



# Narrowing the GAP: Enhancing gem5's GPU Memory Bandwidth Accuracy

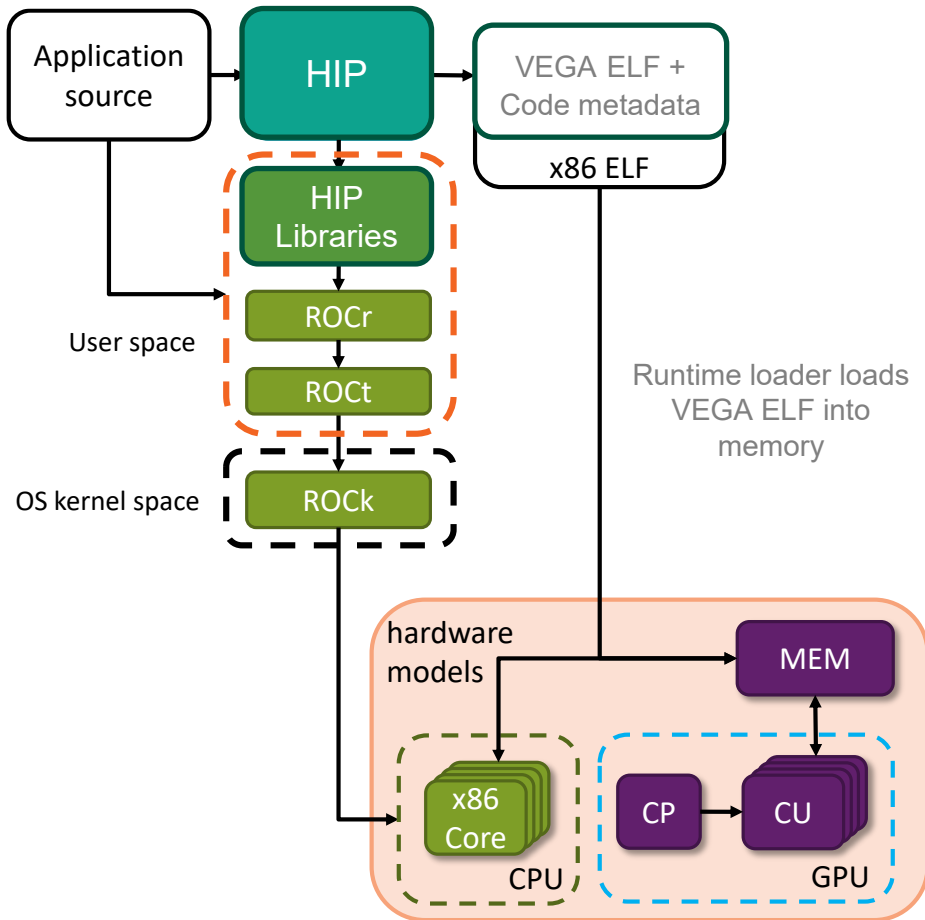
**Yu Xia\***, Vishnu Ramadas\*, Matthew Poremba\*\*, Matthew D. Sinclair\*

\*University of Wisconsin-Madison, \*\*AMD Research

[xia73@wisc.edu](mailto:xia73@wisc.edu)



