COL 380
ASSIGNMENT 2 REPORT

Goalla Varsha(2018CS10334)
Kavya(2018CS10350)

STRATEGY 1

Approach:

In this we have parallelized three for loops. First we used ‘parallel for’ for the loop assigning value 1 to U[i][i]. We have a for loop iterating on j. But in this loop at each iteration, values calculated in the previous iteration are used. There is loop dependency. So, we can't parallelise this loop. This for loop contains 2 loops iterating on i. And iterations in these loops are independent of each other. There are no loop dependencies. Hence we used ‘parallel for’ for the loop where we are calculating L[i][j] values {for i in (j,n). Then we parallelized the for loop where we are calculating U[j][i] values (i=j to n).

Locations having potential data races and correctness by avoiding data races :

As mentioned earlier, the outer loop iterating on j had loop dependency. So we didn’t parallelise that loop. But parallelised the inner 2 loop iterating on i with no loop dependencies. So there were no data races and this guarantees the correctness of our program.

STRATEGY 2

Approach:

If we use section constructs they need to be independent . There should not be data dependencies between loops in different section constructs. We know that the outer loop which iterates on j cannot be parallelised. We have two inner for loops iterating on i . They have a data dependency between them. L[j][j] calculated in the first inner loop is used in the second inner loop. To be able to place these for loops in different sections we separated the i=jth iteration from the first inner loop. We have now placed the two inner loops into two sections with i=jth iteration separated.

Since we have separated the iteration i=j from the first inner loop there are no dependencies in the remaining loops. We have used sections in the first for loop where we are writing the diagonal values of U matrix i.e. U[i][i]=1. And also in the separated iteration of the first inner loop that is i=jth iteration we have used sectioning in the for loop it contains.

Locations having potential data races and correctness by avoiding data races :

As mentioned earlier ,L[j][j] calculated in the first inner loop is used in the second inner loop. So there is a data dependency . As to put the two inner loops into sections there must not be any data dependency . So to avoid that we separated the i=jth iteration from the first inner loop.Other than that as mentioned in strategy 1 ,the outer loop iterating on j had loop dependency. So we didn’t parallelise that loop. So there were no data races and this guarantees the correctness of our program.

STRATEGY 3

Approach:

It is the same as strategy 2 we have used only 2 sections i.e. one section for each inner loop(Note that i=jth iteration of first inner loop is seperated) and inside the both sections we used omp 'parallel for' for each inner loop to parallelise each for loop. Instead of using sections to parallelise the for loop which is writing diagonal elements of U, we directly used omp parallel to parallelise that for loop. Similarly, in the separated iteration of the first inner loop (i.e. i=jth itr) instead of using sections we used omp parallel to parallelise the for loop iterating from k=0 to j.

Locations having potential data races and correctness by avoiding data races :

As mentioned in strategy 1,the outer loop iterating on j had loop dependency. So we didn't parallelise that loop. But parallelised two inner for loops as there were no data dependencies. As for sectioning there was only one data dependency i.e. L[j][j] calculated in the first inner loop is used in the second inner loop. So to avoid that we separated the i=jth iteration from the first inner loop. So now there is no data dependency in our code and this guarantees the correctness of our program.

STRATEGY 4

Approach:

In the first inner loop that is iterating from i=j to i=n-1, a process calculator L[i][j] iff i%(no_of_process)=rank of the process. So, each process calculates only nearly n/no.of.processes values. After completion of the loop, the process broadcasts the L values it calculated. After broadcast is done, all processes have the same L.

Now in the second inner loop that is iterating from i=j to n-1,a process calculator U[j][i] iff i%(no_of_process)=rank of the process. So, each process calculates only nearly n/no.of.processes values. After completion of the loop, the process broadcasts the U values it calculated. After broadcast is done, all processes have the same U.After this next iteration of outer loop continues. Since we cant distribute iterations of outer loop among the processes, we have distributed work in each iteration of outer loop among the processes.

Locations having potential data races and correctness by avoiding data races :

As mentioned earlier the outer for loop iterating over j which contains two inner for loops iterating over i. The outer for loop has data dependencies so we cannot parallelise it.

Each process has its own copy of L. Each process calculates only some values of Lin the inner loop. For the second inner loop we need the L[j][j] value for each process. L[j][j] is the dependency between both inner loops. But the L[j][j] value is calculated by only 1 process. So here other processes can get wrong values of L[j][j]. Since we have used MPI broadcast function and sent L[j][j] and other L values to all other processes, we are guaranteed that we have correct L values. And again after completion of an iteration of outer loop, the L,U values calculated are used in the next iteration. We already know that all processes have correct L values. Same as the 1st inner loop, in the 2nd inner loop only some U values are calculated by a process. So each process can have different U values after completion of iteration. But we have used MPI broadcast to send calculated U values to all processes. It ensures that all processes have correct U values at the end of iteration.