Efficient Memory Sharing Between CPU and GPU in Gem5

Stephen Singh, Austin Kee, Thomas Tymecki, Andrew Femiano

Original Proposal Description

Heterogeneous Memory Sharing

Explores efficient memory sharing and cache coherence in CPU-GPU systems within gem5.

Objectives

Develop a shared memory model that uses both cpu and gpu caches. Design mechanisms for unified memory access or cache coherency.

Original Proposal Description

Proposed Approach and Team Roles

Proposed Approach
Build a CPU-GPU system in gem5 with shared memory and efficient TLB management to maximize efficiency in models.

Team Roles
Team members specialize in gem5 setup of different CPU/ GPU models, memory optimization, performance analysis, and debugging.

Project Progress Summary

GPU Modeling **Advantages**

Gem5-GPU:

- Detailed Heterogeneous models
- Built around already existing Gem5 builds with x86 support
- OpenCL support for parallelism and custom kernels

GPGPU:

- Detailed Heterogeneous System Architecture
- Uses Cuda toolkit for Nvidia Drivers and has Nvidia graphics cards
- Gem5 samples built in for various cache coherence statistics

Full System AMD GPU:

- Heterogeneous models using AMD CPU
- Built around already existing Gem5 and up to date
- Current active community development

Project Progress Summary`

GPU Modeling **RoadBlocks**

Gem5-GPU:

- Outdated versions and unsupported past Ubuntu 16.04
- Poor Documentation of expected run results
- Requires different configuration files in gem5 that is not found to work with different cache models

GPGPU:

- Difficult to get working on a VMware
- Cuda Toolkit is only supported to work with Gem5 in Ubuntu models before 16.04 and Cuda 8.0 even though Cuda is at version 13
- Documentation on how to properly setup and get a GPU to work in Linux is very limited or non existent

Full System AMD GPU:

- requires knowledge of how to use and setup a KVM
- Very intensive workloads and long simulation/setup times
- very limited working samples which are under development and not ready to use unless very proficient with system

Project Progress Summary

GCN3 GPU Model

1.Compute Units (CUs)

- Simulates multiple compute units, which are the basic building blocks of GPU computation.
- Each CU contains:
  - **Scalar Units**: For lightweight, scalar operations.
  - **SIMD Units**: For highly parallel, vectorized operations.
  - **LDS (Local Data Store)**: For shared memory within a CU.

2. Memory Hierarchy

- Includes a detailed memory model:
  - **Private Memory**: Each thread's local memory.
  - **Shared Memory**: Accessible by all threads within a workgroup.
  - **Global Memory**: Accessible by all threads across all workgroups.
- Simulates memory latency, bandwidth, and contention, enabling accurate performance analysis.

3. Heterogeneous System Architecture (HSA) Support

- Supports **Heterogeneous Unified Memory Access (hUMA)**, allowing both CPUs and GPUs to share a unified memory space.
- Facilitates easy data sharing between CPUs and GPUs.

4. ROCm Integration

- Works with AMD's ROCm (Radeon Open Compute) platform.
- Supports OpenCL and HIP (Heterogeneous Interface for Portability) workloads, enabling realistic heterogeneous application simulations.