# CPU-GPU SE Mode Support in gem5



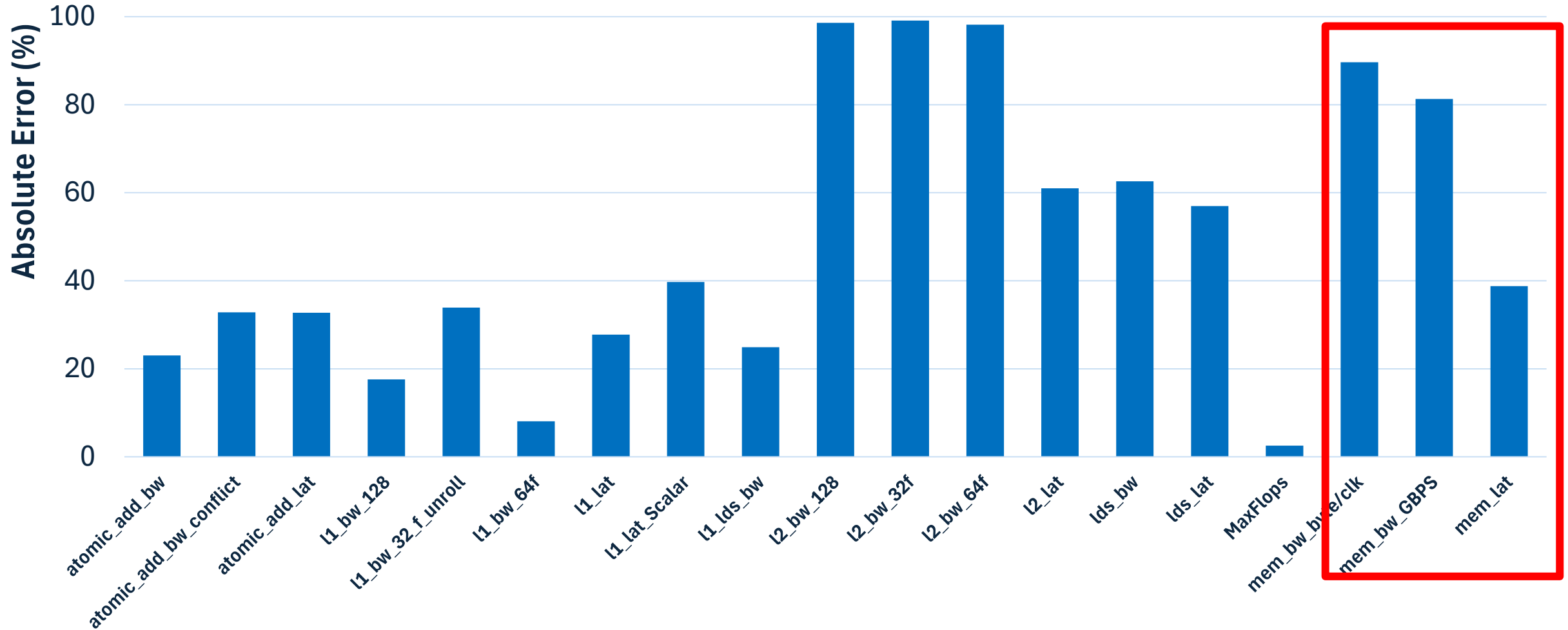
[Gutierrez et al., HPCA '18]

- Architects rely on simulators to rapidly prototype ideas
- gem5 widely used (execution-drive, cycle level), supports CPU-GPU systems
- AMD updated GPU support [Gutierrez, et al. HPCA '18]
  - Simulates end-to-end HIP application (AMD's GPGPU programming language) binaries
- Recent updates (by us and AMD)
  - Runs unmodified ROCm user stack
    - ROCm 4.0 SE mode, ROCm 6.4 FS mode
  - **Full-system mode with GPUs connected via PCIe now supported** (+ PyTorch) [Ramadas, et al. OSCAR'24]
  - Supports ML apps [Roarty & Sinclair gem5 Wkshop'20]
  - Recently added support for MI200/300 GPUs

