CS259 - LAB1

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

Part1 - Documentation

We recommend starting by reading 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 and MICRO tutorials are valuable resources as well.

- 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 OpenGPU on your development machine using the instructions provided in the `README.md` on GitHub. Follow the full instructions, including running the demo example. Note: An Ubuntu 20.04 system is required. You can use WSL 2.0 on Windows or a VM on macOS to set up the environment. You may also use the Dockerfile in `vortex/miscs/docker` if preferred.

Part3 - GPU Programming (2 pts)

You will implement a Parallel Bitonic Sort on the OpenGPU using C++.

A typical OpenGPU applications consist of two parts:

1. **Host CPU code (main.cpp)** – This implements the host-side code that runs on your CPU. It handles GPU resource allocation, memory transfers, and launching the kernel. The code uses the GPU runtime API (Vortex Runtime API) to interact with the GPU device.
2. **GPU kernel code (kernel.cpp)** – This contains the kernel that runs on the GPU hardware. OpenGPU kernels use the GPU kernel library (Vortex Kernel Library) for accessing the kernel API.

Steps:

- Clone the existing parallel sort benchmark from Vortex Tests into a new folder named `bsort` to create your GPU application.
- Update the `Makefile` by renaming the `PROJECT` value to match the folder name.
- Modify `gen_ref_data()` in `main.cpp` to generate the test gold data for verification.
- Implement the parallel reduce sum in `kernel_body()` in `kernel.cpp`.
- Execute your new application using the *blackbox* utility:
  `$ blackbox.sh --driver=rtlsim --app=<test> --perf=1`

Part4 - Evaluation (3 pts)

Evaluate your application using the RTLSIM driver by executing a 16K input data with the following configurations:

1. **Cores Comparison:** Run the program on 1, 2, and 4 cores (each core with 4 warps, 4 threads), then plot the results in one graph.
2. **Threads/Warps Comparison:** Run the program on a single core with varying configurations (4x4, 8x8, 16x16 warps x threads, with the issue width=4), then plot these results in another graph.

Enable the performance counter when running your tests by passing `--perf=1` to the *blackbox* command line. Capture the IPC report in a table and plot the results.

Configuring Hardware Parameters:
You can adjust the hardware configuration by using the following flags:

1. **NUM_CORES:** Number of cores to use in the simulation.
2. **NUM_WARPS:** Number of warps per core.
3. **NUM_THREADS:** Number of threads per warp.
4. **ISSUE_WIDTH:** Number of issued Instructions per cycle.

To pass these flags to the blackbox test, use the following command:

```
$ CONFIGS="-DNUM_CORES=4 -DNUM_WARPS=2 -DNUM_THREADS=2
-DISSUE_WIDTH=2" blackbox.sh --driver=rtlsim --app=<test> --perf=1
```

For more details on configuring the GPU, check out the CI regression suite in `/ci/regression.sh`.

Part5 - Profiling (5 pts)

You will add a new performance counter to the GPU RTL to capture GPU efficiency when running your kernel. Specifically, you will measure the **warp execution efficiency**, which represents the ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor.

Performance counters in OpenGPU are implemented via RISC-V CSRs. Here are the key locations where counters are defined:

- `vx_dump_perf()` in `/runtime/common/utils.cpp` – Processes and displays counters on the console.
- `vx_csr_data()` in `/hw/rtl/core/VX_csr_data.sv` – Exposes counters as CSR registers for kernel software to access.
- `VX_pipeline_perf_if.sv` – Implements the interface for pipeline counters, where you will add your warp efficiency logic.

After implementing the counter, plot both warp efficiency and IPC using the single-core GPU configuration with 4x4, 8x8, 16x16 warps x threads with the issue width=4.

Submission Guidelines:

Please submit the following two files with **“exact”** names:

1. **source.diff:** A diff file showing the changes you made to the source code. Ensure the diff reflects all modifications, including the kernel and host code changes.
2. **report.pdf:** A written report detailing your evaluation results. The report should include a summary of your approach, performance analysis, graphs, and insights from your profiling experiments.