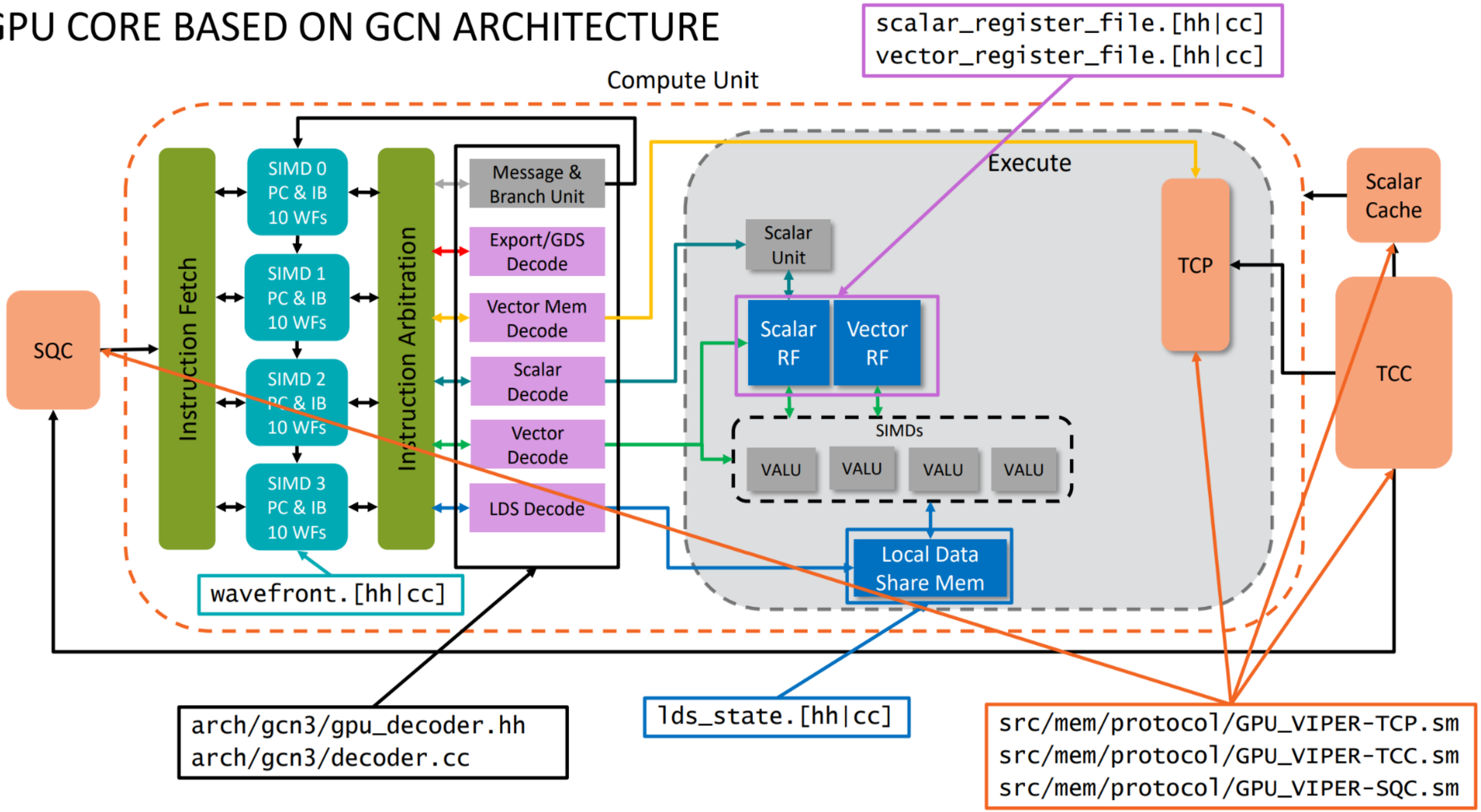
5. Instruction Set Architecture (ISA)

- Implements the AMD GCN3 ISA, providing a detailed and accurate representation of GPU instruction execution.
- Includes support for operations like ALU instructions, branching, and memory operations.

6. Full-System Simulation

- Can be used in a full-system gem5 setup to simulate a complete computing environment, including CPUs, memory, I/O, and GPUs.
- Enables end-to-end analysis of heterogeneous workloads.

