SMVS
Page No.
Date : / /

Partitioning and Divide-and-Conquer Strategies

1 Discuss Partitioning Strategies with domain and functional decomposition. Consider the problem of adding n numbers on m processors.

=> Partitioning:

Partitioning divides a problem into smaller parts to enables concurrent execution, either of data or function.

There are Main Two Types of Partitioning.

(1) Domain Decomposition
(2) Function Decomposition

1 Domain Decomposition:

Domain Decomposition is also known as Data Partitioning.

In the Partitioning, data is divided into smaller segments and each processed by a separate process.

This method is simple and efficient For large datasets.

Date : / /

This is preferred for problems with large data that can be divided into independent part.

-> Example: Adding n Numbers on m Processors.

Step 1: Divide the n ~~m~~ numbers into m parts.

Step 2: Assign each part to a separate processor

Step 3: Combine the Partial sums.

2 Functional Decomposition:

It divides the program into independent functions and each executed concurrently on different processor.

It can be used where tasks can be divided into distinct functions.

It Provides Flexibility in the design of parallel solutions.

SMVS
Page No.
Date : / /

-> Example: Adding n Numbers on m Processors.

Instead of dividing the data, the addition process itself could be split into different functions.

Each Function executes independently and the results are combined after all functions are completed.

2 How would you do Partitioning in a master-slave approach?

=> Partitioning using master-slave approach involves dividing the workloads into smaller, independent tasks.

It distributing them among multiple slave processes and then aggregating the results.

-> Steps :

1 Divide the workloads:

The master process splits the

input data into partitions, each assigned to a specific slave.

2 Distribute the Data:

- Individual Sends: The master sends each partition directly to the corresponding slave using send().

Each slaves receives its assigned data via recv().

- Broadcast / Multicast: Alternatively, the master broadcast the entire dataset to all slaves.

3 Process Data on Slaves:

Each slaves perform its assigned computation.

4 Aggregates Results:

The slaves send their results back to the master using send().

The master accumulates these partial results to compute the Final output.

-> Example : Add n numbers.

Master:

```
s = n/P;
For (i=0; x=0; i<P; i++,
        x = x+s)
    send(&numbers[x], s, P;);
sum = 0;
For (i=0; i<P; i++)
    recv(&part_sum, PANY);
    sum = sum + part_sum;
```

Slave:

```
recv(numbers, s, Pm);
part_sum = 0;
For (i=0; i<s; i++)
    part_sum = part_sum +
               numbers[i];
send(&part_sum, Pm);
```

3 Explain M-ary Divide and Conquer method in detail.

-> The M-ary Divide and Conquer method

Date : / /

is a generalized from of the traditional divide and conquer method.

In this method, the problem is recursively broken into m part, each of which is solved independently.

The solution to the sub-problems are then combined to solve the original problem.

-> Method :

1 Division into Multiple Parts :

In M-ary method, task or problem is divided into m sub-problems.

2 Recursive Definition :

Each sub-problem is solved recursively.

3 Tree Representation :

The recursive division forms an m-ary tree, where each node has m childream.

SMVS
Page No.
Date : / /

-> Example: Add Four Numbers.

```
int add(int *s)
{
  if(n(s)=<4)
    return(n1+n2+n3+n4);
  else
  {
    Divide(S, S1, S2, S3, S4);
    p_S1 = add(S1);
    p_S2 = add(S2);
    p-S3 = add(S3);
    p-S4 = add(S4);
    return(p-S1+p-S2+p-S3
           +p-S4);
  }
}
```

-> Application:

1 Used in Image Processing

2 3D Space Division

3 Used For Parallel Computing.

4 Explain Sorting Using Bucket Sort in detail using divide and conquer.

=> Bucket sort is a sorting algorithm that divides input numbers into several "buckets" and sorts each bucket individually.

The final sorted list is obtained by concatenating all sorted buckets.

=> Sequential Algorithm:

1 Initialize Buckets:

Divide the range of input numbers [0, a-1] into m equal bucketes.

Each bucket corresponds to an interval.

