U-2 (HPC).
Parallel Algo Designs

Principles of Parallel Algorithm Design.

Preliminaries -

- Decomposing problems into tasks.
- Multiple ways of decomposition
- Tasks of diff size.
- Task dependency graph.

- Decomposition → To divide a ~~task~~ computation into sub computations to execute them parallely.

- Task → programmer defined unit of computation
  - generated by subdividing main composition by decomposition.

- Task dependency graph →
  all possible dependencies among tasks & order of execution of task is shown pictorially.

- Granuality - size of tasks is expressed as granuality of parallelism.
  Fine or coarse.

  Decompⁿ larger no. of smaller task → Fine grained granuality
  " smaller no. of larger " → Coarse grained granuality

- Degree of concurrency - no. of tasks that can be executed in par
  - max. degree of concurrency → max. no. of tasks executed si
  - avg degree of concurrency → avg no. of tasks executed con

DOMS Page No.
Date / /

* Critical path – determines average degree of concurrency for given granularity. path b/w any pair of start & finish node of nodes
* Critical path length – sum of weights along this path.
  wt. of nodes → size or amt of work assosiated with corresponding task

* Decomposition techniques –

decom

1) Recursive Techniques.
- divide & conquer strategy
- well suited for problems with inherent recursive structure
- may not be applicable to all problems
- divide problems into sub problems of same type
- eg → sorting algo (Quicksort).

2) Exploratory decomposition (search space partitioning)
- partitions sol^n space for concurrent exploration
- useful for problems with larger sol^n space
- ~~High performance if dependencies are low & predictions are~~.
- May not guarantee finding optimal sol^n complex design.
- eg → Machine learning (exploring parameters)

3) Speculative Decomposition
- creates subtasks speculatively (without guaranteed independance)
- high performance if dependencies are low & predictions are accurate
- Risk of wasted computation if speculation are wrong.
- eg → Branch prediction in modern processors.

4) Hybrid Decomposition-
- combines diff elements of diff techniques
- Leverages strength of diff techniques to optimize for problem's characteristic
- complexity can increase as rate of the design involves managing multiple decomposition strategies.
- eg → program might use recursive decomposition for main algo structure & data decomposition for processing larger datasets within each recursive step.

✱ Task Interaction Graph -

- The pattern in which task interact with each other.
- represents relationship & dependencies b/w tasks in a parallel program.
  - Nodes → Represent individual tasks.
  - Edges → represent interaction between tasks

Benefits - • Visualize communication patterns.
• Identify potential bottlenecks
• Design communication strategies
• Analyze potential for parallelism

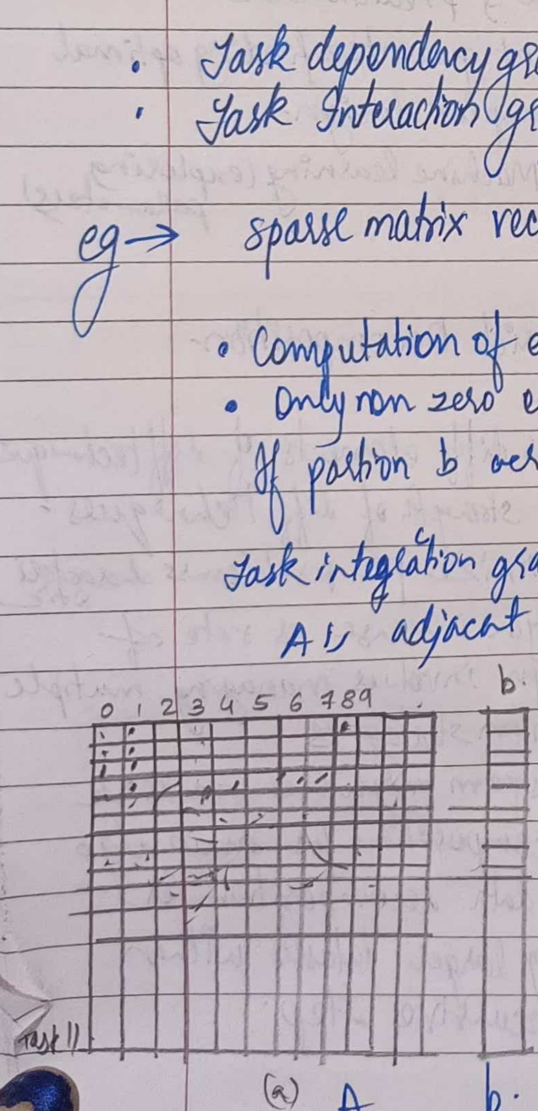
- Task dependency graph represent → control dependencies
- Task Interaction graph represent → data dependencies

