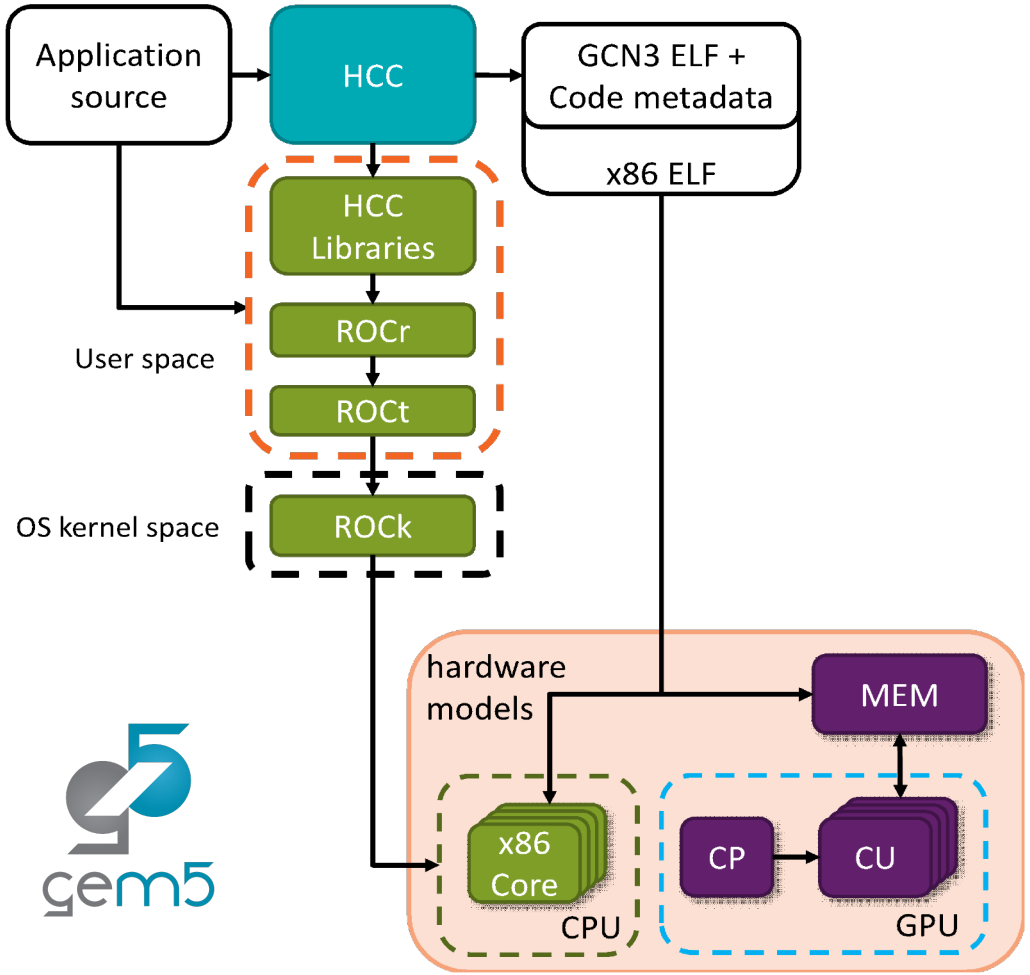
Closing the GAP: Improving the Accuracy of gem5's GPU Models

Vishnu Ramadas, **Daniel Kouchekinia**, Ndubuisi Osuji, Matthew D. Sinclair

University of Wisconsin-Madison, AMD Research

Prior CPU-GPU SE Mode Support in gem5



[Gutierrez et al., HPCA '18]

