Improving gem5’s GPUFS Support

Vishnu Ramadas*, Matthew Poremba^, Bradford M. Beckmann^, and Matthew D. Sinclair*^  
*University of Wisconsin-Madison, ^AMD Research  
vramadas@wisc.edu
Outline

• Introduction
• Proposal
• Progress
• Conclusion and Future Work
Introduction: Challenges in Application Scaling

Simulating entire workloads would take months (or years) in modern gem5

How do we make it faster?

Source:
Introduction: Prior CPU-GPU Support in gem5

- Execution-driven, cycle-level
  - Models complex CPUs & GPUs
  - Rapid prototyping of new features
  - Validate simulation with execute-in-execute

- Prior work [Gutierrez et al. HPCA '18]
  - Runs unmodified ROCm 1.6 user stack
  - Simulates HIP and HCC applications
  - HCC/HIP are AMD’s GPGPU languages

Solid foundation, but does not support ML workloads
Introduction: ML Support in gem5 CPU-GPU system

We have started to add this support [Alsop IISWC ‘19], [Roarty gem5 Workshop ‘21]
Introduction : GPUFS Support

- Introduced in gem5 v22.0
  - Previously only supported SE mode with ROCm 4.0
  - FS mode supports ROCm 4.2

- Running in SE mode required either a specific host environment containing the ROCm stack or a Docker container that encapsulated this environment
  - GPUFS removes all host requirements

- Improves simulation speed by functionally simulating memory copies

- Adds KVM CPU-GPU support
Introduction: What is KVM CPU

- Kernel-based Virtual Machine (KVM):
  - Open-source virtualization technology built into Linux. Turns Linux into a hypervisor that allows the host machine to run a virtual machine

- KVM CPU allows simulation to fast-forward by running the CPU instructions directly on the virtual machine, instead of timing CPU models
  - Requires the application binary to be compiled for the host machine architecture

- Can be used in CPU-GPU systems to fast forward through CPU code
Outline

- Introduction
- Proposal
- Progress
- Conclusion and Future Work
Our Vision to Run Large-Scale Workloads

- Not all parts of the application are equally interesting
  - Some functions/code blocks are “more important” to its behavior

- Applications are simulated multiple times when evaluating new ideas

- **Key Insight** – some regions of the application can be run with low fidelity without affecting the way the other parts interact with the underlying hardware
  - Can use KVM CPU support in GPUFS to do this
Mixed Fidelity for Less Important Application Phases

- May not want to fully simulate certain phases of applications
- Solution: leverage gem5’s KVM CPU to functionally simulate these phases
Outline

• Introduction
• Proposal
• Progress
• Conclusion and Future Work

Improving gem5’s GPUFS Support
Using KVM CPUs: How Much Does This Help?

- First Step: Utilized KVM support to fast forward through CPU code
Using KVM CPUs: How Much Does This Help?

- Cycle Level GPU Simulation: 10-50 KIPS

- Functional KVM Simulation: 100s MIPS
  - KVM CPU emulating GPU: 10s MIPS

- Conservative speedup for a kernel containing 2B SIMD instructions:
  - 11 hours of cycle-level GPU simulation
  - 3 minutes to execute on KVM CPU – single threaded

On-going Work: full set of results for GPU workloads
Further Refinement: Checkpoints

• Users often simulate the same application many times

• Can speedup the execution by not redoing the less important parts

• Solution: create checkpoints (ala CPU SimPoints)
  • Capture the state of the execution when a checkpoint is taken
  • Restore this state the next time the application is run
  • Resume execution from the next instruction after restoration

• Previously only possible for CPUs
  • Added support in GPUs, leveraging gem5’s FS mode and m5 operations
Can We Do Even Better (Faster)?

• Current Task: convert less-important GPU kernels into CPU code

• Update LLVM GPU backend to emit CPU code for kernels

• Use KVM CPU (low fidelity) or another CPU model (medium fidelity)

• Most important phases get max fidelity, others get less fidelity
Can We Do Even Better (Faster)?

- Functionally simulate GPU kernels on CPU
- Preliminary results: only 1.58x – 3x slower on KVM vs bare metal (1 thread)

**Mixed Fidelity makes gem5 much closer to real HW**
Outline

• Introduction
• Proposal
• Progress
• Conclusion and Future Work
Conclusion and Future Work

• Large-scale applications that run on the GPU models take extremely large simulation times

• Our updates are the first in a series to significantly reduce runtime for such workloads

• Significantly improves usability and reduce barriers to entry for simulation

• Future Work
  • Profile ML workloads to find regions that can be annotated for checkpointing
  • Integrate other accelerators into mainline gem5
  • Support accelerator fast-forwarding and checkpointing
  • Additional publicly available applications and resources