

HPC – Unit 2 (Parallel Algorithm Design) – IN-SEM PYQ Answers

Q1. Explain any three-decomposition technique with example. [6]

Q2. Explain with example: i) Recursive decomposition, ii) Data decomposition, iii) Exploratory decomposition [6]

1) Recursive Decomposition

In **Recursive Decomposition**, a problem is divided into smaller sub-problems recursively until the base condition is reached. Each sub-problem can be solved in parallel.

- Based on divide-and-conquer strategy
- Suitable for hierarchical problems

Example:

Parallel Merge Sort

- Divide array into two halves
- Sort each half recursively in parallel
- Merge sorted halves

Used in tree-based computations and quicksort algorithms.

2) Data Decomposition

In **Data Decomposition**, the data is partitioned into smaller segments, and the same operation is performed on each segment in parallel.

- Suitable for data-parallel applications
- Same computation applied to different data blocks

Example:

Matrix Addition

- Divide matrix rows among processors
- Each processor computes addition for its assigned rows

Used in matrix multiplication, image processing, and vector operations.

3) Exploratory Decomposition

In **Exploratory Decomposition**, parallelism is achieved by exploring different parts of a solution space simultaneously.

- Used in search problems
- Each processor explores a different branch

Example:

Parallel Depth First Search (DFS)

- Different processors explore different branches of a search tree
- Stops when solution is found

Used in AI search problems, combinatorial optimization, and game tree exploration.

4) Speculative Decomposition

In **Speculative Decomposition**, processors execute tasks that may or may not be needed in future.

- Used when future control flow is uncertain
- If speculation is correct → time saved
- If incorrect → results discarded

Example:

Speculative execution of branches in parallel.

5) Hybrid Decomposition

In **Hybrid Decomposition**, two or more decomposition techniques are combined.

- Improves load balancing
- Used in complex parallel algorithms

Example:

Matrix multiplication using data decomposition within recursive decomposition.

Q3. Describe mapping technique for load balancing? [5]

Q4. Explain classification of Dynamic mapping techniques. [5]

Q5. Explain different schemes for Static Mapping. [5]

1. Static Mapping

- In static mapping, clients are assigned fixed mappings to servers, which remain unchanged over time.
- For example, clients may be evenly distributed across servers based on their IP addresses or geographical locations.
- While static mapping offers simplicity in setup and management, it may lead to uneven loads on servers, especially if the workload distribution changes over time.

Advantages:

- Easy to apply and understand as well.
- Predictable distribution of tasks.

- Important Factors:
- It is most appropriate where the demand for the resource is stable and the amount of work required does not fluctuate greatly.
- It is not scalable as it can only be designed to cope with a limited amount of work at any given time.

Use Cases:

- Stable workload distribution.
 - Well-understood and predictable workload characteristics.
 - Fixed number of clients or servers.
 - Suitable for small-scale environments with consistent traffic patterns.
- Different schemes of static mapping:
 - 1) Block Mapping
 - Tasks are divided into contiguous blocks.
 - Each processor is assigned one block of tasks.
 - Example: If 100 tasks and 4 processors → each processor gets 25 consecutive tasks.
 - Advantages:
 - Simple implementation
 - Good for uniform workload
 - Limitation: Load imbalance if task execution times vary
 - 2) Cyclic Mapping
 - Tasks are distributed to processors in round-robin fashion.
 - Processor P_i gets tasks: $i, i+p, i+2p, \dots$
 - (where p = number of processors)
 - Example:
For 8 tasks and 4 processors:
 $P_0 \rightarrow T_0, T_4$
 $P_1 \rightarrow T_1, T_5$
 $P_2 \rightarrow T_2, T_6$
 $P_3 \rightarrow T_3, T_7$
 - Advantage: Better load balancing for non-uniform tasks
 - 3) Block-Cyclic Mapping
 - Combination of block and cyclic mapping.
 - Tasks are divided into small blocks and blocks are assigned cyclically.
 - Advantage:
 - Balances load while preserving locality
 - Widely used in parallel matrix computations

2. Dynamic Mapping

- Dynamic mapping adjusts mappings dynamically based on current server loads, network conditions, or other factors.
- For instance, a dynamic load balancer continuously monitors server loads and redirects incoming requests to less busy servers to ensure balanced utilization.
- Dynamic mapping offers better load balancing and scalability compared to static mapping but requires more overhead for monitoring and reconfiguration.