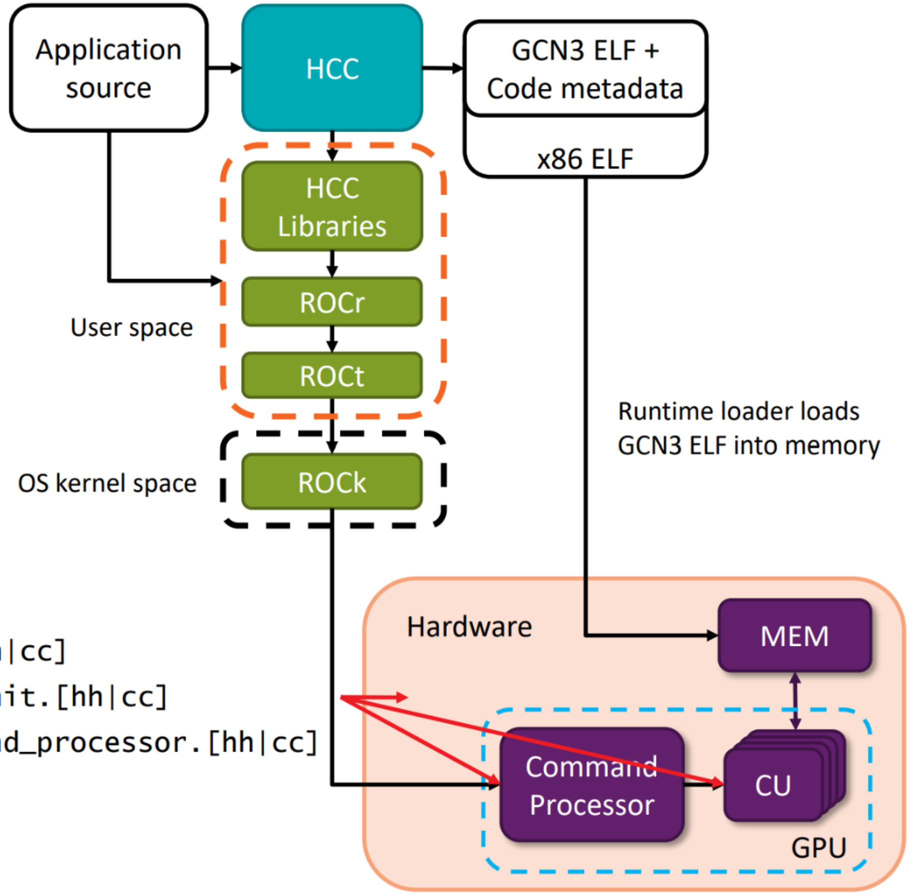
GPU CORE BASED ON GCN ARCHITECTURE



Result and Deliverable

- HCC
  - Clang front end and LLVM-based backend
    - Direct to ISA
    - Multi-ISA binary (x86 + GCN3)
- ROCm Stack
  - HCC libraries
  - Runtime layer – ***ROCr***
  - Thunk (user space driver) – ***ROCt***
  - Kernel fusion driver (KFD) – ***ROCk***
- GPU is a HW-SW co-designed machine
  - Command processor (CP) HW aids in implementing HSA standard
  - Rich application binary interface (ABI)
- GPU directly executes GCN3 ISA
  - Runtime ELF loaders for GCN3 binary

```
shader.[hh|cc]
compute_unit.[hh|cc]
gpu_command_processor.[hh|cc]
```



