Description | Code | Questions | Attempts | History

Vector Addition

The coding deadline is 26 days from now (2015-07-10T23:00:00Z) .

Objective

The purpose of this lab is to get you familiar with using the CUDA API by implementing a simple vector addition kernel and its associated setup code.

Prerequisites

Before starting this lab, make sure that:

- You have completed all week 1 lecture videos
- You have completed "Lab Tour with Device Query" MP
- You have looked over the tutorial (/help) document.
- Chapter 3 of the text book would also be helpful

Instruction

Edit the code in the code tab to perform the following:

- Allocate device memory
- Copy host memory to device
- Initialize thread block and kernel grid dimensions
- Invoke CUDA kernel
- Copy results from device to host
- Free device memory
- Write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

Suggestions

- The system's autosave feature is not an excuse to not backup your code and answers to your

questions regularly.

- If you have not done so already, read the tutorial (/help)
- Do not modify the template code provided – only insert code where the `//@@` demarcation is placed
- Develop your solution incrementally and test each version thoroughly before moving on to the next version
- Do not wait until the last minute to attempt the lab.
- If you get stuck with boundary conditions, grab a pen and paper. It is much easier to figure out the boundary conditions there.
- Implement the serial CPU version first, this will give you an understanding of the loops
- Get the first dataset working first. The datasets are ordered so the first one is the easiest to handle
- Make sure that your algorithm handles non-regular dimensional inputs (not square or multiples of 2). The slides may present the algorithm with nice inputs, since it minimizes the conditions. The datasets reflect different sizes of input that you are expected to handle
- Make sure that you test your program using all the datasets provided (the datasets can be selected using the dropdown next to the submission button)
- Check for errors: for example, when developing CUDA code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do {                                                  \
        cudaError_t err = stmt;                                             \
        if (err != cudaSuccess) {                                           \
            wbLog(ERROR, "Failed to run stmt ", #stmt);                     \
            wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));  \
            return -1;                                                      \
        }                                                                   \
    } while(0)
```

An example usage is `wbCheck(cudaMalloc(...))` . A similar macro can be developed while programming OpenCL code.