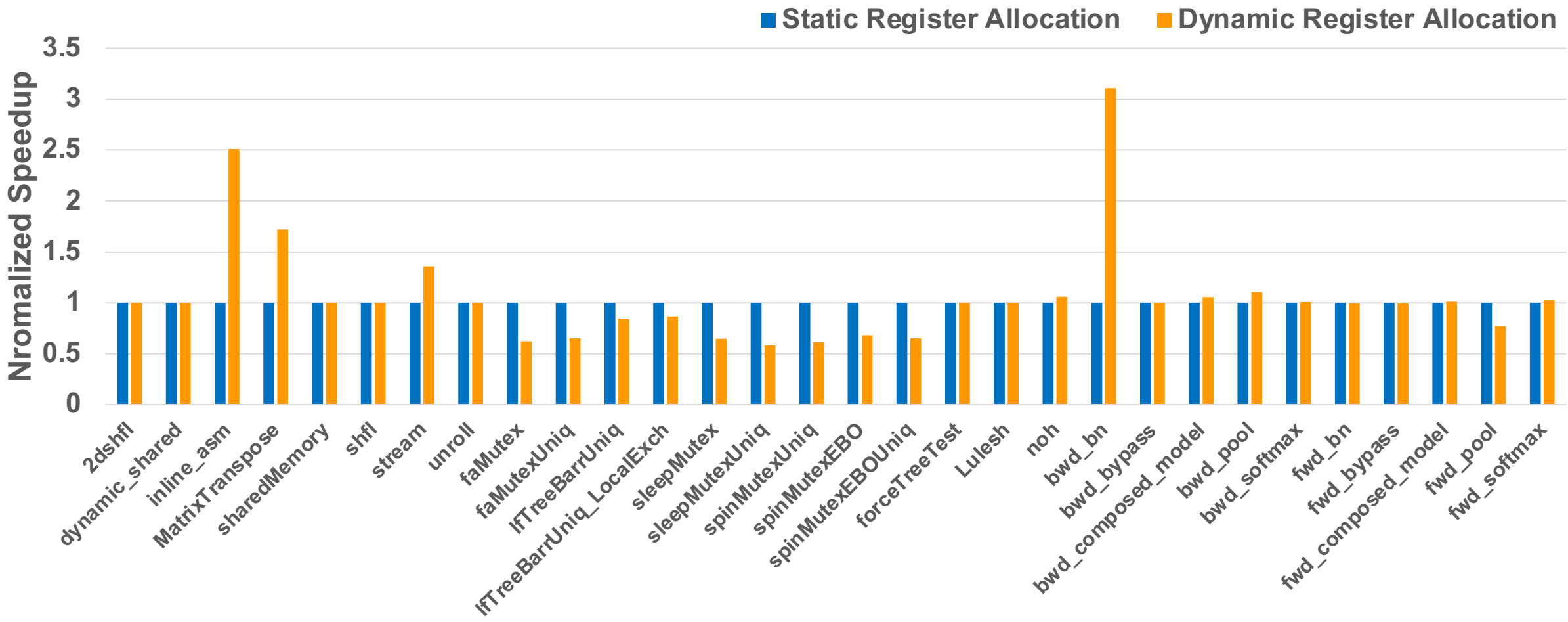
- Execution-driven, cycle-level
  - Models complex CPUs & GPUs
  - Rapid prototyping of new features
  - Simulates HIP (AMD's GPGPU language) applications
- UW HAL Group
  - Creating, validating and releasing docker image to run GPU models with updated versions of ROCm
  - Released support for several GPU workloads in gem5-resources, enabled CI testing
- Publicly-available support focuses on Carrizo- and Vega-Class
  - Do not always provide high accuracy relative to equivalent real GPUs (hazardous)

Improving Register Allocation Support

- Simple register allocation – only 1 wavefront/CU at a time
  - Even if sufficient registers are available for more WFs
- Issue: unrealistic relative to real GPUs
- Solution: add dynamic register allocator [Bruce et al. ISPASS '20]
  - If enough registers available, schedule additional WFs concurrently/CU
  - Potentially can utilize all WF slots depending on register requirements
  - More complex, higher performance designs possible

