CSCE 659 Fall 2017

HW 4: Dense Matrix Computations with OpenMP II

Due: 11:59pm Saturday, Wednesday 22, 2017

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.

A Matlab code of the implementation is given below. The code should be saved to a file called `Rinverse.m`. The routine `Rinverse()` creates a well-conditioned random R and calls `compute_inverse()` to compute the inverse of the upper triangular matrix recursively.

```
function [R, Ri] = Rinverse(n)
  A = rand(n);
  A = A + diag(sum(abs(A)));
  R = triu(A);
  Ri = compute_inverse(R);
  fprintf('Error in computing inverse: %e\n', ...
          norm(Ri*R-eye(size(R))));
return

function Ri = compute_inverse(R)
  n = size(R,1);
  if (n < 16)
    Ri = inv(R);
  else
    n1 = round(n/2);
    Ri = R;
    Ri(1:n1,1:n1) = compute_inverse(Ri(1:n1,1:n1));
    Ri(n1+1:n,n1+1:n) = compute_inverse(Ri(n1+1:n,n1+1:n));
    Ri(1:n1,n1+1:n) = - Ri(1:n1,1:n1) * Ri(1:n1,n1+1:n) ...
                        * Ri(n1+1:n,n1+1:n);
  end
return
```

1. (40 points) You are required to develop a parallel C/C++ implementation of the above algorithm that uses OpenMP directives to parallelize the `compute_inverse` routine. You should use the task directive for the recursive tasks.
2. (5 points) Describe your strategy to parallelize the algorithm. Discuss any design choices you made to improve the parallel performance of the code.
3. (5 points) Determine the speedup and efficiency obtained by your routine on 1, 2, 4, 10, and 20 processors. You may choose appropriate values for the matrix size to illustrate the features of your implementation.

Submission: You need to upload the following to eCampus:

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. Source file(s) are available on ecampus.
2. Load the Intel software stack prior to compiling and executing the code. Use:

   ```
   module load intel/2017A
   ```
3. 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:

   ```
   bsub < batch_file
   ```