Exercise 2

360.252 - Computational Science on Many-Core Architectures
WS 2020

October 21, 2020

The following tasks are due by 23:59pm on Tuesday, November 3, 2020. 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!

"I've been imitated so well I've heard people copy my mistakes." — Jimi Hendrix

There is a dedicated environment set up for this exercise:

`https://gtx1080.360252.org/2020/ex2/`.

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

Basic CUDA (5 Points total)

Write a program that initializes two CUDA vectors (double precision floating point entries) of length N: One consisting of the numbers

$$0, 1, 2, 3, \ldots, N-1$$

and the other consisting of the numbers

$$N-1, N-2, \ldots, 0\ .$$

Investigate the following:

(a) Measure the time to allocate (`cudaMalloc`) and free (`cudaFree`) a CUDA array for different sizes N. (1 Point)

(b) Compare the following three options to initialize the vectors:

- Initialize directly within a dedicated CUDA kernel
- Copy the data via `cudaMemcpy()` from a host array (e.g. from a `malloc`'ed array or from `std::vector<double>`).

- Copy each individual entry by calling `cudaMemcpy` for each entry.

  Provide the time to complete the initialization for each option and compare the effective bandwidth (in megabytes per second) obtained in each case. (1 Point)

(c) Write a CUDA kernel that sums the two vectors. Make sure that the kernel works for different values of N. (1 Point)

(d) Measure and plot the execution time of the kernel for different values of N (e.g. 100, 300, 1000, 3000, 10000, 30000, 100000). What do you observe for small values of N and large values of N, respectively? (1 Point)

(e) Try different grid sizes and block sizes as kernel launch parameters. For simplicity, consider the values 16, 32, 64, 128, 256, 512, and 1024. Which values lead to a significant reduction in performance for large ($N = 10^7$)) vectors)? (1 point)

Dot Product (5 Points total)

Write a program that computes the dot-product of two vectors of variable size N in different ways:

(a) Use two GPU stages: In the first kernel, each thread block computes the partial dot-product and writes the partial result to a temporary array. Then, a second kernel sums the obtained partial results. (2 Points)

(b) Use a GPU and a CPU stage: In the first kernel, each thread block computes the partial dot-product and writes the partial result to a temporary array. Then, copy the temporary array to the CPU and sum it there. (1 Point)

(c) Use a single stage: Reuse the first kernel above, but this time use `atomicAdd`[1] inside the kernel to add the partial result of each thread block to the CUDA result variable. (1 Point)

Compare the performance of the three versions. When is each of the three versions most appropriate? (1 Point)

Halloween Bonus: Trick AND Treat (no Points, but Chocolate)

Scare your lecturer by coming up with creative ways of making GPUs *appear* super-fast (100x speed-up and beyond) for vector additions. To demonstrate, write a program that benchmarks a (slow) *CPU version* of the vector addition from above and compare it to a *GPU version* you coded up above. You can play all tricks to make the CPU version appear slow and the GPU version appear fast. The result array has to be correct in both cases, but any other benchmarking crimes are permitted.

The output of the program should be similar to the following:

```
Summing vectors of size N=10000
CPU time: 1.202 sec
GPU time: 0.001 sec
Speedup: 1202x
```

Fun and creative submissions will be awared a Zotter chocolate bar.

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