GAUTHAM SRINIVASAN
927008557

CSCE 689 HW5

1. **In this assignment, you have to develop GPU code to compute f(x,y) for a given point(x,y). The code should use a single processor of the device but should be parallelized to exploit all the cores within the processor. The code should initialize the grid points and observed data values using equations (1) and (2) on the host and move these values to the GPU device. Next, the matrix K' = (tI+K) should be computed on the device. This should be followed by LU factorization of K' on the device. These factors should be used to compute the solution of the system K'z=f using the L and U factors obtained in the previous step. Finally, the predicted value should be computed as f(x,y) = k T z. You must develop your own code to compute the LU factors and to solve the triangular systems. LU factorization can be replaced by Cholesky factorization, which is a more efficient algorithm for symmetric positive definite matrices.**

   Compilation:
   nvcc -arch=compute_35 -code=sm_35 -o GPR.exe GPR.cu

   Execution:
   ./GPR.exe 2 0.5 0.5 *//where 2 indicates matrix dimension (2x2), 0.5 and 0.5 are xy coordinates*

2. **Describe your strategy to parallelize the algorithm for a single multiprocessor of the GPU. Discuss any design choices you made to improve the parallel performance of the code.**

   Gaussian Process Regression is a statistical model where every point in some continuous input space is associated with a normally distributed random variable. It can be used to predict the values of a function at a point from observations at other points in the domain.

   The algorithm design is explained below:

   - Used a structure containing the x-y coordinates to initialize the grid points as given by the input dimension using the equation (below). This is done by initialize_points function.

$$(x_i, y_j) = (ih, jh) \text{ where } h = 1/(m+1)$$

- Calculated the observed data using the equation (below) which is done by calculate_observed_data function

$$f(x_i, y_j) = 1 - [(x_i - 0.5)^2 - (y_j - 0.5)^2] + d_{ij}$$

- Finally, k and K matrix is calculated by compute_K_matrix and compute_k function using

  $K(r, s) = exp(- || r - s||^2)$ *and* $k(r) = exp(- || r - r^*||^2)$

- Matrix K' = (tI + K) is computed. It is factorized into a set of upper or lower triangular matrices and its inverse is found using Cholesky factorization. All the parallel processes have been scheduled to be executed onto the threads such that if the number of threads available are t, the first t processes would be run in parallel and then the next t threads and so on.

  In the Cholesky factorization function, all the threads are synchronized using the block level synchronization barrier __syncthreads() at every stage.

  Cholesky factorization is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. Every Hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition.
- Finally, run_solver function computes the predicted value by matrix multiplication. This is done using executing the multiplication of two matrices parallely with multiple threads.

3. **Compute the flop rate you achieve in the factorization routine and in the solver routine using all the cores. Compare this value with the peak flop rate achievable on a single core, and estimate the speedup obtained over one core and the corresponding efficiency/utilization of the cores on the node.**

   Flop rate = Number of floating point operations / execution time

   Flop rate in solver routine = 20000 / (3.13 ms) = 6.6 MFlops
   Flop rate in Cholesky routine = 50000 / (68.64 ms) = 735 KFlops

General information		
Market segment	HPC / Server	
Manufacturer	NVIDIA	
Model	Tesla K20c	Tesla K20m
Architecture / Interface		
Die name	GK110	
Architecture	Kepler	
Fabrication process	28nm	
Bus interface	PCI-E 2.0 x 16	
Cores / shaders		
CUDA cores	2496	
ROPs	40	
Pixel fill rate	28.24 gigapixels/s	
Texture units	208	
Texture fill rate	146.85 gigatexels/s	
Single Precision performance	3524.35 GFLOPS	

From the NVIDIA data, we get that
Peak flop rate = 3524.35 GFLOPS

Following observations made for the matrix of the size 10. For single processor, the execution time is taken by executing the mini-project code with NUM_PROCESS = 1

Processors	**Cholesky Time (sec)**	**Solver Time (sec)**
1	95.3 ms	10.99 ms
192	68.64 ms	3.13 ms
Speedup	1.4	3.5
Efficiency	0.007	0.01