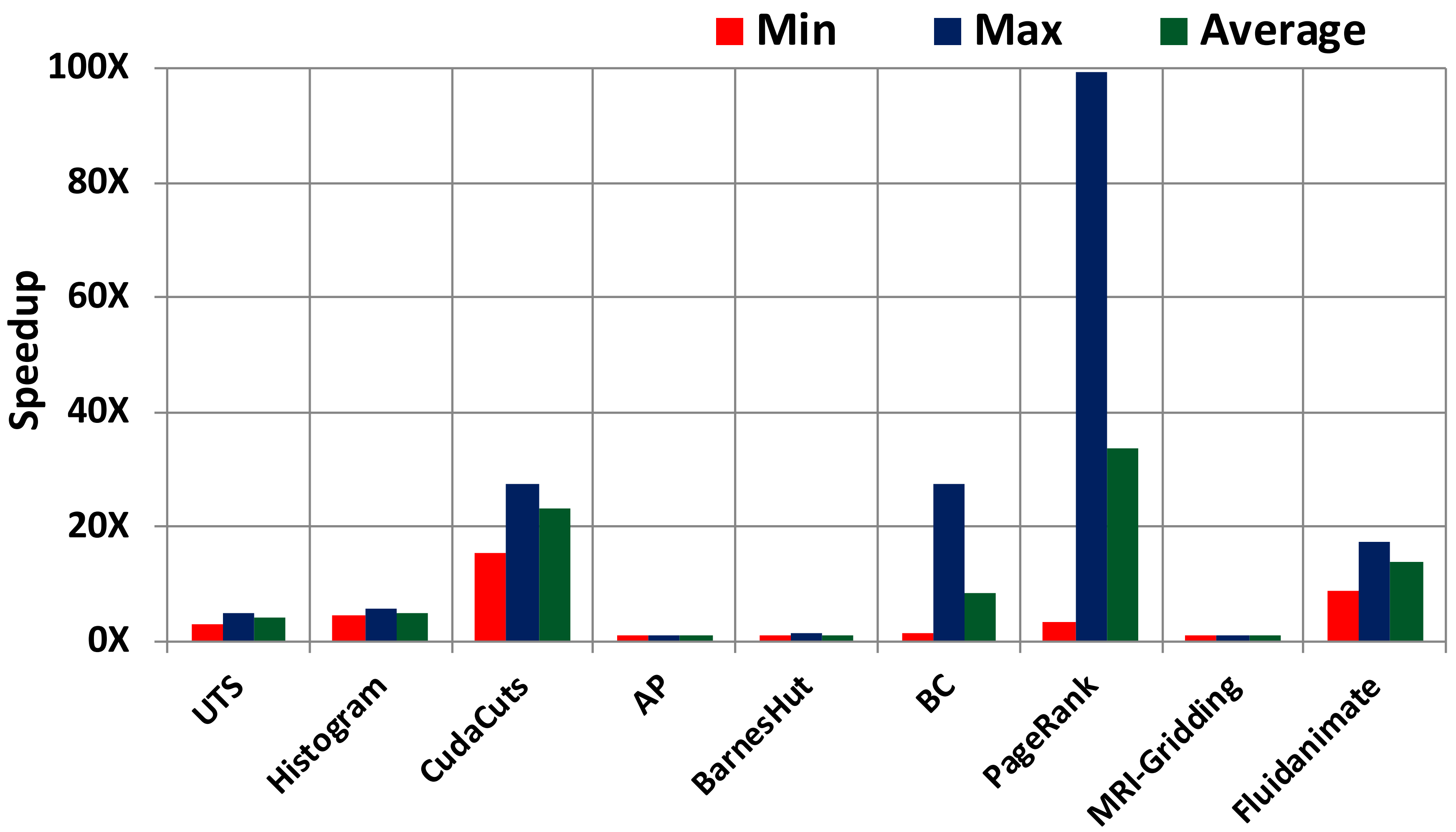


- Threads simultaneously access counters
  - Some threads update their counter
  - Other threads read all counters to get the current partial sum
  - Counter accesses race, so must use atomics

Quantum – Split Counter (Cont.)

