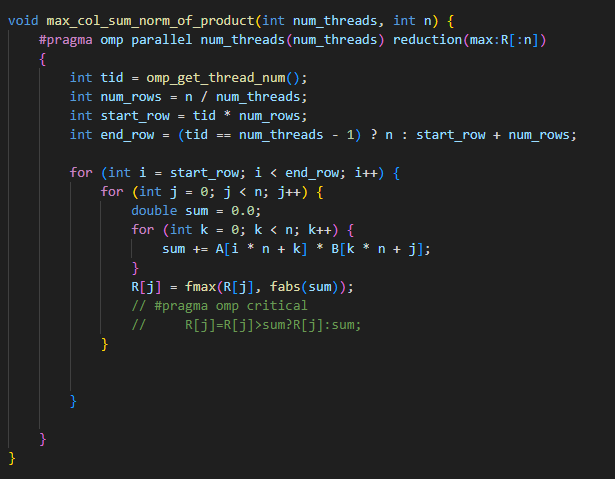
**ASSIGNMENT 4**

**Variants 1b,2a,3a**

The assignment is about using open while doing matrix multiplication and finding the maximum column sum norm of the resultant matrix

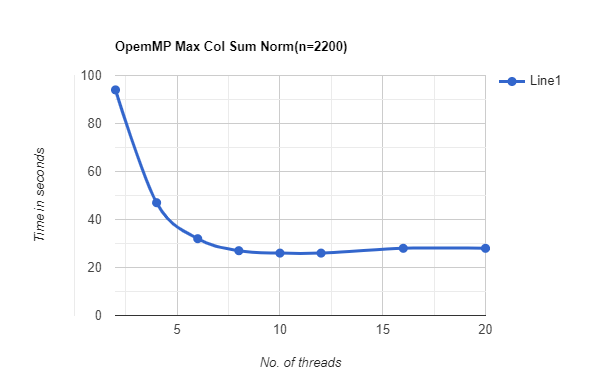


The core of the program is in the above function. It can be summarized as

* The max\_col\_sum\_norm\_of\_product is called from the main application thread..
* All the parameters are declared globally(The input matrices and the resultant max col sum norm array). Regardless of this, each thread makes its own local copies of the respective matrices
* This function takes 2 params - the number of threads and matrix dimension
* The whole function is made a parallel region
* The pragma directive is
  + #pragma omp parallel num\_threads(num\_threads) reduction(max:R[:n])
  + We specify the number of threads. Each thread gets a set of rows from the fist matrix**(n/num threads)**. The remaining number of rows get allocated to the last thread.
  + A reduction clause is applied for the entire result array. This reduction clause is applied across all local copies of R in multiple threads
  + Within the parallel region, for each row the thread is allotted from the first matrix, it is multiplied with all the columns of the second matrix. For each cell value generated, max col sum norm is determined by comparison with result array R
  + At the end of each parallel region reduction is applied across threads on R by OpenMP

**EXPERIMENTATION**

1. **N=2048**

****

Naive implementation time ~ **210 seconds**

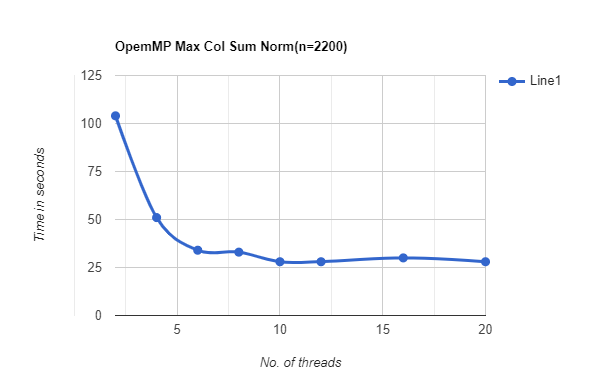
Even with just 2 threads the time taken is halved. Peak performance is observed at around 10-12 with ~26 seconds. With further increase in the number of threads, eventual saturation is reached.

1. **N=2200**

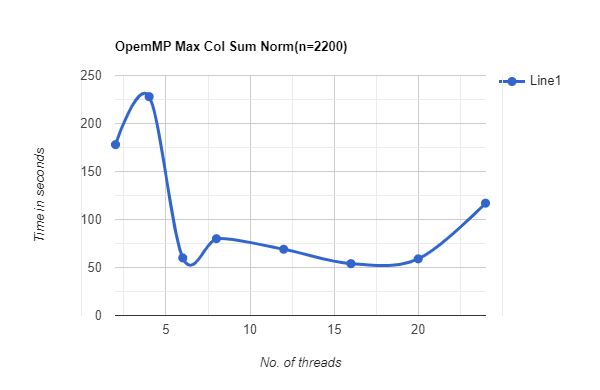
Single threaded implementation time ~ 219

Maximum performance observed at about 10 threads ~ **28 seconds**

There is a sharp increase in performance up until 10 threads beyond which the execution time remains the same.



