Exercise 4

360.252 - Computational Science on Many-Core Architectures
WS 2023

November 6, 2023

The following tasks are due by 23:59pm on Tuesday, November 21, 2023. Please document your answers (please add code listings in the appendix) in a PDF document and submit the PDF together with the code in TUWEL.

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!

> "After some time passed in studying - and even imitating - the works of others, I would recommend the student to endeavour to be original, and to remember that originality should not be undiscovered plagiarism." — Henry Peach Robinson

There is a dedicated environment set up for this exercise:

https://rtx3060.360252.org/2023/ex4/

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

Dot Product with Warp Shuffles (4 Points)

Given a vector x of size N, you need to compute

- the sum of all entries,
- the sum of the absolute value of all entries (1-norm),
- the sum of the squares of all entries (squared 2-norm),
- the number of zero entries.

Your friend recommends to call the respective functions in CUBLAS, but you suspect that you can implement a faster version that computes all these values in a single kernel.

Implement and compare different versions of such a kernel for a fixed block size of 256:

(a) Using shared memory like for the dot product. (1 Point)

(b) Using warp shuffles only (no shared memory) with a fixed number of 256 thread blocks. (1 Point)

(c) Using warp shuffles only with one thread per entry (i.e. the number of thread blocks depends on N) . (1 Point)

(d) Compare the performance of these variants for different N. Also compare with the execution time for the dot-product $\langle x, x \rangle$. (1 Points)

The RTX 3060 GPU supports atomics[1] for writing to global memory so that only a single kernel call is required. Also, transfer (and check) all result values to the host with a single call to `cudaMemcpy`.

Multiple Dot Products (4 Points total)

Given a vector x (in double precision) of size N and K vectors $y_k, k = 1, \ldots, K$ of size N and K being a multiple of 8, your friend coded up a method to compute the dot products

$$\langle x, y_1 \rangle, \ldots, \langle x, y_K \rangle$$

by calling the vendor-tuned CUBLAS library. It turns out that this operation is the bottleneck for an iterative solver; your friend is proud that this approach is faster than a previous hand-rolled implementation and claims that "there's nothing one can do to make it any faster".

Based on the code provided at the URL above, demonstrate that you already have the skills to beat a vendor-tuned library approach:

(a) Write a specialized kernel to compute 8 dot products concurrently. (1 Point)

(b) Add a loop around that kernel to call it $K/8$ times to correctly compute the K dot products $\langle x, y_1 \rangle, \ldots, \langle x, y_K \rangle$. (1 Point)

(c) Plot the execution times for vectors of size N between 10^3 and 10^6 for values of $K = 8, 16, 24, 32$. (1 Point)

(d) Outline (but do not code) an approach that is similarly fast for general values of K. (1 Point)

Make sure that your code works for general values of N. Feel free to use `atomicAdd` on the RTX 3060.

[1] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions