Lab Submission System

We will be using a web based lab development and submission system through this course. The purpose of this web system is two fold:

1. it allows you to develop CUDA applications without the need to buy and/or setup a CUDA system, and
2. it allows us to grade your more efficiently, uniformly, and promptly.

This document describes the online system, walking through the pages that you will see, and describing a sample submission.

Web System

This section describes the interface and system you will be using for this course.

System Interface

When you login to your account, you are directed to the first lab as shown in the following figure

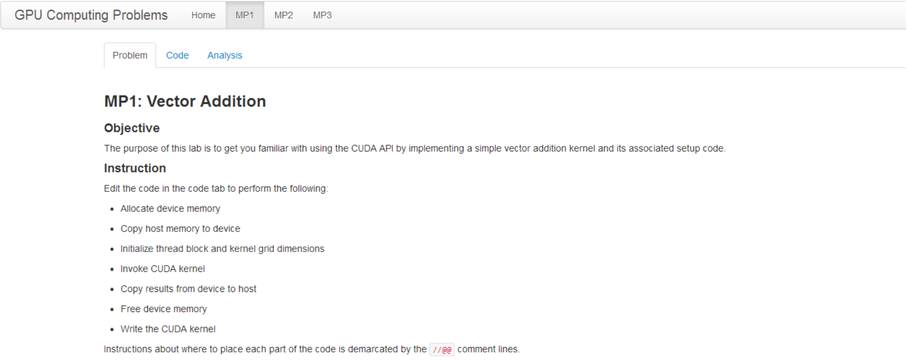
The lab's objective, instructions, and possibly the grading policy is shown in this tab. The second tab is the code view, this where you'll be developing your lab.

Once you have a solution, you click on the submit tab. This will first save your work on the server and then submit it to be processed. In case the server is overloaded, your computation would be placed in a queue (this is a great motivation to not start the lab at the last minute).

The third tab (if shown) shows the previous attempts made at this problem

Insertion Points

Most of instructions for the lab are found in the code. We use `//@@` to demarcated where code needs to be inserted. In lab 1, for example, you see code such as:



MP1: Vector Addition

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.

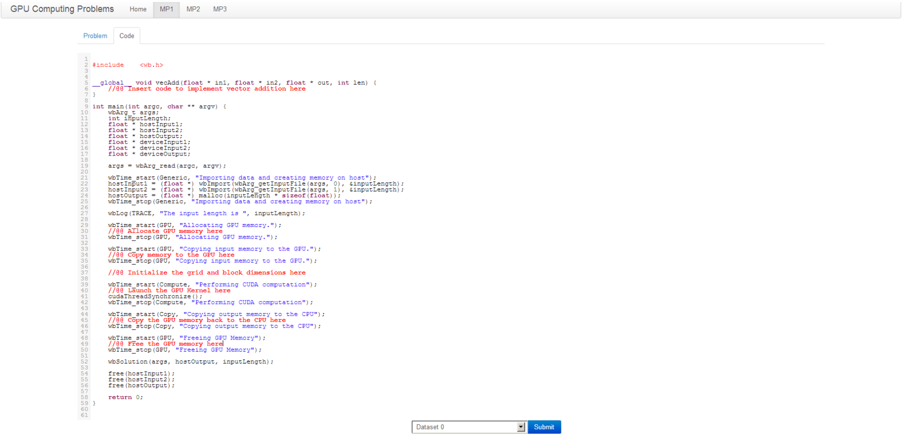
Instruction

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

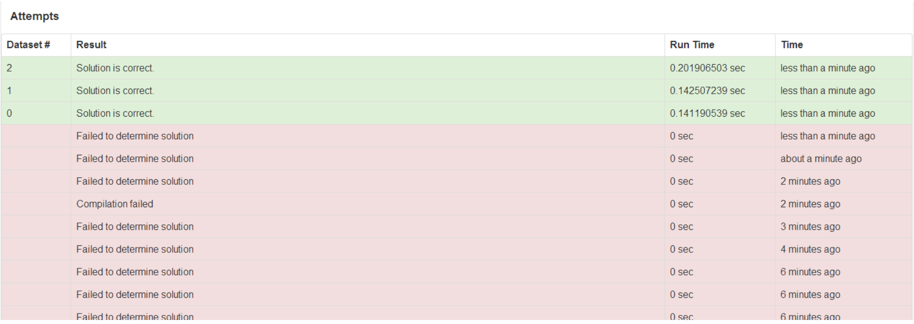
Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