1. **N = 2500**

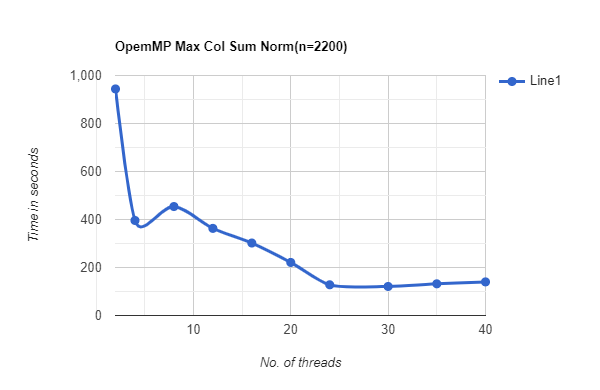
****

Single threaded performance ~ 337 seconds

OpenMP peak performance ~ **54 seconds.**

Best performance is observed around 15 - 20 threads. Around **5-6 times the performance is observed.**

1. **N = 3000**

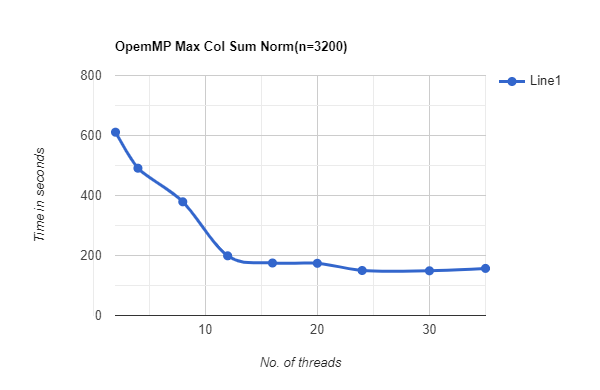


Single threaded implementation execution time ~ 681

Peak execution time using OpenMP ~ **121sec**

Best performance is observed with around 25-30 threads with the fastest time recorded at 121 seconds. **That is almost 6 times the performance compared to a single threaded implementation**

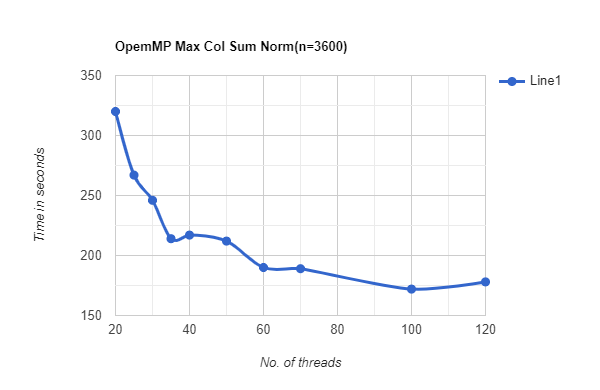
1. **N = 3200**



Single threaded execution time ~ 700 sec

Max OpenMP execution time ~ **149 sec**

OpenMP speeds up the execution time by almost 5 times as compared to its serial counterpart. The performance gradually saturated beyond 30 threads.

1. **N = 3600**

Serial execution time ~ 1176 sec

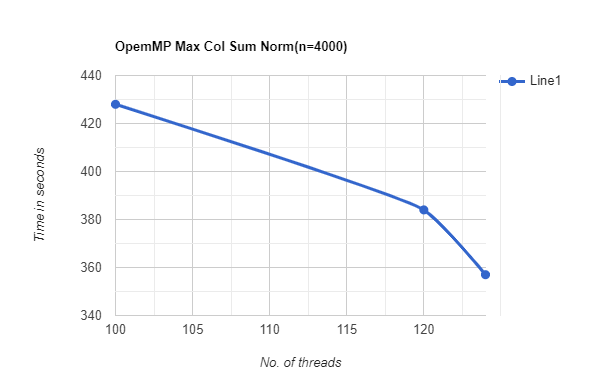
OpenMP execution time ~ 170 sec

With the increase in matrix size there is a need for much larger number of threads. With ~ 120 threads this gives a peak performance of 170 seconds almost 7 times better than the serial one.

1. **N = 4096**

Serial execution time ~ 2500 sec

Peak OpenMP execution time ~ 350 seconds

Speedup ~ **7 times better performance.**

**CONCLUSION**

OpenMP significantly improves the performance compared to its serial counterparts. With smaller matrix sizes an eventual saturation is reached beyond a certain number of threads. However for larger matrices a minimum of ~120 threads is needed to achieve peak performance.

The peak performance with OpenMP is about 8 times faster. On an average it is observed to give about 5-6 times better performance.