Advantages:

- Adaptable to changing workloads.
- Better resource management requires proper alignment with one's abilities, skills, and knowledge.
- Important Factors:
 - requires extensive tracking and the optimal use of complicated patterns.
- The main disadvantage of this method is that it can introduce overhead due to constant changes being made.

Use Cases:

- Fluctuating or unpredictable workload distribution.
- Large-scale web applications or cloud environments.
- Continuous monitoring of server loads and network conditions.
- Dynamic adjustment of mappings to optimize resource utilization and performance.
- Classification of Dynamic mapping techniques:
 - 1) Centralized Dynamic Mapping: In centralized mapping, a single master processor maintains global load information and assigns tasks to processors.
 - Working:
 - Master keeps track of task queue
 - Idle processors request tasks
 - Master allocates tasks dynamically
 - Characteristics:
 - Simple implementation
 - Global knowledge of system state
 - Low decision overhead
 - Limitations:
 - Single point of failure
 - Scalability limited
 - Bottleneck at master node
 - Example: Master-worker model in parallel processing.
 - 2) Distributed Dynamic Mapping: In distributed mapping, each processor participates in task allocation without a central controller.
 - Working:
 - Processors maintain local load information
 - Idle processor requests tasks from other processors
 - Load is shared through cooperation
 - Characteristics:
 - No central bottleneck
 - Scalable for large systems
 - Higher communication overhead
 - Advantages:
 - Better fault tolerance
 - Suitable for large distributed systems

3. Hierarchical Mapping

- Hierarchical mapping organizes servers into a hierarchical structure, such as clusters or tiers, where requests are first directed to higher-level nodes before being forwarded to specific servers.
- This approach enables scalability and fault tolerance by distributing the load handling responsibilities across multiple levels of the hierarchy.
- For example, a request might first be directed to a regional node, then to a data center, and finally to a specific server within the data center.

Advantages:

- Scalable for large systems.
- The reason is that they can manage even the most complex of environments effectively and are capable of devising effective workaround solutions on the fly.

Important Factors:

- As discussed earlier, there is a need to pay more attention to the planning of the hierarchy.
- Ideal for multiple levels or tiers.

Use Cases:

- Large-scale distributed systems or networks.
- Multiple tiers or layers of infrastructure.
- Cloud computing environments or content delivery networks (CDNs).
- Scalable and fault-tolerant load distribution across hierarchical structures.

4. Hash-Based Mapping

- Hash-based mapping calculates a hash value for each incoming request and uses it to determine the target server.
- This ensures consistent mapping for the same request, enabling easy scalability and fault tolerance as servers can be added or removed without affecting the mapping of existing requests.
- For example, a consistent hash function may be used to map requests to servers based on their content or metadata.

Advantages:

- Simple and efficient distribution.
- Ensures even load balancing.

Important Factors:

- It depends on the hash function used. That means that the degree to which collisions can occur mainly depends on the hash function used.
- May need to be shuffled if the number of resources increases or decreases on the platform.

Use Cases:

- Consistent and deterministic mapping of requests to servers.
- Scalability and fault tolerance.
- Suitable for distributed systems where servers can be added or removed dynamically.
- Commonly used in distributed databases, content delivery networks (CDNs), and peer-to-peer networks.

5. Adaptive Mapping

- Adaptive mapping employs algorithms that adaptively adjust mappings based on dynamic factors such as server loads, network conditions, or historical usage patterns.

