

Q.  $p$  processors takes  $(\log n)^2$  time

$$\text{Serial time} = n \log n$$

Speed up =  $\frac{n}{\log n}$

Efficiency =  $\frac{1}{\log n}$

Obs  $\rightarrow$  Efficiency decreases as problem size increases

Cost complexity  $\rightarrow n(\log n)^2$

Obs  $\rightarrow$  Not cost optimal by a factor of  $\log n$

- If  $p \ll n$

$$\text{parallel time} = \frac{n(\log n)^2}{p}$$

Speed up =  $\frac{p}{\log n}$

Efficiency =  $\frac{1}{\log n}$

(Speedup decreases as problem size increases)

Algorithm (Sorting)	$A_1$	$A_2$	$A_3$	$A_4$
Number of processors	$n^2$	$\log n$	$n$	$\sqrt{n}$
$T_p$	1	$n$	$\sqrt{n}$	$\sqrt{n} \log n$

Given serial time =  $n \log n$

Speedup	$n \log n$	$\log n$	$\sqrt{n} \log n$	$\sqrt{n}$
Efficiency	$\log n / n$	1	$\log n / \sqrt{n}$	1
Cost	$n^2$	$n \log n$	$n \sqrt{n}$	$n \log n$

Both  $A_2$  and  $A_4$  are good; based on the number of processors available choose either of ~~two~~  $A_2$  or  $A_4$

Q1. Parallel program on 2 processor system.

Problem size =  $W$

Serial  $\rightarrow$  1 processor cache - 64 KB, Cache hit rate - 80%  
\* Latency of cache - 2 ns, latency of DRAM  $\rightarrow$  100 ns  
\* Memory bound computation  $\rightarrow$  1 FLOP / memory access

Parallel \* Each processor does  $W/2$   
\* Cache hit rate = 90%, 8% from RAM, 2% from remote DRAM (communication overhead)  $\rightarrow$  latency of 400 ns

Comment on expected speedup.

Ans In serial  $\rightarrow$   $W$  memory access

$$\left[ \frac{80}{100} \times 2 + \frac{20}{100} \times 100 \right] = 21.6 \text{ ns}$$
$$\text{Performance} = \frac{1}{21.6} = 46.3 \text{ M FLOP}$$

In parallel  $\rightarrow$   $W/2$  memory access

$$\left[ \frac{90}{100} \times 2 + \frac{8}{100} \times 100 + \frac{2}{100} \times 400 \right] = 17.8 \text{ ns}$$

$$= \frac{1}{17.8} = \text{Performance per processor} = 56.18 \text{ MFLOP}$$

$$\text{Thus speedup} = \frac{46.3}{56.18} = 0.824$$

Q2. Image Processing (Convolution)

\*  $n \times n$  fixed image

\* Convolution with  $3 \times 3$  mask.

\* Each multiply add takes ' $t_m$ ' time

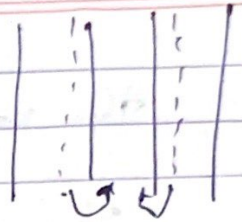
\* Distributed system: Each node has its subimage and applies the mask on the subimage.

\* Each pixel  $\rightarrow$  One word communication

\* Communication latency  $\rightarrow t_s + t_w$



where  $t_s$  = startup time  
 $t_w$  = per word transfer time.

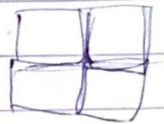


- Comment on efficiency in terms of  $n, p, t_s, t_w, t_c$ .

Ans Serial time =  $n^2 \times 9 \times t_c = 9n^2 t_c$

Parallel time =  $\frac{9n^2 t_c}{p} + \left[ \frac{t_s + t_w}{p} \right] n^2 ((t_w \cdot n + t_s) 2)$

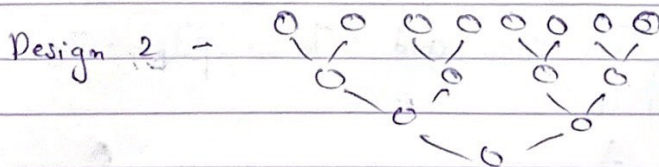
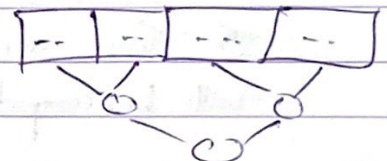
Thus efficiency =  $\frac{T_s}{T_p} \times \frac{1}{p}$



### Effect of granularity on performance.

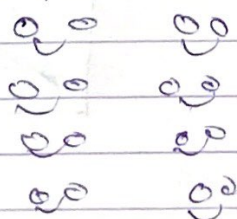
- Adding  $n$  number on  $p$  processor  
 $n = 16, p = 4$  (powers of 2)

Design 1  $\rightarrow O\left(\frac{n}{p} + \log p\right)$   
 Cost =  $(n + p \log p)$



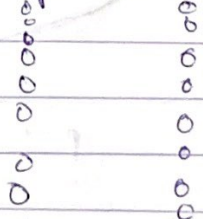
~~$O\left(\frac{n}{p} + \log p\right)$~~

4 processor working



$p \rightarrow 1 \ 2 \ 3 \ 4$

2 processor working

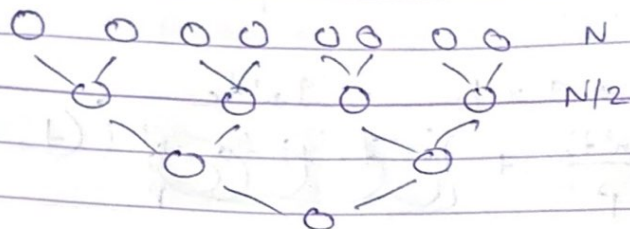


1 2 3 4

1 processor

Thus running time =  $O\left(\frac{n \log p}{p} + \frac{n}{p}\right)$

Thus cost =  $O(n \log p)$



There would be  $\log p$  steps.

Step 1  $\rightarrow n/p$

Step 2  $\rightarrow n/p$

(As <sup>half</sup> processors get deactivated after every step)

For  $\log p$  step  $\rightarrow n/p$

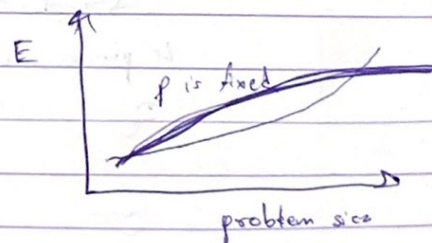
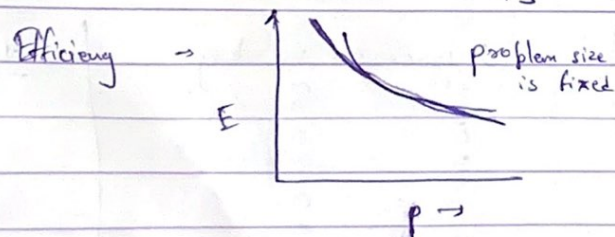
~~And for the last one~~ For  $n^{\text{th}}$  step

For  $i^{\text{th}}$  step, no of computations =  $\frac{N}{2^{i-1}} = \cancel{N/p}$

Thus after  $\log_2 p$  steps  $N/p$  computations are left which will be computed on 1 processor.

Efficiency =  $\frac{S}{p} = \frac{T_s}{pT_p}$  and  $T_o = pT_p - T_s$

$\therefore E = \frac{1}{1 + \frac{T_o}{T_s}}$





→ Iso efficiency

$$W = \frac{E}{1-E} T_0(n, p)$$

Problem size = Min number of computations in serial case

- For adding  $N$  number of  $p$  processors

$$\begin{aligned} T_0 &= pT_p - T_s \\ &= p\left(\frac{N}{p} + \log p\right) - N \\ &= p \log p \end{aligned}$$

$$\therefore W = K p \log p \quad \text{where } K = \frac{E}{1-E}$$

Q If we increase  $p$  from  $p$  to  $p'$ . Find the rate of increase in problem size

