TDT4200 Problem Set 6

Cooperative groups in CUDA

Maren Wessel-Berg and Claudi Lleyda Moltó

Deadline: November 7, 2023 by 22:00
Evaluation: Graded (10%)

- **This assignment has two parts.** In the first part, you will solve programming tasks. These tasks are described in the Tasks section. In the second part, you will answer questions about your implementation and/or from the curriculum. These questions are stated in the Questions section.
- **This assignment is graded and will count towards 10% of your final course grade.** You must receive a pass grade on this assignment for you to sit the final exam in the course.
- **This assignment must be done individually and without help from anyone but the TDT4200 staff.** All sources found on the internet or elsewhere must be referenced. We encourage that you post clarification questions on Piazza so all can benefit. However, make sure you do not post full or partial solutions on Piazza.
- **The assignment requirements are explained in the Evaluation section.**
- **How and what to deliver is explained in the Delivery section.** Do not deliver any other files than those specified. Code should not use external dependencies aside from the ones already included in the handout code.

Finite difference approximation of the 2D heat equation using CUDA cooperative groups

In this exercise you will take a sequential implementation of the Red/Black Gauss-Seidel algorithm for solving the 2D heat equation and write a GPU version using CUDA cooperative groups. The sequential implementation is provided. A skeleton for your CUDA implementation is provided in `heat_parallel.cu`. Information on how to install the dependencies and run the program is provided in the `README.md` of the handout code.

Although we provide a Makefile in the handout code, we encourage you to inspect the Makefile to see how to compile and run the program.

Tasks

The following tasks all have corresponding TODO-comments in `heat_parallel.cu`. We recommend that you ensure the program compiles after completing each task even though the program might not run correctly between tasks.

1. Declare device pointers to store host-side data on the GPU.

    Tip Prefix the names with `d_` to separate them from their host-side counter-parts. This will make it easier to manage data between the GPU and CPU.

2. Allocate space for the device buffers using the device pointers and `cudaMalloc()`.

3. Copy the necessary data from the CPU to the GPU using `cudaMemcpy()`.

4. Change the function `time_step()` to be a CUDA kernel. One thread should be responsible for calculating one grid point.

    Hint Use the tools provided in the `cooperative_groups` namespace to sync all the blocks when switching from red to black squares.
    Hint Define a variable that gives the global index of a thread in the grid using `blockIdx`, `blockDim` and `threadIdx`.
    Note You might need to change the macros if you want to use them in the kernel.

5. Change the function `boundary_condition()` to be a device function. Call it from the `time_step`-kernel and choose appropriate threads to set the boundary values.

6. Launch the `time_step`-kernel with as many threads per thread block as possible and a sufficient number of thread blocks.

7. Copy the results from the GPU to the CPU before the `domain_save()` function.

8. Free device memory using `cudaFree()`.

Questions

1. (10%) What are the limitations of using cooperative groups to sync the whole grid at once?

   Hint Try to increase the simulation domain size.

Evaluation

This assignment is graded and will count towards 10% of your final course grade and is graded as follows:

- **CUDA implementation: 80%**

  The CUDA implementation will count towards 80% of the problem set grade, i.e., 8% of your final course grade.

  – Your implementation should run error-free and produce correct output the default configuration. Your solution can be compared for correctness with the correctness script in the handout code.

  – Partial solutions will be given some percentage points if the intent behind the code is clear and well-documented.

- **Documentation and code clarity: 10%** Documentation and clarity of the code will count towards 10% of the problem set grade, i.e., 1% of your final course grade. You should document each function and what is does, use sensible variable names, etc. This only applies to the code that **you write**, i.e., you do not need to document functions that are already implemented or clarify existing code.

- **Questions: 10%** Answer to the question above will count towards 10% of the problem set grade, i.e., 1% of your final course grade. The possible percentage points for the question is indicated in the Questions section. You should answer the question properly, but there is no need to write long essays.

Delivery

Deliver the file `heat_parallel.cu` in BlackBoard. Answers to the questions should also be delivered in BlackBoard in a separate PDF file. Do **NOT** upload a ZIP file.