Intuition: Dynamic allocator significantly improves accuracy

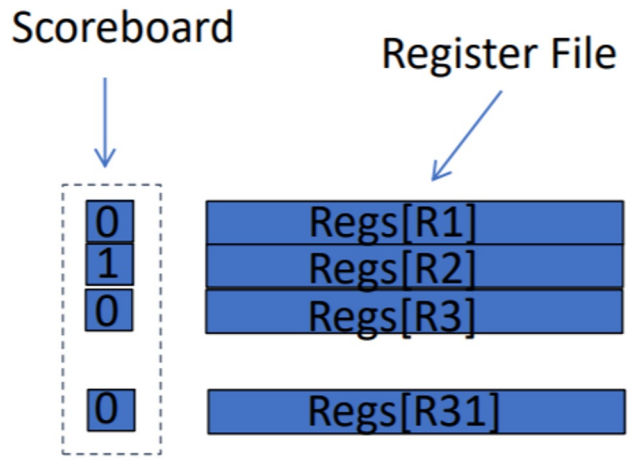
Dynamic Register Allocator Performance



Reality: dynamic register allocator 6% worse than simple – why?

Issue: Dependence Tracking

- GPU model did not track dependencies well → many stalls
  - Result: optimizing register allocation in isolation was insufficient
- Issue: Proprietary GPU dependence checking sols unknown
- Solution: simple, in-order scoreboard
  - Bit per register to track use status
  - Cleared on instruction completion
  - Checks for RAW/WAW hazards



Result: up to 44% reduction in stalls

Issue: Unknowns in Proprietary Solutions

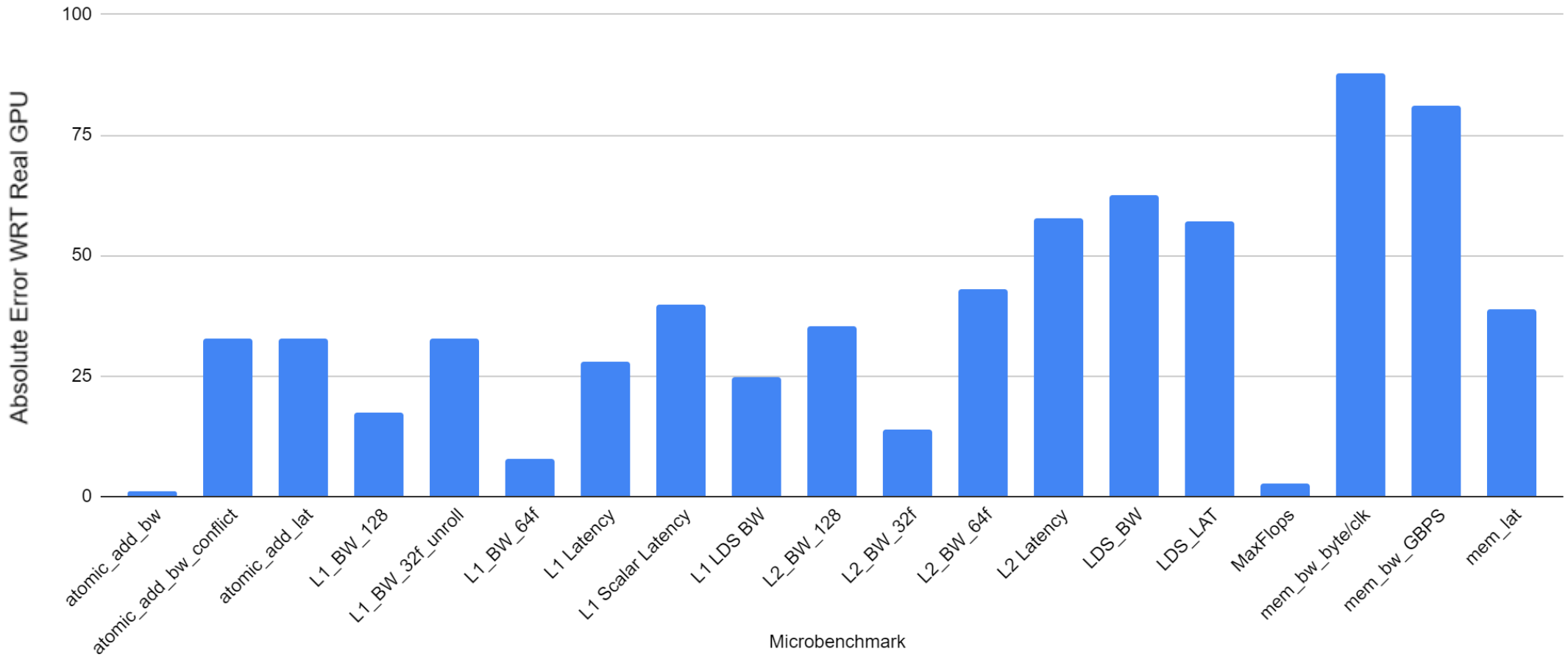
- Point solution (not scalable)
- Need for examining GPU behavior at a finer granularity
- Goal: Isolate behavior of different components to attack inaccuracies at a more digestible level
  - Targeting specific corresponding statistics in gem5 and the ROC profiler

