Unit 2: Parallel Algorithm Design

Principles of algorithm design:

For designing parallel algos following steps are followed:

1. Find pieces of work that can be done concurrently.
2. Tasks are then mapped to multiple processors.
3. Management of access of shared data.
4. Synchronisation of processors at various points in parallel execution.

Decomposition, Tasks and Dependency graphs:

a) Decomposition

- Decomposition → Process of dividing a computation into smaller parts.
- Number and size of tasks → determines the granularity of the decomposition.
- Computation → Large num of small tasks → Fine grained
- Course grained → Small number of large tasks.

b) Tasks

- Sequential unit of computation.

c) Task dependency graphs

- Describe the relationship between tasks.

- Can be represented in a graph where nodes represent tasks and edges indicating the result required for processing the next.
- Features of a dependency graph
    - Critical path → Longest directed path betwen start and finish node.
    - Total work → Total work done by all nodes
    - Maximum degree of concurrency → Number of tasks executed simultaneously.
    - Average degree of concurrency → total work / critical path length.

Decomposition Techniques

1. Data decomposition:
    - Used to execute a prb having a lot of data parallely.
    - Data is partitioned across various tasks.
    - 2 steps: a) Data is divided into sub parts b) Subparts are executed differently by different processors.
    - eg Matrix multiplication.

2. Recursive Decomposition
    - Concurrency but in divide and conquer strategy.
    - Solve the prb directly if its small otherwise decompose the prb into subprbs to solve.
    - eg Merge sort8

3. Exploratory Decomposition
    - Decomposition goes hand in hand with execution.

- Prb is in running state and in the running state it is also decomposed.
- eg Puzzle prb

4. Speculative Decomposition
    - It is impossible to identify independent tasks when dependencies are not known in advance.
    - eg. Switch case

Characteristics of tasks and interactions

1. Generation of tasks
    a. Static task → Before computation if tasks are known its static task. eg. Data or recursive decomposition.
    b. Dynamic task → In some cases, tasks are to be executed dynamically based on decomposition of data.

2. Size of task
    a. Uniform task → same size tasks. eg. matrix multiplication
    b. Non uniform → eg. merge sort

3. Knowledge of task size → how much time would be required to complete the task
4. Data size

Mapping techniques for load balancing

1. Mapping techniques are used to solve overheads such as load imbalance and inter-process communication.
2. Techniques of mapping
    a. Static mapping → Tasks are distributed into processes, if sizes are unknown, serious imbalance can be caused. Hence static mapping only for those tasks whose size is known in advance.
    b. Dynamic mapping → tasks into processes at runtime, and 2 types of mapping
        - Centralised dynamic mapping → all tasks are maintained in a common data structure. Master → Managers the tasks, Slave → Depends on master to obtain work
        - When a slave process has no work, it takes a portoin of work from master.
        - Main prb → When many processes are used, master process may become bottleneck.
        - Distributed Dynamic mapping → Tasks are distributed among processes which exchange tasks at runtime to balance work.
        - Each process can send or recieve work from other processes.
        - Main prb → Sending and recieveing processes are paired together at runtime and who will initiate the work transfer.

Methods for containing interaction overheads

- Techniques to reduce the interaction overheads are:
    1. Maximising data locally
    2. Minimising contention and hot spots
    3. Overlapping computation and interaction.
    4. Replicating data or computation.
    5. Using optimised collective interaction operations.

a. Maximising data locality

- Some parallel prgs require access to common input data, which increases interaction overheads.
- The overhead can be reduced by promoting the use of local data that has been recently fetched.
- solutions → minimise the access frequency, maximise the reuse of recently accessed data.

b. Minimising contention and hot spots

- Contention occurs when → Multiple tasks access the same resource concurrently.
- Multiple simultaneous access to the same block.
- Multiple processes sending messages to the same process at the same time.

c. Overlapping computation with interaction

- The amount of time the processes spend waiting for shared data to arrive can be reduced by doing some useful computation during the time.
- Simplest technique is to initiate an interaction early so that its completed before its needed.

d. Replicating data or computation

- Replication of data → Replicate a copy of shared data structure
- All subsequent accesses to this data are free of overhead.
- Replicating computation → Effective to compute intermidiate results than to get them frm another process.

