15418 Project Checkpoint: GSim

Group Members: Prithvi Cherabuddi (pcherabu) & Shashwat Gupta (shashwag)

Work Completed So Far

Note: Please refer to the project site for an upto date schedule. At the time of this writeup, the current schedule is as follows:

- Week 1.0: Explored the NVidia Cuda compiler flow with NVCC. We primarily focused this half of the week in taking a look at the CUDA binary utilities that exist for Nvidia's developer community. The focus revolved around cuobjdump (operating from x86 environments only).
- Week 1.5: Found a couple of bottlenecks in our work flow; this stemmed from our realization that we can't simply use cuobjdump and .cudin files as inputs to our simulator. Primarily, we have realized that we would require to make our own "input files" for GSIM. We have termed these files as "gimage files".
- Week 2.0: Week 1 was full of problems understanding the nvcc toolchain in order to get it to work and develop our own toolchains from interstage nvcc compiler outputs. Hence, we decided to split the work and bring the Proposed Week 3 schedule of developing a scheduler and the interface between our front end and back end of our simulator. Thus, work for generating gimages from .cudin files and developing the scheduler are going on silmuntaneously.
- Week 2.5: Scheduler work presses on, still coding up MInst and AInst classes and their member functionality. Core design of a single core is saved to Week 3 as initially proposed by the schedule.
- Week 3.0 (Current week... Checkpoint Update): Scheduler work is on hold and the entire team is on the front end problem. The current design plan is to have our front-end be a wrapper around nvbit (because it does most of the work we're trying to do from scratch).

Work To Be Done

- Week 3.5: Integrate NVBit and SASSI to have GSIM "replace" its front end with options that will call different dynamic instrumentation tools integrated with these PINs. Our simulator will now generate its own trace files by taking in normal .cu code files and giving it into NVBit (for example) which will output certain instrution information to use as our front-end.
- Week 4.0: Finish writing all dynamic instrumentation tools on NVBit to generate instruction traces. Start up scheduler work and wrap it up (at this point we're done with all credit checks on core resources and shedule out blocks to each core).
- Week 4.5: Start on the core model (the actual simulation of the instruction itself). Start developing cache and memory subsystem model.
- Week 5.0: Come back on the proposed schedule and draft a rough model of shared memory at high level caches and the device memory. This means simulating coherency manager and introducing invalidation protocols on our L1 cache. This final week will be the wrap up on calculating AI.
- Week 5.5: Finish up the CM and shared L2 (have ideal dynamic memory if time does not permit). Introduce flow statistics here to calculate AI and histogram support via maps.

Summary:

- ➔ Developed Simulator Infrastructure (Front-End support and Back-End Support).
- ➔ Developed queues that have input and output cycle latencies
- ➔ Developed latches for o cycle latency signals to communicate between stages in our simulator
- ➔ Developed FInst, MInst, and AInst packet classes that are transmitted through the scheduler from fetch till completed execution.

Project Deliverables

The Project deliverables remain the same as the proposal. However, the design methodology changed. We still aim to develop a cycle approximate model; nevertheless, our front-end doesn't do as much as estimated in our project proposal. It rather is a wrapper around existing tools that will feed in our cycle-approximate back end model. We recognize that we will lack correlation at the memory architecture side of things, but we will simulate everything else in order to calculate relevantly accurate Arithmetic Intensities of any .cu inputted.

It would be "nice to have" a front-end that does the work we wanted to in the proposal. However, this requires forward progress in our tool-chain support to generate "gimages" into our simulator. This has been a massive time-dump in the first two weeks and is better to explore on a less time-sensitive deadline. However, having this in makes our simulator a complete "cycle accurate model" (something like GPGPUsim).

Demo

As in the project proposal, we'll be able to dynamically show simulation statistics of code. This includes dynamically calculating Arithmetic Intensity, showing histogram statistics of any micro-architectural flow in the back-end, and scheduler decisions made due to the nature of the code.

The input of the simulator can be any valid build CUDA binaries as supported by this timeline's NVCC versions.

Concerns

Due to the already listed concern of generating gimages from our own toolchains operating on NVCC's intermediate outputs, we have shifted our front-end to be designed around existing instrumentation tools. This hasn't been scoped and may cause delays that will make it difficult to model accurate shared caches (which is upcoming shortly in 1.5 weeks). This means we have to compromise on accuracy for the simulator version presented by the deadline.

Note: Project Webpage is https://pcherabu.github.io/gsim.github.io/