Microbenchmarks

- Hand-tuned HIP assembly kernel μBenches
  - Atomic operations latency & bandwidth (with and without conflicts)
  - L1 I$ size & latency
  - L1 scalar and vector D$ size, latency, & bandwidth
  - LDS (scratchpad) latency & bandwidth
  - L2 $ latency & bandwidth
  - Main memory latency & bandwidth
  - TLB/Page Table latency & bandwidth
  - Max FLOPs, Arithmetic latency for various operations, …
- Compare μBench output and GAP script [Jamieson gem5 Workshop ‘22] analysis to identify underlying inaccuracies

µBench Results Before Tuning

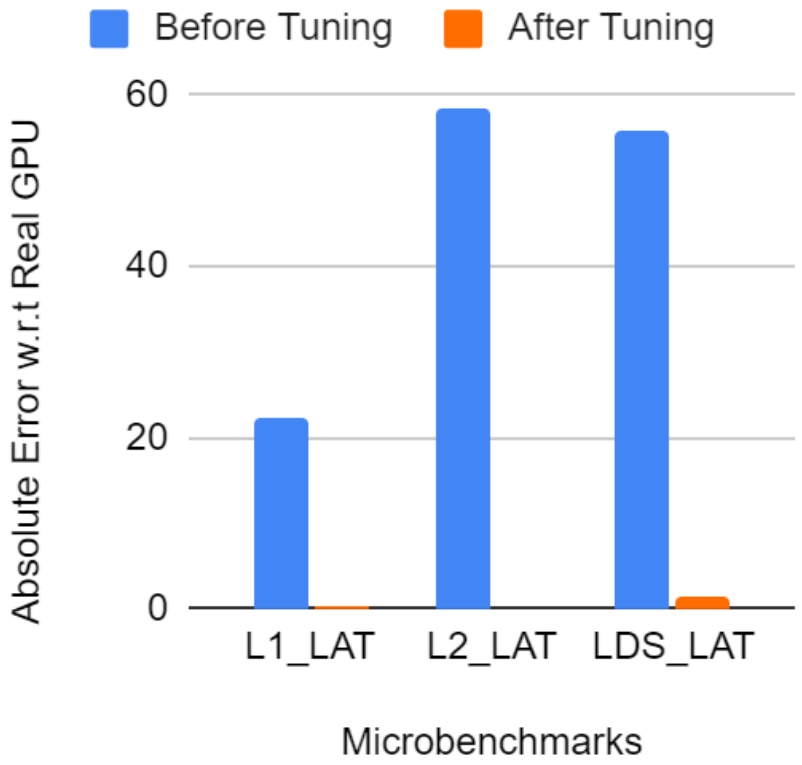
(Vega 20)



