Sai

PRINCIPLES OF PARALLEL ALGORITHM DESIGN

Sequential Algorithm : Recipe for solving a Problem using a Serial Computer

Parallel Algorithm : Recipe for solving a problem using multiple processors.

Terms

1. Decomposition : Process of dividing a computation into smaller parts, some or all of which, may potentially be executed in parallel

2. Tasks : Programmer defined units of computation into which the main computation is subdivided by means of decomposition
   - All tasks need not be of the same size
   - Once defined, are considered as indivisible units of computation

3. Processes : Logical computing agents that perform Tasks.

4. Processors : Hardware units that physically perform computations.

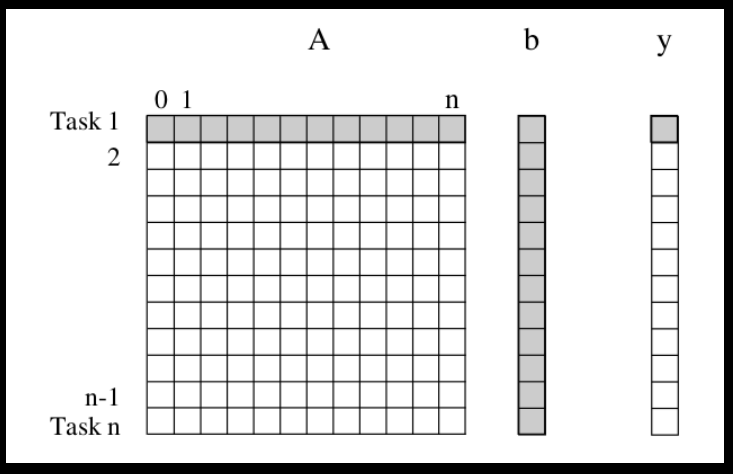
Steps involved to design (and implement) a parallel algorithm:

→ Identify Concurrent Tasks
→ Map Tasks to processes running in parallel
→ Distribute input, output, intermediate data to processes
→ Manage accesses to shared data by multiple processes
→ Synchronize processes at various stages of parallel program execution

Example I: Dense Matrix-Vector multiplication

$$y[i] = \sum_{j=1}^{n} A[i][j] * b[j]$$

Task i = Computation of y[i]

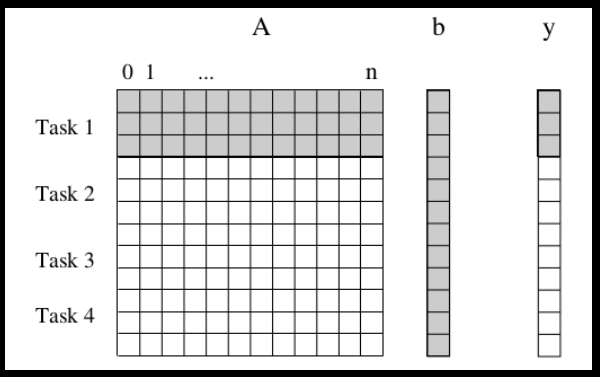


→ All tasks are independent and hence can be executed simultaneously (any sequence)

Granularity : Number and Size of tasks into which a problem is decomposed into.

→ Fine-grained - Large no. of Small tasks
→ Coarse-grained - Small no. of large tasks

How much more fine-grained? Any limit?



Each task computes n/4 elements of y.

Maximum Degree of Concurrency : The maximum number of tasks that can be executed simultaneously in a parallel program at any given time.

Average Degree of Concurrency : Average number of tasks that can be run concurrently over the entire duration of execution of the program.

Granularity Vs Degree of Concurrency

→ Generally increases
→ Depends on task dependencies

Example II : Database Query Processing

Query : MODEL = "civic" AND YEAR = "2001" AND (COLOR = "Green" OR COLOR = "White")

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

(1) MODEL = "Civic" ⇒

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

(2) YEAR = "2001" ⇒

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

(3) COLOR = "White" ⇒

ID#	Color
3476	White
6734	White

(4) COLOR = "Green" ⇒

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

(5) COLOR = "White" OR COLOR = "Green" ⇒

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

(6) YEAR = "2001" AND (COLOR = "White" OR COLOR = "Green")

ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

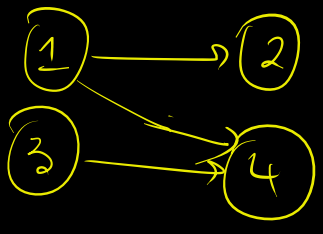
(7) MODEL = "Civic" AND YEAR = "2001" AND (COLOR = "White" OR COLOR = "Green")

ID#	Model	Year	Color
6734	Civic	2001	White

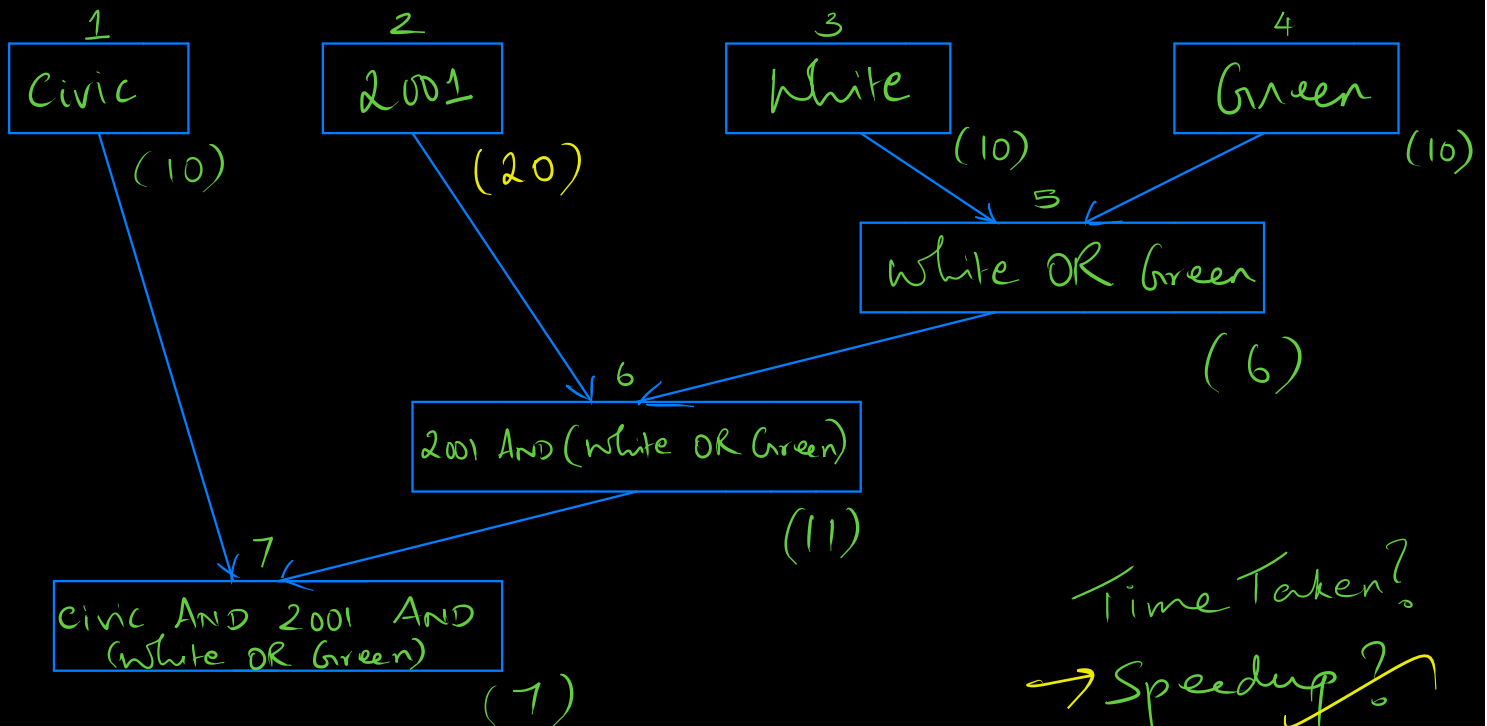
Task Dependency Graph: Directed Acyclic Graph in which nodes represent tasks and the directed edges indicate the dependencies amongst them.

→ Edge set can be empty

→ Can be disconnected



→ Task corresponding to a node can be executed when all tasks connected to this node by incoming edges have completed



Time Taken?
→ Speedup?