- These algorithms continuously analyze the workload distribution and adjust mappings to optimize load distribution and resource utilization.
- Adaptive mapping enhances performance and efficiency by dynamically optimizing load balancing strategies for changing workload conditions, ensuring optimal resource allocation and responsiveness.

Advantages:

- Highly efficient and responsive to changes.
- an optimize resource utilization based on predictive analytics.
- Important Factors:
- Requires sophisticated algorithms and data analysis.

Best suited for environments with highly variable and unpredictable workloads.

Use Cases:

- Dynamic workload conditions.
- Optimization of load balancing strategies based on real-time data.
- Continuous adaptation to changing workload patterns.
- Enhanced performance and efficiency in highly dynamic environments.

Q6. What are characteristics of task, interaction and Inter-Task Interactions? [4 each]

1) Characteristics of Task [4]

A **task** is a unit of computation in a parallel program.

Characteristics:

- **Computation Size:** Amount of work performed by the task (granularity).
- **Task Dependencies:** May depend on results of other tasks.
- **Execution Time:** Can be uniform or variable across tasks.
- **Task Type:** Can be independent or cooperative.
- **Granularity:** Fine-grained (small tasks) or coarse-grained (large tasks).

2) Characteristics of Interaction [4]

Interaction refers to communication or synchronization between tasks.

Characteristics:

- **Frequency of Communication:** How often tasks exchange data.
- **Volume of Data:** Amount of data transferred during communication.
- **Synchronization Requirement:** Blocking or non-blocking communication.
- **Communication Pattern:** One-to-one, one-to-many, many-to-many.

- **Locality:** Local or remote communication.

3) Characteristics of Inter-Task Interactions [4]

Inter-task interaction defines the relationship and communication structure among tasks.

Characteristics:

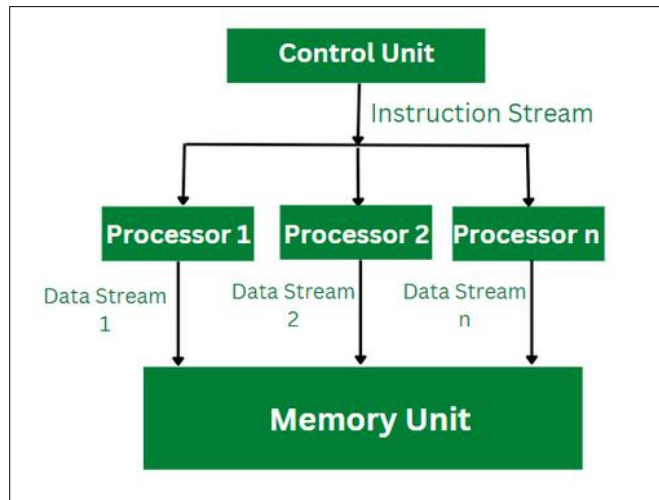
- **Static vs Dynamic Interaction:** Fixed communication pattern or runtime-dependent.
- **Regular vs Irregular Pattern:** Structured (e.g., grid) or unstructured communication.
- **Degree of Dependency:** Strongly dependent or loosely coupled tasks.
- **Communication-to-Computation Ratio:** Determines efficiency and scalability.
- **Synchronization Overhead:** Impacts overall parallel performance.

These characteristics influence load balancing, scalability, and performance of parallel algorithms.

Q7. Explain in detail parallel algorithm models? [6]