**Validated simulator configurations are crucial**



# Initial gem5 GPU Fidelity (vs. Vega-class GPUs)

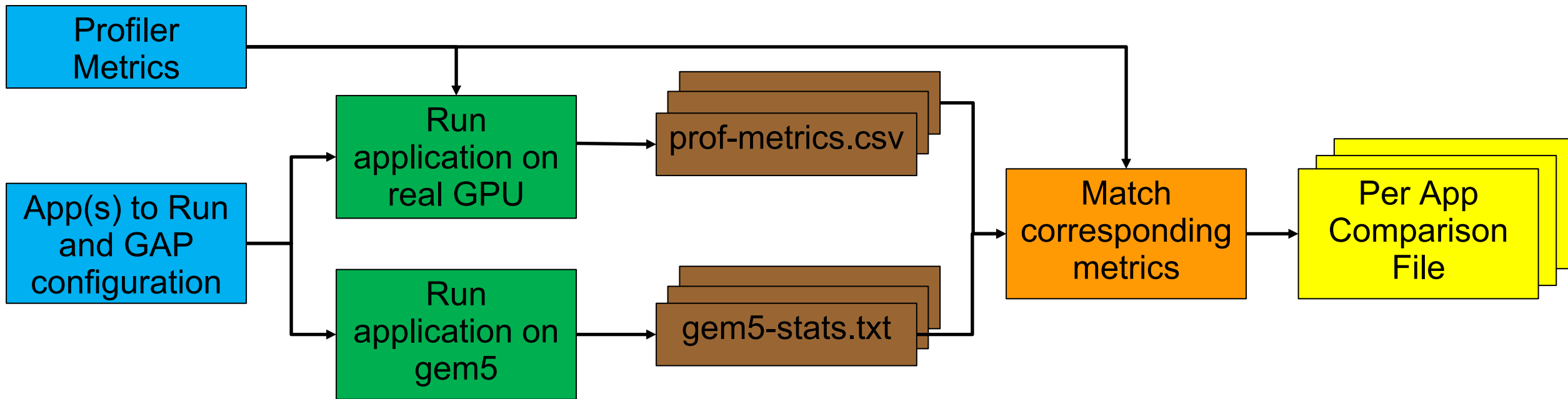


Most have significant error relative to real GPU (37% geomean error)



# How to Address These Shortcomings?

- Our prior work: start addressing these inefficiencies [Jamieson, et al. gem5 Wkshp'22], [Ramadas, et al. gem5 Wkshp'23], [Ramadas, et al. YArch'24]
- Issue: need standard, systematic approach for validating configurations
- Solution: gem5 GPU Accuracy Profiler (**GAP**)



