Take-Home Exam CUDA_HTT

Parallel Programming & Architectures

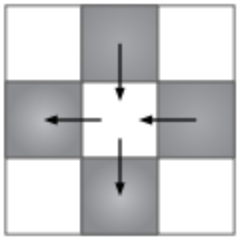
Consideration

- ✓ Your code is automatically graded using a script, and therefore, if your file/folder names are wrong you will receive a grade of **zero**. Please read and follow the instructions carefully. Common mistakes include
  - Different file or folder names
  - Different formatting of input or output
  - Not paying attention to case sensitiveness of C++ and Linux
- ✓ Go to the folder ~/the/cuda_htt/ in your home directory on the server and put your codes in this directory and remove any compiled binaries and test cases.
- ✓ Make sure your code compiles and runs without any error **on the server**. Your grade will be **zero** if any compile or runtime error occurs on the server. **Any!**
- ✓ The provided test cases, examples and sample codes (if any) are only to better describe the question. They are **not** meant for debugging or grading. It is your responsibility to think of and generate larger and more complex test cases (if necessary) in order to make sure your software works correctly for all possible scenarios.
- ✓ Start early and don't leave everything to the last minute. Software debugging needs focus and normally takes time.
- ✓ Just leave your final programs on the server. **Don't** email anything!
- ✓ Your grade is divided into several parts. In all cases, if you miss **correctness** (i.e. your code doesn't satisfy desired functionality), you miss other parts (e.g. speed, coding style, etc.) too. This rule is applied separately for each section of a take-home exam. So for example, in cuda_mm, your code might not be correct for M >= x but still you will get your grade for lower M values.
- ✓ Talking to your friends and classmates about this take-home exam and sharing ideas are *OK*. Searching the Internet, books and other sources for any code is also *OK*. However, copying another code is **not OK** and will be automatically detected using a similarity check software. In such a case, grades of the copied parts are **multiplied by -0.5**. Your work must be 100% done only by yourself, and you **should not** share parts of your code with others or use parts of other's codes. Online resources and solutions from previous years are part of the database which is used by the similarity check software.

***bmm.cu* – Heat Transformation with Texture Memory**

Grade: *20%* correctness, *80%* speed

Consider a two-dimensional room that is divided into cells, with each cell having a temperature randomly ranging between *20* to *30* degrees Celsius. As each cell has a different temperature from its neighbors, heat transformation occurs between these cells and their neighbors in order to establish thermal balance.



Heat dissipating from warm cells into cold cells

And we can model this heat transformation with an equation like this:

$$T_{\text{new}} = T_{\text{old}} + K \times \sum_{\text{neighbors}} \{T_{\text{neighbors}} - T_{\text{old}}\}$$

In this equation, the new temperature of each cell depends on the temperatures of its neighboring cells, incorporating the constant 'K' over a specific duration.

So, in a two-dimensional scenario, we consider neighbors located at the top, bottom, right, and left, and therefore, we utilize this equation.

$$T_{\text{new}} = T_{\text{old}} + K \times \left[\left(T_{\text{top}} - T_{\text{old}}\right) + \left(T_{\text{bottom}} - T_{\text{old}}\right) + \left(T_{\text{right}} - T_{\text{old}}\right) + \left(T_{\text{left}} - T_{\text{old}}\right)\right]$$

Consider matrices of size N × N, where $N = 2^M$, with each array cell representing a cell within the room. Our goal is to calculate the temperature of each cell after undergoing 5 times heat transformations using texture memory. To learn how to utilize texture memory in CUDA, the following resources can be helpful:

1. Jason Sanders et al., 'CUDA By Examples' Book, Chapter 7

2. https://github.com/ricsonc/linear_vs_texture_memory_cuda

To utilize texture memory, as mentioned in the book, you can employ either 1D or 2D texture memory for calculations, and you can choose between them based on their speed, as both are correct options.

Your program must work correctly for any value *10 <= M <= 13*. Note that larger arrays, for example M=14, may not fit into the GPU global memory.

Use the provided gpuerrors.h, *gputimer.h, htt_main.cu, htt.h* and *htt.cu* files to start your work. Only modify *htt.cu*. Check the speed of your calculations as shown in the provided *htt_main.cu* file.

Compile: *nvcc -O2 htt_main.cu htt.cu -o htt*

Execute: *./htt M*

Note that *N* is equal to 2^M.