3) Output dependency :-

statement 'i' precedes 'j'
and 'i' computes a value that 'j' also computes.

① $x = 1$ ② $y = x + 2$ ③ $y = 2 - w$ ④ $x = y/2$.

Dependencies:
① → ③
① → ④
③ → ④

4) Input dependency :-

statement 'i' precedes 'j'
and 'i' uses a value that 'j' also uses.

Dependencies :-
① $x = 1$ ② $y = x + 2$ ③ $x = 2 - w$ ④ $x = y/2$.

Dependencies :- (Input dependency not a problem)
③ → ④.

- the most challenging problem is how to remove the dependencies.
- It is very difficult to remove true dependency.

Dependency flows from 'i' to 'j'
↓ ↓
source sink.

- We can have data and task dependency. use DAG, directed graph.

In a data dependency graph:-

1) Nodes are the statements
2) Edges are the dependent relations.

```
a[i] = b[i] + c[i].   → There is a flow dependency,
d[i] = a[i].            but no loop carried dependency.
                        → Dependence distance = 0
```

loop carried dependency.
loop independent dependency.

```
a[i] = b[i] + c[i]   → Dependence distance = 1.
d[i] = a[i-1].       → This is a loop carried
                        dependency
```

Try to parallelize the loop that does not have a dependency.

```
For(i=0 ; i<N; i++)  → No problem (can be parallelized)
    a[i-1] = b[i]
for(j= 0; j<N; j++)  → No problem (can be parallelized).
    c[j] = a[j].
```

```
for( i=0; i<N; i++)  → Problem: cannot be parallelized,
    a[i-1] = b[i].       but locality is better as
    c[i] = a[i].         efficient use of spatial locality.
```

loop fusion

In the first case, the locality is bad

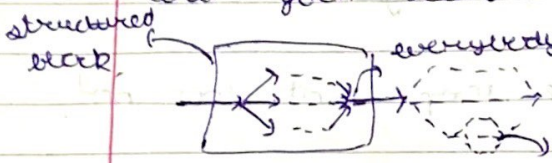
Loop fusion:- To fuse 2 loops into one, it is done for better use of spatial locality.

Loop distribution :- It is a technique to introduce parallelization.

Directives are special preprocessor instructions, instructions may or may not get compiled.
- simple and limited set of directives.

shared - global memory. (access to program).
distributed - private memory. (memory management difficult.)

* Fork join concept:- Fork creates concurrency and join removes concurrency.



If one of the threads are stuck, there is a deadlock.
- This is a concurrency control mechanism.
- In a structured block, there is one part of entry and one part of exit, openmp will parallelize this structured block.
- Put exit() condition.
- Branching is not allowed, explicit barrier one of the threads may be stuck.

~~We need to understand model.~~
- shared memory with thread based parallelism.
- The user must specify parallel execution, user directive are not ignored even if wrong.

Thread:- An independent instruction stream allowing concurrent operation.

- Open MP program starts single threaded.

Data scoping:- Scope is the access part.
- The time for which the data is available.

Private variables:-
1) Exist only within the new scope.
2) other threads cannot access the private variables.
3) loop index variables are also private variables.

```
#pragma omp parallel
{
  code to be executed by each thread.
}
```

compile : -fopenmp.
Intel : -openmp. setenv/export
Specifying no. of threads: OMP_NUM_THREADS = 4, 4 threads are assigned, other ways also to specify this.

```
#pragma omp parallel num_threads(4) : Introduces no. of thread inside the code.
```

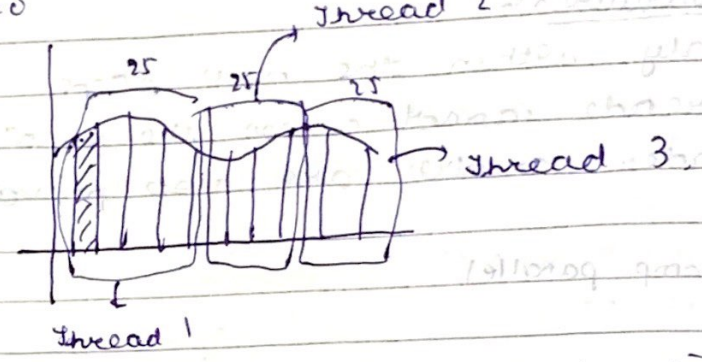
- 3 categories of compiler directives:-
1) control constructs
2) Data scoping
3) Synchronization

```
int ID = omp_get_thread_num();
```
- The output can be produced in any random arbitrary order.