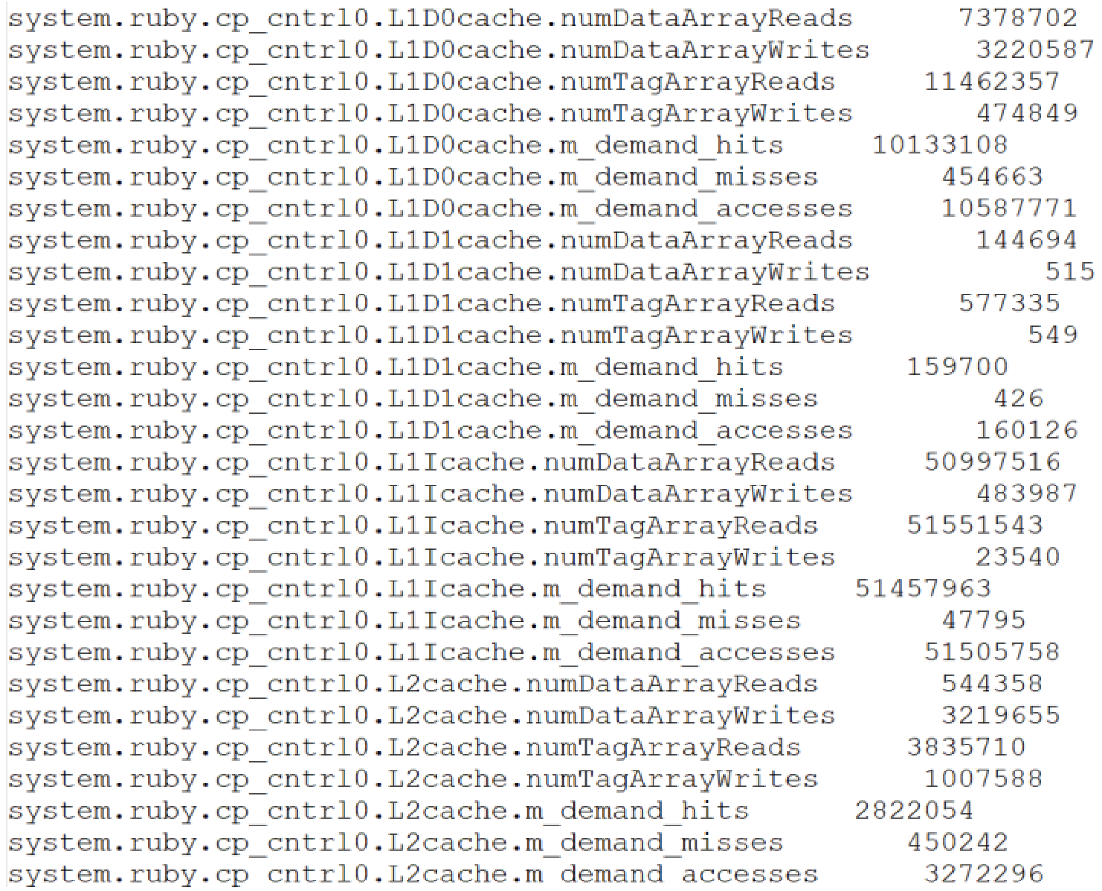
Results Standard APU workload

L1DCache:

- Data Array Reads: ~7.37M
- Data Array Writes: ~3.22M
- Tag Array Reads: ~11.46M
- Tag Array Writes: ~4.74M
- Demand Hits: ~10.31M
- Demand Misses: ~454K

L1ICache:

- Data Array Reads: ~50.99M
- Data Array Writes: ~483K
- Tag Array Reads: ~51.55M
- Tag Array Writes: ~23.54K
- Demand Hits: ~51.45M
- Demand Misses: ~47.79K



```
system.ruby.cp_cntrl0.L1D0cache.numDataArrayReads      7378702
system.ruby.cp_cntrl0.L1D0cache.numDataArrayWrites      3220587
system.ruby.cp_cntrl0.L1D0cache.numTagArrayReads     11462357
system.ruby.cp_cntrl0.L1D0cache.numTagArrayWrites       474849
system.ruby.cp_cntrl0.L1D0cache.m_demand_hits     10133108
system.ruby.cp_cntrl0.L1D0cache.m_demand_misses       454663
system.ruby.cp_cntrl0.L1D0cache.m_demand_accesses     10587771
system.ruby.cp_cntrl0.L1D1cache.numDataArrayReads       144694
system.ruby.cp_cntrl0.L1D1cache.numDataArrayWrites           515
system.ruby.cp_cntrl0.L1D1cache.numTagArrayReads       577335
system.ruby.cp_cntrl0.L1D1cache.numTagArrayWrites           549
system.ruby.cp_cntrl0.L1D1cache.m_demand_hits       159700
system.ruby.cp_cntrl0.L1D1cache.m_demand_misses          426
system.ruby.cp_cntrl0.L1D1cache.m_demand_accesses       160126
system.ruby.cp_cntrl0.L1Icache.numDataArrayReads     50997516
system.ruby.cp_cntrl0.L1Icache.numDataArrayWrites       483987
system.ruby.cp_cntrl0.L1Icache.numTagArrayReads     51551543
system.ruby.cp_cntrl0.L1Icache.numTagArrayWrites        23540
system.ruby.cp_cntrl0.L1Icache.m_demand_hits     51457963
system.ruby.cp_cntrl0.L1Icache.m_demand_misses       47795
system.ruby.cp_cntrl0.L1Icache.m_demand_accesses      51505758
system.ruby.cp_cntrl0.L2cache.numDataArrayReads       544358
system.ruby.cp_cntrl0.L2cache.numDataArrayWrites      3219655
system.ruby.cp_cntrl0.L2cache.numTagArrayReads      3835710
system.ruby.cp_cntrl0.L2cache.numTagArrayWrites      1007588
system.ruby.cp_cntrl0.L2cache.m_demand_hits     2822054
system.ruby.cp_cntrl0.L2cache.m_demand_misses      450242
system.ruby.cp_cntrl0.L2cache.m_demand_accesses      3272296
```

