Homework 4 - GPU Transpose

Philip Nelson

5th November 2021

I implemented four functions/kernels for this assignment: CPU transpose, CUDA transpose, CUDA tiled transpose, and CUDA copy. Their implementations can be found in `transpose/transpose.cu`. The first transpose kernel uses the naive approach which reads an input from global memory, then writes to the transposed output location in global memory. The memory access pattern leaves a lot of performance on the table because it does not have coalesced writes to the output. This happens because the input is read row-wise (coalesced access), but the output is written column-wise (non-coalesced access). The tiled transpose improves this access pattern by reading the input row-wise and storing it in a transposed shared tile. Since the shared memory is much faster, the colum-wise store does not hurt performance as much. Then this tile can be read and written into the output in global memory but with a row-wise access pattern again, taking advantage of coalesced memory access.

The CPU transpose was used for verification and the CUDA copy was used to compare effective bandwidth.

This project includes a CMakeLists.txt. The built executable accepts optional commandline arguments for the raw input image and its height and width in pixels. Default values are `gc_1024x1024.raw 1024 2014` respectively. The following commands will build and run the transpose project.

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make transpose
./transpose/transpose

# Transposing gc_1024x1024.raw [ 1024 x 1024 ]
#
# Copy
# Execution Time     : 1.01417 ms
# Effective Bandwidth: 6.20357 GB/s
#
# SUCCESS
#
# Transpose Naive
# Execution Time     : 2.25422 ms
# Effective Bandwidth: 2.79097 GB/s
#
# SUCCESS
#
# Transpose Tile
# Execution Time     : 1.49972 ms
# Effective Bandwidth: 4.19507 GB/s
#
# SUCCESS
```

Timing results

To compare the performance of the two methods I implemented a simple copy operation to get a baseline for effective bandwidth. Then I ran the three kernels averaging the timing results over 100 trials and calculated the effective

bandwidth at 1x1, 2x2, 4x4, 8x8, 16x16, and 32x32 sized blocks. The results of these tests can be seen below in Figure 1. The results clearly show that the tiled transpose outperforms the naive implementation.

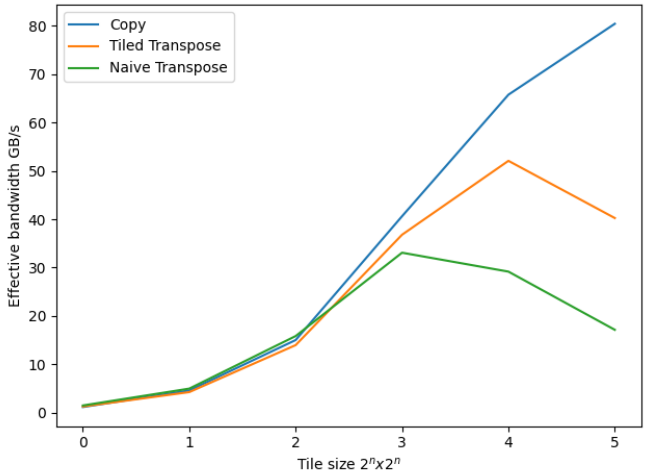

Figure 1: Effective Bandwidth Comparison on NVIDIA Tesla K80