Granularity : 7 tasks (Coarse grained) No. of Processors?

Maximum Degree of Concurrency : 4

To find average degree of Concurrency :

Start Node : Node with no incoming edges

Finish Node : Node with no Outgoing edges

Critical path : Longest weighted directed path between any pair of Start and finish nodes.

Weight of a Node : Size or amount of work associated with the corresponding task.

Critical Path Length : Sum of weights of nodes along the critical path.

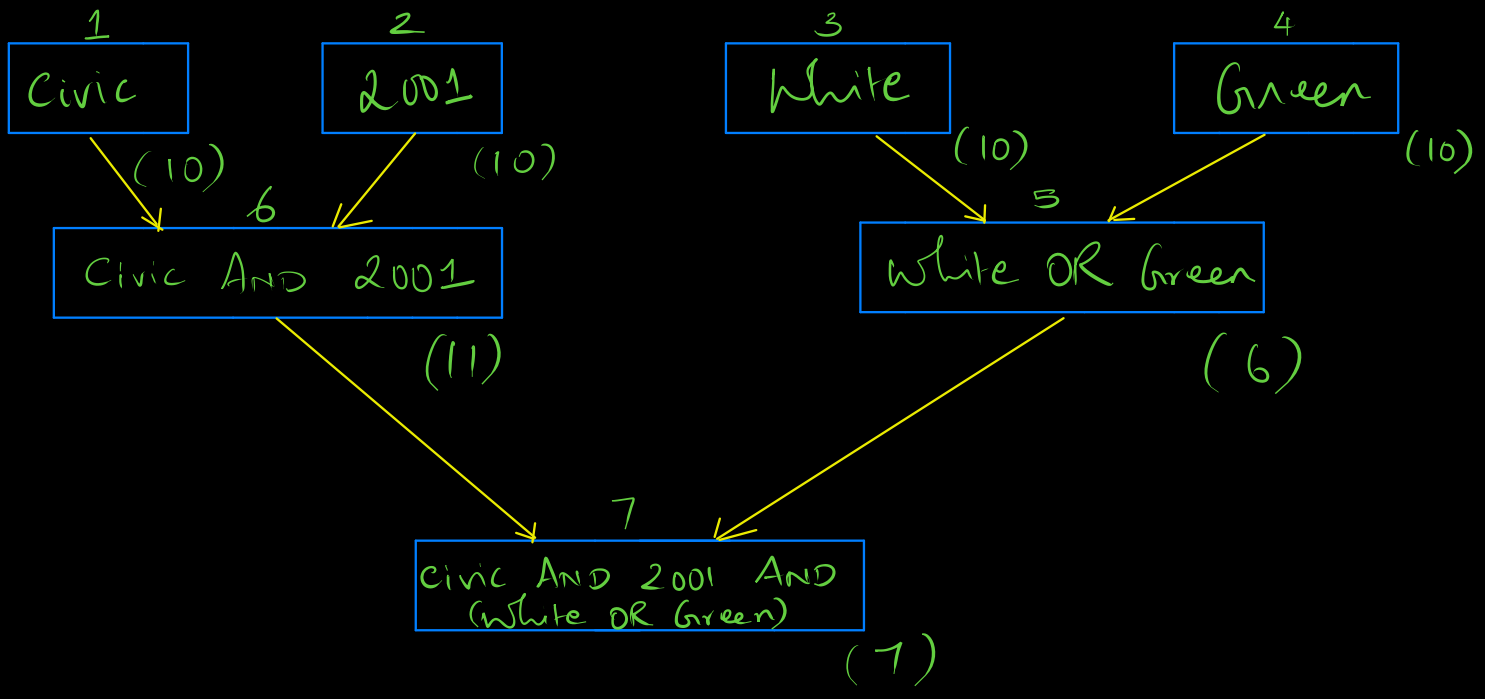
⇒ Average Degree of Concurrency = $\frac{\text{Total amount of work}}{\text{critical path length}}$

(Max Speedup)

⇒ Shorter critical path favors a higher degree of Concurrency.

$\therefore$ ADC $= \frac{64}{34} = 1.88$

Better idea ?