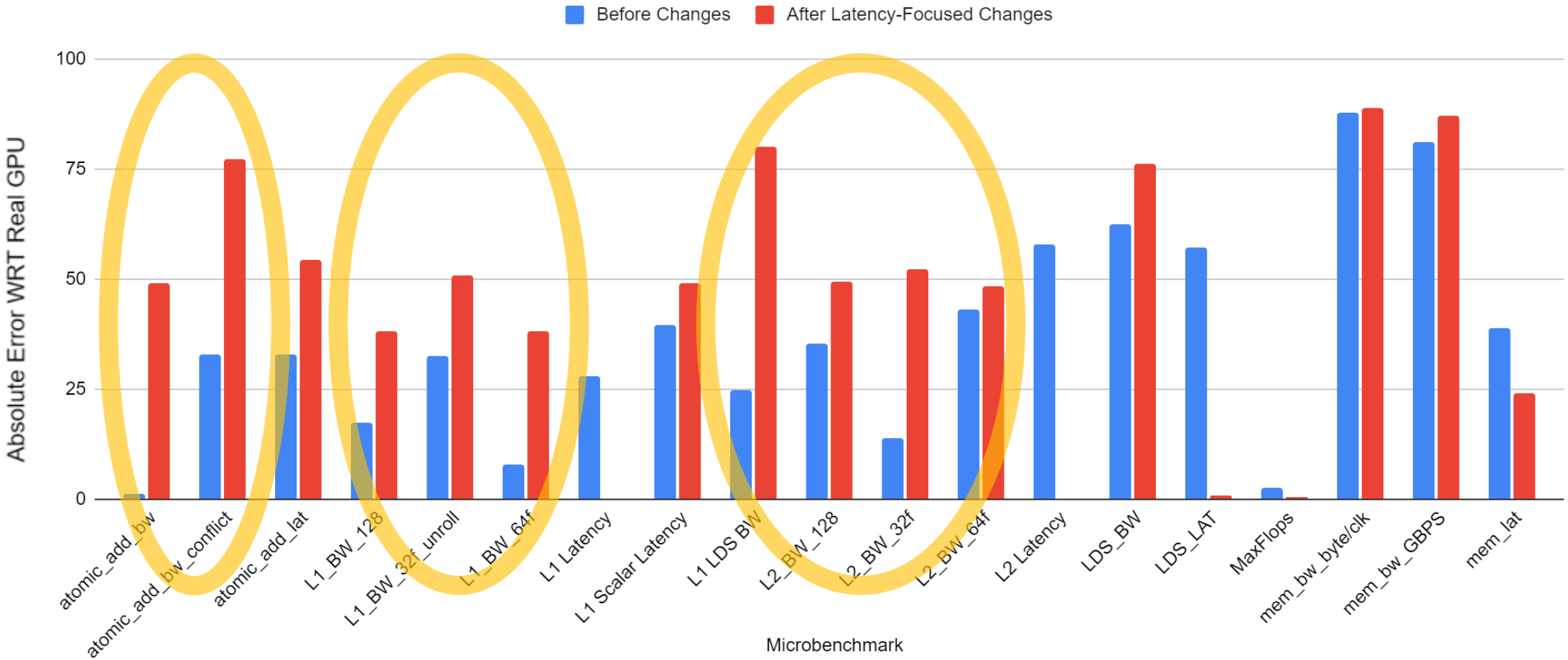
Most microbenchmarks see significant error when compared with real GPU.

µBench Discrepancy: L1, L2, & LDS Latency

- Issue: L1, L2, & LDS clocked twice as fast as they should be
  - Result: Lower latencies than actual GPU
- Refinement of cache parameters including latency and size



