Class Activity

(a)

Ali is given the job of writing a program that will get a speedup of 19 on 20 processors. He makes the program 95% parallel and goes home dreaming of a big pay raise. Assuming the program will always be run with the same problem size regardless of the number of processors or cores used, and ignoring communication costs, what speedup will Ali actually get? Show your work.

(b)

For Ali's program in (a), assuming the problem size can expand as the number of processors increases, what is the expected speedup on 20 processors? Show your work.

(c)

The table below shows the performance measurement of a parallel program. Is this program likely to achieve a speedup of 10 on 12 processors?

***N* (no. of processors)**	**4**	**8**
Ψ	3.9	6.5

(d)

A parallel algorithm is used to sum n numbers on p processors. The algorithm works in two phases; (1) each processor sums n/p elements locally, and (2) the local sums are combined using a binary tree reduction (with $\log p$ steps).

(i) Derive the parallel runtime T_p as a function of n and p.
(ii) Using the expression for T_p, derive the parallel overhead T_o.
(iii) Determine the isoefficiency function of the algorithm, clearly stating what it tells you about the scalability of the algorithm.
(iv) Based on your isoefficiency result, rate the scalability of this algorithm as ***Poor***, ***Moderate***, or ***Good***. Justify your answer.

Solution:

(a)

$$Speedup \leq \frac{1}{(0.05) + 0.95/20} \approx 10.26$$

Ali will get a speedup of approximately 10.26, not 19. Ali overestimated the potential speedup by not applying Amdahl's Law — even a small serial fraction (5%) limits the maximum achievable speedup, no matter how many processors you add.

(b)

$$Speedup \leq N + (1 - N)S$$
$$Speedup \leq 20 + (1 - 20)(0.05)$$
$$Speedup = 19.05$$

(c)

The experimentally determined serial fraction e should be at least 6.5 (that of p = 0.033).

$$0.033 = \frac{1/\Psi - 1/12}{1 - 1/12}$$

$$\Psi = 8.8$$

No, this program is unlikely to achieve a speedup of 10 on 12 processors. The Karp-Flatt metric shows diminishing returns due to increasing serial fraction

(d)

Derive the parallel runtime $\boldsymbol{T_p}$ as a function of $\boldsymbol{n}$ and $\boldsymbol{p}$.

Local sum on each processor ➔ $T_{local} = \Theta(\boldsymbol{n/p})$
Reduction of p partial sums using binary tree ➔ $T_{reduce} = \Theta(\boldsymbol{log\ p})$
Hence $T_P = \Theta(\frac{\boldsymbol{n}}{\boldsymbol{p}} + \boldsymbol{log\ p})$

Using the expression for $\boldsymbol{T_p}$, derive the parallel overhead $\boldsymbol{T_o}$.

Sequential time for adding n numbers ➔ $T_1 = \Theta(\boldsymbol{n})$
Parallel overhead ➔ $T_0 =$ p·$T_p - T_1$

$T_0 =$ p·$(\frac{\boldsymbol{n}}{\boldsymbol{p}} + \boldsymbol{log\ p}) - n = \Theta(\boldsymbol{p\ log\ p})$

Determine the isoefficiency function of the algorithm, clearly stating what it tells you about the scalability of the algorithm.

To maintain constant efficiency: $W \geq \Theta(\boldsymbol{T_0(N)})$ (W = T_1)

$$T_1 \geq \Theta(\boldsymbol{T_0(N)})$$
$$n \geq \Theta(\boldsymbol{p\ log\ p})$$

To maintain efficiency while increasing p, the **problem size n must grow proportionally to p log p**.

Based on your isoefficiency result, rate the scalability of this algorithm as ***Poor***, ***Moderate***, or ***Good***. Justify your answer.

Moderate, because as p grows large, communication overhead log p starts to dominate unless problem size grows quickly.