eg → sparse matrix vector multiplication - $A \times b$

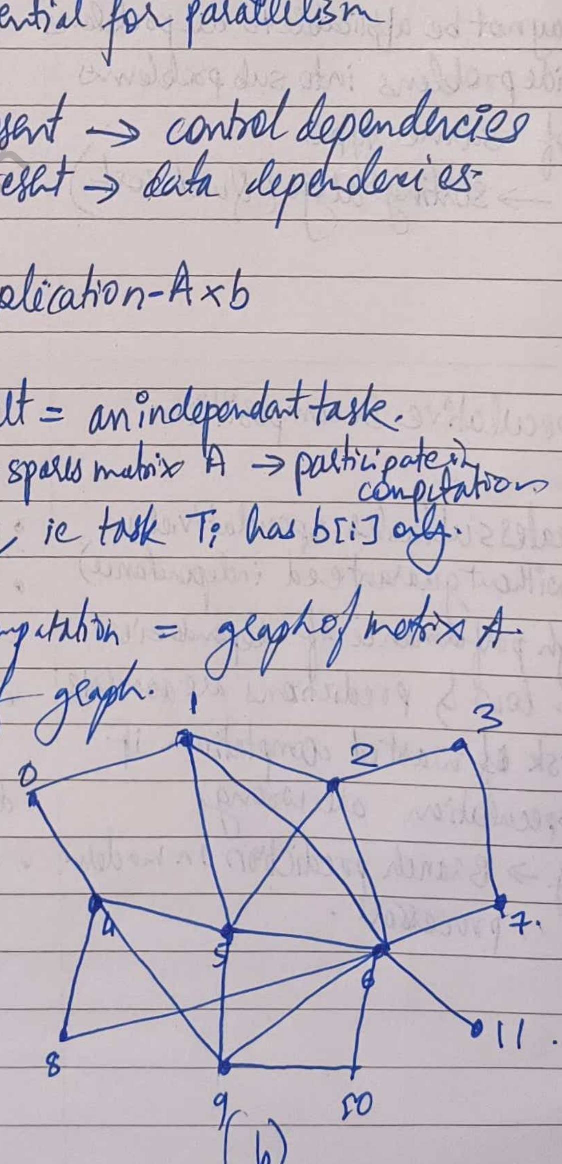
- Computation of each result = an independant task.
- Only non zero element of sparse matrix A → participate in computations

If partion b across tasks, ie task T_i has $b[i]$ only.

Task integration graph of computation = graph of matrix A.
A is adjacent matrix of graph.



(a) A b.



(b)

Processes & Mapping.

- Processes -
  - Refers to a processing or computing agent that performs tasks.
  - use code & data corresponding to task to produce output of task within time limit.
  - exchange of data → communicate with other processors.
  - speedup in parallel formulation → if more than one process remains activite at a time, performing mutiple tasks.

- Mapping -
  - assignment of task to process.
  - good mapping scheme → based on task dependecy
  - min. interactions → mapping tasks
  - mapping scheme → exploit max concurrency & min. execution time.
  - single task → single process (no time wasted in interaction, but no speedup
  - efficiency to be acheived in parallel processing.

Characteristics of Task & Interactions.

- → Task generation → static & Dynamic task generations
- → Task Sizes → uniform & non uniform tasks.
- → Knowledge of task sizes → .
- → size of data assogiated → If size & location of data is known.

Mapping Techniques for load balancing.