µBench Results After Clock Fixes

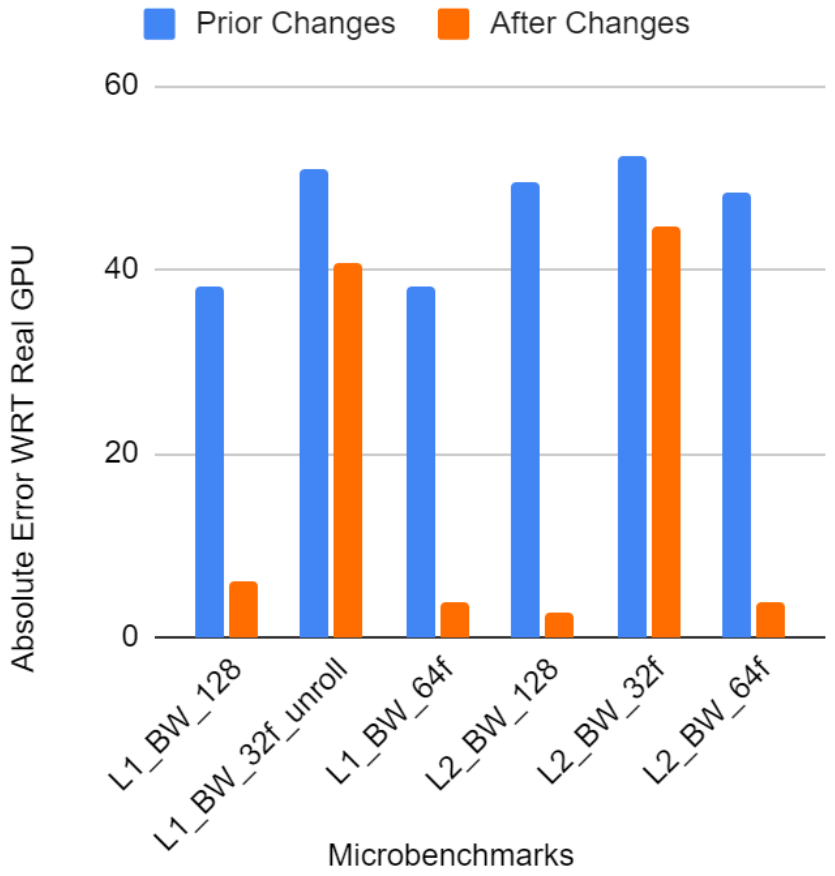


L1, L2, LDS latency errors reduced, accurate but others (especially BWs) increased

