26/4

Page No.
Date

U-6 (HPC)

* Scope of Parallel Computing -
- Tackling large Problems
- Scientific & Engineering Applications.
- Data Processing & Analysis
- Advanced Graphics & Simulations.
- Financial Modeling & Risk Analysis
- Beyond Traditional computers.

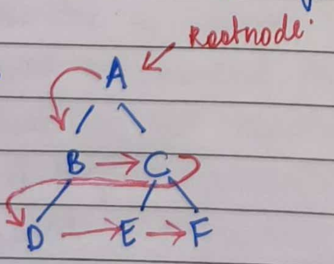
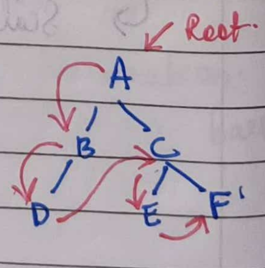
Parallel Sorting
- Bubble
- Merge

Distributed Computing.
- Document classification -
  - Frameworks
    - Kubernets
    - GPU Appl^{n}
    - Parallel Computing for AI/ML.

Parallel Search Algo
- BFS.
- DFS.

* Issues in sorting on parallel computers.
- where input & output is stored?
- How comparisons are perform

- Data Distribution.
- Communication Overhead
- Load Balancing.
- Algorithm choice.
- Granularity
- Synchronization
- Scalability

Breddth First Seach	Depth- First Search
• Uninformed Search	• Uninformed Seaed
• Queue based	• stack - based
• explores all neighbor nodes before moving deeper.	• explores as far as possible along each ~~node~~ branch
• QUEUE (FIFO)	• Stack (LIFO).
• more memory intensive.	• less memory intensive
• Level by level	• Follows one path deeply.
• slow for large graphs.	• fast for specific nodes.
• tree level by level	• subtree by subtree
• Appl^n → • finding shortest paths • min spanning trees. • connected components in unweighted graphs.	• Appl^n - • Topological sorting • finding cycles • pathfinding in some cases
• Is slow	• Is fast
• Backtracking isn't allowed	• Backtracking ~~are~~ is allowed
• space complexity is more critical critical as compared to time complexity.	• lesser space complexity, at a time needs store only single path from root to leaf node.
eg → 	eg → 
A, B, C, D, E, F	A, B, D, C, E F.

Parallel BFS	Parallel DFS.
Level wise exploration	Depth first exploration.
Queue (FIFO)	Stack (LIFO).
concurrent level explolation	concurant branch explolation.
typically balanced workload	potential load imbalance
Dense scalability (regular structures)	sparse, irregular structures.
efficient for high branching	efficient for low branching.
~~[illegible]~~ all neighbors of a level	Multiple paths concurrently.
distributed queue/work stealing	need to multiple stacks.
level managemt (synchronization)	need ed to avoid conflicts.
easier with a level	tricky for loadbalancing.
communication → high (distributed queue)	High (shared paths).
suitable for → shortest paths (unweighted, connected components.	cycles, connected components
scales well with efficient comm^n	Limited by communication.

	Parallel Bubble Sort	Parallel Merge Sort.
efficiency	Low	high.
Time complexity	$O(n^2)$	$O(n \log n)$.
Reasoning	Relies on adjacent elements, limiting independent work.	Divide & conquer for independent sub array sorting
datamovement	high due to element swapping.	lower due to merging sorted.
speedup	Limited (large datasets)	Significant.
stability	typically stable	stable
Commun^n	High	Lower
Scalability	Poor	Good
Better for Parallel sorting	No	Yes

* Distributed Computing

- processing tasks across multiple interconnected computers or nodes
- document classification is distributed comp appl^n into predefined classes or cate
- It leverages parallelism to large volumes of text data efficiently.
- distributes workload among multiple processing nodes.
- subset of documents, result are aggregated to produce final classif
- enhance scalability & performance by leveraging resources;
- Application - information retreival content filtering & sentiment analys
- distributed computing frameworks like Apache Hadoop & spark Facilitate distributed document classification.
- constributes to faster & more accurate information processing in diverse domains.

* Frameworks -

1) Kubernetes -

- open source system for handling development lifecycle.
- Helps in automating deployment, scaling & management of contanerzie app^n that also support distribute computing.
- Helps is not compromising security or reliability.
- saves time on infrastructure management.
- Acts as an orchestrator for containers.
- portable, extensible opensource platform for managing containerised workload
- designed to help developers & administrators manage app^n & services.
- works by dividing app^n & services into smaller isolated components called containers
- popular & widely used framework for automating deploment, scaling & manage
- features → automatic scaling, self healing & rolling update
  Integrates with range of tools.

App^n →
- service discovery & load balancing
- Distribute network traffic
- automated rollout & rollb
- automated bin packing
- self healing kubernet

* GPU computing

- Graphics ~~comp~~ processing unit (GPU) to perform highly parallel calculat
- essential to graphical rendering
- allows enormous parallelization.

strengths & weakness.

- → excels at some task may or may not
- → increases throughput of data & no. of concurrent calculation.
- → Arithmetic Intensity
- → good for GPU acc rate of math operation → 10 : 1.
- → high degree of parallelism.
- → sufficient GPU memory.

* GPU computing Appl^{n} –

- features numerous cores optimized for parallel processing for data intensive computat
- leverage parallelism to accelerate computations & handle large datasets
- CUDA & OpenCL → developers to harness GPU resource for parallel comp
- accelerate ML frameworks 'like Tensorflow & Pytorch'
- deployed in cloud environment.
- keeps workflows simple, portable & scalable.

1) Deep Learning.
2) Drug Design
3) Seismic Imaging
4) Automative design.
5) Astrophysics
6) option pricing
7) weather forecasting
8) Data Analytics
9) Machine Learning.

- Parallel computing for AI/ML - Kubeflow
  - ML toolkit for kubernetes
  - keeps workflows simple, portable & scalable
- supports Jupyter notebooks & tensorflow model convinient.
- adds postibility in ML Deploy
- GPU accelerate parallel programming ment
- Realtime influence & prediction

challenges →
- synchronization
- communication overhead
- load balancing

Odd even Jransposition-

- Sorts array into phases ie odd & even.

1) immediate right index pair.
2) compare exchange. (small goes towards left
   big no → towards right)
3) then iteration 2 (even)
4) consider even index.
5) 3rd iteratin (odd index).

after 8phases → output generated