* Numerical integration:-

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi.$$

$$\sum_{i=0}^{n} F(x_i)\,\Delta x \approx \pi$$

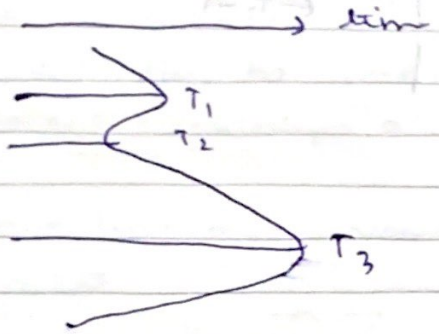


- All must get same no. of points, i.e. proper thread values.
- SPMD is used.
- All threads write sum in private memory, before exiting, write it in the global memory.
- We must ensure a principle of mutual exclusion where 2 people do not access the same data.

Load balancing:- Load balancing is simple prob but it may not be there.
- Not necessary that all threads will take the same amount of time.

1000 → 250 | 250 | 250 | 250.
- We can decide that whoever is free will accept the work.

Time perspective



T_3 will give its work to T_2.

components of openmp :- Directives, runtime library routines, environment variables.

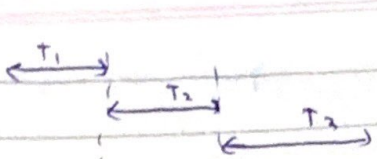
Runtime library routines will help manage parallel programs.

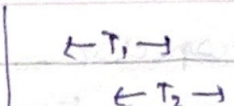
compile : gcc serial-trap.c -fopenmp -o serial-trap
output : ./serial-trap.

Index variable of a worksharing loop is private by default.

synchronous	asynchronous.
←1→ ←2→ ←3→	←1→ ←2→ ←3→

Important in threads : communication and computation
- Whenever there is communication, there should not be computation.
- Synchronous is a blocking mechanism, asynchronous is a non-blocking mechanism.





N=10, threads = 4.

$T_1 = 1$	5	9
$T_2 = 2$	6	10
$T_3 = 3$	7	
$T_4 = 4$	8	

Another scheduling mechanism:-

$T_1 - 12$

$T_2 - 34$

$T_3 - 56$

$T_4 - 78$.

No scheduling mechanism done till now, when no proper division, scheduling mechanism will become important.

3 main data scoping clauses:-

1) Shared
2) Private.
3) Reduction - Exhibits both shared and private storage behaviour.

#pragma omp for: It is a worksharing construct commonly used with sharing constructs for

1) for: shares iterations of a loop across a group of threads
2) sections: breaks work into separate sections
3) critical: serializes a section of code

Scheduling mechanism: static, dynamic, guided, default.

#pragma omp parallel shared(a,b,c,chunk) private(i)
i → private variables different from global variables
It is possible to input more than 1 private variables.

Data scope context, coordination scope, synchronization scope, load balancing, work sharing, scheduling are the contexts

Thread synchronization:-

communication: through read/write operation
- without synchronization, multiple threads may cause conflicts.

2 main forms of synchronization:-

1) Implicit event synchronization
2) Explicit event synchronization

Race condition: 2 asynchronous threads access the same variables, different outcomes are possible.
#pragma omp critical.

#pragma omp single. - only one thread does it

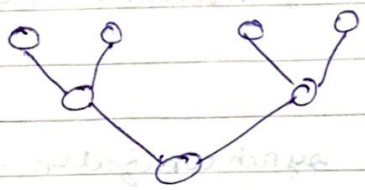
#pragma omp barrier - collective operation.

