Exercise 4

360.252 - Computational Science on Many-Core Architectures
WS 2021

November 10, 2021

The following tasks are due by 23:59pm on Tuesday, November 16, 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!

"Plagiarism is the fear of a blank page." — Mokokoma Mokhonoana

There is a dedicated environment set up for this exercise:

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

1. Using shared memory like for the dot product. (1 Point)
2. Using warp shuffles only (no shared memory). (1 Point)
3. Compare the performance of these variants for different N. Also compare with the execution time for the dot-product $\langle x, x \rangle$. (2 Points)

In all cases, use atomics[1] for writing to global memory so that only a single kernel call is required. Also, transfer all result values with a single call to `cudaMemcpy`.

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

Conjugate Gradients (5 Points total)

The conjugate gradient algorithm for solving a system of equations $Ax = b$ with symmetric and positive definite system matrix A can be formulated as follows:

Algorithm 1: Classical CG

```
1  Choose x_0;
2  p_0 = r_0 = b − A x_0;
3  for i = 0 to convergence do
4      Compute and store A p_i;
5      Compute ⟨p_i, A p_i⟩;
6      α_i = ⟨r_i, r_i⟩/⟨p_i, A p_i⟩;
7      x_{i+1} = x_i + α_i p_i;
8      r_{i+1} = r_i − α_i A p_i;
9      Compute ⟨r_{i+1}, r_{i+1}⟩;
10     Stop if ⟨r_{i+1}, r_{i+1}⟩ is small enough;
11     β_i = ⟨r_{i+1}, r_{i+1}⟩/⟨r_i, r_i⟩;
12     p_{i+1} = r_{i+1} + β_i p_i;
13 end
```

Implement this algorithm in CUDA using double precision arithmetic and data types. Implement

1. a kernel for the matrix-vector product in line 4, (1 Point)
2. kernels for the vector operations in lines 5 and 9 as well as 7, 8, and 12. (1 Point)

When your implementation is completed, analyze the following:

1. Plot the time needed for convergence (reduction of the residual norm r by a factor of 10^6 compared to r_0) and the number of iterations for different system sizes. (1 Point)
2. Break down the total solution time for small system sizes (about 1000 unknows) and large system sizes (about 10^7 unknowns) on a per-kernel basis. Which parts of the implementation are worthwhile to optimize in later work? (1 Point)

Finally, develop a simple performance model based on latency and memory bandwidth for the required data transfers (assume five nonzero entries per row in the A). Compare it with the actual timings obtained above for different system sizes. Which approximate values for the latency and memory bandwidth do you get? (1 Point)

Hints

- Check `https://gtx1080.360252.org/2021/ex4/` for a code skeleton to start from.
- A common choice for x_0 is to use a vector of zeros and thus avoid the computation of Ax_0 in line 2.
- Consider starting with a sequential CPU-version and then incrementally move computations to the GPU.

Bonus point: Early Submission

Hand in your report by 23:59pm on Monday, November 15, 2021.