Figure 1: Lab 1 Description



```

#include    <wb.h>


__global__ void vecAdd(float * in1, float * in2, float * out, int len) {
    //@@ Insert code to implement vector addition here
}

int main(int argc, char ** argv) {
    wbArg_t args;
    int inputLength;
    float * hostInput1;
    float * hostInput2;
    float * hostOutput;
    float * deviceInput1;
    float * deviceInput2;
    float * deviceOutput;

    args = wbArg_read(argc, argv);

    wbTime_start(Generic, "Importing data and creating memory on host");
    hostInput1 = (float *) wbImport(wbArg_getInputFile(args, 0), &inputLength);
    hostInput2 = (float *) wbImport(wbArg_getInputFile(args, 1), &inputLength);
    hostOutput = (float *) malloc(inputLength * sizeof(float));
    wbTime_stop(Generic, "Importing data and creating memory on host");

    wbLog(TRACE, "The input length is ", inputLength);

    wbTime_start(GPU, "Allocating GPU memory.");
    //@@ Allocate GPU memory here
    wbTime_stop(GPU, "Allocating GPU memory.");

    wbTime_start(GPU, "Copying input memory to the GPU.");
    //@@ Copy memory to the GPU here
    wbTime_stop(GPU, "Copying input memory to the GPU.");

    //@@ Initialize the grid and block dimensions here

    wbTime_start(Compute, "Performing CUDA computation");
    //@@ Launch the GPU Kernel here
    cudaThreadSynchronize();
    wbTime_stop(Compute, "Performing CUDA computation");

    wbTime_start(Copy, "Copying output memory to the CPU");
    //@@ Copy the GPU memory back to the CPU here
    wbTime_stop(Copy, "Copying output memory to the CPU");

    wbTime_start(GPU, "Freeing GPU Memory");
    //@@ Free the GPU memory here
    wbTime_stop(GPU, "Freeing GPU Memory");

    wbSolution(args, hostOutput, inputLength);

    free(hostInput1);
    free(hostInput2);
    free(hostOutput);

    return 0;
}


```

Figure 2: Lab 1 Code



Attempts

Dataset #	Result	Run Time	Time
2	Solution is correct.	0.201906503 sec	less than a minute ago
1	Solution is correct.	0.142507239 sec	less than a minute ago
0	Solution is correct.	0.141190539 sec	less than a minute ago
	Failed to determine solution	0 sec	less than a minute ago
	Failed to determine solution	0 sec	about a minute ago
	Failed to determine solution	0 sec	2 minutes ago
	Compilation failed	0 sec	2 minutes ago
	Failed to determine solution	0 sec	3 minutes ago
	Failed to determine solution	0 sec	4 minutes ago
	Failed to determine solution	0 sec	6 minutes ago
	Failed to determine solution	0 sec	6 minutes ago
	Failed to determine solution	0 sec	6 minutes ago

Figure 3: Lab 1 Attempts

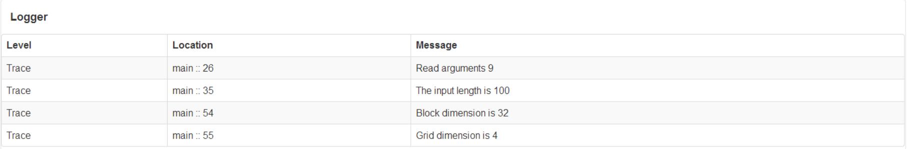
```
wbTime_start(GPU, "Copying input memory to the GPU.");
//@@ Copy memory to the GPU here
wbTime_stop(GPU, "Copying input memory to the GPU.");
```

This means that you need to insert a CUDA memory copy operation where the comment is located.