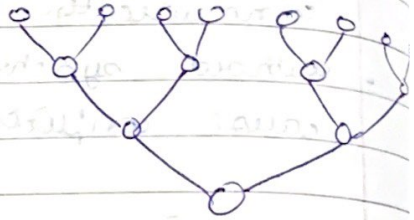
```
sum = 0.0
#pragma omp parallel for reduction(+:sum)
    for(i=0;i<20;i++)
        sum = sum + (a[i] * b[i]);
```

Floating point operations are not associative
i.e. $(a+b)+c \neq a+(b+c)$
Identity element needs to be there.

Introduction to OpenMP: Tim Mattson.

* Parallel Programming:
1) Problem decomposition.
2) Task and mapping of task
3) Mapping of dependency. → task dependency graph.





This is the parallel task dependency.
The amount of work done is same.

This is the serial task dependency.

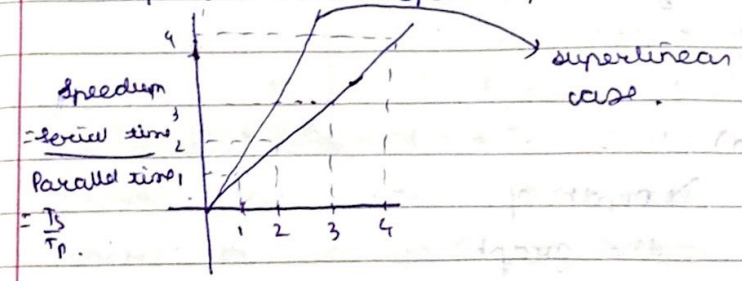
The time complexity will change, but work complexity is same.

- Map the output to the processor.
- The whole of b matrix is used everytime, we have to be careful.
- ~~The~~ A lot of data is shared.
- The control dependencies have to be removed.

Degree of concurrency :- No. of tasks that can be executed in parallel.

- Fine and coarse grain granularity, the degree of concurrency will increase as the granularity is fines.

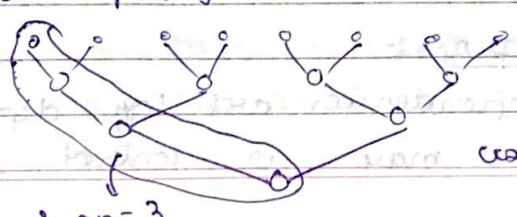
Super linear speedup :-



This superlinear speedup is not possible with the cost model.

- When combined cache is more compared to serial cache, we get a superlinear speedup.

Serial cost (order of time)
Time complexity
Parallel cost (order of time × no. of processors)
Time complexity



$\log_2 n$. = time complexity.
cost = $n \log_2 n$.

For serial problem:-

Time complexity = n
cost = n.

In serial, time and cost complexity is same, but in parallel it is different.

Span / depth of graph: longest path in graph.

Work = total no. of nodes / tasks / vertices if all nodes do same amount of work.

Work / Depth = average amount of parallelism.

$T_p(n) \geq D(n)$.
↓ ↘ Depth of the graph
Execution time.

Work span law:-

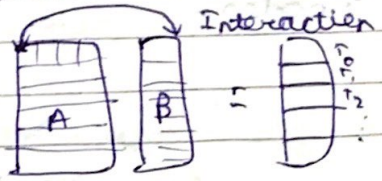
$T_p(n) \geq \max[D(n), W(n)/p]$.

Critical path length:- longest path length in a task dependency graph.