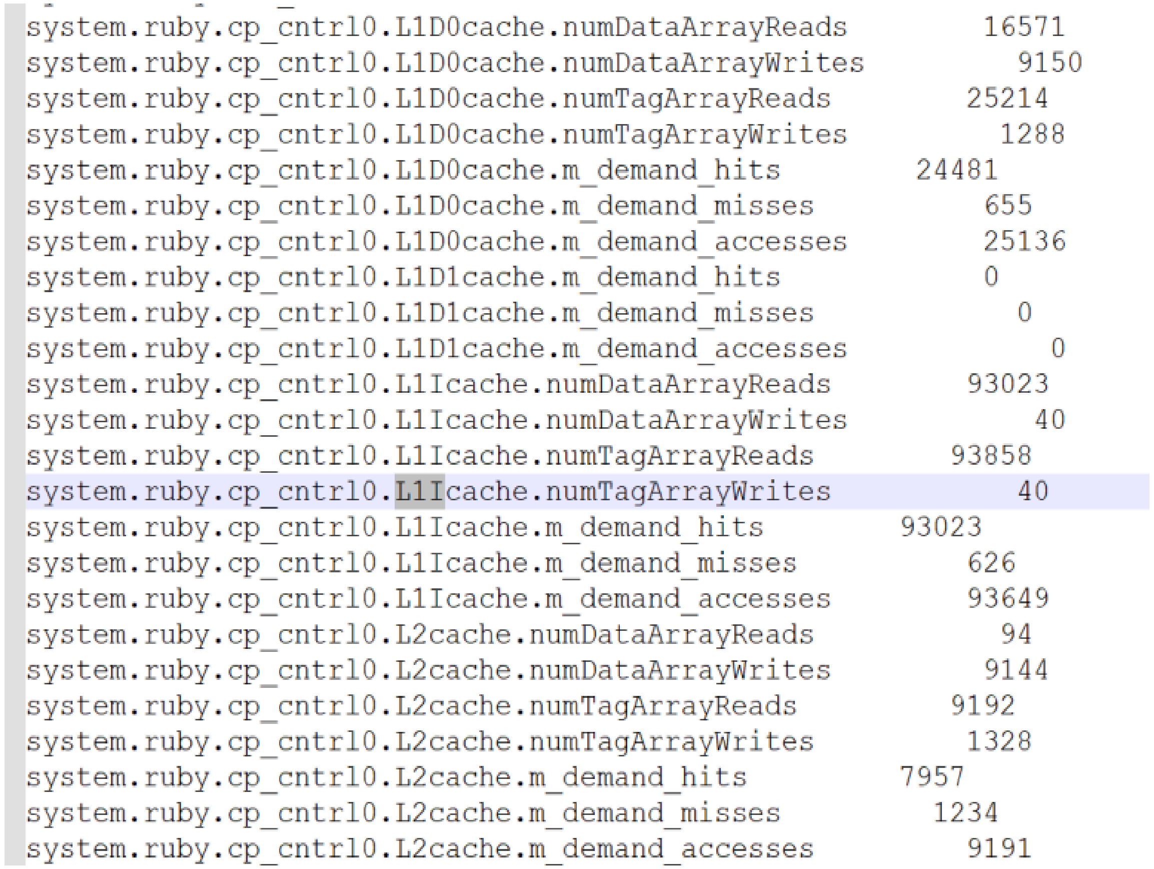
Results Custom APU workload

L1DCache:

- Data Array Reads: 16,571
- Data Array Writes: 9,150
- Tag Array Reads: 25,214
- Tag Array Writes: 1,288
- Demand Hits: 24,481
- Demand Misses: 655

L1ICache:

- Data Array Reads: 93,023
- Data Array Writes: 40
- Tag Array Reads: 93,858
- Tag Array Writes: 40
- Demand Hits: 93,023
- Demand Misses: 626