Points to need to iteratively refine

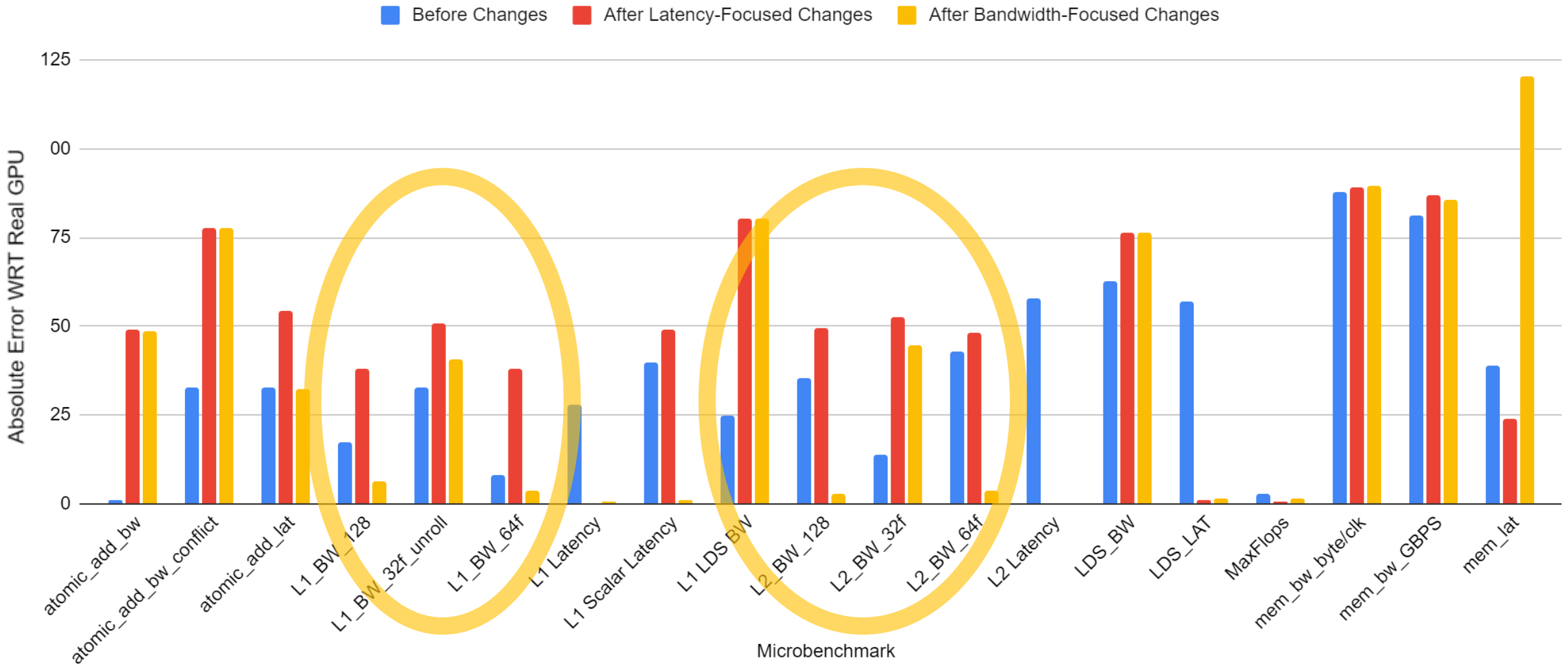
µBench Discrepancy: L1 & L2 Bandwidths

- Tuned coalescer, number of banks, and L1 latency parameters
- Issue: Lack of cache bypassing for GLC and SLC loads and stores
  - GLC*-set instructions should not cache in L1
  - SLC**-set instructions should not cache in L1 or L2