~~Task interaction graph:-~~
- some types of dependencies are loop dependency, data dependency and main is control dependency

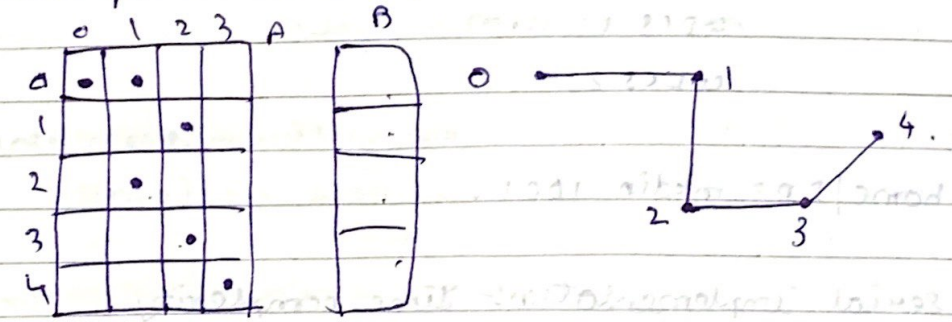
- Structure dependency is more related to hardware.

+ Task interaction graph:



Multiplication of sparse matrix A with a vector B.

- Only non-zero elements of matrix A will participate in computation.



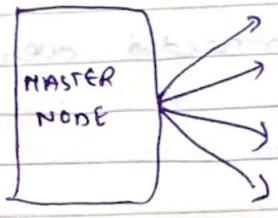
- Divide the nodes in such a way that there is a good performance.
- Task interaction graph will deal with data dependency.
- If granularity of decomposition is finer, overhead increases.

Processes and mapping: Mapping of tasks must be done to processes.

- Mappings are determined by task dependency and task interaction graphs.
- Goal of task dependency graph is to minimize interaction with other processors.

IP address for HPC class: 10.100.71.130.

10.100.71.130

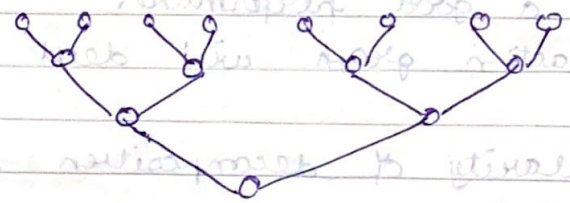


- enter into master node.
  ssh ID @ 10.100.71.130.

  DO SSH GICS0.
  GICS1
  GICS2.

  home/ID = mkdir 1001.

* Serial implementation:- Time complexity
  Implementation approach :- why this implementation



Problem size = n.

~~work complexity~~

	S	P
work complexity	$O(n)$	$O(n)$.
time complexity	$O(n)$	$O(\log_2 n)$
cost complexity	$O(n)$	$O(n \log_2 n)$.

Theoretical speedup = $\frac{\text{serial time } (T_s)}{\text{Parallel time } (T_0)}$

$= \frac{n}{\log_2 n}$.

Estimated serial fraction : No need to include this.

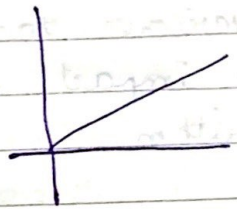
Tight upper bound on Amdahl's law: Don't include this.

Number of memory accesses:- Put a counter to get no. of memory accesses.

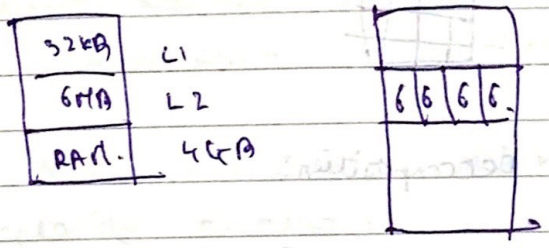
No. of computations : calculate this.

* Plots we will make:-

S(n, p) → speedup curve.



Linear increase not always necessary.



Time wise: (no. of processors wise)

Time wise: (problem size increases)

Efficiency wise analysis:

Efficiency = $\frac{\text{speedup}}{\text{no. of processors}}$

Karp-Flatt : (not to be done now.)

