CS259 - LAB1

The objective of this lab is to introduce you to the basics of parallel programming, execution and evaluation on GPU.
We will be using the Vortex OpenGPU (https://github.com/vortexgpgpu/vortex) for this experiment.

Part1 - Documentation

We recommend reading to first read the main Vortex publication to learn about the OpenGPU ISA, microarchitecture, and software stack. The vortex MICRO 2023 tutorial is also useful. The github documentation an MICRO tutorials are also valuable resources.

- Publication: https://dl.acm.org/doi/10.1145/3466752.3480128
- Documentation: https://github.com/vortexgpgpu/vortex/blob/master/docs/index.md
- Tutorials: https://github.com/vortexgpgpu/vortex_tutorials

Part2 - Installation and setup

Install the OpenGPU on your development machine using the instructions in the README.md in github. Follow the full instructions, including running the demo example. A Ubuntu 18.04 system will be required. You can use WSL 2.0 on Windows or a VM on MAC to set that up.

Part2 - Programming (4pts)

You will implement a parallel reduction sum kernel on the Vortex GPU.
You will be implementing it directly using Vortex C++ API.
Vortex applications consist of two parts: 1) host CPU code (main.cpp), and 2) GPU kernel code (kernel.cpp). main.cpp implements the host code that compiles and executes on your CPU. This program handles the allocation of resources the GPU needs, the transfer of those resources together with the kernel binary to the GPU. The code uses the Vortex runtime API (https://github.com/vortexgpgpu/vortex/blob/master/runtime/include) to query, allocate memory, copy memory, and execute kernel on GPU device.You will see those runtime API calls inside main.cpp. Kernel.cpp implements the GPU device kernel, this program is compiled into RISC-V and executes the GPU hardware/simulator. Vortex GPU kernels use the Vortex kernel library (https://github.com/vortexgpgpu/vortex/tree/master/kernel/include) to access the kernel API. A common kernel API is *vx_spawn_tasks* that is used for scheduling a task for execution on the GPU hardware threads. A typical Vortex kernel program includes a *main()* function that is executed on a single thread at launch, and a *kernel_body()* function that implements the parallel code of the kernel.

a) Clone the parallel *sort* test under https://github.com/vortexgpgpu/vortex/tree/master/tests/regression into a new folder named “psum” to create your new test

b) Rename PROJECT value in Makefile to psum
c) Modify *gen_ref_data*() in *main.cpp* to generate the psum gold data for verification
d) Modify *kernel_body*() in *kernel.cpp* to implement the parallel reduce sum.
e) Execute your new application using the blackbox utility as follows: "$ blackbox.sh –driver=rtlsim –app=psum" –perf=1

Part3 - Evaluation (2pts)

You will evaluate your new application using Vortex RTLSIM driver and executing 16K entries *psum* on 1, 2, 4, 8, 16 cores (each core having 4 warps, 4 threads, and issue width=1) as one graph, then running it on a single core with 4x4, 8x8, 16x16, 32x32 warp x threads with issue width=4 in another graph. Enable the performance counter when running your tests by passing (--perf=1) to the blackbox test. Capture IPC report into a table and plot..

Part4 - Profiling (4 pts)

You will add a new performance counter in the GPU RTL to capture the GPU efficiency when running your kernel. The counter, warp_efficiency, will capture the percentage of warps that are executing instructions as opposed to being idle due to stalls. A performance counter is implemented in Vortex via RISC-C CSRs. The Vortex hardware performance counters are implemented in the following locations: a) vx_dump_perf() in /runtiem/common/utils.cpp processes all the counters and displays their result on the console. b) vx_csr_data() in /hw/rtl/core/VX_csr_data.sv exposes the counters as CSR registers for the kernel S/W to access them. c) VX_pipeline_perf_if.sv implements the interface holding the pipeline counters. That is also where you will need to add the information you need to capture at runtime.
With the counter implemented, plot the warp efficiency and IPC with the single core GPU configuration with 4x4, 8x8, 16x16, 32x32 warp x threads with issue width=4

What to submit?
1- zip file containing kernel.cpp, main.cc, Makefile, report.pdf.
The report should be 1-2 pages only showing the generated plots.