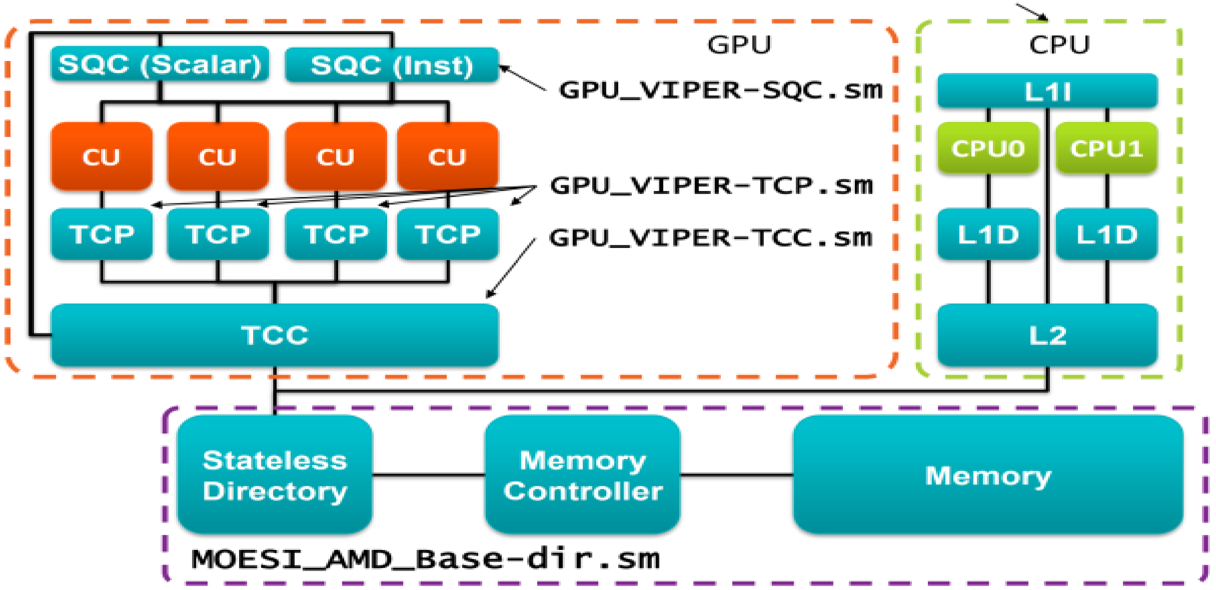
```
system.ruby.cp_cntrl0.L1D0cache.numDataArrayReads        16571
system.ruby.cp_cntrl0.L1D0cache.numDataArrayWrites         9150
system.ruby.cp_cntrl0.L1D0cache.numTagArrayReads         25214
system.ruby.cp_cntrl0.L1D0cache.numTagArrayWrites          1288
system.ruby.cp_cntrl0.L1D0cache.m_demand_hits         24481
system.ruby.cp_cntrl0.L1D0cache.m_demand_misses          655
system.ruby.cp_cntrl0.L1D0cache.m_demand_accesses        25136
system.ruby.cp_cntrl0.L1D1cache.m_demand_hits            0
system.ruby.cp_cntrl0.L1D1cache.m_demand_misses            0
system.ruby.cp_cntrl0.L1D1cache.m_demand_accesses            0
system.ruby.cp_cntrl0.L1Icache.numDataArrayReads         93023
system.ruby.cp_cntrl0.L1Icache.numDataArrayWrites            40
system.ruby.cp_cntrl0.L1Icache.numTagArrayReads         93858
system.ruby.cp_cntrl0.L1Icache.numTagArrayWrites             40
system.ruby.cp_cntrl0.L1Icache.m_demand_hits         93023
system.ruby.cp_cntrl0.L1Icache.m_demand_misses           626
system.ruby.cp_cntrl0.L1Icache.m_demand_accesses         93649
system.ruby.cp_cntrl0.L2cache.numDataArrayReads            94
system.ruby.cp_cntrl0.L2cache.numDataArrayWrites           9144
system.ruby.cp_cntrl0.L2cache.numTagArrayReads          9192
system.ruby.cp_cntrl0.L2cache.numTagArrayWrites          1328
system.ruby.cp_cntrl0.L2cache.m_demand_hits         7957
system.ruby.cp_cntrl0.L2cache.m_demand_misses         1234
system.ruby.cp_cntrl0.L2cache.m_demand_accesses          9191
```

