Validating gem5's Memory Components

Mahyar Samani, Jason Lowe-Power

Architecture simulators are a common and powerful tool in computer architecture design. It is important that results reported by simulators are trustworthy. The accuracy of simulators should be evaluated by comparing their reported statistics and results to their counterparts in real hardware. We will present our methodology and tools for evaluating gem5's memory subsystem components and the results of our validation of gem5's current memory system components.

I. Methodology

The methodology focuses on isolating the component under test from other components in the system. We only focus on the components of the memory subsystem (i.e. DRAM and cache). Moreover, for each component, we only consider measurements that directly effect the performance of other components in the system. To that end, we use bandwidth and average latency as the metric for comparison. We use synthetic traffic to stress each component to factor out any inaccuracy that might originate from processor models. We use DRAMSim3 as the reference of comparison in validating DRAM models. DRAMSim3 is a cycle accurate memory simulator that has been validated by comparing against real hardware. In addition, DRAMSim3 has already been integrated to work work with gem5.

II. Tools and Utilities

We use gem5's builtin synthetic traffic generators, PyTrafficGen, and our new traffic generator, GUPSGen, to generate different types of traffic and measure bandwidth and average latency of accesses.

PyTrafficGen provides a configurable interface for creating synthetic traffic. It can create sequential and random access patterns with configurable demand bandwidth, access range, and access granularity.

We developed GUPSGen to create traffic specified by HPCC RandomAccess benchmark. It will execute a key-value store program without the need for a processor model in the system. RandomAccess benchmark has been recognized as a common benchmark for the memory subsystem.

III. Validating DRAM Models

To evaluate DRAM models, we use different configuration of PyTrafficGen (e.g. demand bandwidth, access pattern, access range) and DRAM models (e.g. address mapping, paging policy) and use bandwidth and average latency measurements as metrics to compare gem5's models with respective models in DRAMSim3. We validated the accuracy of DRAM models in gem5. Figure 1 shows a comparison of bandwidth readings from a DDR4 model between gem5 and DRAMSim3.

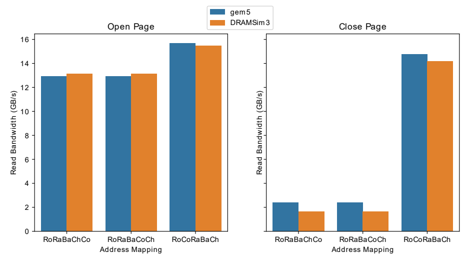

Fig. 1. Comparison of DDR4 in gem5 and DRAMSim3 under linear traffic. gem5 measurements are within 5% of DRAMSim3 measurements.

IV. Validating Ruby Caches

We validate the accuracy of a two level cache hierarchy set up using Ruby caches. We use SimpleMemory as a deterministic model for the backing DRAM of the tested cache hierarchy. Using SimpleMemory allows us to isolate the cache hierarchy without the requirement for a validated memory model. We validate bandwidth and average latency measurements under different traffic patterns that cause different hit rates in each level of the cache hierarchy. Figure 2 shows the effect of working set size on measured bandwidth and average latency.

Next, we configure a cache hierarchy using available information on Intel Skylake architecture. To create a complete memory subsystem we use a validated DDR4 model as the main system memory. We use GUPSGen to measure the performance of the memory subsystem and compare the results to the readings from real hardware.

V. Conclusion

Overall, we validate the accuracy of DRAM models in gem5 and report a 10% difference between GUPS measurements from our tested cache hierarchy and real hardware.

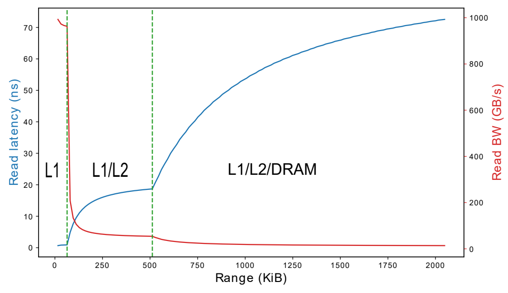

Fig. 2. Effect of hierarchical memory on access latency and bandwidth.