Midterm - CS671

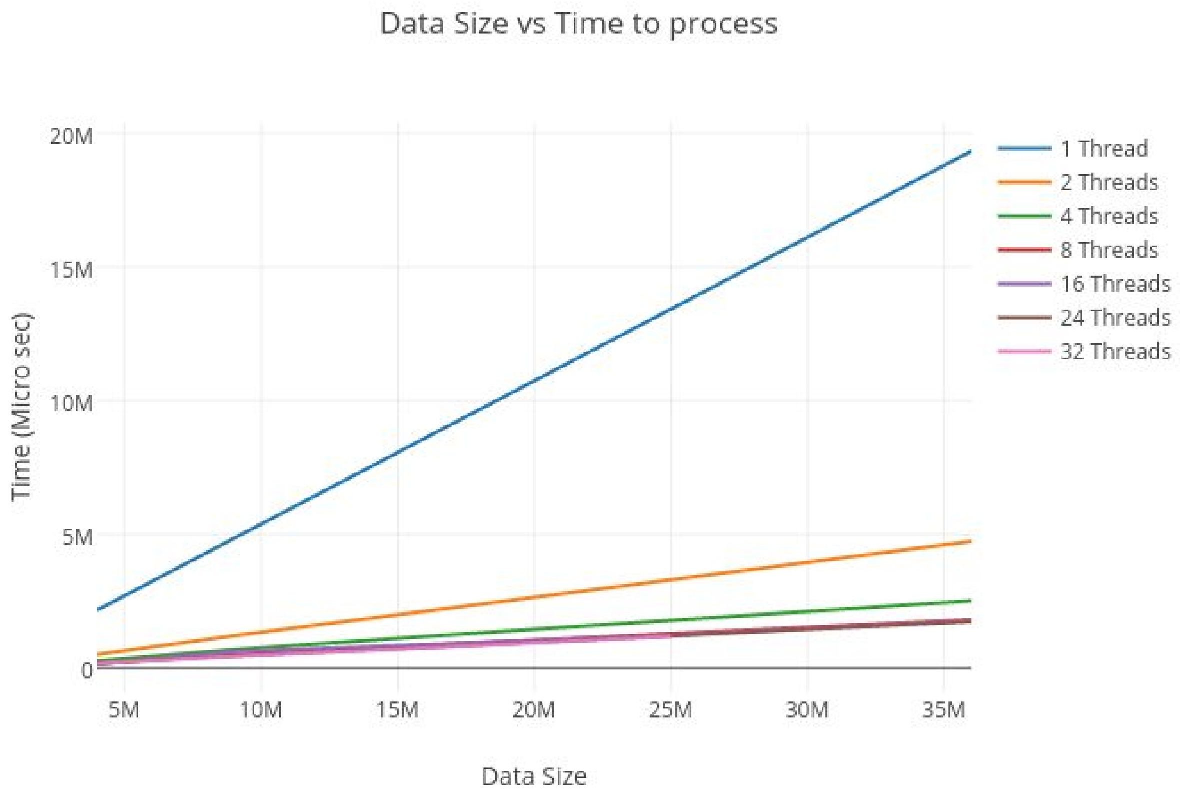
Midterm

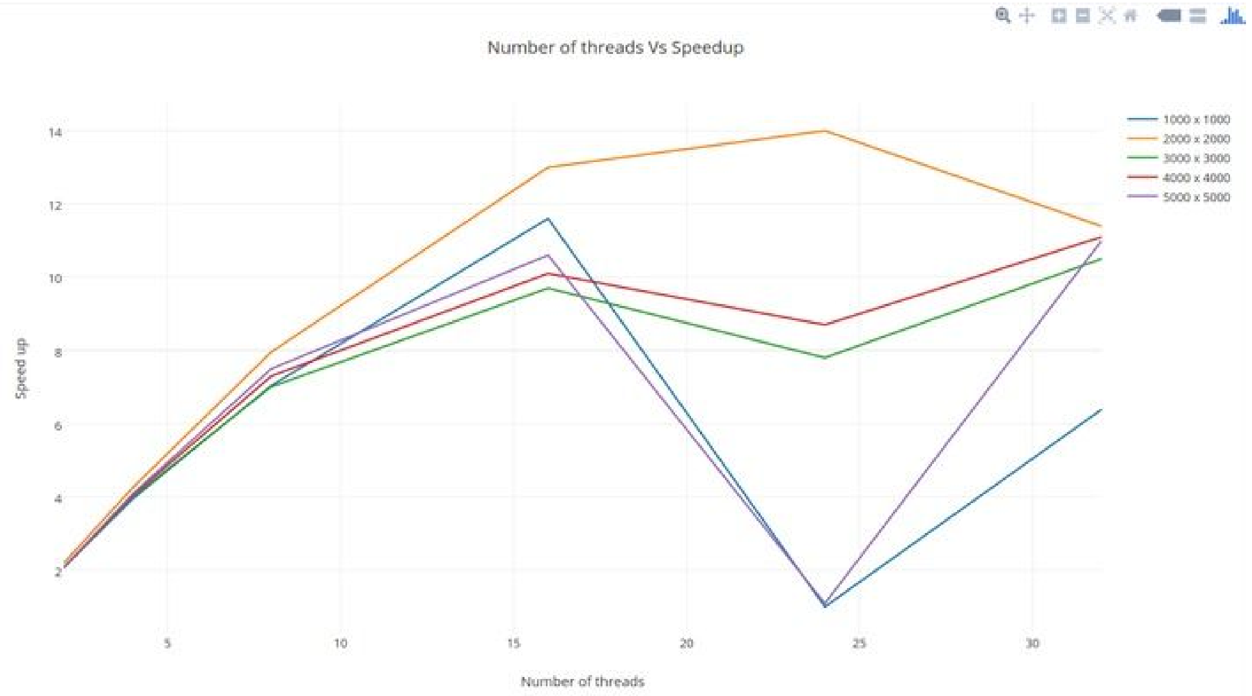
- Ashish Jindal (AJ523)

