Contents

HIP Lecture Series

Introduction to Occupancy Exercises

For the HIP Lecture Series, the examples can be retrieved from this repository.

```
git clone https://github.com/olcf/hip-training-series
```

Below are instructions for doing the exercises on OLCF Frontier. We are using portable makefiles as in the first two lectures. So the code will also compile on NERSC Perlmutter. But the AMD tools will not run on NERSC Perlmutter. On Perlmutter, the exercise will be to evaluate the performance impact of each of the variations of the kernel. For an even deeper dive, the Nvidia performance tools can be run on Perlmutter to see what they reveal.

For NERSC Perlmutter, see the instructions for Nvidia below. NERSC has installed the AMD ROCm software stack to enable developers on Perlmutter to begin developing more portable applications with a single repository. This can greatly reduce code porting and maintenance efforts.

This markdown document is located at `'Lecture3/03 Exercises for Occupancy.md'` contains the instructions to run the examples. You can view it in github for better readability or download the pdf file `'Lecture3/03 Exercises for Occupancy.pdf'` which has been generated from the markdown document.

For the first interactive example, get an slurm interactive session on Frontier (see further below for NERSC Perlmutter):

```
salloc -N 1 -p batch --reservation=hip_training_2023_09_18 --gpus=1 -t 10:00 -A <project>
```

Outside the reservation window or if you're not on the reservation list, you can do: `salloc -N 1 -p batch --gpus=1 -t 10:00 -A <project>`

Use your project id in the project field. If you do not remember it, run the command without the -A option and it should report your valid projects.

```
module load PrgEnv-amd
module load amd
module load cmake
export CXX=${ROCM_PATH}/llvm/bin/clang++
```

Occupancy example

```
cd hip-training-series/Lecture3/Occupancy
```

Examine files here – README, Makefile, CMakeLists.txt and occupancy_mxv.cpp. Notice that the Makefile requires ROCM_PATH to be set. Check with module show rocm or echo $ROCM_PATH. The Makefile builds and runs the code. We'll do the steps separately. Check the HIPFLAGS in the Makefile. There is also a CMakeLists.txt file to use for a cmake build.

For the portable Makefile system

```
make occupancy_mxv
srun ./occupancy_mxv
```

This example also runs with the cmake system

```
mkdir build && cd build
cmake ..
make
srun ./occupancy_mxv
```

Now clean up from these exercises before the next part.

```
cd ..
make clean
rm -rf build
module unload PrgEnv-amd
module unload amd
module unload cmake
```

Run all the different versions

```
module load PrgEnv-amd
module load amd
module load cmake
export CXX=${ROCM_PATH}/llvm/bin/clang++

./run_maketests.sh
grep GFLOPS */*.out
```

You should get something like the following:

```
Occupancy_naive/occupancy.out:0.0217501 Time - GFLOPS 987.345
Occupancy_shmem/occupancy.out:0.0192239 Time - GFLOPS 1117.09
Occupancy_shmem_A/occupancy.out:0.0224112 Time - GFLOPS 958.218
Occupancy_shmem_batched/occupancy.out:0.019235 Time - GFLOPS 1116.45
Occupancy_shmem_batched_unroll/occupancy.out:0.0136489 Time - GFLOPS 1573.38
```

Versions corresponding to the talk

Use the Occupancy_shmem_batched either in the main file or the subdirectory. Note that the V2 and V3 are equivalent to the Occupancy_shmem version where there is no batch parameter (i.e. batch = 1)

Modify the defines at the top of the file

#define batch 1 #define Nm 4 #define nThreads 256 #define Nunroll 4

Here is an example of the results on Frontier

V0 batch 32 nThreads 128 – GFLOPS 60.68 VGPR 74 LDS 65536 Occupancy 2 V1 batch 16 nThreads 256 – GFLOPS 105.08 VGPR 42 LDS 65536 Occupancy 4 V2 batch 1 nThreads 128 – GFLOPS 1168.98 VGPR 38 LDS 2048 Occupancy 8 V3 batch 1 nThreads 256 – GFLOPS 1171.88 VGPR 38 LDS 4096 Occupancy 8

Try the same exercise on the Occupancy_mxv_shared_batch_unroll version

Add the profiling tools

Now let's run the example with the profiling tools. First let's use the rocprof tool.

```
module load PrgEnv-amd
module load amd
module load cmake
cd $HOME/HPCTrainingExamples/HIP/hip-stream
make
srun ./occupancy_mxv
rocprof --stats ./occupancy_mxv
```