- Can reorder, overlap relaxed atomics from same thread
  - Results may not be SC – programmers ok with approx values
- DRFrlx
  - Distinguish quantum atomics
    - Quantum atomic loads logically return approximate value
  - Program is DRFrlx if DRF1 and no races in new program

Relaxed Atomics on Discrete GPUs



Cost of staying away too high!

Incorporating Other Use Cases Into DRFrlx

Work Queues

Seqlocks

Reference Counters

Split Counters

Flags

Incorporating Other Use Cases Into DRFrlx

Work Queues

Seqlocks

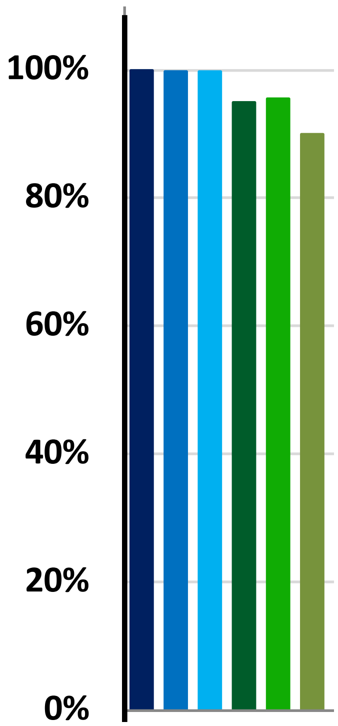
Reference Counters

Flags

Split Counters

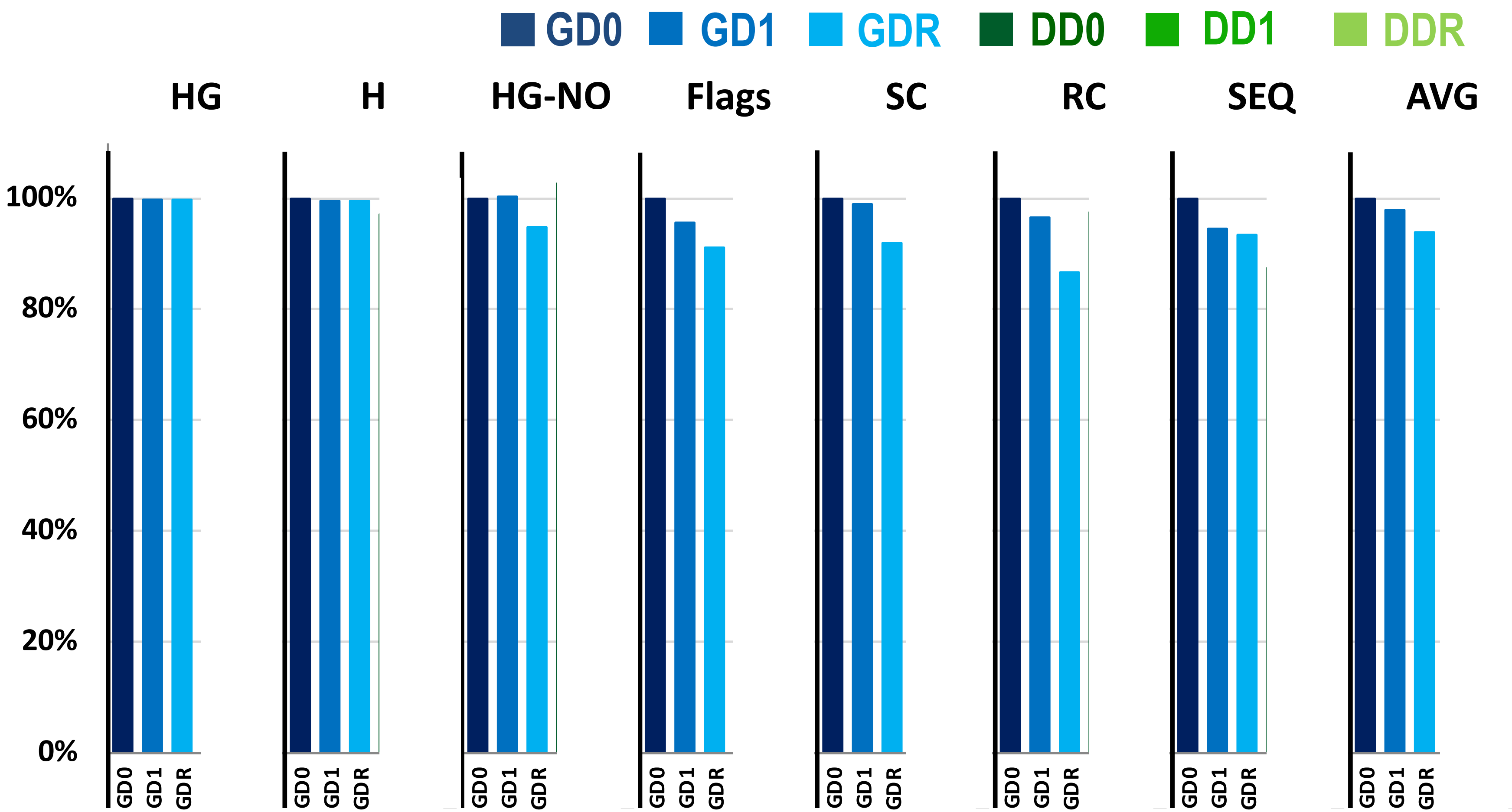
Use Case	Category	Semantics
Work Queues Flags	Unpaired Non-Ordering	SC
Event Counters Seqlocks	Commutative Speculative	Final result always SC
Ref Counters Split Counters	Quantum	SC-centric: non-SC parts isolated