**Use GAP to iteratively refine gem5 GPU models**

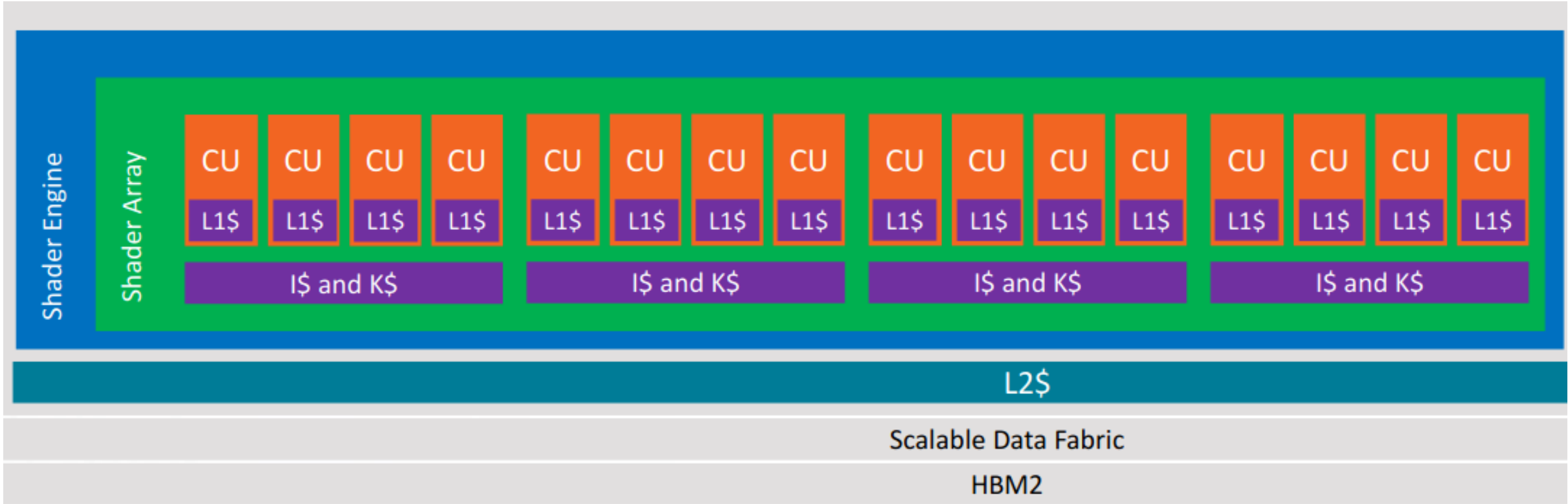


# GAP Methodology

- Created ~60 hand-tuned assembly-kernel microbenchmarks
  - Enables isolating behavior of different components
  - Target latency, bandwidth, size, and/or # of levels
  - Atomic operations (with/without conflicts), L1 I\$, L1 D\$, L2\$, LDS, Main Memory, TLBs/Page Table, Max FLOPs, MFMA operations, arithmetic latency, ...
- Compare microbenchmark gem5 vs. real GPU output with GAP
  - Target corresponding stats in gem5 and ROC profiler from real GPU
  - Use results to **iteratively refine**
  - Leverage patents, publicly disclosed info, and reverse engineering



# First Target: Vega-class GPU

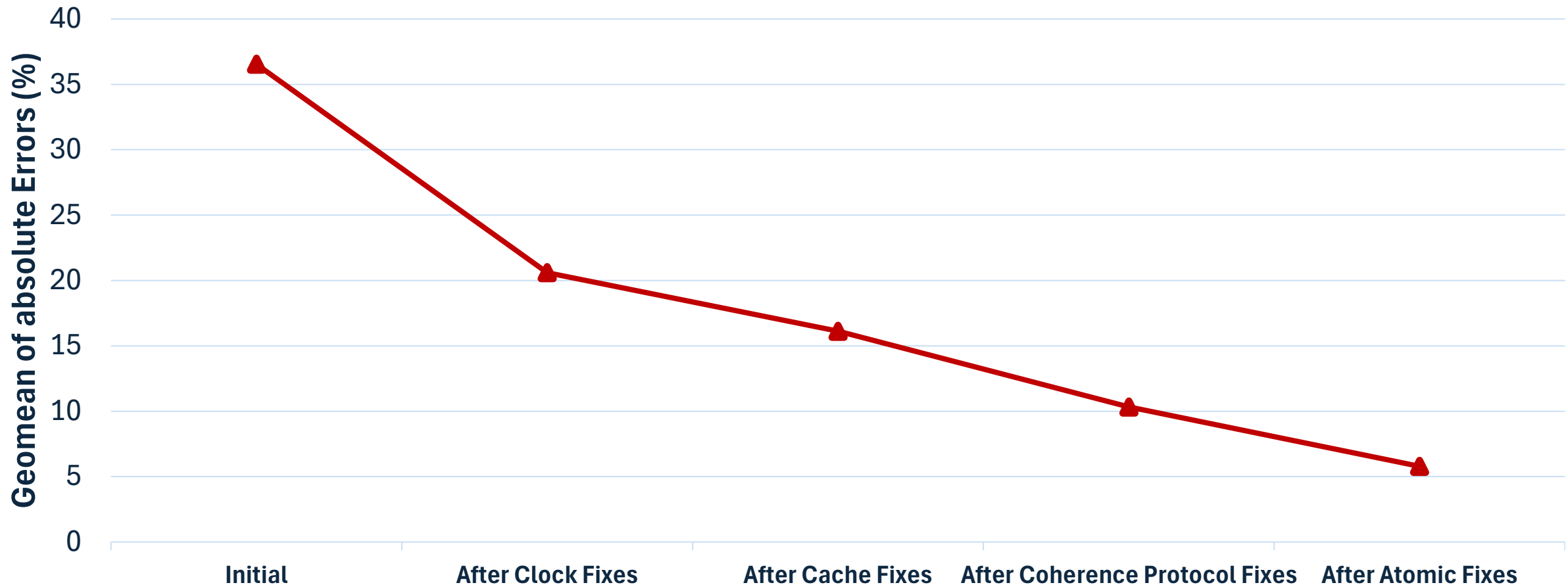


Source: AMD

**Added new HW support & configurations to model Vega GPUs  
Extending to MI200/300**



# Prior GAP Improvements



**Overall, much better fidelity (from 37% to 6% geomean)!**  
**But ... main memory still has huge error (e.g., 88% for memory BW)**  
**This Work: reduce main memory error (for GPUFS)**



