CSCE 735 Fall 2021

Minor Project (OpenMP)

Due: 11:59pm Nov 15, 2021

The inverse of an upper triangular matrix R with a block 2x2 structure can be represented as shown below.

$$R^{-1} = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}^{-1} = \begin{bmatrix} {R_{11}}^{-1} & -{R_{11}}^{-1} R_{12} {R_{22}}^{-1} \\ 0 & {R_{22}}^{-1} \end{bmatrix}$$

Thus, the inverse of R can be computed recursively by computing the inverse of the two diagonal blocks R_{11} and R_{22} followed by the off-diagonal block. The recursion terminates when the leaf-level matrix is less than or equal to `leaf_matrix_size`, a user provided input.

You are provided with a C++ code in `Rinverse.cpp` that initializes an upper triangular matrix R and computes it inverse in place, i.e., overwrites R with its inverse. You need to parallelize the following routines using OpenMP directives to obtain a parallel code:

- `compute_matrix_inverse_recursively`: a recursive routine that computes the inverses of R_{11} and R_{22} recursively, then computes the off-diagonal block by calling `compute_off_diagonal_block`. This routine should be parallelized using OpenMP `tasks` directive.
- `compute_off_diagonal_block`: compute computes the off-diagonal block by multiplying R_{12} with the inverse of R_{11} from the left and R_{22} from the right. This routine should be parallelized using OpenMP `for` directive.

1. (75 points) You are required to parallelize the C++ implementation provided in `Rinverse.cpp` using the strategy outlined above.
2. (15 points) Determine the speedup and efficiency obtained by your routine on up to 48 processors. You may choose appropriate values for the matrix size and leaf matrix size to illustrate the features of your implementation.
3. (10 points) Explore the performance of your code by experimenting with different values of `leaf_matrix_size`. Describe what you observe as you change this parameter for different problem sizes.

   Notice that `leaf_matrix_size` allows your algorithm to vary between a purely recursive algorithm (with leaf matrix size of 1) and a non-recursive/iterative algorithm implemented in `invert_upper_triangular_matrix_block`. While the project does not require you to parallelize the latter routine, it can be a challenging to attain good load balance in that routine. The recursive-iterative hybrid approach in this project is typical in parallel computing since it allows efficient single-processor implementations (iterative routine) to be combined into a parallel algorithm (recursive routine).

Submission: You need to upload the following to Canvas as a **single zip file**:

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

Helpful Information:

1. Load the Intel software stack prior to compiling and executing the code. Use:
```
module load intel
```
2. The run time of a code should be measured when it is executed in dedicated mode. Create a batch file as described on hprc.tamu.edu and use the following command to submit it to the batch system:
```
sbatch < batch_file
```