Relaxed Atomics Microbenchmarks – Execution Time



- GD0 = GPU coherence + DRF0
- GD1 = GPU coherence + DRF1
- GDR = GPU coherence + DRFrlx
- DD0 = DeNovo coherence + DRF0
- DD1 = DeNovo coherence + DRF1
- DDR = DeNovo coherence + DRFrlx

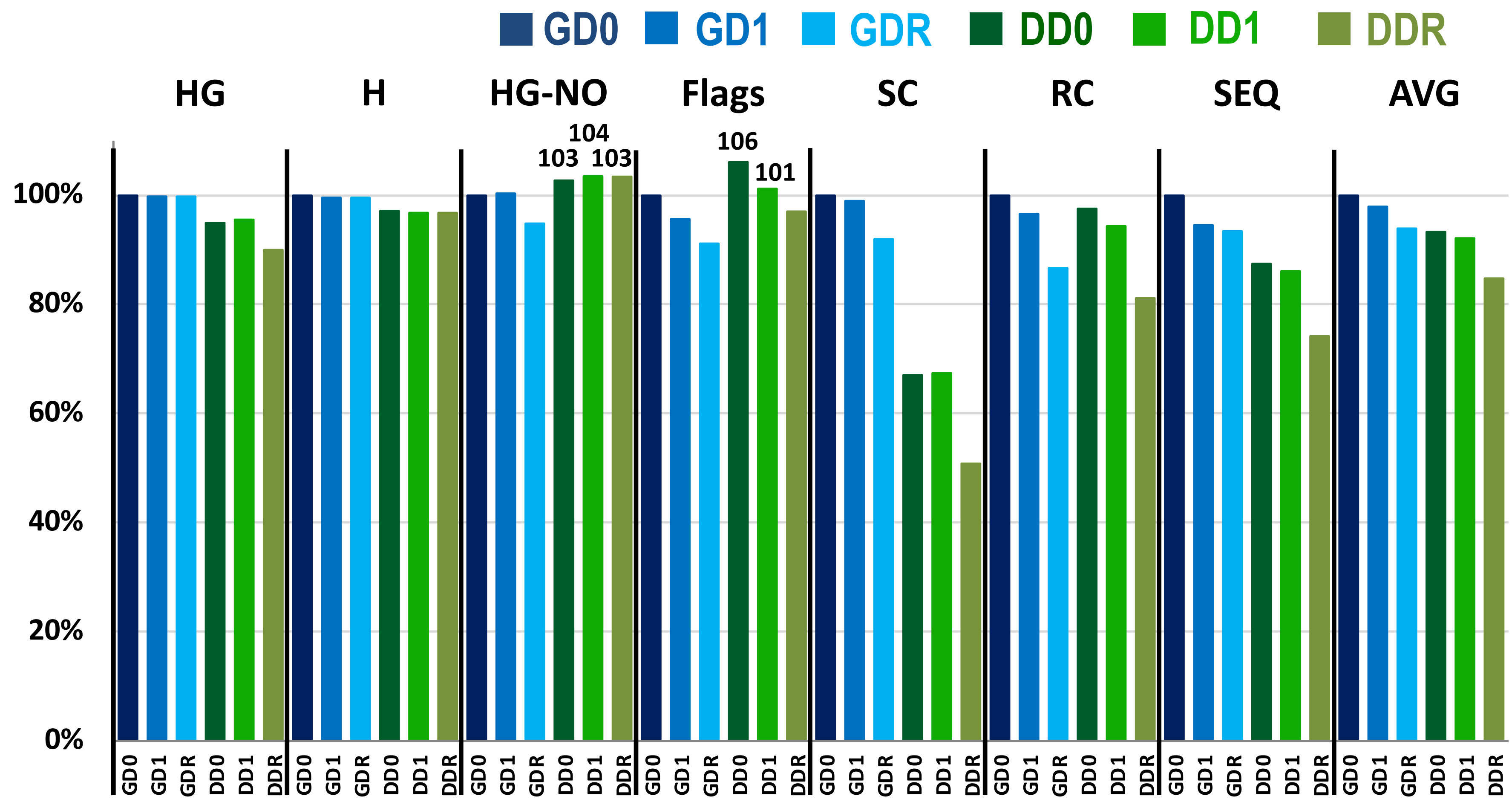
Relaxed Atomics Microbenchmarks – Execution Time



Weakening the consistency model does not significantly improve perf

DRFrlx allows atomics to be overlapped (7% avg improvement for GPU)