# Outline

- Motivation
- **Design**
- Conclusion

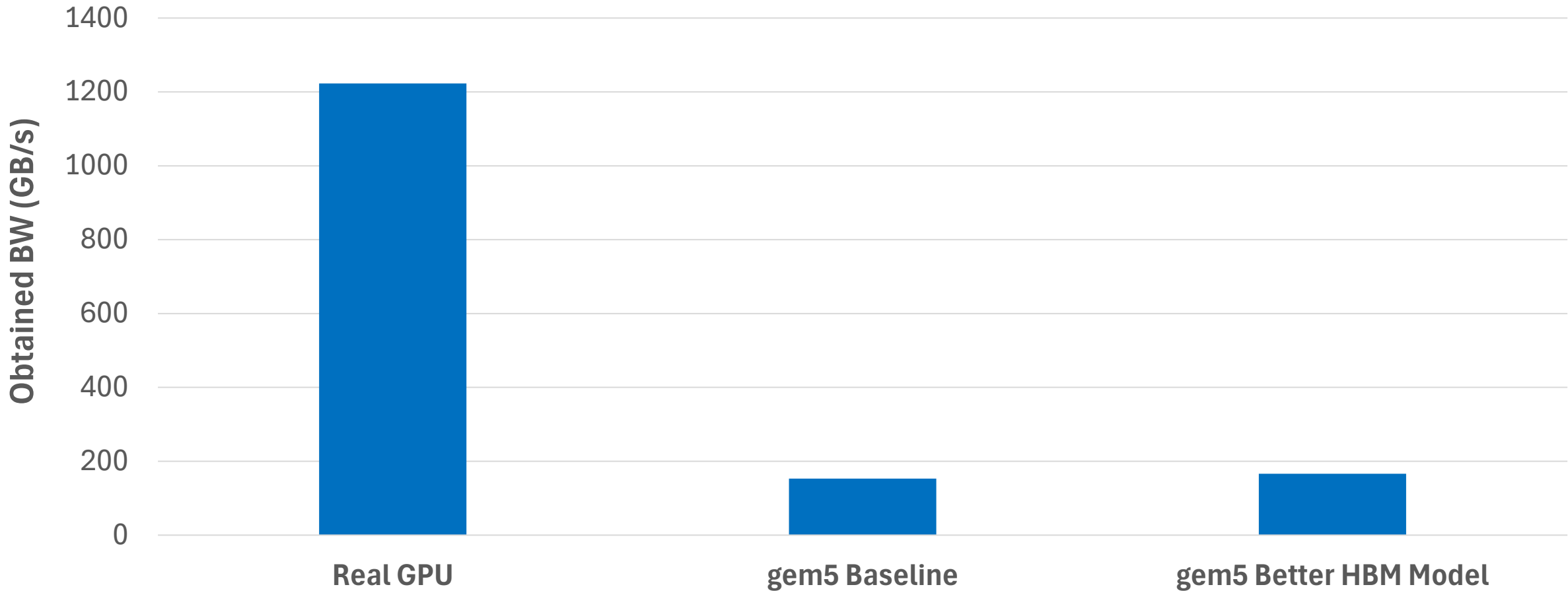


# Issue #1: Insufficient HBM2 Support

- Initial main memory BW error: 88% (gem5: 153 GB/s, real GPU: 1223 GB/s)
- Observation #1: GPU model supported HBM(2), but not at high fidelity.
- Solution #1:
  - Modify GPU model to support more accurate HBM2 [Akram, et al. gem5 Wkshp'22]
  - Validation: achieve target BW when directly connecting HBM2 to traffic generators
  - New HBM model supports 1 TB/s like real GPUs!
- Unfortunately, less effective sending traffic through the gem5 GPU to HBM2 model



# After HBM2 Support



Result #1: 153 GB/s → 166GB/s (86% error), although mem. latency error is better  
**Something between CUs & memory controller must be the bottleneck**