Logging and Debugging

Logging is facilitated through the use of a logging API. The logging function `wbLog` takes a level which is either `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, or `TRACE` and a message to be printed. To print the value of variable `x`, for example you type `wbLog(TRACE, "The value of x = ", x)`. Note that variables are serialized to a string when placed in the `wbLog` function, so you can place reals, integers, and strings in the logging function.

The result of the logging messages is shown once you submit your code.



Logger

Level	Location	Message
Trace	main :: 26	Read arguments 9
Trace	main :: 35	The input length is 100
Trace	main :: 54	Block dimension is 32
Trace	main :: 55	Grid dimension is 4

Figure 4: Logging View

Debugging is done in the form of logging, so you cannot place break points, etc…

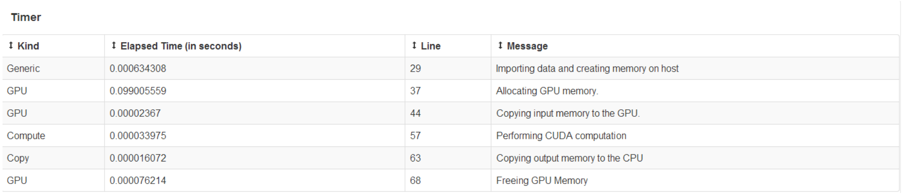
Timing Code

Logging is facilitated through the use of a timing API. The timing functions `wbTime_start` and `wbTime_stop`. To time a section of code, you write `wbTime_start(tag, msg)` and then `wbTime_stop(tag, msg)`, where the `tag` and `msg` must match. In the template code given for Lab 1, for example, we do

```
wbTime_start(GPU, "Allocating GPU memory.");
//@@ Allocate GPU memory here...
wbTime_stop(GPU, "Allocating GPU memory.");
```

the above would time the amount it takes to allocate GPU memory. The timing result in shown in the attempts tab after you submit the program.

The timing information includes information the tag and message you specified when you timed your code, and also which line you are timing. Since is important to understand the performance of your code, so it is valuable to spend time to understand the output of the timer.



Timer

↕ Kind	↕ Elapsed Time (in seconds)	↕ Line	↕ Message
Generic	0.000634308	29	Importing data and creating memory on host
GPU	0.099005559	37	Allocating GPU memory.
GPU	0.00002367	44	Copying input memory to the GPU.
Compute	0.000033975	57	Performing CUDA computation
Copy	0.000016072	63	Copying output memory to the CPU
GPU	0.000076214	68	Freeing GPU Memory

Figure 5: Timing View

Previous Attempts

All previous attempts are saved and recorded on the server side, but you are only graded on the latest attempt.

Walk Through

TODO: need to add an example (with screenshots) of going through the submission process.

Behaviors

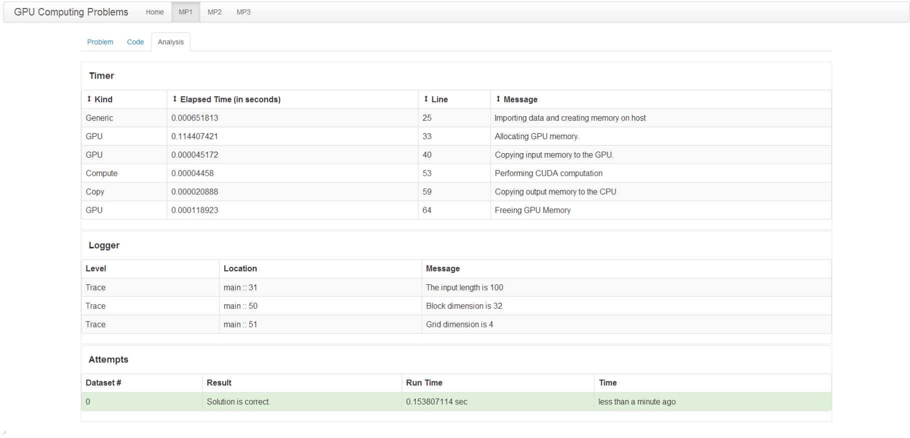
While developing the labs, you may encounter one or more of the following behaviors.

