# Raghav Pandya

# Net id: rp835

# Assignment 1

Problem 1:

A) I) Speedup (n/p) + log (p) ) / ( (n/qp) + log (qp) ) - Old time / New Time

No, because since processors have different speeds, overall expression does not hold

II) Efficiency = Speedup/ N

Speedup = (Serial Time) / Parallel Time

Efficiency = n / ( n + p (log p ) )

If we increase the processors by a factor of k, we need to increase datasize by factor of P

i.e. n /( n + p (log p) ) = P \* n / (P \* n ) + ( kp log (kp) )

P = k \* log(kp) / log (p)

III) Yes it is strongly scalable as efficiency remains constant and does not depend upon number og threads

B) I)

Please find the program attached as a file.

Time taken by sequential merge sort for 50 elements is: 0.000019 seconds

Time taken by sequential merge sort for 500 elements is: 0.000110 seconds

Time taken by sequential merge sort for 5000 elements is: 0.002838 seconds

Time taken by sequential merge sort for 500000 elements is: 0.126275 seconds

II) Assumptions: We assume number of threads is a even number and number of elements in the array should be a multiple of the number of threads to reduce complexity in coding and better understanding

1. First divide the array into n partitions where n = number of threads

2. Sort each block with sequential merge sort function with each thread in parallel

3. As a result of previous step we have, n sorted blocks , now we have to merge the adjacent blocks based on number of threads.

For example,

if n = 2, number of merges = 1; using single thread

if n = 4, number of merges = 3; we use 2 threads to merge 2 pairs each initially and then

1 thread to merge the resultant 2 long pairs, and the sequence goes on

III) Please find attached, Program must be compiled with this command:

gcc -g parallel.c -lpthread -lm -std=c99

IV) The graph below shows the variation with increasing data size and number of threads for both sequential and parallel implementation. Also parallel code is not always faster as we can see the, the cost to create individual threads sometimes exceeds the benefits of running in parallel. Also for big datasets, time is equivalent as sequential nad no benefits are seen.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SIZE** | **SEQUENTIAL** | **THREAD = 2** | **THREAD = 4** | **THREAD = 8** |
| 64 | 11 | 0.000260 | 0.000282 | 0.000584 |
| 3200 | 1575 | 0.00270 | 0.001556 | 0.004196 |
| 6400 | 4567 | 0.000654 | 0.001496 | 0.003862 |
| 64000 | 14416 | 0.016895 | 0.029454 | 0.030084 |
| 640000 | 157634 | 0.180871 | 0.223996 | 0.205358 |
| 1000000 | 236809 | 0.240831 | 0.359618 | 0.326378 |

**Parallel vs Sequential merge sort in clock ticks**

V) No it does not guarantee that one thread would run on one core because a single managed thread would make use of multiple OS threads to support it.

Hyperthreading can improve the runtime for parallel merge sort algorithm as different threads can run on different logical cores as they work on independent dataset.