Efficiency = $\frac{T_s}{T_p} \times \frac{1}{P}$ (how many processors involved in whole process.)
{ amount of processor utilization.

- Bounds of efficiency are the upper and lower bound.

$0 \leq \varepsilon(n,p) \leq 1$.

$$\varepsilon(n,p) \leq \left[\frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p} + k(n,p)}\right] \times \frac{1}{p}$$

$$\leq \boxed{\frac{\sigma(n) + \phi(n)}{p\sigma(n) + \phi(n) + pk(n,p)}}$$

we want an ideal situation where $\sigma(n) = 0$ and $k(n,p) = 0$.

$\varepsilon(n,p) = 1$ (upper bound) , best case

worst case: everything in parallel.

$p \to \infty$, $\phi(n) = 0$.

$\varepsilon(n,p) = 0$ (lower bound) , worst case.

$$\psi(n,p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p} + k(n,p)}$$

$$\Rightarrow \psi(n,p) << \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p}}$$

let us define a function:

$f = \frac{\sigma(n)}{\sigma(n) + \phi(n)}$ (amount of serial part in the code.)

$$\psi(n,p) \leq\leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p}}$$

$f\sigma(n) - \sigma(n) = f\phi(n) - f\phi(n)$

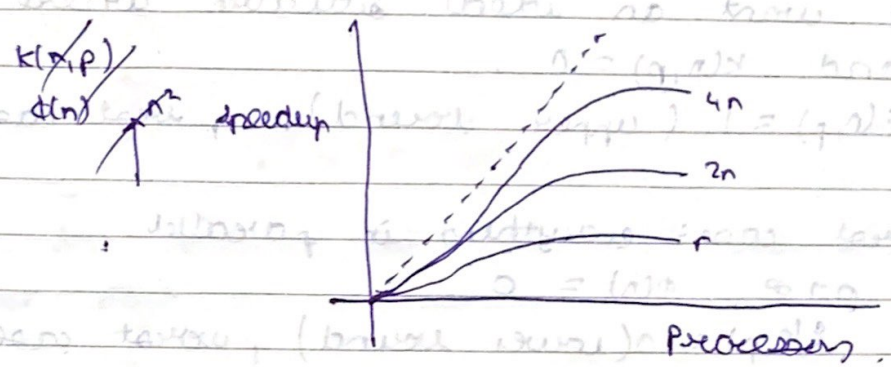
$\Rightarrow \phi(n) = \sigma(n)\left[\frac{1-f}{f}\right]$

$\Rightarrow \phi(n) = \sigma(n)\left[\frac{1}{f} - 1\right]$

$$\psi(n,p) = \frac{\sigma(n)/f}{\sigma(n) + \frac{\sigma(n)(\frac{1}{f}-1)}{p}}$$

$$= \frac{1/f}{1 + (\frac{1}{f}-1)/p} \cdot \frac{T}{f + \frac{f-1}{p}}$$

Amdahl's law will give an upper bound on speedup for a number of processors and problem size.



For a fixed number of processors, the speedup will increase as no. of processors will increase, this is Amdahl's effect (different from Amdahl's law.)

General formula:

$$\psi(n,p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p} + K(n,p)}$$

How good is the code on p processors compared to only one processor.

Given that the time is fixed :-

s → fraction of time spent in the parallel code performing serial work.

$$s = \frac{\sigma(n)}{\sigma(n) + \frac{\phi(n)}{p}} \qquad 1 - s = \frac{\phi(n)/p}{\sigma(n) + \frac{\phi(n)}{p}}$$

$$\sigma(n) = \left[\sigma(n) + \frac{\phi(n)}{p}\right] s$$

$$\phi(n) = \left[\sigma(n) + \frac{\phi(n)}{p}\right] (1-s)\, p$$

$$\psi(n,p) \leq \frac{\left[\sigma(n) + \phi(n)/p\right] s + (1-s)\, p}{\sigma(n) + \phi(n)/p}$$

$$\leq p + (1-p)\, s$$

This will show how the code will perform if it is put on p processors.

Sources of overhead :-

1) creation of threads (process setup).
2) Communication time.
3) extra computation
4) Synchronization (race conditions, etc.).

Experimentally determined serial fraction :-

$$T(n,p) = \sigma(n) + \frac{\phi(n)}{p} + k(n,p)$$

$$T(n,1) = \sigma(n) + \phi(n) \rightarrow \text{serial}$$

(idle time: $(p-1)\sigma(n)$; overhead: $pk(n,p)$)

$$e = \frac{(p-1)\sigma(n) + pk(n,p)}{(p-1)T(n,1)}$$

$$\Rightarrow e = \frac{p\sigma(n) - \sigma(n) + pk(n,p)}{*} \qquad * = pT(n,p) - \phi(n) - \sigma(n) = pT(n,p) - T(n,1)$$

Date

$$pT(n,p) = e(p-1)\,T(n,1) + T(n,1)$$

$$pT(n,p) = T(n,1)\,ep - T(n,1)\,e - eT(n,1)$$

$$T(n,p) = T(n,1)\,e - T(n,1)\left(1 - \frac{e}{p}\right)$$

$$T(n,p) = T(n,p)\,\psi e + T(n,p)\,\psi\left(\frac{1-e}{p}\right)$$

$$\Rightarrow 1 = \psi e + \frac{\psi(1-e)}{p}$$

$$\Rightarrow \frac{1}{\psi} = e + \frac{1-e}{p}$$

$$\Rightarrow \frac{1}{\psi} = e + \frac{1}{p}$$

$$\Rightarrow \boxed{e = \frac{1/\psi - 1/p}{1 - 1/p}}$$ (This is for experimental observations.)

$$\psi = \frac{T(n,i)}{T(n,p)}$$

p	2	3	4	5	...	8
ψ	1.82	2.5	3.08	3.57	...	4.71
e	0.1	0.1	0.1			0.1

- We are getting a speedup which is not linear.
- Problem: Most of the processors are idle, or there is a some overhead.

$$e = \frac{(p-1)\,T(n) + pk(n,p)}{(p-1)\,T(n,1)}$$

If $pk(n,p)$ is removed,
e = constant.

to the value of e is constant, it is no longer possible to further improve the code.

p	2	3	4	5	...	8
ψ	1.87	2.61	3.2	3.73		4.7
e	0.07	0.75	0.88	0.81		0.1

For a fixed problem size, efficiency will decrease with increase in number of processors.

* Inefficiency metric :-

$$\psi(n,p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n)/p + k(n,p)}$$

It is the ability to increase performance as the number of processors increases.

$$\Rightarrow \psi(n,p) \leq \frac{p[\sigma(n) + \phi(n)]}{\sigma(n) + \phi(n) + (p-1)\sigma(n) + pk(n,p)}$$

$T_o(n,p) = (p-1)\sigma(n) + pk(n,p)$: overhead.

efficiency :

$$\varepsilon(n,p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n) + T_o(n,p)}$$

$$\leq \frac{}{1 + T_o(n,p)/T(n,1)}$$

$$\frac{T_o(n,p)}{T(n,1)} \leq \frac{1 - \varepsilon(n,p)}{\varepsilon(n,p)}$$

$$\boxed{T(n,1) \geq \frac{\varepsilon(n,p)}{1 - \varepsilon(n,p)} T_o(n,p)}$$

↳ Rate at which problem size has to be increased to maintain constant efficiency.

99%
97%
90%
} Cache hit.

Cache hit : getting data from the cache
- If data is not in cache, penalty must be paid to get the data from the RAM.

Cache hit time = 1 cycle.
Miss penalty ~ 100 cycle.
- If you not get something at your nearest location, you have to pay a penalty.

Average access time = $\frac{99}{100} \times 1 \text{ cycle} + \frac{1}{100} \times 100$

$= 1 + 0.99$

$= 1.99$.

If cache hit rate is 97%.

