ELEC 522 Project 1 CUDA Code Problem.

Fall 2024

A CUDA example program ***Proj1_kernel.cu*** has been created to highlight the structure of solving a problem in parallel on a GPU. The program is the basic addition of two vectors *a* and *b* leading to a result *c*.

The code is structured in several parts listed below:

1. Lines 1 through 12 are the header files needed for C on CPU and CUDA on GPU.
2. Lines 15 and 15 declare the CPU Helper function *addWithCuda*
3. Lines 18 through 27 are the actual CUDA kernel definition to run on the GPU. Notice that there is no *for* loop here as would be used in CPU code. There is one statement that defines the variable *i* and then one equation for *c* parameterized on *i*. In a GPU we will allocate parallel threads and each thread will execute seemingly in parallel. So here the hardware scheduler understands how to perform each value from 0 to the maximum number of threads in parallel on the available CUDA hardware cores. This becomes more complicated in very big problems.
4. Line 30 begins the *main* program on the CPU through line 70
    a. Lines 33 through 42 initialize the data for this example using increasing values.
    b. **NOTE: The *arraySize* variable on line 33 sets the problem size. We will experiment with this further in this problem and it is one of the values that you will change. (It is initially 10 in the code posted.)**
    c. Lines 44 to 51 call the CPU *addWithCuda* Helper function which does most of the work in this example. It will be listed further in the file.
    d. Lines 53 to 58 print the input values and also the output values read back from the GPU. Note what happens as the problem *arraySize* exceeds the number of total threads.
    e. Lines 60 to 70 do some CUDA error checking and end the program.
5. Lines 73 through 236 are the CPU Helper function *addWithCuda*
    a. Lines 83 to 93 are important problem configuration statements. In this example, we will do just one iteration of the vector addition. If we wanted more accurate average timing results, then we would repeat the problem multiple time to deal with runtime variation.
    b. **NOTE: Lines 88 and 93 set important variables for GPU execution and you will change them in the experiments listed below. The initial number of GPU *threads* is set to the *size* of the vectors passed in from *main*. The initial number of *blocks* is set to 1. If the *arraySize* passed in as *size* exceeds, 1024, there will be an error since the maximum number of threads per block is 1024 in the CUDA specifications. I did not make the code "automatic" to recognize this issue (other than a warning) since I wanted you to experiment and edit the code. The *threads* can then be set to 1024**

and the *blocks* increased from 1 to a larger number. You can see what happens to the result values in the variable *c* as the problem size exceeds the total threads allocated.

c. Lines 95 to 99 check to see if this is a multi-GPU system.
d. Lines 102 to 120 create buffers in the GPU memory to hold the input and result arrays using calls to *cudaMalloc.*
e. Lines 123 to 137 do the actual transfer of data to the GPU through *cudaMemcpy.*
f. Lines 139 to 164 are for performance instrumentation. The GPU can track various events through *cudaEventCreate* and a start time on the GPU is marked.
g. Line 167 to 171 are the actual call to start the computation kernel on the GPU in calling *addKernel* and passing the data variables and the runtime configuration of threads and blocks.
h. Lines 174 to 190 record the stop time and are a synchronization barrier to be sure that all data on the GPU has been updated.
i. Lines 193 to 199 copy the result vector *c* back from the GPU to the CPU memory.
j. Lines 201 to 209 now compute the elapsed time difference between the start and stop timestamps that have been made.
k. Lines 213 to 228 first check if any errors occurred in launching the GPU kernel**. If no errors, then several performance metrics of the kernel execution time on the GPU are printed.** (Note here that we did not time the transport time, that is the time to copy data from CPU to GPU memory, or the time to copy back from GPU to CPU. We could have done this with additional calls to the event timers.)
l. Lines 230 to 236 free up GPU memory and conclude the helper function and return to the *main* program.

Compile the initial Project 1 CUDA program.

1. Add this file ***Proj1_kernel.cu*** to a new CUDA project in Visual Studio 2022. Follow the procedure in the demo and example from class on Canvas at Files / CAD_Tool_Documentation / Visual_Studio_CUDA_start.pdf
2. With the defaults of ***arraySize = 10*** on line 33, ***threads = size*** on line 88 and ***blocks = 1*** on line 93, “Build / Build Solution.” There should be no CUDA errors. Ignore the single error in the CUDA syntax that Visual Studio intellisense does not understand.

Carefully document your results in your project report for the following experiments:

1. Then “Debug / Start without Debugging” to execute the program. Save a screenshot of the result window and upload with your solution to Canvas. Create a table of performance results that you will also upload to Canvas. Record for this case and the cases below: Performance in Mops/s, Time, Size, Threads, Blocks, and Total Threads.

2. Modify the code to increase the *arraySize* of 10 on line 33 in multiple steps of 100, 200, 500, 750, 1000, and 1250. After each step change of *arraySize* then Build again, Debug again, and record performance results in your table.
3. What happens at *arraySize = 1250* with the original *threads* and *blocks* value? Explain.
4. Further increase the *arraySize* to 1500, 2000, 5000, 10000, 25000.
5. Also save a screenshot of the Debug results for *arraySize* = 25000.
6. Note that you will need to change the threads and blocks sizes for the larger values.
7. With the minimum *threads* and *blocks* sizes appropriate for *arraySize* = 25000, increase the *arraySize* to 64000. What are the values of the final two values of *c* ? Explain. How would you fix this problem and add your (correct) data to the table.
8. From the data in your table of the various experiments from *arraySize* of 10 to 64000, what is the percent increase in performance, time, and size. Explain the causes and trends.
9. Upload your final ***Proj1_kernel.cu*** file to Canvas after your final changes.

URLs for additional suggested readings on CUDA and GPUs

https://developer.nvidia.com/blog/easy-introduction-cuda-c-and-c/

https://developer.nvidia.com/blog/how-implement-performance-metrics-cuda-cc/

https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/

https://developer.nvidia.com/blog/even-easier-introduction-cuda/

https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

https://docs.nvidia.com/cuda/index.html

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html