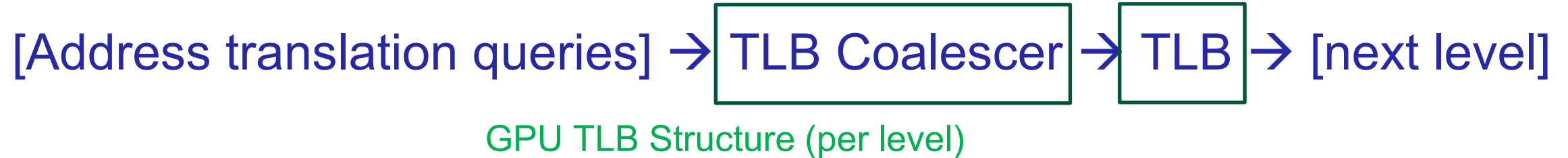
# Issue #2: Larger Pages

- Observation #2: ROCm 5 switched default to 2 MB GPU pages (from 4KB)
  - Current gem5 GPU TLB model only fully supports 4 KB pages
  - Consequence: many redundant translation requests when larger page used
- Reality: Majority of GPU memory accesses are to 2 MB pages
  - 99.85% L3 GPU TLB misses were to 2 MB pages (511 redundant translations each)
  - These redundant translations hurt memory system performance
- Solution #2: add multi-page size support to gem5 GPU
  - Larger page support reduces TLB misses
  - Should also increase address translation performance



# Issue #2: Larger Pages (Cont.)

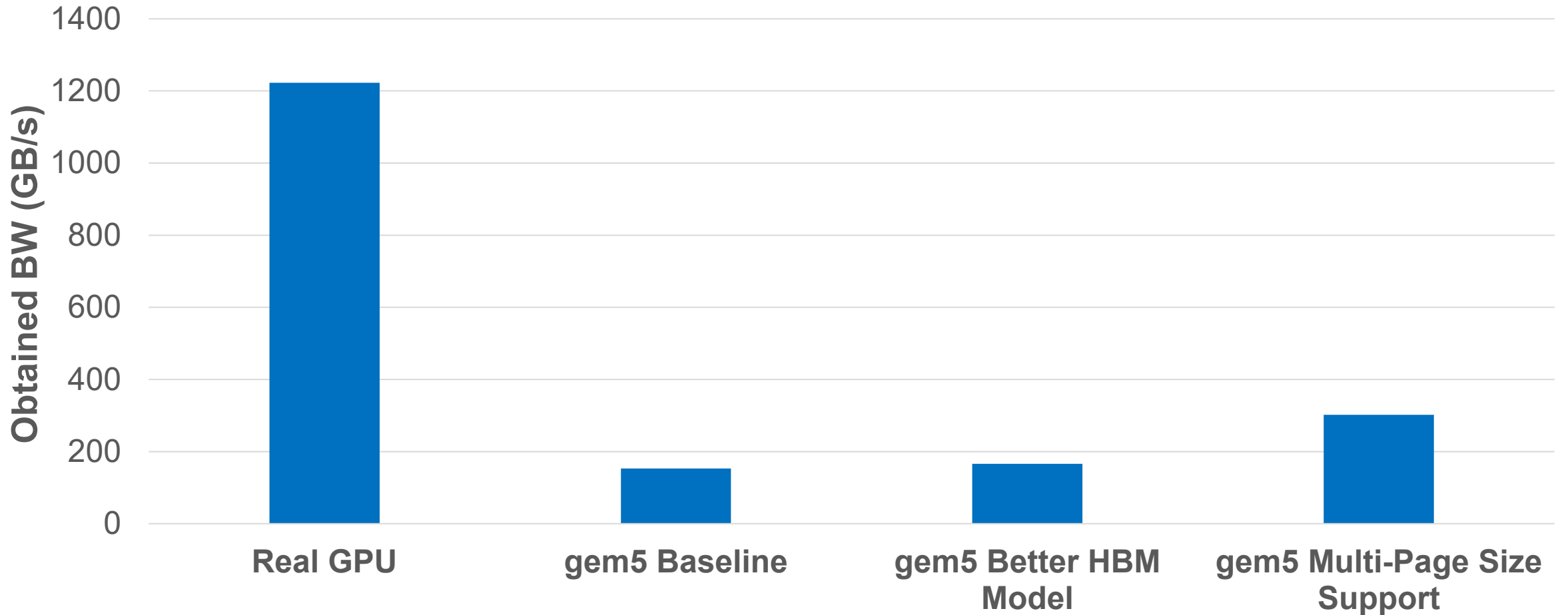
Both need upgrades



- Implementation:
  - Coalescer does not know the request's page size until request is returned
  - Solution #2A: Coalescer buffers all req(s) potentially belonging to same larger page
    - Trades off: increasing critical path if not a match vs. avoiding redundant requests
  - Solution #2B: TLB properly maps the VA translation according to its page size