Average access time = $\frac{97}{100} \times 1 + \frac{3}{100} \times 100$

$= 3.97$

The access time will reduce by a factor of around 2.

$\frac{80}{100} \times 4 + \frac{20}{100} \times 100 = 20.8$.

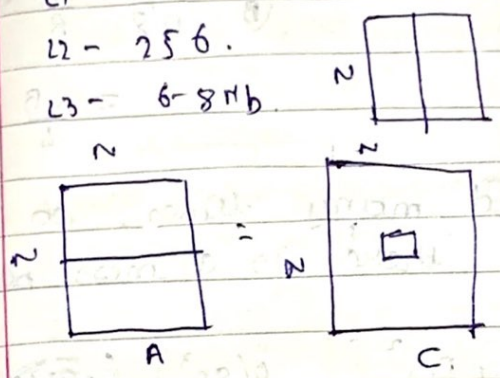
Miss rate = 1 - cache hit.

Cache hit time : time taken for data to be transferred from cache to processor.

L1 - 30

L2 - 256.

L3 - 6-8 Mb.



Matrix elements = 8 bytes.

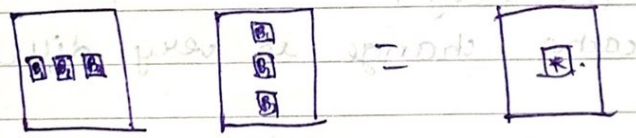
Total no. of operations in software: $O(N^2)$.

cache block size: A chunk of data is recieved instead of a single block of data, generally of 64 or 32 bytes.

No. of cache misses = $\frac{n}{8} + n$ (row, column).

cache size << Problem size (N).

- For the first access, there will be a miss but not for the others.
- In C, the data is stored in row major, so there will be cache misses for ~~any~~ only one element in row but for all elements in the case of column.



(Multiplying 2 blocks, we get resultant block.)

Block size = $B^2 = B \times B$

- Assume that the cache can hold only 3 blocks but not more than 3 blocks.
- condition for $3B^2$ < cache size.

cache miss for each block $= \left(\frac{n}{B} + \frac{n}{B}\right)\frac{B^2}{8} = \frac{2n}{B} * \frac{B^2}{8}$

$= \frac{nB}{4}$

- one block is reused many times, so only for the first time, there is a miss penalty.

Total no. of blocks $= \left(\frac{n}{B}\right)^2$, $O\left(\frac{9n^3}{B}\right)$

Thus, $\left(\frac{n}{B}\right)^2 * \frac{nB}{4} = \frac{n^3}{4B}$

$\frac{9n^3}{8}$	$\frac{n^3}{4B}$ (n is fixed). value of B will change.
	$B=4 : \frac{n^3}{16}$

By doing block matrix multiplication, the increase in speed $= \frac{9n^3}{8} * \frac{16}{n^3}^2 : 18.$

Thus, algorithm will be 18 times faster.
Now, if $B = 8$:

$\frac{9n^3}{8} * \frac{32}{n^3} = 36$ times faster

This won't always be true if done on a machine practically.
- Thus 2% cache change is very different.

General lesson: Keep working data set very small, this can be achived by dividing into blocks so that it is possible to fit the data into cache and so the cache hit rate increases.

- Keep the strike sizes small.
- Focus on pattern in innermost loop.

1) n processors takes $(\log n)^2$ time in parallel.
serial time = $n \log n$. special case ($p = n$).

speedup = $\frac{n}{\log n}$

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

cost = $n(\log n)^2$.

- efficiency will decrease with increase in the problem size.
- the algorithm is not cost optimal by a factor of $\log n$.
- let us make a real assumption where $p << n$.

$T_p = \frac{(\log n)^2}{p}$ $\frac{n(\log n)^2}{p}$

New speedup = $\frac{p}{\log n}$

the speedup decreases as the problem size increases for a fixed value of the problem size because the algorithm is not cost optimal.

* sorting a list → serial → $(n \log n)$.