Date

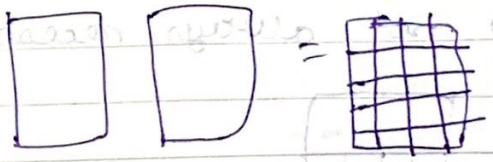
* Further detailed analysis: (Don't do this right now.)

Decomposition techniques :-

1) Recursive decomposition
2) Data decomposition
3) Exploratory decomposition
4) Speculative decomposition.

1) Divide and conquer.
- Problem divided into sub problems.
- Finding minimum number also a divide and conquer approach.

2) Data identifies on which computations are performed.
- Partition of data across various tasks.
- Partitioning of data will impact the performance of the algorithm.



Output Data Decomposition:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

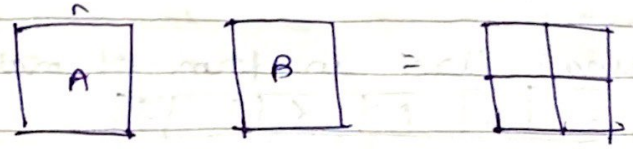
$10000 \times 8 = 80000$ bytes.

algorithm is work inefficient if:
work complexity of parallel algorithm > work or complexity of serial algorithm

Page Date

state complexity of parallel < state complexity of serial.

Owner Computes Rule:- Process assigned a particular data item is responsible for all computations associated with it.



3) Exploratory decomposition: Problem goes hand in hand with its execution.
- Involve exploration of a state space of solutions.
- Variety of discrete organisation problems, theorem proving, game playing, etc.
- Superlinear speedup because we terminate on finding the solution

4) Speculative decomposition: Dependencies are not known, impossible to identify independent tasks.

Hybrid decomposition:- A mix of decompositions for decomposing a problem.

3 main things which need to be understood
1) Complexity analysis
2) Dependency graphs.
3) Decomposition techniques

Today = NO lecture.

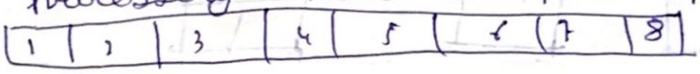
10.100.56.71 : export ☺ for hpc

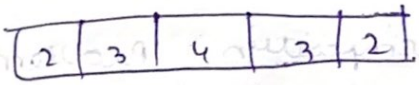
Multicore data preprocessing in openmp.

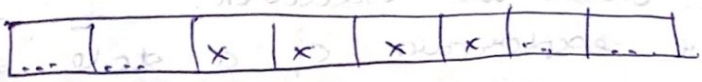
- convolution : stencil computation

$$\frac{f(x+h) - f(x-h)}{2\Delta x} = \frac{\partial f}{\partial h}$$

- Image processing : Pixel in form of matrix.

ID : 

Mask : 

Output 

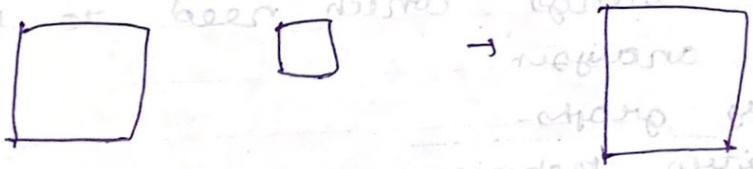
The output is symmetric

- there can be overlap while reading, but not during writing.
- convolution transfers signal into more desirable signal.
- Not much speedup is acquired.
- No dependency.

Input Data + convolution mask → output

2D image convolution:

N input N mask Output



→ Output is N × N.

5 × 5 × 8 = 200 bytes

this is a spatial locality problem.

- Image warping: spatial transformation of image.

$$x' = (x - C_x)\cos\theta + (y - C_y)\sin\theta + C_x$$
$$y' = -(x - C_x)\sin\theta + (y - C_y)\cos\theta + C_y$$

- libraries to convert image to pixel.

Scan operation and parallelization:-

- sum reduction takes $O(n)$.
- sum reduction is work operation as same order of work is done.

Scan algorithm:-

Input: $A = [a_0\ a_1\ a_2 \dots a_{n-1}]$.

