Lab 6: Matrix Multiplication on GPUs

Introduction

The purpose of this assignment is to become familiar with different ways to parallelize code for execution on NVIDIA and AMD GPUs. We will also compare those executions against serial and shared memory parallel implementations on Intel and AMD CPUs as well.

Problem

You are given example serial and OpenMP GPU-based code for the problem of matrix multiplication. You will need to create several parallel implementations of those codes, including:

- Shared memory parallel CPU-based solution via OpenMP (In addition to the lecture, you can find tutorials and OpenMP documentation here and here.)
- NVIDIA CUDA kernel GPU-based solution
- AMD HIP kernel GPU-based solution (In addition to the lecture we went through in class, you can find additional tutorials and information about HIP here and here. You can also learn about the wider ROCm software platform AMD is providing for interacting with their GPUs here.)
- OpenACC GPU-based solution (In addition to the lecture, you can find tutorials and OpenACC documentation here.) *Note: You will not be able to compile or test this on the HPC system currently as the compiler module is not currently available. Still, write your implementation based on class notes.*

You will then need to compile and test your codes on the WAVE system on both Intel and AMD CPUs and NVIDIA and AMD GPUs. After writing your codes, follow the steps below to complete all experiments. I would suggest you implement the verification step in all your programs and test it on smaller matrices (e.g., 1000x1000) to ensure your code is executing correctly before you run the experiments below.

Experiments

For all experiments, user /usr/bin/time to time the execution of the program and store the results of the experiment in a log file with the file name matmult_<type>_<node_type>.log. For example, for the matmult_cpu program, when you run it on the compute node, you can execute,

```
/usr/bin/time -v ./matmult_cpu 3000 4000 5000 > matmult_cpu_compute.log 2>&1
```

which will store the execution results and timing information in the matmult_cpu_compute.log file. In all experiments, you will use a matrix A of size 3000x4000 with a matrix B of size 4000x5000. Change directory to your code directory before executing the example commands below.

1. Intel CPU

Execute serial and OpenMP codes on a compute node in the cpu partition. Example interactive partition and execution commands:

```
srun --partition=cpu --cpus-per-task=96 --mem=24G --nodes=1 --pty /bin/bash
make clean
make matmult_cpu
make matmult_omp_cpu
export OMP_PROC_BIND=true
export OMP_PLACES=cores
export OMP_NUM_THREADS=48
/usr/bin/time -v ./matmult_omp_cpu 3000 4000 5000 > matmult_omp_cpu_cpu.log 2>&1
/usr/bin/time -v ./matmult_cpu 3000 4000 5000 > matmult_cpu_cpu.log 2>&1
exit
```

2. AMD CPU

Execute serial and OpenMP codes on a compute node in the amd partition, on a node with an AMD CPU. Example interactive partition and execution commands:

```
srun --partition=amd --cpus-per-task=96 --mem=24G --nodes=1 --pty /bin/bash
make clean
make matmult_cpu
make matmult_omp_cpu
export OMP_PROC_BIND=true
export OMP_PLACES=cores
export OMP_NUM_THREADS=48
/usr/bin/time -v ./matmult_omp_cpu 3000 4000 5000 > matmult_omp_cpu_amd.log 2>&1
/usr/bin/time -v ./matmult_cpu 3000 4000 5000 > matmult_cpu_amd.log 2>&1
exit
```

3. NVIDIA GPU

Execute serial, OpenACC and CUDA codes on a compute node in the hub partition, on a node with an NVIDIA GTX 1080 GPU. Example interactive partition and execution commands:

```
srun --partition=hub --cpus-per-task=8 --mem=24G --nodes=1 --pty /bin/bash
module load CUDA
```

```
make clean
make matmult_cpu
make matmult_cuda
/usr/bin/time -v ./matmult_cuda 3000 4000 5000 > matmult_cuda_hub.log 2>&1
/usr/bin/time -v ./matmult_cpu 3000 4000 5000 > matmult_cpu_hub.log 2>&1
exit
```

Optionally, if one is available, repeat this experiment on a node in the gpu partition, which has NVIDIA V100 GPUs. Example interactive partition and execution commands:

```
srun --partition=gpu --cpus-per-task=8 --mem=24G --nodes=1 --gres=gpu:1 --pty /bin/bash
module load CUDA
make clean
make matmult_cpu
make matmult_cuda
/usr/bin/time -v ./matmult_cuda 3000 4000 5000 > matmult_cuda_gpu.log 2>&1
/usr/bin/time -v ./matmult_cpu 3000 4000 5000 > matmult_cpu_gpu.log 2>&1
exit
```

4. AMD GPU

Execute the HIP version of the program on a compute node in the amd partition, which has the AMD MI100 GPU. Example interactive partition and execution commands:

```
srun --partition=amd --cpus-per-task=8 --mem=24G --nodes=1 --gres=gpu:1 --pty /bin/bash
make clean
make matmult_hip
/usr/bin/time -v ./matmult_hip 3000 4000 5000 > matmult_hip_amd.log 2>&1
exit
```

Deliverable

Create a short 1-2 page report including a description of the problem and a chart showing speedup (compared to serial on the hub node) for all experiments. You can find the execution time for each program in the log file in the line starting with "Elapsed (wall clock) time". Make sure to label your experiments correctly in the chart, including the code type and the system they were executed on, so they can be properly differentiated. Draw some conclusions from the comparison. I've included a jupyter notebook I used in my own experiments last year for reference.

Create an archive file called lab06.tar.gz or lab06.zip containing your source code for all solutions, log files from all experiments, and your report. Upload your archive to Canvas.