Amdahl's Law – Math of Multi Threading

- **Speedup enhanced** –improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program
- If the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. --- Speedup enhanced.
- *Speedup Enhanced is always greater than 1.*
- Moore's Law v/s Amdhal's Law v/s Niklaus Writh's law

- T(n) represent the time a program takes to execute with n processors
- speedup computed by Amdahl's law is a comparison between T(1), the time on a uniprocessor, and T(n), the time on a multiprocessor with n processors.
- **S(n) = T(1) / T(n)**
- . If s is the serial fraction of T(1), then it takes **[(1-s)*T(1)]/n]** units of time to run the parallel part and (s) ·T(1) for the serial part

T(n) = T(1) { s + (1-s)/n }

S(n) = T(1) / T(n)

= T(1) / T(1) { s + (1-s)/n }

= 1 / {s+(1-s)/n}

- Assume serial component of a task is 25% and the setup is parallelized with N=2.
- Then **Speed up = 1 / S+(1-S) / N**

= 1 / {.25 + .75 / 2} = 1.6

Speed Up is 1.6 times becoz of Multithreading!!

Same setup with N=4 cores wud imply a speed up of 2.28

Theoretically N can reach infinity

Speed up = 1 / S ; assume S =40% ; speed up is 2.5 times

Serial component of a task will always impact the overall speedup despite the high number of processors available

Quiz Time

- Assume an application is run on a 64-processor machine and that 80 per cent of the application can be parallelized. Compute the expected performance improvement
- Assume a cluster computing setup with 500 processors, what must the serial and parallel component be to obtain an efficiency of 90%
- Given an algorithm where the fraction that the program is serial (f) is .15, what is the speedup on 2, 4, 8, 16 and an infinite number of processors.

Threading – General Discussion

- Despite feel of infinite parallelization system will always have a limited number of kernel threads mapped to user threads request
- Thread creation , initialization has associated overhead
- Initially n threads are created at process start up **{Thread Pool}**
- Threads are allocated on user (process) requests
- **On completion** threads return to Thread Pool for more user requests
- No Kernel Thread available ; User Thread will have to wait
- Avoids multiple thread initialization and memory overhead involved operations
- Memory leak as a result of infinite thread creation is avoided!