$scan(A) = [a_0\ \ a_0 \oplus a_1\ \ a_0 \oplus a_1 \oplus a_2 \dots a_0 \oplus a_1 \oplus \dots \oplus a_{n-2}]$

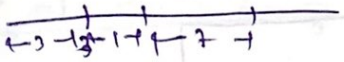
$\oplus$ is addition operation.

3	1	7	0	4	1	6	3

Sum:

0	3	4	11	...

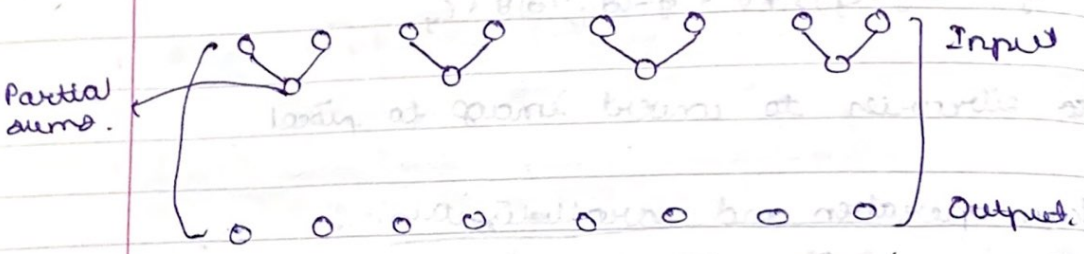
Rope cutting method.



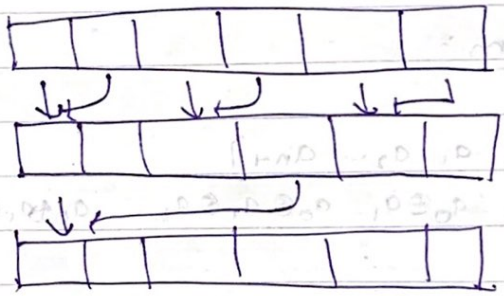
If we do a scan, we get the locations

where to put the letter, and do all the operations in parallel, this is also called prefix sum.

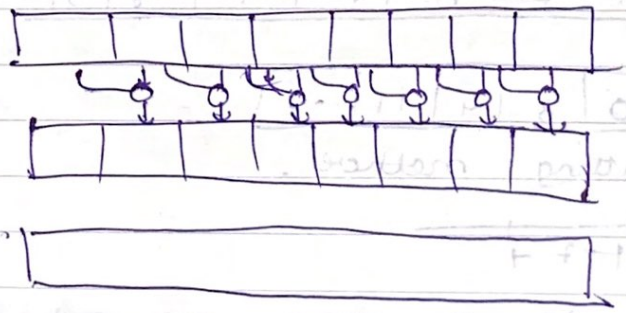
- Scan is a building block for many parallel algorithms.
- In scan there is a dependency on every point, so it is difficult to parallelize, but in reduction, there is no dependency between points.



- Add the partial sums and get a global sum.



+ steele algorithm :-



Iterate $\log(n)$ times, each thread will add an offset to its own value.

How to reach $O(n)$.

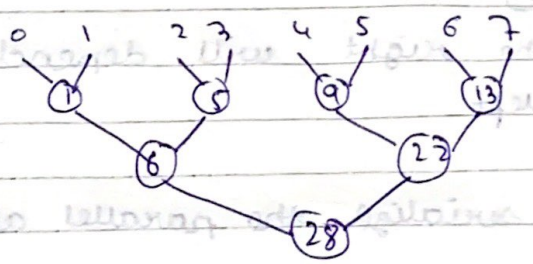
$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + \infty = n$.

Not alternate threads are deactivated

$n = 8$:

	S	P.
	8	$24 - 7 = 17$.
	1024.	$1024 \times 16 - 1024 \approx 9000$

- When parallel algorithm takes more than serial algorithm = work inefficient

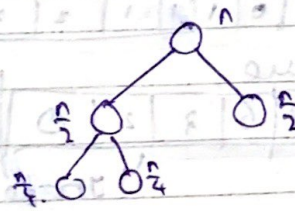


- It consists of an up sweep and a down sweep.