Algorithm	A_1	A_2	A_3	A_4
No. of processes	n^2	$\log n$	n	$\sqrt{n}$
T_p	1	n	$\sqrt{n}$	$\sqrt{n} \log n$

Speedup	$n \log n$	$\log n$	$n \log n$	$\sqrt{n}$
Efficiency	$\frac{\log n}{n}$	1	$\frac{\log n}{n}$	1
cost	n^2	$n \log n$	$n n$	$n \log n$

d) A_4 should be chosen.

A_1 won't be chosen because the cost is very high.

A_3 won't be chosen as speedup is high.

We may choose between A_2 and A_4.

e) Parallel program on 2 process system.

- Problem size = W.
- serial → 1 processor, cache = 64 kB, cache hit rate = 80%
- latency of cache = 2ns, latency of DRAM = 100 ns.
- Memory bound computation → 1 flop/memory access.

Parallel → each processor does W/2.

- cache hit rate = 90%, 8% from RAM, 2% from the remote DRAM (communication overhead) → latency of 400 ns.

Comment on expected speedup.

A: Expected speedup = 2.42

The speedup is a superlinear speedup. This is due to a combined cache effect solution:

$(0.8 \times 2) + (0.2 \times 100) = 21.6$ ns.

Performance = $\frac{1}{21.6 \text{ ns}}$ = 46.3 MFlops (serial). (effective memory access time)

Parallel performance:

$(0.9 \times 2) + (0.08 \times 100) + (0.02 \times 400) = 17.8$ ns.

Performance = $\frac{1}{17.8 \text{ ns}}$ (flops

Page
Date

Processing time = $2 \times 17.8 = 35.6$ ns

performance per processor = $\frac{1}{17.8} = 56.18$ MFlops

performance = 2×56.18

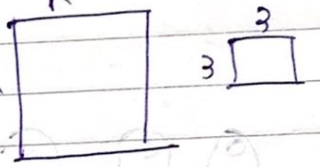
Speedup = $\frac{56.18 \times 2}{46.3} = \boxed{2.42}$

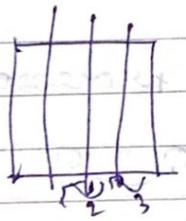
* Image processing (convolution).

- $n \times n$ pixel image
- convolution with 3×3 mask.
- each multiply-add takes 't_c' time.
- Distributed system → each node has its subimage and applies the mask on subimage.
- each pixel → one word communication
- communication latency → $t_s + t_w$.
  - t_s → startup time
  - t_w → per word transfer time

comment on efficiency?

i:





Efficiency in terms of n, p, t_s, t_w, t_c

serial time = $n^2 9 t_c$

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

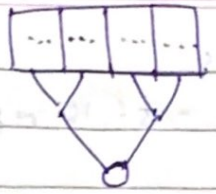
we always assume that $p = n$.

$p < n$: change granularity of problem.

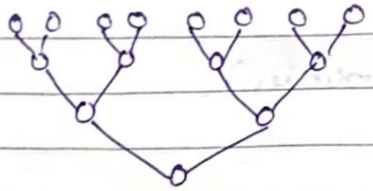
effect of granularity on performance.

n numbers on p processors

$n = 16$, $p = 4$ (power of 2). valid only for power of 2.



Design 1: $\frac{n}{p} + \log p$.



n elements = n processors. → $\log n$.

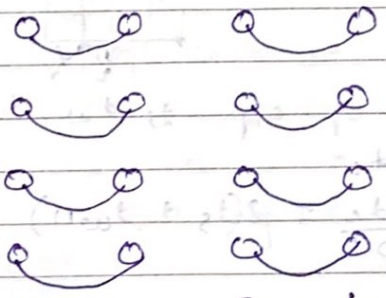
n elements → p processors with the same design.

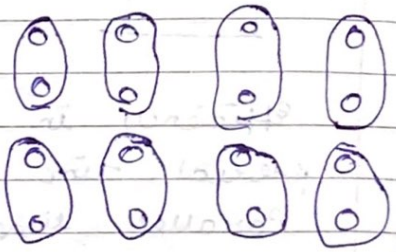
cost → $O(n + p \log p) \sim O(n)$. : this is cost optimal.