* Global-Level Coherence
** System-Level Coherence

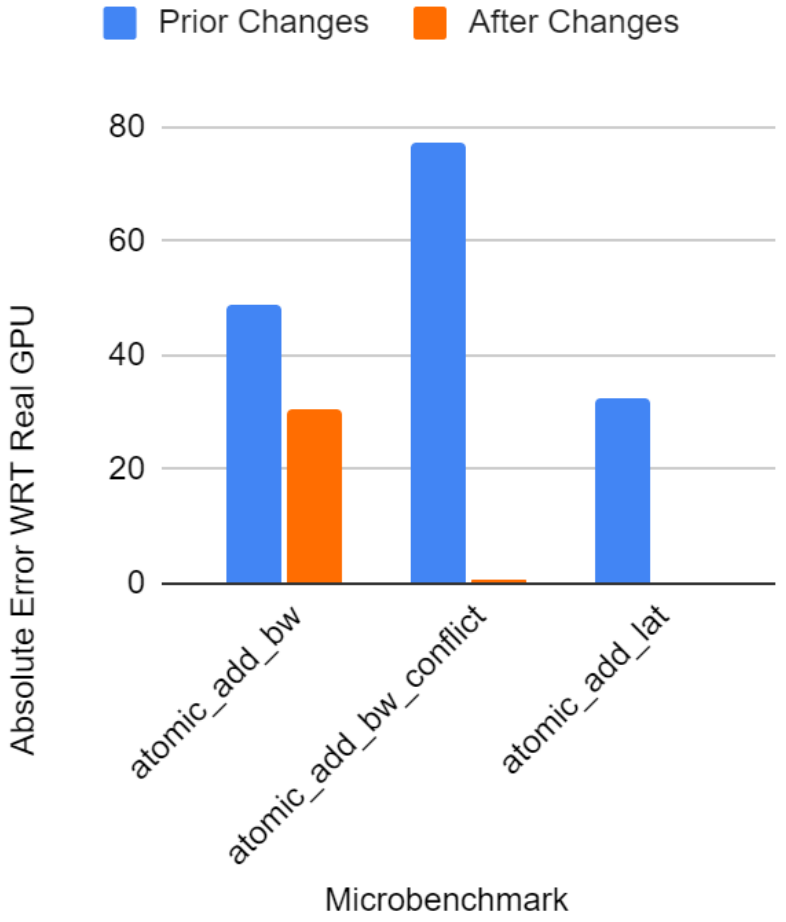
µBench Results After BW-Related Changes



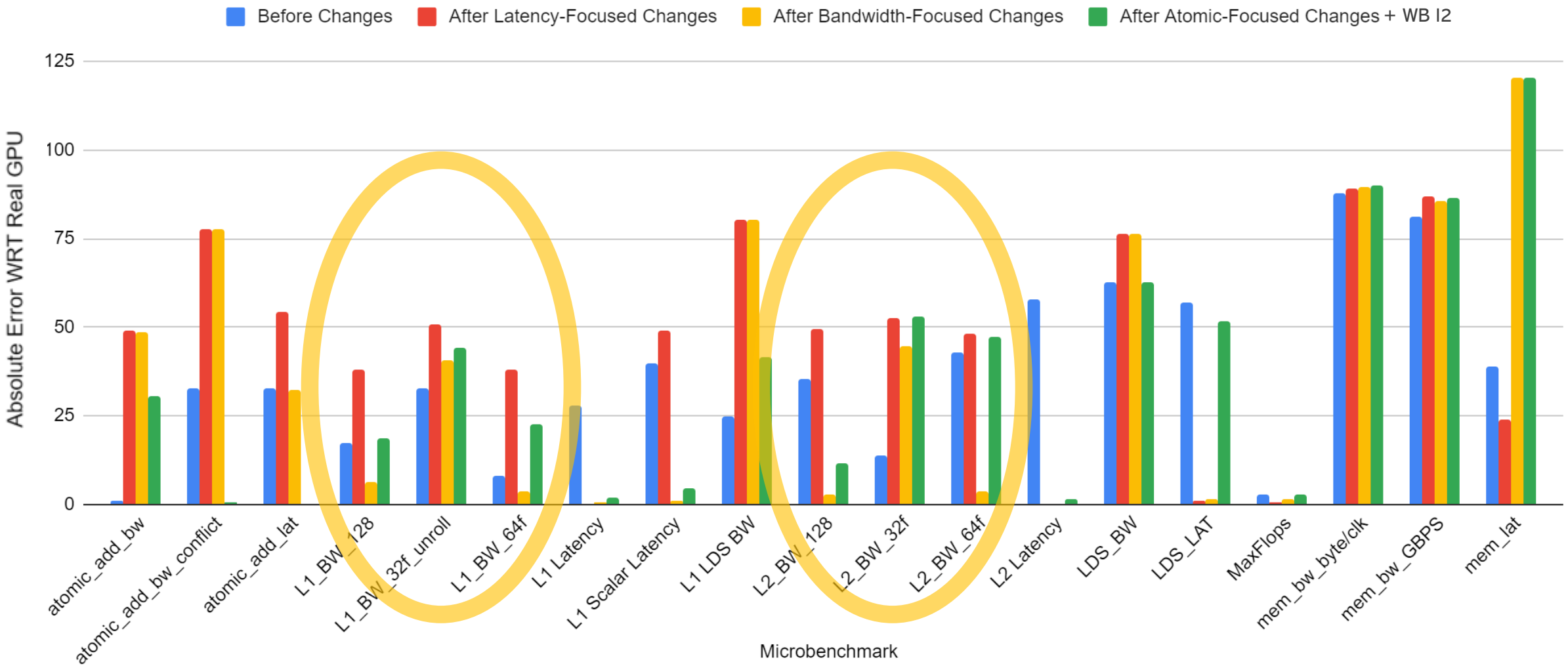
Improvements to L1 and L2 bandwidths, with no impact elsewhere

µBench Discrepancy: Atomics

- Issue: No GLC atomic support
- Consequence: all atomics treated as system-scope atomics
- Added GLC atomic handling into VIPER coherence protocol
  - GLC atomics now performed at L2
  - Additional fixes for GPU WB L2 caches



µBench Results After Atomic Changes



Atomics improved, but increased inaccuracies of other bandwidths

Next Steps: Further Iterative Refinement

- Continue to use µBenches to guide iterative improvements
  - Continue to improve accuracy with additional microbenchmarks
- Update model to provide additional features
  - Update main memory HBM model to use multiple channels
  - Atomic ALU constraints, TLB and I$ refinement
  - Add additional support and HW features as uncovered by tests
- After µBenches obtain high fidelity:
  - Validate larger benchmarks
  - Add known good models for this and other GPUs

Conclusions

- Having validated gem5 models is important
  - Existing GPU model does not always behave intuitively
  - Point solutions **insufficient**
- Solution: Iterative refinement through μBenches
  - Use microbenchmarks to tune for minimum absolute error in GPU model
  - Validate and release model improvements publicly
    - We've already released some patches!
  - Integrate performance regression testing into gem5


WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON