# Assignment 1

27-02-2021


Naman Jhunjhunwala

**2017MT10737**


**COL380**

# Strategies

In this I have implemented two strategies which are following

1. **Strategy 0** - Divide the work of calculating the sum in p threads and finally accumulate the result
2. **Strategy 1** - Using divide and conquer , At each level divide elements in pair and accumulate the result to find the entries of the next level

# Results

- To compute the execution time of the program I have taken average of 40 readings,
- In all strategies I got sum = 500500, 5000050000, 50000005000000 for N = 1000, 100000, 10000000 respectively

1. Serial version:

| N | Time in seconds |
|---|---|
| 1000 | 0.000009145 |
| 100000 | 0.000286739 |
| 10000000 | 0.0206957 |

2. Strategy 0:

N = 1000

| No. of Threads | Time in seconds | Speedup = Tserial/Tparallel |
|---|---|---|
| 2 | 0.00002468 | 0.370543 |
| 4 | 0.00063609 | 0.014377 |
| 8 | 000135799 | 0.067342 |

N = 100000

| No. of Threads | Time in seconds | Speedup |
|---|---|---|
| 2 | 0.000191914 | 1.494102 |

| | | |
|---|---|---|
| 4 | 0.000597539 | 0.479867 |
| 8 | 0.000312337 | 0.918044 |

N = 10000000

| No. of Threads | Time in seconds | Speedup |
|---|---|---|
| 2 | 0.0133308 | 1.552472 |
| 4 | 0.0117622 | 1.759509 |
| 8 | 0.0112195 | 1.844619 |

3. Strategy 1:

N = 1000

| No. of Threads | Time in seconds | Speedup |
|---|---|---|
| 2 | 0.000039431 | 0.231924 |
| 4 | 0.000068332 | 0.133832 |
| 8 | 0.000718653 | 0.012725 |

N = 100000

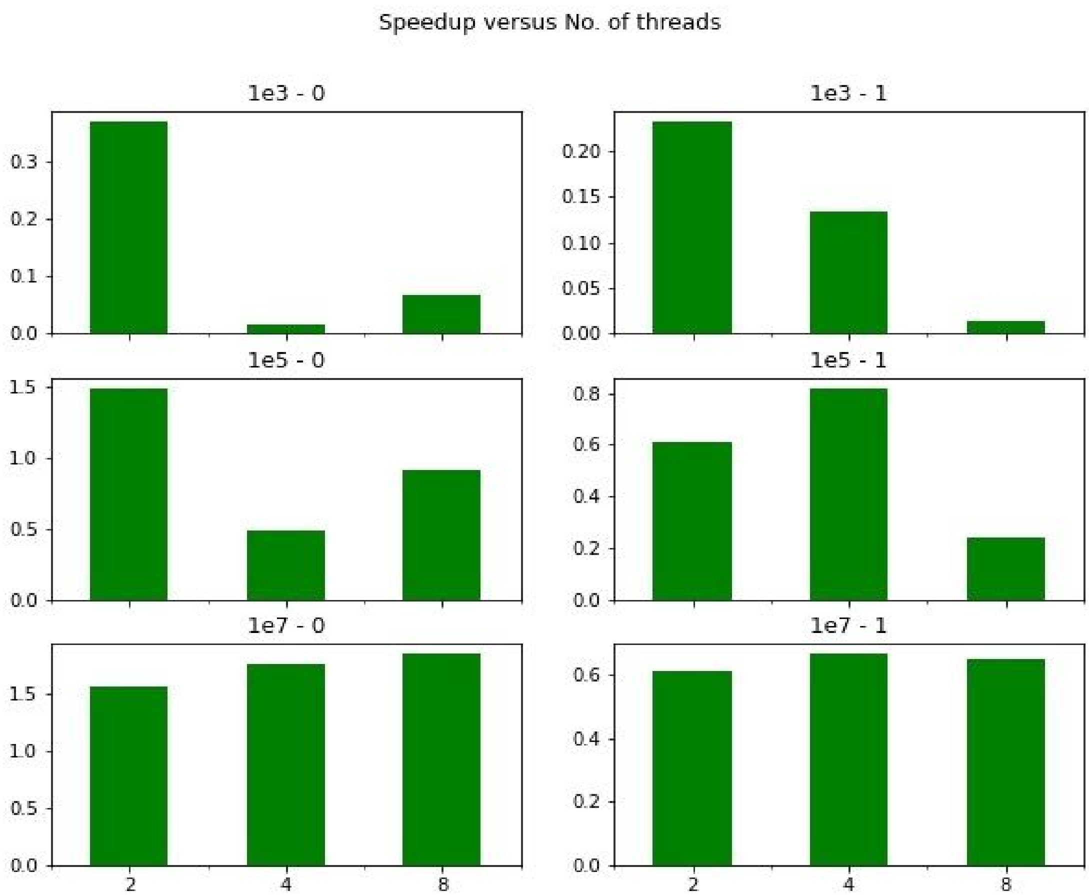
| No. of Threads | Time in seconds | Speedup |
|---|---|---|
| 2 | 0.046876 | 0.611697 |
| 4 | 0.000349572 | 0.820257 |
| 8 | 0.00119756 | 0.239436 |

N = 10000000

| No. of Threads | Time in seconds | Speedup |
|---|---|---|
| 2 | 0.0338894 | 0.610684 |
| 4 | 0.0311918 | 0.663498 |
| 8 | 0.0320108 | 0.646522 |

# Graph:

X -axis and Yaxis contains number of threads and total time taken respectively




Observations/ Analysis:

1. Speed up is greater than 1 for the strategy 0 for N = 10^7 indicating that the overhead is minimal in comparison to the time saved by parallelising the program
2. We see strategy 1 is fairly slow even though it is parallelised version of the input program, It can be due to the following reasons:
    - Overhead of creating a new array to store the temporary sum.
    - This strategy parallelizes the program for each level of the tree so it creates thread more than once ( ~O(logn)), Hence overhead of parallelisation is more in this case
3. Speedup for small N is less than 1 because overhead of parallelisation is comparable to the execution time of the program

4. Strategy 0 is better than strategy 1 because its speed up is more for large inputs

## Amdahl's law:

Amdahl's law states that speed up of a program is upper bounded by (1/(1-f)) where f is the fraction of serial part in the program, Hence the speed up saturates as we increase number of threads in the process even if we have infinite cores

In strategy 0 we computed the solution by first finding the partial sums (Parallely) and then accumulating the result (serially), This serial part acts as a bottleneck for our problem , We can see from the bar graph of strategy 0 for any N speedup does not increases at same pace (For N = 10^3 ad 10^5 due to comparable overhead of parallelisation speed up has decreased for number of threads = 2)

In Strategy1 we parallelized the program for each level but a thread has to wait till all the entries in one level is completed , So levels are calculated serially but entries of each level are calculated parallely , We can see the speedup versus number of threads graph of strategy 1 and conclude that speed up does not increases as we increase number of threads which is in sync with the Amdahl's law