Granularity = 7

Max Degree of Concurrency = 4

Average Degree of Concurrency = $\frac{64}{28} = 2.28$

Factors that limit Speedup in Parallelization:

→ Granularity

→ Degree of Concurrency (Task Dependency)

→ Task Interaction

→ Tasks share input, Output or intermediate data

→ Dependencies in a Task dependency graph
(usually due to Output of one → input of another)

→ Database query

→ Even if tasks are independent, they may need to share data

→ Matrix Vector Multiply

⇒ Task Interaction Graph : Pattern of interaction among Tasks.

→ Nodes represent Tasks

→ Edges connect Tasks that interact with each other

→ Can be weighted (if known)

(Node weight : Amount of Computation a task performs
Edge weight : Amount of interaction along an edge)

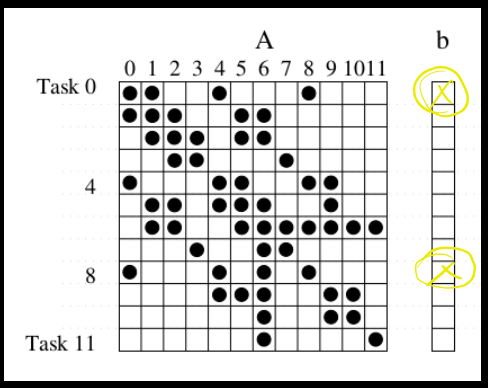
→ Can be undirected or directed graph

→ Edges of Task Interaction graph ⊇

- Edges of Task dependency graph
(usually, not always)
(Database Query)

- All tasks in Matrix Vector multiplication are independent but all tasks need a copy of vector b (messages to be sent)

Example III : Sparse Matrix Vector Multiplication



$y = Ab$

$$y[0] = A[0][0] * b[0] + A[0][1] * b[1] + A[0][4] * b[4] + A[0][8] * b[8]$$

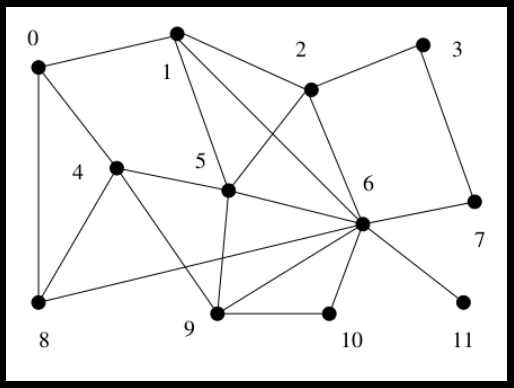
Partition Output Vector y and have each task compute an entry of y ⇒ 12 Tasks

Assume Task i is owner of A[i][*] and b[i] (input split across tasks)

But for Task i to compute y[i], it needs other elements of b.

Task 8 needs b[0] from Task 0,
b[4] from Task 4,
b[6] from Task 6.

Task Interaction Graph :



(But each task is independent)

Granularity Vs Interaction:

Let time to Compute = 1 unit of time

" " " Communicate = 1 unit of time

⇒ Task 0 Computes in 1 unit of time and Communicates in 3 units of time

What if we assign Task 0 to Computing elements 0, 4 & 8 of y?

⇒ Task 0 already has with it b[0], b[4] and b[8]

⇒ Needs 3 units of time to Compute.

How many units of time for Communication?

4

Compute : Communication ratio which is favourable?

1 : 3

3 : 4

Process: Logical Computing agents that perform Tasks.

→ Abstract entity that uses the code and data corresponding to a task to produce the output of that task within a finite amount of time after the task is activated by the parallel program.

Mapping: Mechanism by which Tasks are assigned to Processes for execution.

Good Mapping: [Task Dependency + Task Interaction graphs]

→ Maximize Concurrency by mapping independent Tasks to different Processes T_1 T_2 P_0

→ Minimize total Completion time by ensuring Processes are available to execute Tasks on the critical path as soon as such Tasks are ready to be executed

→ Minimize interaction among Processes by mapping Tasks with high degree of mutual interaction onto the same Process.

One Task + One Process ⟶ n Tasks + n Processes

KEY: Finding a balance that optimizes Parallel Performance.

Decomposition ⇒ Degree of Concurrency

Mapping ⇒ Efficiency of the resulting Parallel Algorithm