# After Multi-page Size Support



Result #2: 166GB/s → 302 GB/s (75% error)

**Significant improvement, but something else still causing issues**

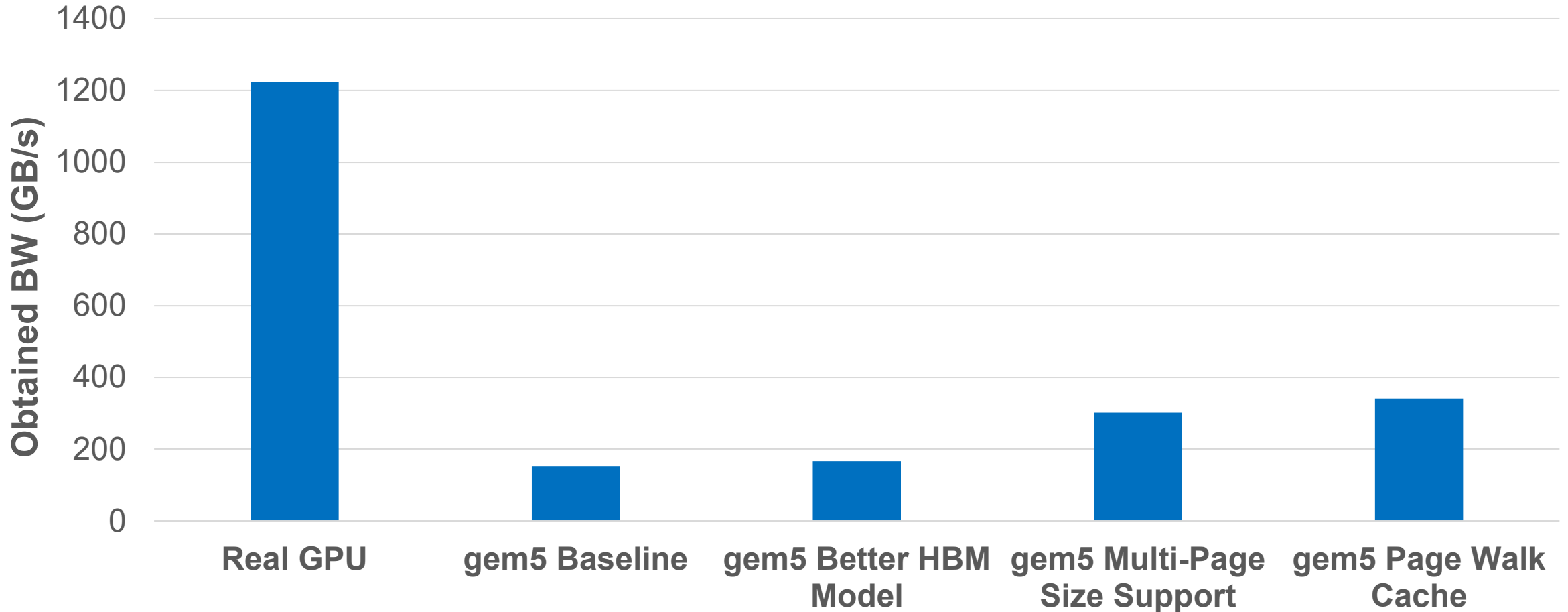


# Issue #3: Page Walk Overhead

- Observation #3: Page walks expensive in multi-level page tables
  - Each translation has multiple memory accesses
  - Tuning page table latency helps, but does not resolve root problem
- Solution #3: Add small page walk cache (PWC) to GPU page table walker
  - PWC caches frequently accessed intermediate nodes in multi-level page table
  - Reduces number of main memory accesses for each L3 GPU TLB miss
  - Experimentally tuned size of PWC – 64-entries enough