Parallel Algorithm Models

6 types of models:

1. Data Parallel model
2. Task graph model
3. Task pool model
4. Master/ Slave model
5. Pipeline/ producer consumer model
6. Hybrid model

1. Data Parallel Model
    a. Tasks are statically mapped onto processes, each task performs similar operations.
    b. Result of identical operations applied concurrently on data items is called parallelism.
    c. Work is done in phases.
    d. Interaction overhead can be minimised
    e. Degree of data parallelism increases with size.
    f. Solves large prbs efficiently.
    g. eg. Dense matrix multiplication

2. Task Graph Model
    a. The interrelationship between tasks is used to promote locality or to reduce interaction cost.

  b. Used to solve problems where amount of data associated with tasks is large.
  c. Tasks are statically mapped to optimise the cost.
  d. eg. Parallel quick sort
  e. This parallelism that is expressed by independent tasks in a task dependency graph → Task parallelism

3. Task pool model
   a. Dynamic mapping of tasks onto processes for local balancing.
   b. No premapping is done.
   c. Pointer to task may be stored physically in a shared list or a queue, hash table, tree.
   d. Used when amount of data associated with task is small.

4. Master - Slave model
   a. One or more master processes generate work and allocate to worker processes.
   b. Tasks can be allocated prior if manager can estimate the size of the task.
   c. Workers are assigned small pieces of work at different times.
   d. Its a hierarchial model in which top level manager sends chunks of tasks to low level managers.
   e. Used to message passing paradigm.

5. Pipeline / Producer consumer problem
   a. A stream of data is passed on through the sucession of processes each of which performs the same task.

b. Simultaneaous execution of different programs of a data stream is called stream parallelism.

c. Pipeline is a chain of producer and consumer.

d. Each process in the pipeline can be viewd as a consumer of sequence of data for process preceding it and producer of data for processor following it.

6. Hybrid Model
    a. More that one model
    b. Composition of multiple models.
    c. Task dependency graph is used
    d. Eg. Parallel quicksort

Complexities

1. Sequential and Parallel Computational Complexity
    a. Algorithm is evaluated on the basis of its Time and Space complexity.
    b. Sequential complexity → Amount of time and resources to execute on a single processor.
    c. Parallel complexity → Amount of time and resources to execute on multiple processors simultaneously.
    d. TC → Number of steps required to complete the algorithm as a function of input size. Big O notation is commonly used.
    e. Parallel algorithm complexity is measured in terms of speedup, which is ratio of time required to execute sequentially to the time required to execute in parallel.

f. Parallel speedup → influenced by factors like number of processors, communication overhead and granularity.

- Speedup : How well it performs.
    - Speedup = Worst case execution time of fastest known sequential algorithm / Worst case execution time of a parallel algorithm.
- Number of processors used: Cost of the solution is directly proportional to the number of processors an algorithm uses to solve a problem.
    - As number of processors increase , TC decreases
    - However in some cases the increasing the number might not provide a significant impact in performance.
    - Therefore the number of processors depends on the size of the problem, nature of algo, hardware and computation goals.
- Total cost → TC * Number of processors used
    - Also takes into account the cost of using multiple processors.
    - With TC, optimal number of processors to use will depend on size of prb to be solved, nature of algo.
- Work Efficiency
    - If an algo performs the same amount of work as the fastest sequential algorithm within a constant factor we refer it to be work efficient.

2. Anomalies

a. Refer to unexpected behavior that can occur during execution of parallel algos.

b. Common anomalies that occur:

- Deadlock - When 2 or more processes are blocked waiting for each other to release a resource.
- Race condition - Outcome of a operation depends on the timing or order in which proceses are executed, when 2 or more processes access a shared resource simultaneously.
- Starvation: Process is unable to access a shared resource.
- Load imbalance: Occurs when some processes are heavily loaded.
- Communication overhead: Time and resources required to exchange data between parallel processes.

c. Deadlock :

- Deadlocks occur in parallel algos due to improper resource allocation or insufficient synchronisation.
- Solution → Use locks and semaphores to enforce mutual exclusion.
- Also its important to release resource when they are no longer needed.
- Careful testing and debugging help identify potential deadlocks.

d. Race Conditions

- Race conditions occur due to incorrect use of locks and semaphores.
- Solution → Carefully design parallel algorithms.
- Usage of synchronisation primitives such as locks, semaphores, barriers to coordinate and ensure that only 1 thread can access a resource at a time.

e. Starvation

- Starvation occurs when multiple processes or threads are competing for limited resources such as time, memory and bandwidth.
- To prevent starvation → Resource allocation policies, priority scheduling and fair queueing.

f. Load Imbalance

- Load balancing techniques such task stealing, where idle processing elemnts take on additional tasks of busy ones, or task migration where tasks are moved from overloaded processing elements to idle ones, dividing workload into smaller chunks.

g. Communication overhead

- Refers to the additional time and resources required for processing elements.
- To prevent this, techniques such as message passing, shared memory is used.
- Message passing → Sending and receiving data between elements,
- Shared memory → Common memory space