No Error

This means that no errors were found while running the program. Your program has been checked against the expected solution of the dataset you selected. Note that it might be the case that the program may run correctly on one dataset and not the other.

Solution is Incorrect

This means that while the program did compile and run without errors, the solution did not match the expected results. This problem could stem from either having an implementation error in your algorithm, or from having incorrect logic in the host code. Logging the state of your program at different points would help you debug the problem.



Timer

↕ Kind	↕ Elapsed Time (in seconds)	↕ Line	↕ Message
Generic	0.000651813	25	Importing data and creating memory on host
GPU	0.114407421	33	Allocating GPU memory.
GPU	0.000045172	40	Copying input memory to the GPU.
Compute	0.00004458	53	Performing CUDA computation
Copy	0.000020888	59	Copying output memory to the CPU
GPU	0.000118923	64	Freeing GPU Memory

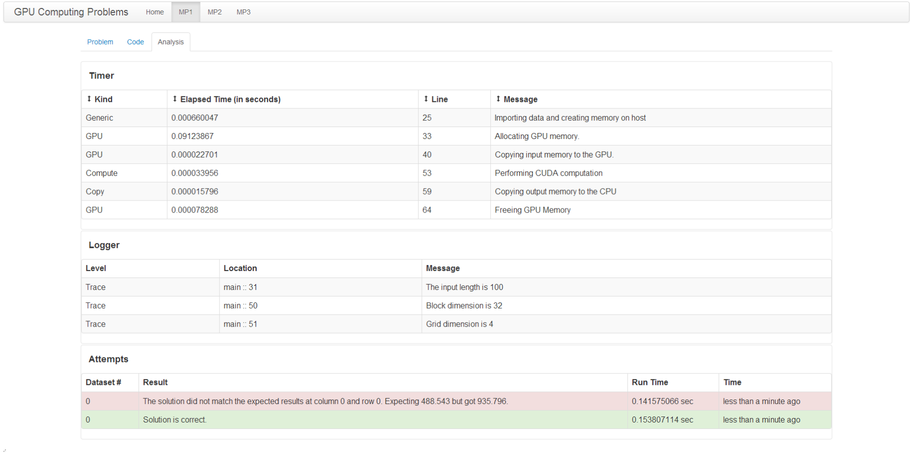
Logger

Level	Location	Message
Trace	main :: 31	The input length is 100
Trace	main :: 50	Block dimension is 32
Trace	main :: 51	Grid dimension is 4

Attempts

Dataset #	Result	Run Time	Time
0	Solution is correct.	0.153807114 sec	less than a minute ago

Figure 6: Solution is Correct



Timer

↕ Kind	↕ Elapsed Time (in seconds)	↕ Line	↕ Message
Generic	0.000660047	25	Importing data and creating memory on host
GPU	0.09123867	33	Allocating GPU memory.
GPU	0.000022701	40	Copying input memory to the GPU.
Compute	0.000033956	53	Performing CUDA computation
Copy	0.000015796	59	Copying output memory to the CPU
GPU	0.000078288	64	Freeing GPU Memory