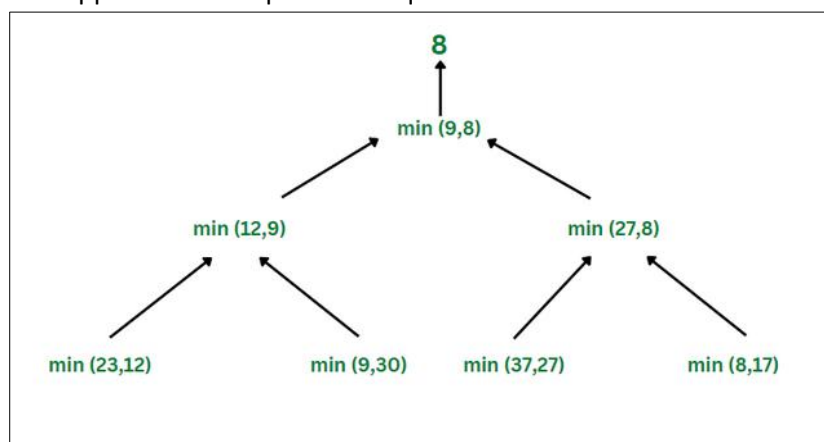
1. The Data-Parallel Model

- The data-parallel model algorithm is one of the simplest models of all other parallel algorithm models.
- In this model, the tasks that need to be carried out are identified first and then mapped to the processes.
- This mapping of tasks onto the processes is being done statically or semi-statically.
- In this model, the task that is being performed by every process is the same or identical but the data on which these operations or tasks are performed is different.
- The problem to be solved is divided into a number of tasks on the basis of data partitioning.
- Here data partitioning is being used because all the operations performed by each process are similar and proper uniform partitioning of data followed by static mapping assures the proper load balancing.
- Example: Dense Matrix Multiplication:
 - The instruction stream is being divided into the available number of processors.
 - Each processor computes the data stream it is allocated with and accesses the memory unit for read and write operation.
 - As shown in the below figure, the data stream 1 is allocated to processor 1, once it computes the calculation the result is being stored in the memory unit.



2. The Task Graph Model

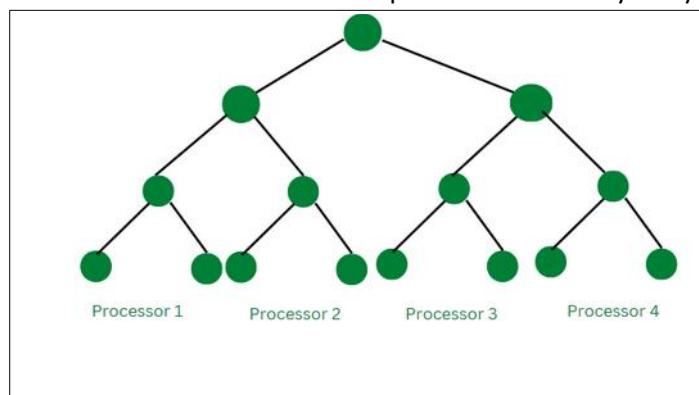
- The task dependency graph is being used by the parallel algorithms for describing the computations it performs.
- Therefore, the use of interrelationships among the tasks in the task dependency graph can be used for reducing the interaction costs.
- This model can be used effectively for solving problems in which tasks are associated with a large amount of data as compared to that actual computation.
- The parallelism that is described with the task dependency graph where each task is an independent task is known as task parallelism.
- The task graph model is majorly used for the implementation of parallel quick sort, a parallel algorithm based on divide and conquer.
- Example: Finding the minimum number
 - The task graph model works parallelly in order to find the minimum number in the given stream.
 - As shown in the above figure, the minimum of 23 and 12 is computed and passed on further by one process, similarly at the same time the minimum of 9 and 30 is calculated and passed on to the further process.
 - This approach of computation requires less time and effort.



3. Work Pool Model

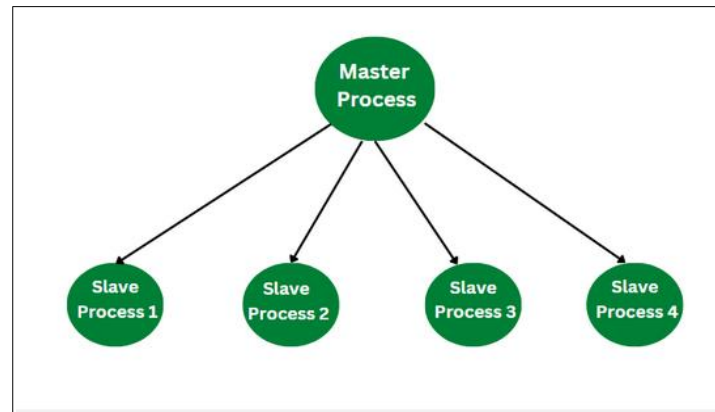
- The work pool model is also known as the task pool model.

