Multicore Architecture and Parallel Programming
Assignment on CUDA Programming
Due: November 5th

1. The most effective way to fully understand a programming tool is to develop programs or real applications. There are many problems, which can be effectively solved with CUDA, are suitable for a novice as exercise. Here is one example.
   As you may know, Kernel Method (KM) is used to avoid computation and complexity in Support Vector Machine (SVM). RBF kernel is one of the famous kernel functions. Its definition is
   $$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$
   Where x_i and x_j are n-dimensional vectors with each representing a piece of n-dimensional feature. It's clear that calculating RBF kernel function using single CPU thread may be slow and it can be effectively parallelized using CUDA.
   PROBLEM:
   Please write CUDA kernel functions to effectively compute the RBF kernel of two matrices. You may need to write wrapper functions to accomplish the task.
   (Hint: if you have any problems, please first refer to matrix multiplication algorithms)

2. Have you ever experienced the power of GPU computing? To achieve the maximum performance, there are many tricks based on the processor and memory architecture. I want to show you a trick that you may never see in most CUDA documentations released by NVIDIA. The memory hierarchy of CUDA architecture brings us global memory (including constant memory, texture memory and surface memory), shared memory and registers. You must realize that shared memory should be used if possible to avoid high latency accessing global memory. But shared memory is again far slower than registers. How can we achieve the extreme performance? Please read reference [2] and optimize your program to get more speedup.

References

[1]. http://svr-www.eng.cam.ac.uk/~kkc21/thesis_main/node31.html
[2]. http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf

NOTICE:
1. You have to write a makefile to compile your code.
2. Send your final version to TA at yukang.tian@outlook.com. You should archive your source code and makefile with **StudentID_Name_CUDA**.tar.gz(or any archive file types). Do not include binary file.
3. It should be a standalone function to perform the computation, i.e., everybody can reuse your function to do the similar job with variable configurations.
4. Should you have any questions, please feel free to contact TA at yukang.tian@outlook.com.