No. of steps: $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots = 2n - 1 = O(n)$.

Assume n is exactly a power of 2.

If we have n operations:



$\frac{n}{b^k} = 1$, $\frac{n}{4} = \frac{n}{b^2}$.

Finally, it will converge to 1.

$\log n = x \log b$:

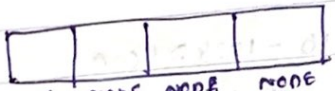
$\Rightarrow x = \frac{\log n}{\log b} = \log_b n$.

$n + (n-1) + (n-2) + (n-4) + \ldots$

$= O(N \log N - N - 1)$: this is not work efficient

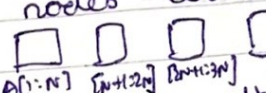
$= O(N \log N)$.

- In new algorithm we divide scan algorithm into 2 phases.



scan.

All nodes will do a scan.



The node on the right will depend on the node on the left.

A[N]

- Dependency will serialize the parallel algorithm.

N_1	N_1+N_2	$N_1+N_2+N_3$

: scan can be used to parallelize scan itself.

Problem:-

old index:

0	1	2	3	4	5	6	7

flag:

0	1	0	0	1	1	0	1

1 : Keep the value

0	1	1	1	2	3	3	4.

0 : Discard the value.

new values

1	4	5	7	6	3	2	0

These are the new values

HPC.
(continued)

Performance analysis:- there are 4 general formulas.

1) Amdahl's law.
2) Gustafson - Barsis law.
3) Karp - Flatt metric.
4) Isoefficiency matrix.

1) Amdahl's law will decide whether parallization is possible on serial code or not.

2) This is for evaluation of a parallel program.

3) Reason for not getting the best performance in the code.
- overhead is one part. (problem size), no of processors)
- serial part in code which is always present.
- Parallel part.
- this metric will help us find if serial part or overhead causes problem.

4) scalability of code.

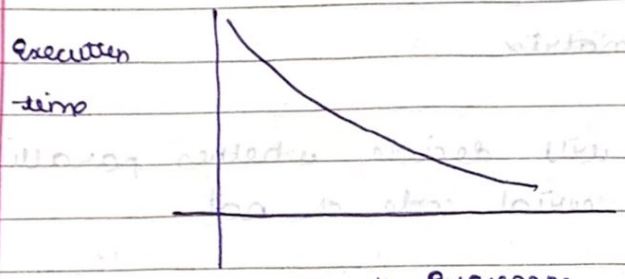
sequential time = $\sigma(n)$.
n → problem size
p → no. of processors.
Parallel time = $\phi(n)$.
Overhead time = $k(n,p)$.

Page
Date

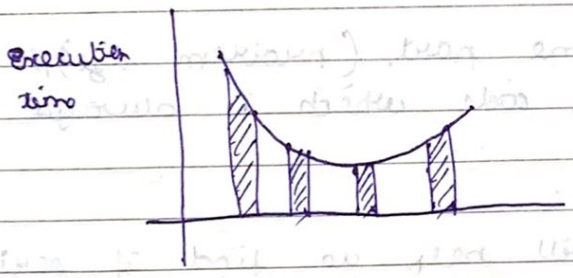
$$\text{Speedup} = \Psi(n,p) \leq \frac{\sigma(n) + \phi(n)}{\sigma(n) + \frac{\phi(n)}{p} + k(n,p)} \quad \text{i.e.} \quad \leq \frac{1}{f}$$

Condition for equality: work has to be perfectly divided.

$p \uparrow$ Communication time $\uparrow$
Computation time $\downarrow$



In the actual case, we will see something like this.



- After some time, the communication time will start increasing.
- The problem size will be fixed.
- For a given fixed problem size, there is an optimal no. of processors giving us the least execution time.

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.)

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

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

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