Lab 5

CSEN 145L: Parallel Programming

CUDA Programming in C++

In this lab, we will explore the fundamentals of CUDA programming for GPU-based parallel computing. We will focus on understanding how to structure CUDA kernels, manage memory between host (CPU) and device (GPU), and optimize performance through techniques like thread synchronization, shared memory usage, and minimizing data transfers. Through solving a series of practical problems, you will gain hands-on experience in writing, debugging, and optimizing GPU programs for real-world applications.

You are provided with a CUDA file, CUDA_problems.cu and you are required to complete GPU related code for each problem. The code already contains helper functions from #include <cuda_runtime.h> that will help manage CUDA related errors, copying data from host memory to GPU memory and vice versa.

The code also has a host implementation (CPU) and your task is to solve each problem by utilizing the power of the GPU.

To get Started, Clone or download the lab05 folder from github.

1. Start implementing the problems inside **CUDA_problems.cu,** serial function's logic that runs on CPU (host) are already implemented, Your task is to implement the kernel functions of each problem to run that on GPU.

2. To compile/execute and test your code, you will need to request resources from WAVE HPC by either requesting interactive (srun method) or non-interactive (sbatch) resources.

A. Srun method:

srun --partition=gpu --nodes 1 --ntasks 1 --cpus-per-task 8 --mem 2G --time 01:00:00 --gres gpu:0 --pty /bin/bash

- You can modify other flags as per your requirements but make sure to include --gres gpu:0 to request GPU.

The partition flag is set to gpu since that partition is required to access NVIDIA V100 GPUs. However, these GPUs are often busy. Alternatively you can use the hub partition. Note that the hub partition does not require the –gres flag since they only have 1 GPU and hub node allocations are exclusive.

srun --partition=hub --nodes 1 --ntasks 1 --cpus-per-task 8 --mem 2G --time 01:00:00 --pty /bin/bash

After you are allotted with resources your current node will change from login1/2/3 to gpuXX or hubXX.

3. Now to compile and execute your code, you will need to compile it using **nvcc**

Note: The code implementation is in C++ but since we need to use CUDA to utilize GPU, it is required to save your .cpp file with .cu extension for the compiler to understand. Provided file already has a .cu extension. And Similarly to compile it, we use nvcc instead of g++/gcc.

First, Load CUDA module using module load

```
module load CUDA
```

Then, Compile CUDA_problems.cu

```
nvcc CUDA_problems.cu -o cuda_problems
```

To run your code,

```
./cuda_problems
```

B. Sbatch Method:

This method follows all the steps explained above since they are already present in the given **cuda_batch.sh** bash file.

```
#!/bin/bash
#SBATCH --partition=hub        # Specify the partition, gpu or hub
#SBATCH --nodes=1              # Number of nodes
#SBATCH --ntasks=1             # Number of tasks (1 task)
#SBATCH --cpus-per-task=8      # Number of CPU cores per task
#SBATCH --mem=2G               # Memory allocation
#SBATCH --time=01:00:00        # Max runtime (1 hour)
```

```
#SBATCH --gres=gpu:0          # Request 1 GPU, remove if hub partition
#SBATCH --output=cuda_output.log # Log standard output to file
#SBATCH --error=cuda_error.log   # Log error output to file

# Email notifications
#SBATCH --mail-type=BEGIN,END,FAIL  # Send email on job start, end, or failure
#SBATCH --mail-user=your_email@example.com  # Replace with your email address

# Load the CUDA module
module load CUDA

# Compile the CUDA code
nvcc CUDA_problems.cu -o cuda_problems

# Run the compiled code
./cuda_problems
```

Note: For Step 4, you will need to switch to srun interactive environment for the ease of profiling your code if you use the sbatch method. If you are already implementing in srun, you can follow step 4 directly.

4. Finally, to profile your cuda program use the following:

   nvprof ./cuda_problems

   - To capture the contents of this command in a logfile use:

     nvprof ./cuda_problems > cuda_problems_prof.log 2>&1

   - A new logfile cuda_problems_prof.log will be created and contain profiling results.

   - To view gpu-trace results while profiling, use the following command:

     nvprof --print-gpu-trace ./CUDA_problems

Final Deliverables: Submit the completed **CUDA_problems.cu**, your **cuda_output.log** and **cuda_problems_prof.log** files.