PARALLEL PROGRAMMING ASSIGNMENT 1

Mohit Reddy (14CO118), Guru Pradeep Reddy (14CO246)

Note : The Speed up formula used is the same as the one given in the assignment (For clarity while interpreting the results) = Tn/T1.

Q1) Hello World Program

Normal version : Here each thread will print the hello world separately.
New approach : Printf by each thread may cause I/O overhead, to avoid that we concatenate 'Hello world' to a shared variable. The shared variable is protected using critical section, in this case atomic won't work because string is not a native type. By this approach we can reduce the I/O overhead.

Q2) Hello World Program - Version 2
Approach : Here each thread will print "Hello world" along with the thread number, it is accomplished by implementing a simple function which prints "Hello World" by taking the thread id.

Q3) DAXPY Loop

Normal approach : Here we can simply make the for loop parallel using #pragma for construct.

New approach : The above approach can be improved by Unrolling the loop to do maximum work in one iteration. This will save few thread operations and it reduces compiler work.

No of threads that give Max Speed up : In ideal scenario no_of_cores + 1, should be ideal number of threads that give max speed up, because 1 thread can do I/O operation while rest are doing some task.

When we increase the number of threads beyond this point, performance will decrease. This is because of the context switching. As only only four threads can work at a time on four core machine, if we create more numbers of threads then we have to suspend one thread execution to accommodate new one, this involves saving state of process which is costly, So it will

degrade performance even though the amount of work to be performed by each thread decreases.

Q4) Matrix Multiplication

New approach : Here we make the outer for loop parallel and we unroll the outer loop such that 4 rows are covered in each iteration.

Q5) Calculation of PI
Approach insight : Normal calculation of PI can be calculated by sharing the calculation of area part between the threads, one bottleneck here will be the amount of work in critical section. So ideally we prefer the critical section to have the least amount of work. Our approach does this by keeping only one operation in critical section(i.e is also +) as we calculate the sum of the area covered by each thread separately

Shared Variables : PI (final required value), step_count (No of steps), step (Size of each step along x direction)
Private Variables : X (Co-ordinate of centre of rectangle in x direction), sum (Area to be calculated by each thread), i (for loop iterator)

Q6) Calculation of PI - Worksharing and Reduction

In the previous Implementation of the PI, we have to manually create critical section to calculate the PI value. This process can be simplified by using reduction clause, which reduces the sum using '+' operation, so the final output which we get after this loop will be total sum computed by all threads.

Q7) PI calculation using - Monte Carlo Simulation
Linear Congruential generator was used to generate the random numbers. Values of modulus, multiplier, increment are 2^31 - 1, 1103515245, 12345 (used by glibc aka gcc).

Two challenges faced while solving this problem are :

1) Making the random number generator thread safe : This is solved by making the random_last as thread private so it makes the generator thread safe.
2) Making random number generator numerically correct : If we select the seeds randomly there and generate large number of random numbers using each thread. There is high chance that some random numbers generated overlap. To solve this problem we used Leap Frog algorithm to make the random numbers generated distinct making use of linear congruential generator.

Q8) Producer-Consumer Problem

It is implemented by using Sections. Producer and Consumer will be in different sections and are executed by two different threads. A flag variable is used to notify Consumer when it can perform this operation. For this Consumer infinitely checks if that variable is set. To get the most recent value we use flush operation and atomic read and atomic write to provide the synchronisation.