Static Mapping ~~st~~	Dynamic Mapping.
• distribution is before execution	• distribution time is during execution.
• Info used is static i.e. predefined	• Info used is dynamic. ie realtime
• adaptability is ~~high~~ limited.	• adaptability is ~~low~~ high
• complexity is low	• complexity is high
• suitable for predictable wordloads & stable systems.	• suitable for unpredictable workloads & dynamic systems.
• Is less efficient	• Is more efficient.
• eg → block, cyclic, random	• eg → master-slave, gossiping, work stealing

Methods for Containing Interaction Overheads:-

- parallel algo → efficient → if it has min interaction overhead
- depends of factors → vol. of data exchanged during interactions
  → frequency of interaction
  → spatial & temporal pattern of interactions

Methods are → Maximizing Data Locality
→ Min. Vol of Data exchange
→ Min. Frequency of Interactions
→ Min. Contention & Hot spots
→ Overlapping computation with interactions
→ Replicating data or computation.
→ Using optimized collective Interaction Operators
→ Overlapping Interactions with other Interaction

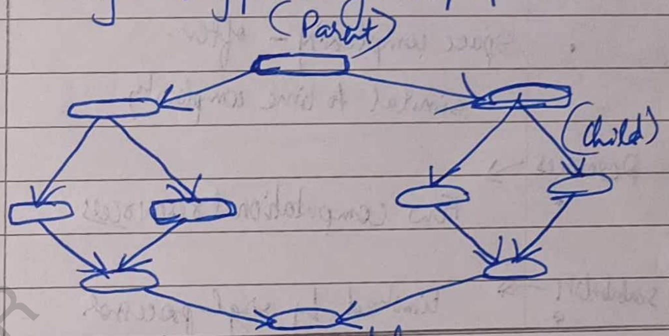
Parallel Algo Models -

1) Data Parallel Model

- Divides data in chunks & processes in parallel across multiple processors.
- same instruction to independent data elements.
- Data dependency - Low
- Minimal communication.
- Automatic (equal workload) load balancing
- Synchronization may be required for shared data.
- eg → sorting a large list.

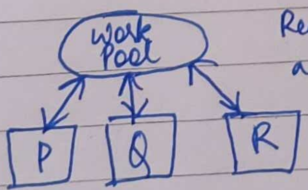
2) Task Graph Model

- Divides problem into independent tasks
- divides tasks or operations, each processor works on diff part of task concurrently
- Data dependency - Medium.
- Medium communication
- Load balancing is Manual (task design for balanced workload)
- each processor → diff task
- eg → Img processing multiple photos.



3) Work Pool Model

- Utilizes a pool of tasks or jobs
- where multiple workers fetch & execute tasks concurrently
- Data dependency - Medium.
- Communication - Medium
- Load Balancing - Automatic.
- Requires efficient task scheduling algo
- eg → Assigning customer service tickets

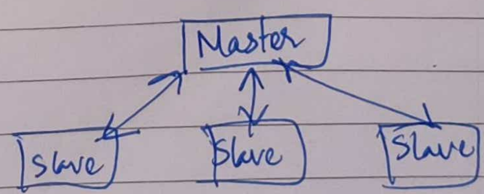


Realized as a global data structure

No. of processes are created

4) Master Slave Model

- Utilizes a master processor to distribute tasks to multiple slave processor for execution
- Data dependency - High (Master control task flow)
- communication - Medium.
- Load balancing - Manual (Master assigns tasks)
- Master distributes tasks & slaves execute
- eg → Video encoding



* Complexities.

	Sequential Complexity	Parallel Complexity
•	single processor execution	• Multiple processor execution
•	performance limited by single processor	• potential for improvement
•	Time complexity - Big O	• requires analysis of parallel execu
•	space complexity - often similar to time complexity	• similar with consideration of parallel execution
Resources →	Few computational resources	• may require more due to multiple processors
Scalability →	limited by single processor	• potential high scalability
Synchronization	Less concern	• Require synchronization me
Complexity Analysis	Simpler	• Complex
•	eg → linear search, Bubble sort	• eg → • Parallel matrix mult • Parallel sorting alg

* Anomalies in Parallel Algorithms -
↳ (unexpected performance in parallel algo

Causes → 1) Load Imbalance — unequal workload distribution.
2) Communication Overhead - too much communication b/w processors
3) Data dependencies — Improper handling of dependencies b/w data.
4) Race Conditions - conflicting access to shared resources.
5) Starvation - unable to access required resources
6) Concurrency bugs - errors from parallel execution
7) Cache Coherence Issues - Inconsistent memory views among processor or core
8) Inefficient Parallelization - overhead parallel execution → exceeds performance gains
9) Deadlocks - Processors or threads unable to progress due to circular dependencies on resources.