Relaxed Atomics Microbenchmarks – Execution Time

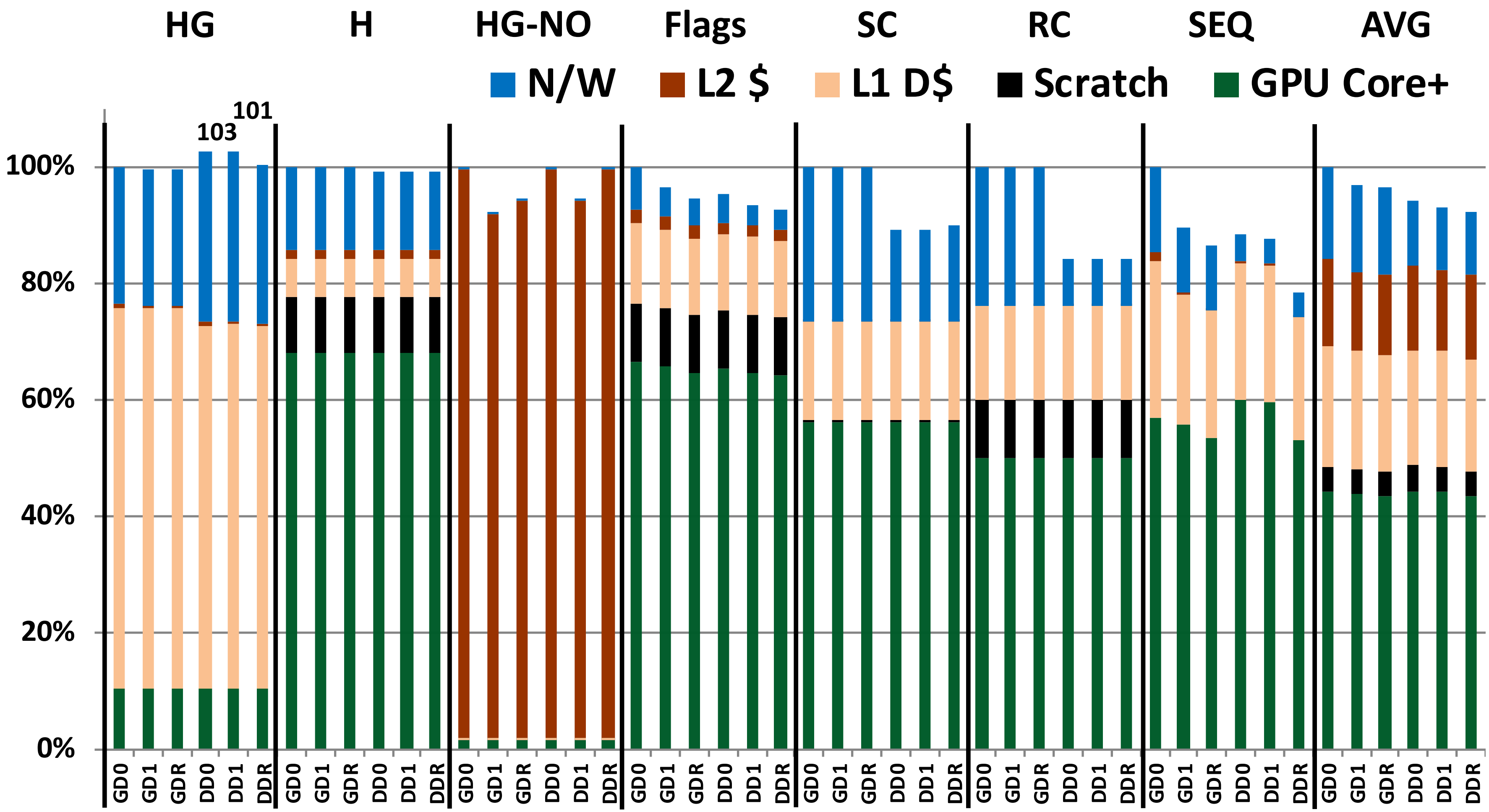


Weakening the consistency model does not significantly improve perf

DRFrlx allows atomics to be overlapped (7% avg improvement for GPU)

DeNovo exploits synch reuse, outperforms GPU (DRFrlx: 10% avg)

