Report COL380 A2

Shubham(2018CS10641), Suyash Singh(2018CS50646)

April 2021

1 Strategy 0

Strategy 0 is the sequential program given to us.

2 Strategy 1

Strategy 1 is developed by using 'parallel for' construct with other appropriate clauses. Making the diagonal entries of U is done using, *"pragma omp parallel for"*, and setting number of threads using "$omp\_set\_num\_threads(nthreads)$" before it. Thus the first loop will be executed parallely by "$nthreads$" threads. Same thing is done in nested loops. *"pragma omp critical"* is used to protect data race condition because it allow only one thread to execute the enclosed code block under *"pragma omp critical"*. *"pragma omp barrier"* is used for synchronization at the end.

3 Strategy 2

Strategy 2 is developed by using '$sections'$ construct with other appropriate clauses. In *'pragma omp parallel sections'* the construct '$parallel'$ creates a set of threads and the construct '$sections'$ distributes multiple sections between existing threads. Making the diagonal entries of U is done by using eight sections using *'pragma omp section'* which are executed according to the number of threads. In each of the 2 nested for loops, 4 sections are used. If the number of threads are greater than than the sections some of the threads will remain idle else if the number of threads are smaller than the number of sections than some of threads will execute multiple sections and there exists no order of execution of threads. By default, there is a barrier at the end of the each section to prevent data races. *"pragma omp critical"* is used while doing *"sum = sum + x"* also, for data race.

4 Strategy 3

Strategy 3 is developed by using both *'parallel for'* and *'sections'* constructs with other appropriate clauses .

Making the diagonal entries of U is done using, *"pragma omp parallel for"*, and setting number of threads using "$omp\_set\_num\_threads(nthreads)$" before it. Thus the first loop will be executed parallely by "$nthreads$" threads.

For the for loops, two sections are created. Each for loop is enclosed in a separate section and *pragma omp parallel for num_threads(nthreads/2)* is used inside each section. $nthreads/2$ threads are given to one section and remaining $nthreads - nthreads/2$ threads are given to other section. By default, there is a barrier at the end of the each section to prevent data races.

In each loop, there is a separate local variable sum, and there is no data dependency in each thread, so no data races in the code.

5 Strategy 4

Strategy 4 we divide the loop into 'n' chunks, where n = *number of processes*. After dividing we make a call to *MPI broadcast* which ensures that the computed value is updated to all the processes.This *MPI broadcast* is thread-safe i.e, this routine can be used by multiple threads without the need for any user-provided thread locks.