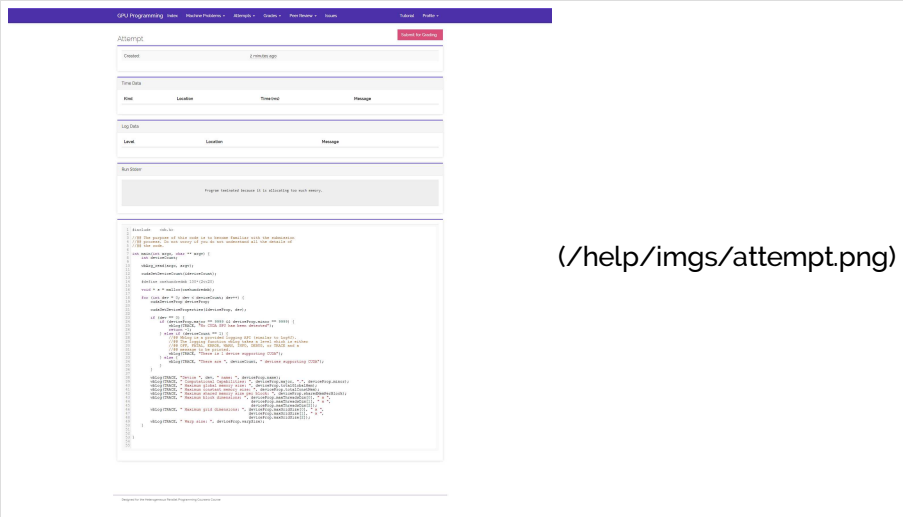
Plagiarism

Plagiarism will not be tolerated. The first offense will result in the two parties getting a 0 for the machine problem. Second offense results in a 0 for the course.

Grading

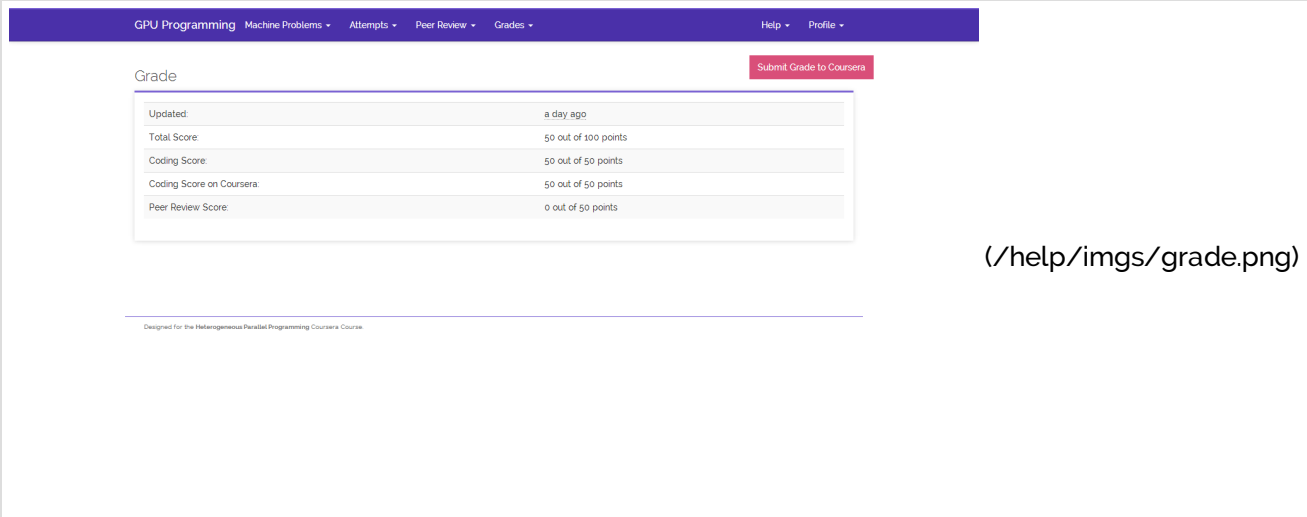
Grading is performed based on criteria that are specific for each lab.

For each attempt in the attempts tab, you'll see a *submit for grading* button.



(/help/imgs/attempt.png)

Clicking on that would tell the system to grade that attempt and redirect you to the grade report page.



(/help/imgs/grade.png)

The grades are automatically sent to Coursera. You will have to submit your grade back to Coursera by clicking the button submit to Coursera button.

Note: you cannot submit code for grading after the corresponding deadlines have passed.

Local Development

While not required, the library used throughout the course can be downloaded from Github (https://github.com/abduld/libwb). The library does not depend on any external library (and should be cross platform), you can use `make` to generate the shared object file (further instructions are found on the Github page). Linking against the library would allow you to get similar behavior to the web interface (minus the imposed limitations). Once linked against the library, you can launch your program as follows:

```
./program -e <expected_output_file> -i <input_file_1>,<input_file_2> -o <output_file> -t <type>
```

The `<expected_output_file>` and `<input_file_n>` are the input and output files provided in the dataset. The `<output_file>` is the location you'd like to place the output from your program. The `<type>` is the output file type: `vector`, `matrix`, or `image`. If an MP does not expect an input or output, then pass `none` as the parameter.

In case of issues or suggestions with the library, please report them through the issue tracker (https://github.com/abduld/libwb/issues) on Github.

Issues/Questions/Suggestions

In case of questions, please post them to the forums. For issues or suggestions, please use the issue tracker (https://github.com/abduld/wb/issues).

Deadline

The coding deadline is 26 days from now (2015-07-10T23:00:00Z) .

Note: you cannot submit code and answer questions for grading nor perform a peer review after the corresponding deadlines have passed.

Grading Rubric

100 out of the 100 possible points is reserved for code development.

Dataset

We provide the dataset (/dataset/1/dataset.zip) to allow you to develop the machine problem on your own system or to examine the data. This is purely optional however. Consult the tutorial (/help) for information about local development.

Designed for the Heterogeneous Parallel Programming (https://class.coursera.org/hetero-004) Coursera Course.