The results will be in …

For a more detailed profile, we use the omniperf tool.

```
module load PrgEnv-amd
module load amd
module load cmake
cd $HOME/HPCTrainingExamples/HIP/hip-stream
make
srun ./occupancy_mxv
omniperf profile -p $HOME/occupancy/workloads --no-roof -n occupancy -- ./occupancy_mxv
omniperf analyze -k 1 -t us -p ./occupancy/mi200 >& ./occupancy_0.txt
```

The results will be in …

Batch submission

We can use a SLURM submission script, let's call it `hip_batch.sh`. There is a sample script for some systems in the example directory.

```
#!/bin/bash
#SBATCH -p batch
#SBATCH -N 1
#SBATCH --gpus=1
#SBATCH -t 10:00
#SBATCH --reservation=hip_training_2023_09_18
#SBATCH -A <your project id>

module load PrgEnv-amd
module load amd
module load cmake
cd $HOME/hip-training-series/Lecture3/Occupancy

make occupancy_mxv
srun ./occupancy_mxv
```

Submit the script `sbatch hip_batch.sh`

Check for output in `slurm-<job-id>.out` or error in `slurm-<job-id>.err`

To use the cmake option in the batch file, change the build commands in the batch file to

```
mkdir build && cd build
cmake ..
make
srun ./occupancy_mxv
```

Compile and run with Cray compiler

```
module load PrgEnv-cray
module load amd-mixed
```

```
module load cmake
CXX=CC CRAY_CPU_TARGET=x86-64 make vectoradd
srun ./vectoradd
```

And with the cmake build system.

```
module load PrgEnv-cray
module load amd-mixed
module load cmake
mkdir build && cd build
CXX=CC CRAY_CPU_TARGET=x86-64 cmake ..
make
srun ./vectoradd
```

Before moving onto another example, first clean up from the previous work.

```
cd ..
make clean
rm -rf build
module unload PrgEnv-cray
module unload amd-mixed
module unload cmake
```

NERSC Perlmutter instructions For the hands-on exercise on the NERSC Perlmutter system, there is a reservation under the account ntrain8. Get an allocation with

```
salloc -N 1 -C gpu -A ntrain8 --reservation=hip_sept18 -q shared -c 32 -G 1 -t 1:00:00
```

Outside of reservation window, you can do:

```
salloc -N 1 -C gpu -A <your_project> -q shared -c 32 -G 1 -t 1:00:00
```

Load the environment for Nvidia. Note that there is an order required. To load the HIP module, the GNU environment must already be loaded. Then load the HIP environment. Once the HIP module is loaded, you can load the Nvidia programming environment.

```
module load PrgEnv-gnu/8.3.3
module load hip/5.4.3
module load PrgEnv-nvidia/8.3.3
module load cmake
```

Build the example

```
cd ~/hip-training-series/Lecture3/Occupancy
HIPCC=nvcc make occupancy_mxv
srun ./occupancy_mxv
```

Run all the tests

```
./run_maketests.sh
grep GFLOPS */*.out
```

You should get the something like the following results

```
Occupancy_naive/occupancy.out:0.0132262 Time - GFLOPS 1623.66
Occupancy_shmem_A/occupancy.out:0.0132137 Time - GFLOPS 1625.2
Occupancy_shmem_batched/occupancy.out:0.0134865 Time - GFLOPS 1592.32
Occupancy_shmem_batched_unroll/occupancy.out:0.0122282 Time - GFLOPS 1756.18
Occupancy_shmem/occupancy.out:0.013485 Time - GFLOPS 1592.5
```

Versions corresponding to the talk

Use the Occupancy_shmem_batched either in the main file or the subdirectory. Note that the V2 and V3 are equivalent to the Occupancy_shmem version where there is no batch parameter (i.e. batch = 1)

Modify the defines at the top of the file

#define batch 1 #define Nm 4 #define nThreads 256 #define Nunroll 4

Here is an example of the results on Perlmutter

V0 batch 32 nThreads 128 – too much shared data (0x10000 bytes, 0xc000 max) V0* batch 16 nthreads 128 – 0.102533 Time - GFLOPS 209.443 V1 batch 16 nThreads 256 – too much shared data (0x10000 bytes, 0xc000 max) V1* batch 8 nThreads 256 – 0.0873428 Time - GFLOPS 245.868 V2 batch 1 nThreads 128 – 0.0134582 Time - GFLOPS 1595.67 V3 batch 1 nThreads 256 – 0.0138559 Time - GFLOPS 1549.87

Cleanup

```
make clean
```

For the cmake build

```
mkdir build && cd build
cmake -DCMAKE_GPU_RUNTIME=CUDA ..
make
srun ./occupancy_mxv
```

Cleanup

```
cd ..
rm -rf build
```