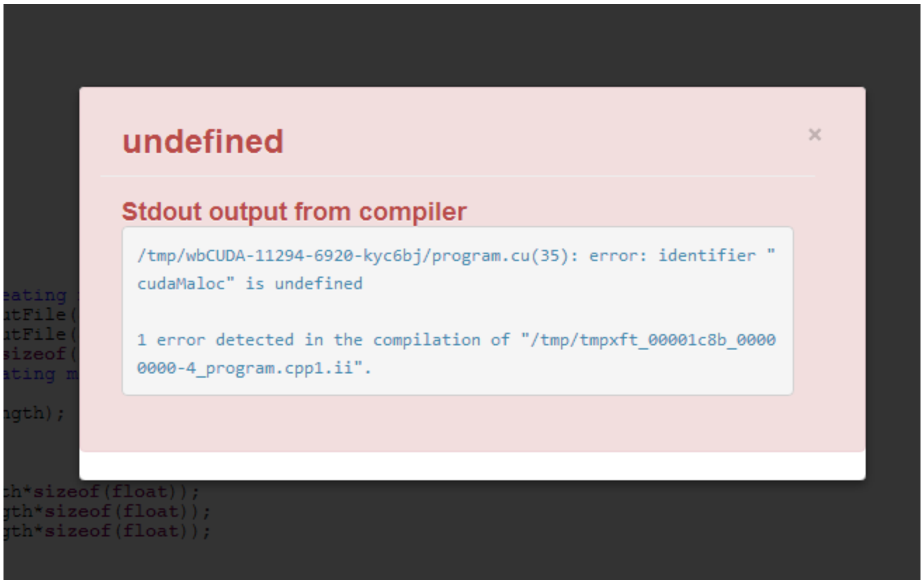
Logger

Level	Location	Message
Trace	main :: 31	The input length is 100
Trace	main :: 50	Block dimension is 32
Trace	main :: 51	Grid dimension is 4

Attempts

Dataset #	Result	Run Time	Time
0	The solution did not match the expected results at column 0 and row 0. Expecting 488.543 but got 935.796.	0.141575066 sec	less than a minute ago
0	Solution is correct.	0.153807114 sec	less than a minute ago

Figure 7: Solution is Incorrect



undefined

Stdout output from compiler

```
/tmp/wbCUDA-11294-6920-kyc6bj/program.cu(35): error: identifier "
cudaMaloc" is undefined

1 error detected in the compilation of "/tmp/tmpxft_00001c8b_0000
0000-4_program.cpp1.ii".
```

Figure 8: Compilation Failed

Compilation Failed

The error message occurs when the program submitted failed to compile. The output from the compiler is shown in the error window.

Program Terminated due to Timeout

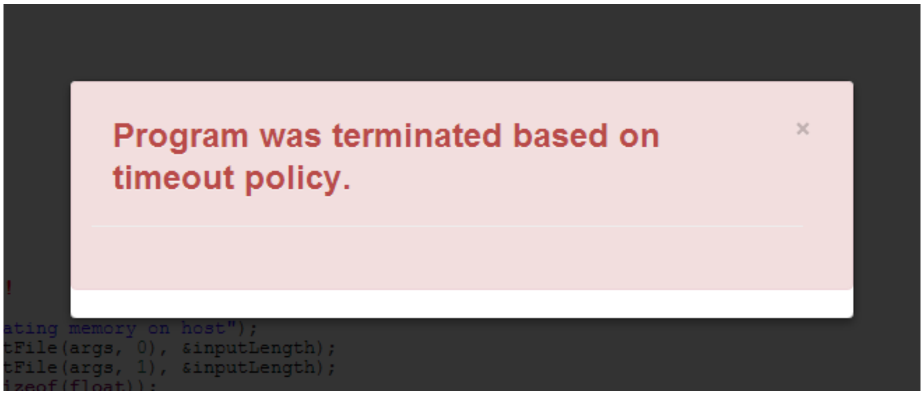
The most likely cause of this error is that you have inadvertently placed an infinite loop either in your CPU or GPU code. Part of the reason for this behavior is that the system ensures fairness (i.e. you should not hog down the machine). To ensure fairness, the system is configured to terminate long running processes. When you see this error, you have hit that timeout limit.