Comparison

1. **Efficiency:**
   - The first setup (first image) shows significantly higher cache activity, indicating a larger workload or a more intensive simulation. This includes more demand hits and higher tag array access rates.
   - The second setup (second image) has lower activity but still achieves a high hit rate relative to its workload.
2. **Hit/Miss Ratio:**
   - For L1DCache in the first image, the demand hit rate is around **95.8%**, while in the second image, it is around **97.4%**.
   - For L1ICache in the first image, the demand hit rate is around **99.9%**, while in the second image, it is around **99.3%**.
3. **Tag Array and Data Array Access:**
   - The first setup has substantially more tag and data array accesses, which may result in more overhead but is also indicative of handling more complex or larger workloads.

Final Results Analysis

- **Better Cache Coherence:** The **second setup** demonstrates better cache coherence for lighter workloads, with a slightly higher hit rate and less overhead.
- **Workload Intensity:** If your goal is to test the system's handling of intensive workloads, the **first setup** appears to be more reflective of a high-demand environment.
- **Choice Depends on Objectives:** Use the first setup for stress-testing or performance benchmarking. Use the second setup if aiming for efficient, coherent memory access in smaller workloads.



Limitations

Quote gem5 documentation:

"The supported cpu_types are X86KvmCPU and AtomicSimpleCPU as timing CPUs do not support the disjointed Ruby network required to simulate a discrete GPU."

The implication here is that simulations currently cannot account for race conditions and competitions for cache access between a threaded CPU application and a discrete GPU, as the CPU is simulated without timing information.

The lack of detailed CPU timing models in GPU simulations suggests that the results can only be interpreted about ideal access conditions.

Future Work

With the current limitations of gem5 established, we propose additional development on gem5 to provide some support for CPU timing together with GPU modelling.

As Unified Memory and zero-copy architectures become more common in the wild for heterogeneous computing (i.e.: ARM Mali, Apple/Imagination PowerVR, Apple M-series), there is greater interest in simulating such architectures.

Since it is claimed that the Ruby network simulation is too complex to include CPU behavior in the model, we propose extending Garnet 2.0 instead with a simplified GPU model.

Future Work

According to the gem5 documentation: some parameters that modify GPU configuration only modify the value in gem5 and not the simulated device

For example: “The dgpu_mem_size parameter does not change the amount of memory seen by the device driver and is hardcoded to 16GB in C++”

Since some parameters cannot be changed on simulated devices, optimizing the system’s architecture on simulated devices are limited

If modifying parameters like dgpu_mem_size is supported, then the effects heterogeneous models can be studied at various memory sizes

Summary

Key Challenges

- **GPU Setup Failures**:
  - GPGPU setup limited to specific Ubuntu versions (16.04 or earlier).
  - Lack of comprehensive documentation for Linux GPU configuration.
  - Outdated gem5-GPU models with poor compatibility and limited cache support.
- **Simulation Constraints**:
  - gem5 lacks proper support for race conditions and competition for cache access.
  - Limited timing CPU support, impacting the realism of heterogeneous workload simulations.

GCN3 GPU Model Highlights

- **Compute Units (CUs)**: Enabled lightweight operations (Scalar Units) and parallel computations (SIMD Units).
- **Unified Memory Access (hUMA)**: Allowed efficient CPU-GPU data sharing.
- **ROCm Integration**: Supported OpenCL and HIP workloads for realistic testing.

Workload Results

- **Standard High APU Workload**:
  - High cache activity, ~95.8% L1DCache demand hit rate.
- **Custom Low APU Workload**:
  - Lower activity but higher efficiency, ~97.4% L1DCache demand hit rate.

Key Findings

- **Efficiency vs. Intensity**:
  - Higher activity in standard workloads reflects stress-testing scenarios.
  - Custom Low workloads favor better cache coherence and lighter simulation overhead.

Future Directions

- Expand **CPU timing models** for realistic GPU simulation.
- Improve parameter customization (e.g., memory size modifications) for flexible architecture testing.
- Enhance simulation support for unified memory in heterogeneous systems.