# After PWC

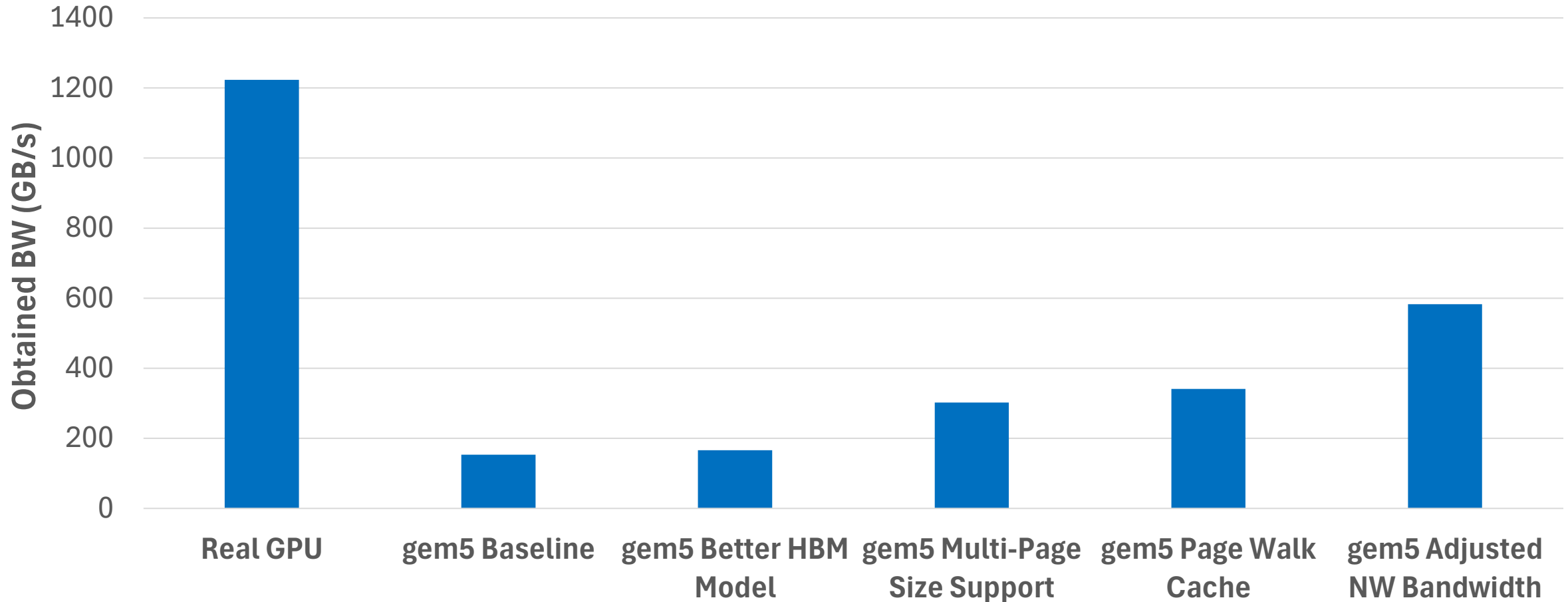


**Result #3: 302 GB/s → 341 GB/s (72% error)**

**There is one more adjustment we can do to Network**



# After Network Adjustments



**Result #4: 341 GB/s → 583 GB/s (52% error)  
Iteratively getting closer to target!**



# Outline

- Motivation
- Design
- **Conclusion**



# Conclusion

- Having validated “known good” gem5 models is important
  - Existing GPU models do not always behave intuitively
  - Point solutions **insufficient**
- Solution: more **automated** framework (**GAP**)
  - Prior GAP work: improve caches, specialized memories, FLOPs
- But significant bottlenecks remain, especially for main memory → fixes:
  - Higher fidelity main memory (HBM) model support
  - Multi-size page support to reduce redundant, unnecessary translations
  - Page walk cache to reduce overhead of page table walk
  - Adjust network bandwidth to better reflect GPUFS system
- **Optimizations significantly improve gem5 GPU’s main memory BW**
  - Ongoing: identify further bottlenecks with main memory BW
  - Ongoing: Continue to push support from this work → “known good” GPU models

