GAUTHAM SRINIVASAN
927008557

CSCE 689 Mini Project

1.In this assignment, you have to develop an OpenMP-based parallel code to compute f(x,y) for a given point (x,y). The code should use a single shared-memory node on ADA or Terra, and should be parallelized to exploit all the cores on the processing node. The code should initialize the grid points and observed data values using equations (1) and (2). Next, the matrix K' = (tI+K) should be computed. This should be followed by LU factorization of K'. 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)=kTz. 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:
icc -qopenmp -o GPR.exe GPR.c

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

Using Job file:
bsub < GPR.job

2. Describe your strategy to parallelize the algorithm. 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

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

- 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. This is done by *get_inverse_by_cholesky* function which uses the *task region* to invoke multiple threads such that each thread has its own data variables for synchronization.
  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 parallely using *omp for pragma*.

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 = 4000 / (1.17064e-04) = 8.54 MFlops
Flop rate in Cholesky routine = 3000 / (6.06060e-04) = 4.95 MFlops

Peak flop rate = (CPU speed in GHz) * (number of CPU cores) * (CPU instruction per cycle) * (number of CPUs per node)

= 2.50GHz * 1 * 4 * 1
= 10 GFlops

Following observations made for the matrix of the size 10,

Processors	**Cholesky Time (sec)**	**Solver Time (sec)**
1	5.14984e-04	1.09911e-04
20	4.63963e-04	7.48634e-05
Speedup	1.11	1.48
Efficiency	0.05	0.07