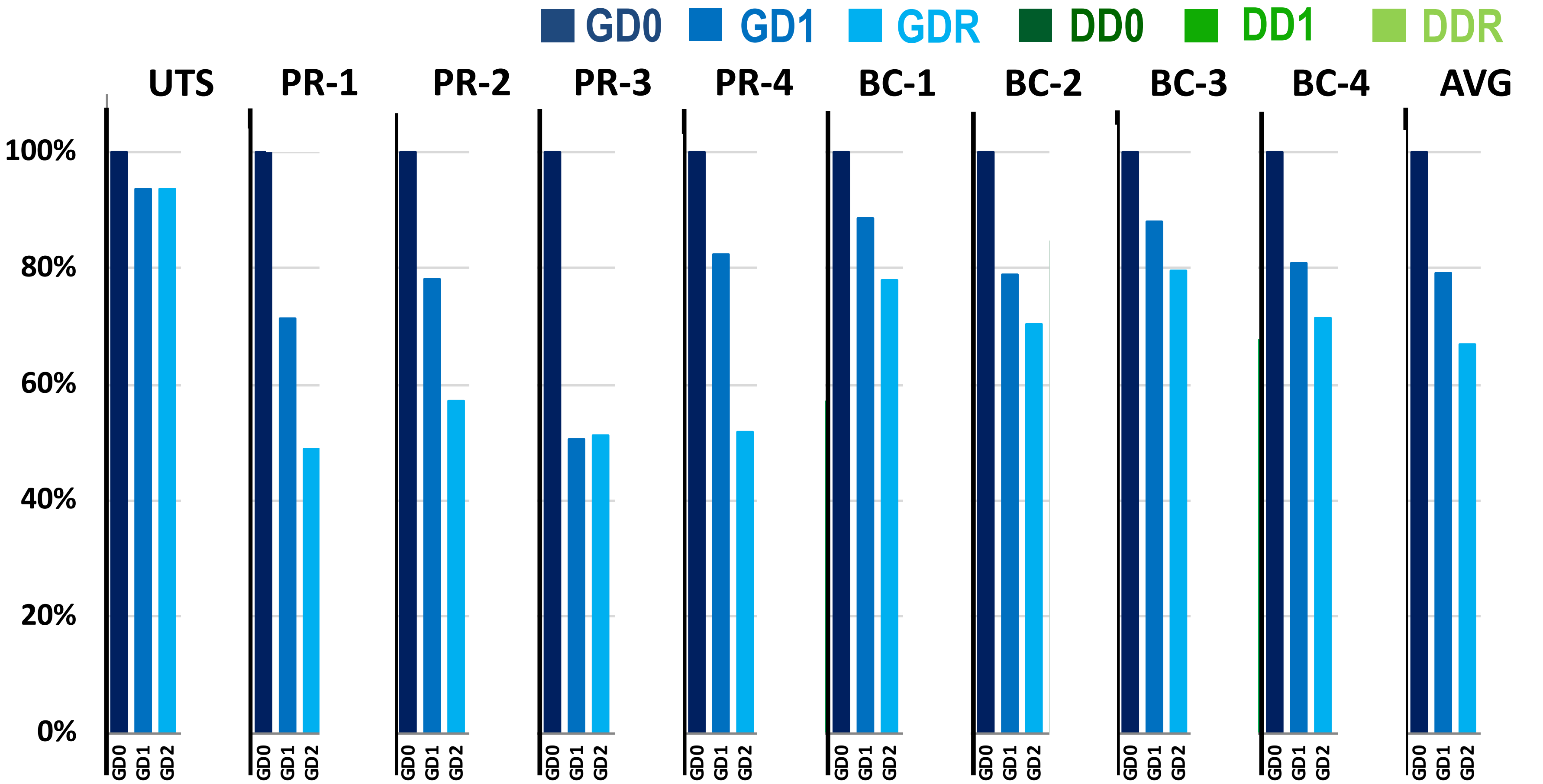
Relaxed Atomics Microbenchmarks – Energy



Energy trends somewhat similar to execution time

DRFrlx: DeNovo reduces energy by 4% over GPU

Relaxed Atomics Applications – Execution Time



Weakening memory model helps a lot (up to 51% for GPU)

DRF1 increases data reuse (21% avg vs. GD0)

DRFrlx overlaps atomics (15% avg vs. GD1)

DRFrlx Summary

- New relaxed atomic type for each category
  - Formalize when an atomic falls into category
  - SC(-centric) semantics if use relaxed atomics correctly

Strongest (SC)

Unpaired	SC: Reorder unpaired with data accesses
Non-Ordering	SC: atomics do not order other accesses
Commutative	Final result always SC
Speculative	Final result always SC (retry violations)
Quantum	Isolate non-SC parts

Weakest (SC-centric)