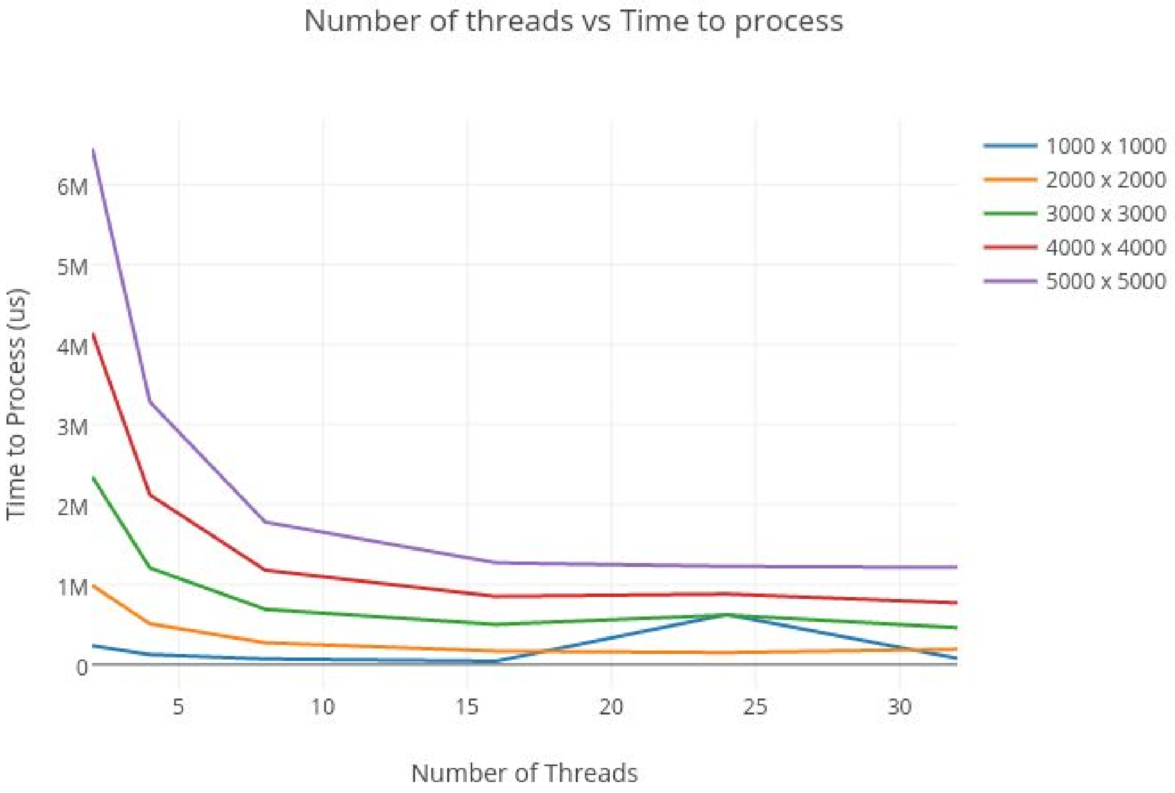
Q 1 -

1. Code - ./main.c - Serial implementation using OpenMP (Working and tested)
2. Code - ./cuda/solver.c - Cuda stencil based implementation (Not tested)
3. Script to run on elf and generate stats - ./run.sh
4. Data - stats*.csv (Check main.c for data interpretation)
5. Input format - <serial - 0 vs openmp - 1> <iterations> <N><Num threads>
6. Output Format - <NxN><iterations><Time><Error if any><serial/openMP>

Q 2 -




Number of threads Vs Speedup
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
Speed up
2
4
6
8
10
12
14
5
10
15
20
25
30
Number of threads


Number of threads vs Time to process
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
Time to Process (us)
0
1M
2M
3M
4M
5M
6M
5
10
15
20
25
30
Number of Threads

Q 3 -

The idea of algorithm is to determine convergence of the matrix after some N iterations. In the serial implementation we are updating the array value as soon as a new value is computed but this methodology doesn’t sit in a good way with parallel implementation as it creates lots of dependencies among the cell values.

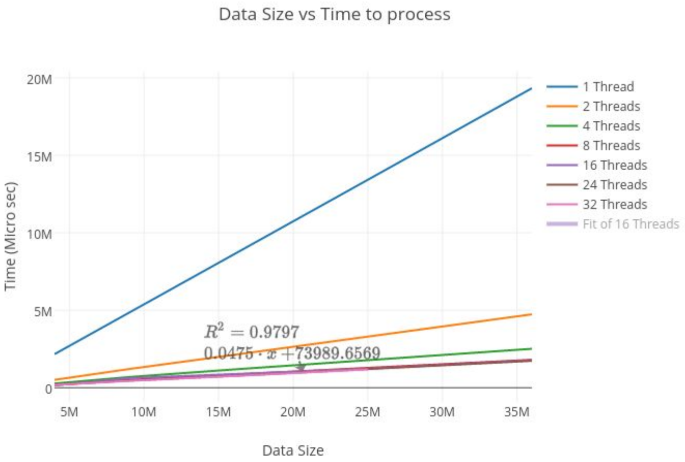
OpenMP Implementation -

- In my algorithms, I take an auxiliary array to support parallel reads without taking any locks.
- Before starting the computation we copy the value from original array to this auxiliary array, in parallel using openMP
- Then we start the openMP threads to do computation in parallel while reading from the previously computed auxiliary array and writing to the original array.
- As we aren't modifying the aux array, so the reads are safe.
- For the next iteration we again update the original array by copying the aux array value to it via openMP.

CUDA implementation -

- The CUDA implementation is also similar to the one described above.
- We maintain a grid of threads on the GPU of size tileSize x blockSize
- Also we have two device memory addresses one for the original computation and other for the auxiliary purpose (parallel reads)
- We read and write to both the memory location alternately for e.g for iteration 0 (even) we read from aux array and write to original array and for iteration 1(odd) we read from original array and write to aux array.
- When the computation is finished we copy one of the arrays from device to host.

Using the following curve -



We can develop a model of following type -

$T_{Comm} = \alpha + \beta M$

1. T_{Comm}= Processing time
2. α = latency
3. β = inverse bandwidth
4. M = Data size

From the resulting graphs we can see that for 16 threads, the values of α and β are as follows -

α = 73989.6589 us
β = 0.0475 us