- This model makes use of a dynamic mapping approach for task assignment in order to handle load balancing.
- The size of some processes or tasks is small and requires less time.
- Whereas some tasks are of large size and therefore require more time for processing. In order to avoid inefficiency load balancing is required.
- The pool of tasks is created. These tasks are allocated to the processes that are idle in the runtime.
- This work pool model can be used in the message-passing approach where the data that is associated with the tasks is smaller than the computation required for that task.
- In this model, the task is moved without causing more interaction overhead.
-
- Example: Parallel tree search:
- that uses the work pool model for its computation uses four processors simultaneously.
- The four sub-trees are allocated to four processors and they carry out the search operation.



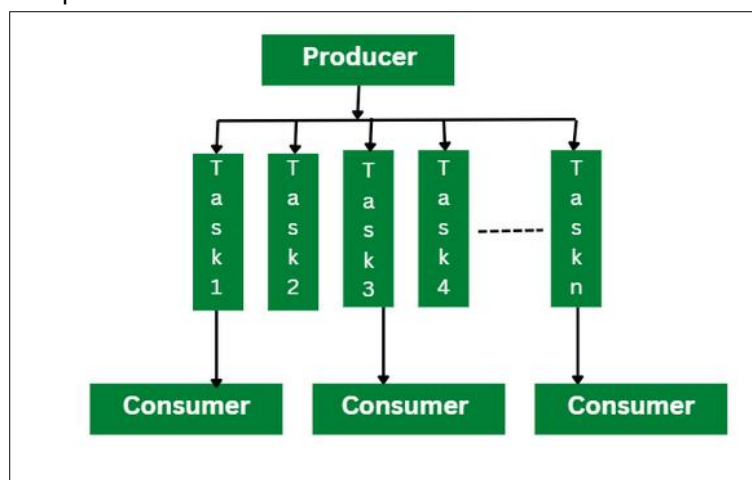
4. Master-Slave Model

- Master Slave Model is also known as Manager- worker model. The work is being divided among the process.
- In this model, there are two different types of processes namely master process and slave process.
- One or more processes acts as a master and the remaining all other processes act as a slave.
- The master allocates the tasks to the slave processes according to the requirements.
- The allocation of tasks depends on the size of that task. If the size of the task can be calculated on a prior basis the master allocates it to the required processes.
- If the size of the task cannot be calculated prior the master allocates some of the work to every process at different times.
- The master-slave model works more efficiently when work has to be done in different phases where the master assigns different slaves to perform tasks at different phases.
- In the master-slave model, the master is responsible for the allocation of tasks and synchronizing the activities of the slaves.
- The master-slave model is generally efficient and used for shared address space and message-passing paradigms.
- Example: Distribution of workload across multiple slave nodes by the master process
 - The distribution of workload is being done across multiple processes.
 - As shown in the below diagram, one node is the master process that allocates the workload to the other four slave processes.
 - In this way, each sub-computation is carried out by multiple slave processes.



5. The Pipeline Model

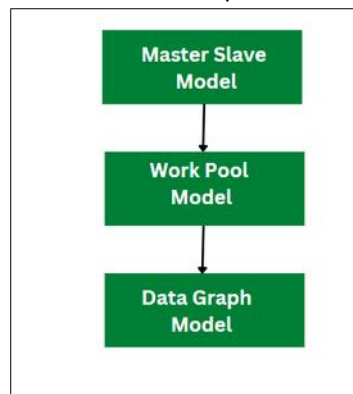
- The Pipeline Model is also known as the Producer-Consumer model.
- This model is based on the passing of a data stream through the processes that are arranged in succession. Here a single task goes through all the other processes.
- They are then accessed by the required processes in a sequential manner.
- Once the processing of one process is finished it goes to the next present process.
- In this model, the pipeline acts as a chain of producers and consumers.
- This pipeline of producers and consumers can also be arranged in a directed graph-like fashion rather than a linear chain.
- The approach of Static mapping is being used for mapping of tasks onto the processes.
- Example: Parallel LU factorization algorithm:
 - The Parallel LU factorization algorithm uses the pipeline model.
 - In this model, the producer reads the input matrix and generates the tasks that are required for computing the LU factorization as an output.
 - The producer divides this input matrix into a smaller size of multiple tasks and shares them into a shared task queue.
 - The consumers then retrieve these blocks and perform the LU factorization on each independent block.