↳ time as long as $n \sim p \log p$.

- To p processors, $\log p$ steps will be done to converge to 1 processor.

4 processors working.



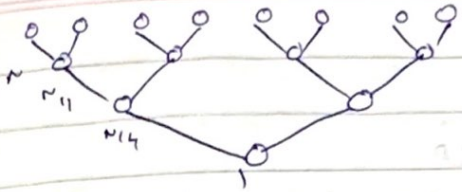


p: 1 2 3 4. 1 2 3 4

cost = $O(n \log p)$ (with p processors.)

this algorithm is non-cost optimal.

n=16, p=4. $\frac{16}{4} \log_2 4 = 4 \times 2 = 8$.



simulate it on p virtual processors.
$p << n$.

After each stage, there is a deactivation.
$n = n/2$ step $\mapsto \frac{n}{p}$.

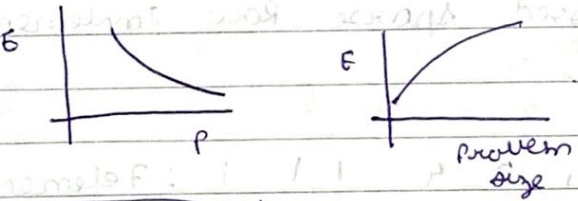
1) Replicate log p steps after the log n steps of the original. (step $\mapsto n/p$)
2) step 2 $\mapsto n/2 \,/\, p/2 = n/p$
3) step 3 $\mapsto n/p$.

$$\frac{n}{p}\log p + \frac{n}{p}$$

cost $= p\left[\frac{n}{p}\log p + \frac{n}{p}\right] = n\log p + n$: this is non-cost optimal.

$E = \frac{S}{p} = \frac{T_s}{pT_p}$ &

$\Rightarrow E = \frac{1}{1 + \frac{T_o}{T_s}}$) $T_o = pT_p - T_s$.



$N = \frac{E}{1-E} T_o(N, p)$ → Problem size. , isoefficiency function.
K

Problem size:- Minimum number of computations in serial case.

Adding N numbers on p processors.

$w = k p \log p$, $\because p' \log p'$.
$p \log p$

$T_o = p^{3/2} + p^{3/4} w^{3/4}$

w = problem size.

p = no. of processors.

* what rate should problem size be increased to maintain same efficiency?

Step 1:- Treat the problem individually.

$w = kp^{3/2}$

$w = kp^{3/4} w^{3/4}$

$w = kp^{3/4}$

$w = k^4 p^3$

Efficiency relation → algorithm, architecture.

Sparse matrix

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

Vector

$$\begin{vmatrix} \\ \\ \\ \\ \end{vmatrix}$$

A × B

CSR

CSR - compressed sparse Row Implementation.

$\frac{7}{16}$.

Data: 3 1 2 4 1 1 1 : 7 elements.

Col pointers: 0 2 1 2 3 0 3 :

Locate: 0 2 2 5 7 : Look into the final matrix

No. of rows: 5 - 1 = 4.

N + N + N = O(N)

$$\begin{vmatrix} N & \diagdown & N \\ & \diagdown & \end{vmatrix} = O(N^2)$$

10000 × 10000 = 100000.

- a huge drop in speedup when there is a parallel generation of random numbers. Why?
- look at the seed, figure out problem. (rate - cards).

- current model which we are using is single Program Multiple Data.
- Multiple Program multiple data works on different data.

SPMD:-

1. Instruction level parallelism.
2. Thread level parallelism.
3. Process level parallelism.

* There is a fixed IP for each processor.

In MPMD, different processors execute different programs on different data.

The major difference in this case is that processors can access all memories, but in the other case it can only access local memories.

Performance depends on data in cache.

cache coherency:- If a piece of data in one processor is updated, it must be updated in the others.

Page
Date

Rule :- The problem must be broken into independent tasks with independent memory.

Challenges are of idling time and giving a wrong value.

