Exercise 3

360.252 - Computational Science on Many-Core Architectures
WS 2021

November 3, 2021

The following tasks are due by 23:59pm on Tuesday, November 9, 2021. Please document your answers (please add code listings in the appendix) in a PDF document and email the PDF (including your student ID to get due credit) to karl.rupp@tuwien.ac.at.

You are free to discuss ideas with your peers. Keep in mind that you learn most if you come up with your own solutions. In any case, each student needs to write and hand in their own report. Please refrain from plagiarism!

> "To steal ideas from one person is plagiarism;
> to steal from many is research." — Steven Wright

There is a dedicated environment set up for this exercise:

https://gtx1080.360252.org/2021/ex3/.

To have a common reference, please run all benchmarks for the report on this machine.

Strided and Offset Memory Access (3 Points)

Reconsider the vector summation from Exercise 2 for vectors of size $N = 10^8$. Modify your GPU kernels so that

1. only every k-th entry is summed, i.e. the parallel equivalent of (1 point)

```
for (int i=0; i < N/k; ++i) z[i*k] = x[i*k] + y[i*k];
```

2. the first k elements are skipped, i.e. the parallel equivalent of (1 point)

```
for (int i=0; i < N-k; ++i) z[i+k] = x[i+k] + y[i+k];
```

and investigate values of k between 0 and 63.

Compute and plot the effective memory bandwidth (in GB/sec) for both cases (1 point). Only consider the entries that are actually touched (N/k and N-k, respectively) for the bandwidth calculation. What do you observe? Which general recommendation can you derive for future performance optimizations in more complicated scenarios?

Dense Matrix Transpose (5 Points)

Your friend is working with a dense matrix $A \in \mathbb{R}^{N \times N}$ and needs to transpose the matrix in-place.

In order to do so, your friend writes a specialized CUDA kernel. Unfortunately, your friend's kernel doesn't produce correct results and you need to help out.

Please help your friend as follows:

1. Run your friend's kernel through `cuda-memcheck` and identify the problem(s). (1 Point)
2. Fix your friend's kernel to yield correct results for all problem sizes, but do not optimize for performance. Determine the effective bandwidth (GB/sec) your code achieves. (1 Point)
3. As a first step to performance optimization, simplify the task by dropping the in-place requirement. That is, read from a matrix A and write the result (A^{T}) to a second matrix B. Optimize your code by avoiding strided memory access to global memory as much as possible. (1 Point)
4. Now reconsider the in-place transposition. Use shared memory to load blocks of size 16×16, transpose them in shared memory, and write the result. Determine the effective bandwidth (GB/sec) you achieve. (1 Point)
5. Compare the performance you obtain for the non-optimized and the optimized kernel(s) for $N = 512, 1024, 2048, 4096$. (1 Point)

For simplicity, assume that N is a multiple of 16. Please make sure that the kernel computes correct results![1]

Bonus point: Early Submission

Hand in your report by 23:59pm on Monday, November 8, 2021.

[1] It is better to have a slow kernel computing correct results rather than having a fast kernel computing wrong results ;-)