6. Hybrid Model

- A hybrid model is the combination of more than one parallel model.

- This combination can be applied sequentially or hierarchically to the different phases of the parallel algorithm.
- The model that can be efficient for performing the task is selected as a model for that particular phase.
- Example: A combination of master-slave, work pool, and data graph model.
 - As shown in the below hybrid model where three different models are used at each phase master-slave model, the work pool model, and the data graph model.
 - Consider the above example where the master-slave model is used for the data transformation task.
 - The master process distributes the task to multiple slave processes for parallel computation.
 - In the second phase work pool model is used for data analysis and similarly data graph model is used for making the data visualization.
 - In this way, the operation is carried out in multiple phases and by using different parallel algorithm models at each phase.



Q8. Explain the different methods for Containing Interaction Overheads. [5]

1) Maximizing Data Locality

Interaction overhead decreases when tasks use local data as much as possible, reducing the need for communication.

- **Minimize Volume of Data-Exchange:**
Structure data partitioning so that each processor accesses fewer shared data elements. For example, higher-dimensional block distributions in matrix operations decrease the total amount of non-local data accessed, lowering interaction cost.
- **Minimize Frequency of Interactions:**
Combine multiple small communication requests into fewer larger ones. Fewer interactions mean the costly startup overhead for communication is amortized over larger data exchanges, reducing overall overhead.

2) Minimizing Contention and Hot Spots

Contention occurs when many processors simultaneously access the same memory location, link, or communication resource, leading to queuing delays.

- Redesign communication and data access patterns so processors avoid simultaneous requests to the same resource.
- For example, in parallel matrix multiplication, reorganizing block accesses prevents all processors from reading the same block at once, eliminating contention.
- Choosing distributed task mapping over centralized mapping also reduces shared resource contention.

3) Overlapping Computations with Interactions

Overlap hides communication latency by performing useful work while communication is in progress.

- Start communication earlier than needed so data arrives by the time it's required.
- Use non-blocking communication primitives (e.g., **MPI_Isend**, **MPI_Irecv**) so the process can compute while messages are in transit.
- On shared-memory systems, prefetch hardware and asynchronous loads help overlap memory accesses with computation.

4) Replicating Data or Computations

Replication reduces repeated remote access by making local copies of commonly accessed data or by replicating computation to avoid communication.

- Frequently read data structures (e.g., lookup tables) are replicated across processors, so subsequent accesses are local.
- Replicating computation for intermediate results may be cheaper than fetching results from remote processors.
- Drawback: increased memory usage, so replication is selective.

5) Using Optimized Collective Interaction Operations

Collective operations handle communication among many processors in a coordinated manner with efficient algorithms.

- Optimized implementations of broadcast, reduction, gather, scatter, etc., use tree-based or pipelined communication to reduce the number of sequential interactions.
- Collective operations minimize redundant communication by aggregating data and exploiting network topology.
- Example: binary tree broadcast sends data in $\log(p)$ steps instead of $p-1$ point-to-point sends.

6) Overlapping Interactions with Other Interactions

Multiple communications can be arranged to proceed in parallel or in pipelined phases.

- When communication patterns allow it, schedule them so that one message is sent while another is being received.
- Pipeline communication stages so that interaction chains overlap rather than block.
- This improves network utilization and reduces idle waiting for communication completions.