

In this project, you will use CUDA to implement the Triangular Solve Operation on GPUs. The goal of the project is to understand the challenges of writing efficient general purpose (ie, not graphics) programs on GPUs. You will compare the performance of your code to the sequential implementation and the highly optimized CUBLAS version.

1 Algorithm

The triangular solve is a basic linear algebra operation used to solve a system of equations involving a triangular matrix. The code snippet below solves a simplified version of the triangular solve problem.

```
for (j = 0; j < n; j = j + 1)
  for (k = 0; k < n; k = k + 1)
    if (B[j * n + k] != 0.0f) {
      for (i = k + 1; i < n; i++)
        B[j * n + i] -= A[k * n + i] * B[j * n + k];
    }
```

As you can see, there is abundant data reuse in this problem, so it is ideal for GPU computation. However, to get good performance, you will need to be careful to balance the load as you assign work to processing units, since the amount of work for different $\langle j, k \rangle$ values is different.

2 Details

You should implement a restricted version of the triangular solve: $A \times X = \alpha \times B$. Here, A and B are square matrices, and A is a lower triangular matrix and is unit triangular (its diagonal elements are all 1s). The CUBLAS call under these restrictions would look something like this:

```
cublasStrsm('U' * leftoperator * '/', 'U' * lowertriangular * '/', 'N' * nottransposed * '/', 'u' *
            unittriangular * '/', N, N, alpha, d_A, N, d_B, N);
```

Input Your code should take three arguments:

1. Input file for A (Check the TA webpage for file format and sample input files)
2. Input file for B
3. alpha
4. A character to indicate which implementation to run: S for sequential, G for your GPU implementation, and C for the CUBLAS implementation

To check the correctness and performance of your code, you should generate random matrices A and B with sizes varying from 10^3 to 10^8

Output Your code should output the following:

1. The resultant matrix X
2. Execution time of the code

3 Using CUDA

Below are some references to get you started on CUDA programming:

- Programming Guide:
http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_Program
- Reference Manual:
http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_Toolkit_Referen

4 Performance Comparison

The sequential implementation of the problem has been provided (check the TA web page). The CUBLAS library contains highly optimized implementations of the basic linear algebra operations on GPUs. You should compare the performance of your implementation with the sequential version on CPU and the CUBLAS version for various input sizes and present the results in your report. You are also encouraged to compare the performance benefits of the various optimizations that you try.

What to Turn In

This default project is due on May 12, 11:59 pm.

Prepare a tar file for your submission. This tar file should have your source code, job script for longhorn, a report and a readme file. The readme file should have your names and instructions for compiling and running your code on longhorn.

To submit your assignment, use the following command:

```
turnin --submit akanksha cudaproject tarfile
```

Final Report In your report, describe your methodology, your work mapping scheme, the optimizations you tried, what worked, what did not work and most importantly, your insight about why things worked or did not work. Also, evaluate the performance of your code by comparing it against sequential implementation and the CUBLAS implementation, and explain your observations.