CSCE 735 Fall 2021

Major Project

Due: 11:59pm Wednesday, December 15, 2021

(No late submission allowed)

Gaussian Process Regression can be used to predict the values of a function at a point from observations at other points in the domain. As an example, consider an m x m grid of points on a two-dimensional unit square. Node coordinates are given by

$$(x_i, y_j) = (ih, jh), i = 1, \dots, m, j = 1, .., m, \quad (1)$$

where h = 1/(m+1) is the mesh width. Observed data value at the point $r(x_i, y_j)$ is given by the function

$$f(x_i, y_j) = 1 - \left((x_i - 0.5)^2 + (y_j - 0.5)^2\right) + d_{ij}, \quad (2)$$

where d_{ij} is a random value between [–0.05,0.05]. For the remaining discussion, it helps to think of the m^2 grid points as a list of $n = m^2$ points obtained by concatenating $(x_i, y_j), i = 1, \dots, m, j = 1, .., m,$ in that order.

The predicted value of the function at a prediction point $r^*(x, y)$ is given by

$$f^* = k_*{}^T (tI + K)^{-1} f, \quad (3)$$

where f is an $n \times 1$ vector. K is a matrix representing correlation between the data points and is defined as follows: for two points $r(x, y)$ and $s(u, v)$,

$$K(r, s) = \frac{1}{\sqrt{2\pi}} e^{-\left(\frac{(x-u)^2}{2l_1^2} + \frac{(y-v)^2}{2l_2^2}\right)}, \quad (4)$$

where e is the base of natural logarithm. Note that K is an $n \times n$ matrix where elements in row r, from left to right, correspond to values obtained using (4) for points s for $(x_i, y_j), i = 1, \dots, m, j = 1, .., m,$ in that order. The one-dimensional vector k_* for the prediction point $r^*(x,y)$ is computed as given below:

$$k_* = \frac{1}{\sqrt{2\pi}} e^{-\left(\frac{(x-u)^2}{2l_1^2} + \frac{(y-v)^2}{2l_2^2}\right)}, \quad (5)$$

for all grid points $s(u, v)$ where the ordering of elements of k_* is similar to f.

Note once again that for an $m \times m$ grid, $n = m^2$. Thus, the vectors f, f^*, k_* in (3) are $n \times 1$ and the matrix K is $n \times n$. Further, t is a noise parameter that is set to 0.01.

We also introduce two hyper-parameters l_1 and l_2 that are be chosen to maximize the likelihood of the prediction being accurate for the given data. In this project, we will use $l_1 = l_2 = 1$.

1. (70 points) In this project, 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 A = (tI+K) should be computed on the device. This should be followed by LU factorization of A on the device. These factors should be used to compute the solution of the system Az=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.

Please refer to the code attached. To run the code on GPU you should below on Grace logins node with GPU (I ran it on Grace2.HPRC.TAMU.EDU).
module load CUDA
nvcc majorProject.cu -o majorProject.exe;./majorProject.exe
Also, I did develop the code for CPU version and it can be run using:
nvcc majorProjectCPU.cu -o majorProjectCPU.exe;./majorProjectCPU.exe

2. (20 points) 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.

The GPU allows for better performance if and only if we can do a same task repeatedly on different element of a matrix. Also, it should be noted that GPU suffers from not having branch prediction and therefore, does not have a good relationship with if statements. Hence, we should avoid if statement in our code as much as possible.
My code (both the GPU and CPU based) starts with initializing some constants including the prediction point coordinates (xR, yR), fineness of the mesh discretization (m), normalization factors (L1 and L2), noise (t) and the memory allocation for arrays (initialization function).
Subsequently, we have to initialize x and y coordinates of the points in the map and their function values f. Next, we need to find the correlation between points (K) and correlation between points and the prediction points (K*). To speed up the process, I did all of these initializations on the GPU as it can be done over the matrix and vector elements repeatedly (cudaInitXY, cudaInitF, cudaInitK, cudaInitKStar functions).
Next, we have to call the main part of the program which is the solver function. It has to first decompose the (tI+K) matrix into L and U (LU decomposition) and then solve and fid the result of $(tI+K)^{-1}F$. The first part of this function is done by LUDecompositionPrep function while the latter one is handled by LUSolverPrep. 1) LUDecompositionPrep calls four kernel function in a loop (over al rows) to find the LU decomposition. To avoid synchronization and race condition, I find the L[i][j]* U[j][k] in cudaMatrixMul (L, U) and then fed the result to cudaUFactorization which updates the upper triangular matrix. As the values of U[j][k] is changed, I re-calculated their multipilication (L[i][j]* U[j][k]) and pass the results to cudaLFactorization to update the lower triangular matrix. After doing these steps for all of the rows, we have to copy the matrix values from GPU to RAM. It should be noted that my LUFactorization code is based on the one available at geeksforgeeks.com but I dissected it into three main parts to enhance the penalization (finding the reulst of L*U, diving each row by diagonal elements in cudaLUNormalizer function and finally finding the actual lower and upper triangular parts) . 2) LUSolverPrep uses L, U and F matrices to find $(tI+K)^{-1}F$. How? Assume that A=tI+K and A== L*U (using LU decomposition). Now, we have to solve $A^{-1}F=Q$ to find Q as we know A and F. This resembles F=AQ and F=LUQ. Now, assume W=UQ→F=LW where W can be found using back-substation (cudaLSolver function). Similarly, we can find Q in W=UQ (cudaUSolver function).
Finally, in the last step, we have to multiply K* into the result of $(tI+K)^{-1}F$ (which we found in the previous step) to get the predicted value for the point.
It should be noted that my code can still get enhanced by removing the redundancy in memcopy calls. But I am really out of time for these final touchups.

3. (10 points) Compute the flop rate you achieve in the factorization routine and in the solver routine. Compare this value with the peak flop rate achievable on the processor, and estimate the speedup obtained over one core and the corresponding efficiency/utilization of the cores on the device. You may choose appropriate values for the grid size to study the features of your implementation.

Submission: You need to upload the following to Canvas:

1. Submit the code you developed.
2. Submit a single PDF or MSWord document that includes the following.
    - Responses to Problem 1, 2, and 3. Response to 1 should consist of a brief description of how to compile and execute the code on the parallel computer

I am out of time to run the program.