Memory Allocation Error

To maintain system stability, users are not allowed to allocate too much memory.

Sandboxing Error

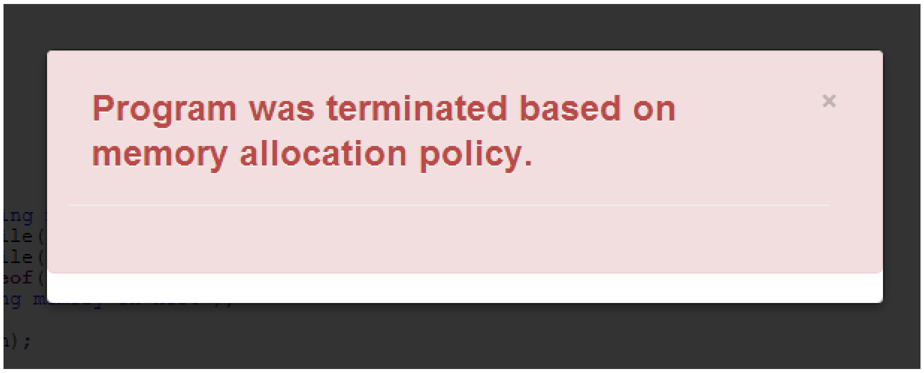
To maintain security, we run your program in a sandboxed mode. This means that you are restriced to using our API functions for certain tasks, rather than



Program was terminated based on timeout policy.

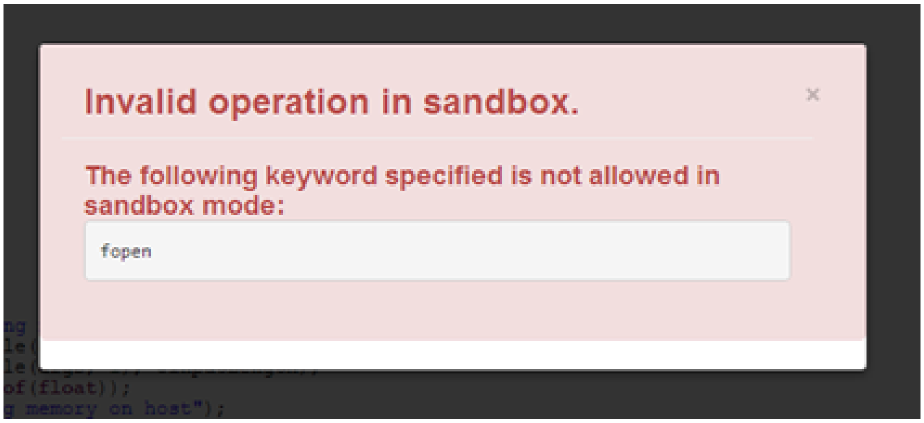
```
ating memory on host");
tFile(args, 0), &inputLength);
tFile(args, 1), &inputLength);
izeof(float));
```

Figure 9: Program Terminated



Program was terminated based on memory allocation policy.

Figure 10: Memory Allocation



Invalid operation in sandbox.

The following keyword specified is not allowed in sandbox mode:

fopen

Figure 11: Sandbox Error

using system calls or C library functions. The sandbox error would sometimes tell you what call is being caught, although that might not always be the case.

System Flow

TODO: need to add some information about how queries are handled and processed. Also, discuss what happens when the system gets overloaded.

Suggestions

The following suggestions are recomended when getting started using the system

- Develop your application incrementaly
- Do not modify the template code written – only insert code where the `//@@` demarcation is placed
- Make sure to test your results against all the datasets provided
- Do not wait until the last minute to attempt the lab.

Grading

Grading is performed based on criteria that are specific for each lab. Aside from the program submitted compiling and running, we also look at how fast the program is in relation to the other submissions.

System Requirements

A recent web browser is the only requirement for using and submitting labs in this course.

Issues/Questions/Suggestions

In case of issues or questions, please post to the forums. Suggestions are welcome and can also be posted on the forums.