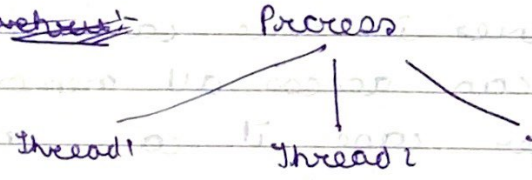
~~Buffered blocking mess~~

	Buffered	non buffered
blocking	② ✓	① ✓
non blocking		

non-blocking : There is no idling time, but in blocking there is idling time.

- Communication overheads will overlap.
- In non-blocking operation, it is unsafe to send data.
- ~~**~~ Anything can be written.

~~* refresh~~



When 1 process has more than 1 thread, there is a hybrid code.

4 stages in MPI program :-

1) MPI include file.
2) Initialize MPI environment.
3) Do work and make message passing calls.
4) Terminate MPI environment.

```
MPI_Init(&argc, &argv);
MPI_Finalize();
```

Initialization :- It will initiate the MPI environment.

Termination :- once we terminate, no MPI routine is called.

MPI processes are identified by their ranks.
- Rank ranges from: 0, 1, 2, ... nprocs-1.
- nprocs do not change during computation.

Communicator :- A group of processes that will communicate with one another.
- They are used as arguments to all message transfer MPI routines.

ways to check if messages are sent correctly or not.

Type of communication :-
1) Point to point
2) Collective.

* Send and receive.

1) Destination / source
2) Size of message.
3) Data type of message (integer, float, etc.).
4) Protocol.
5) Status
6) A tag so that receiver knows which message is to be received.

We are sending data:
A[0:20, 0:30]

- Create argument, convert into 1D and send

MPI: Integration using the trapezoidal rule.

Steps:-

1) Division of work.
Problem size = N
Processors = P

$\frac{N}{P}$.

A[0:Na] : several subprocessors belonging into one huge unit.

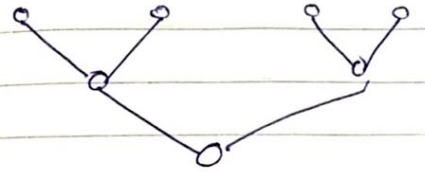
$A\left[0 : \frac{n}{p}\right]$, $A\left[\frac{n}{p}+1 : \frac{2n}{p}\right]$.

2) Idea on what data to work on.
3) Collective communication is used.

Page
Date

Point to point communication:-

- copied from user memory to system buffer.
- sent from buffer over network to buffer of receiving process.
- Receiver copies data from system buffer to local user memory space.



The switches that are used are infiniband switch.

communication protocols are blocking and non-blocking.

- status variable can give information about the errors.

---

* Architecture: Memory hierarchy.
  Pipeline.
  Importance of cache.
* Data dependency.
* Design of parallel algorithms. (complexity analysis.) (PRAMs).
* Parallel patterns → Reduction, scan, convolution, matrix multiplication, block multiplication, complexity analysis.
* Shared memory programming → Open MP.
* Performance Modelling.
* Algorithmic analysis (parallel performance)
* MPI.

25-30 questions.

* Isefficiency analysis:-

$T_o = p^{3/2} + p^{3/4} w^{3/4} = T_o(w, p)$

$w = kT_o(w, p)$

w = problem size

p = no. of processors.

$w = kp^{3/2}$

$w = kp^{3/4} w^{3/4}$

$w^{1-3/4} = kp^{3/4}$

$\Rightarrow w^{1/4} = kp^{3/4}$

$\Rightarrow w = kp^3$

Thus, the problem size must be increased in $O(p^3)$, where p is no. of processors.

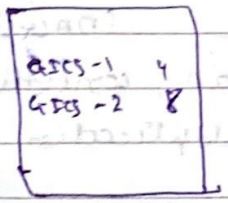
---

* Avoiding deadlock:- cannot move out unless the blocking statement not done.

- for this, circular blocks can be broken down.

use cluster only for MPI.

machine file.h.



mpiexecute -12