Ans Since  $K$  is constant  $\frac{W'}{W} = \frac{p' \log p'}{p \log p}$

Q If  $T_0 = p^{3/2} + p^{3/4} W^{3/4}$ . Find increase in problem size

Ans

$$W = K p^{3/4} W^{3/4} + p^{3/2}$$

$$\Rightarrow \frac{W_2}{p_2^{3/2} + p_2^{3/4} W_2^{3/4}} = \frac{W_1}{p_1^{3/2} + p_1^{3/4} W_1^{3/4}}$$

$$\Rightarrow \frac{W_2 p_1^{3/2} + p_1^{3/4} W_1^{3/4} W_2}{W_2 p_1^{3/2}} = \frac{W_1 p_2^{3/2} + W_1 p_2^{3/4} W_2^{3/4}}{W_1 p_2^{3/2}}$$

Dividing into 2 parts

$$\begin{aligned} W &= K p^{3/2} \\ \Rightarrow \frac{W_1}{W_2} &= \left(\frac{p_1}{p_2}\right)^{3/2} \end{aligned}$$

$$\begin{aligned} W &= K p^{3/4} W^{3/4} \\ \frac{W_1}{W_2} &= \left(\frac{p_1}{p_2}\right)^3 \end{aligned}$$

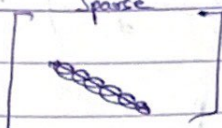
Thus problem size must be increase at  $\frac{p_1^3}{p_2^3}$  to keep  $E$  constant as it is dominant term

Q Why is iso-efficiency important?  
 Ans Iso efficiency relation connects algorithm and architecture.  
 Same algo can be highly scalable in one machine can poorly in another.


- For unscalable algorithm, iso-efficiency is not defined.

- For sparse matrix  $Ax = B$

Sparse



Vector



How to improve cache hit in this scenario?  
 Ans Store it in a compressed form  $\rightarrow$  CSR (Compressed sparse row implementation)

$$\begin{bmatrix} 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Data  $\rightarrow \{ 3, 1, 2, 4, 1, 1, 1 \}$

Col  $\rightarrow \{ 0, 2, 1, 2, 3, 0, 3 \}$

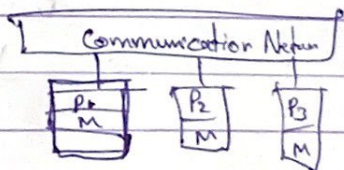
Row pointers  $\rightarrow \{ 0, 2, 2, 5, 7 \}$

Number of rows = Size of Row pointers  
 $= 5 - 1 = 4$

MPI :- Can support both SPMD and MPMD

- In distributed system, processor can access only local memory. Remote memory access requires explicit communication.
- Distributed systems are NUMA
- Cache Coherency :- If a processor modifies data in cache then all other processor's cache too must be updated.

- Number of CPUs are static.
- Message passing programs are often written using asynchronous / loose synchronous.





Problem with send / receive

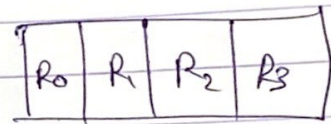
Without use of blocks, value can't be guaranteed to be what is intended.  
If it is block, can wait for lot of time.

- Non buffered Blocking  $\rightarrow$  Idle and Deadlocks can happen
  - Buffered  $\rightarrow$  Removes idling time at the cost of copy operation.
- Limitation of Buffer

- Non Blocking Message Passing

- They are accompanied by a check status operation.
- Since the process are not blocked, it is unsafe to read data being received from receiver side. ~~(Do not update the thing)~~

- In MPI, ~~rank~~ <sup>made</sup> ~~used~~ to data access is ~~used~~ according to rank.  
Data partition is made based on the rank.  
$$\frac{\text{Rank} \times \text{Total Size}}{\text{Number of processes}} + \text{offset}$$



- Communicator: - Group of processes that communicated with one another.
- By Default a process ~~is~~ is part of MPI - comm world.
- A process can be part of multiple communicators.

- Point to Point Communication

- Destination / Source
- Size of the message
- Type of the message
- Protocol (Blocking / Non Blocking)
- Status
- Tag (Sequence Number)

- MPI - Integration using trapezoidal

Step 1 :- Division of work  $\rightarrow N/p$  to each

Step 2 :- for start to end, do work

Step 3 :- If rank = 0, ~~recv~~ else send computed.

- Primary reason of using non blocking  $\rightarrow$  <sup>Do</sup> Computation and Communication parallelly