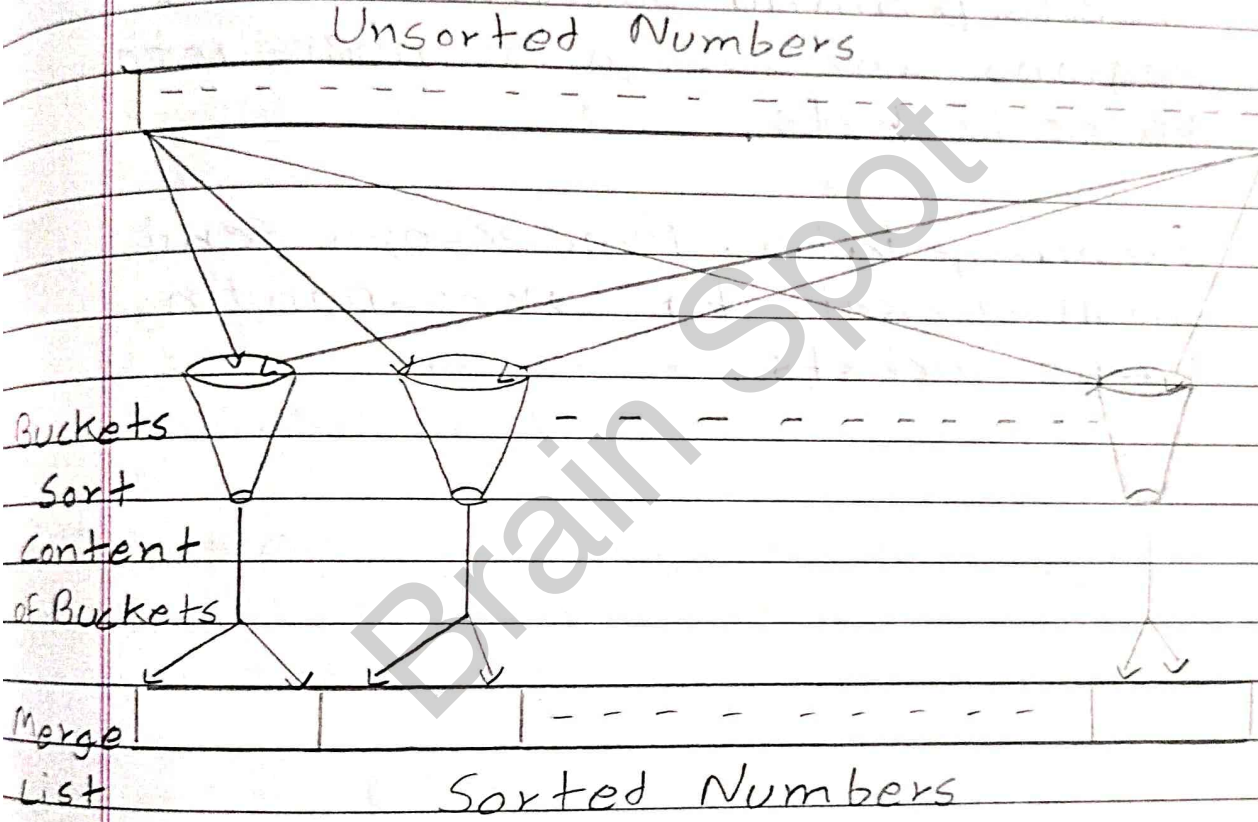
2 Distribute Numbers:

Place each number in the approximate bucket using the formula:

$$\text{Bucket Index} = \left\lfloor \text{number} \times \frac{m}{a} \right\rfloor.$$

3 Sort Each Bucket:

Use a sequential sorting algorithm to sort the numbers within each bucket.



4 Concatenate Buckets:

Combine the sorted numbers from all buckets to from the final sorted List.

Date : / /

=> Parallel Algorithm:

1 Partition Numbers: Divide the input List into p subsets, one for each processor.

2 Assign Small Buckets: Each processor creats p small buckets for its regions and assign numbers into these buckets.

3 Exchange data: Processors send small buckets to corresponding Large buckets.

Each Large bucket receives data from all processors.

4 Sort Large Buckets: Each processor sorts the numbers in its assigned Large bucket sequentially.

5 Merge Bucket: Concatenate all large buckets to from the final sorted List.

5 Explain Numerical Integration Problem in detail.

=> Numerical Integration involves approximating the value of a definite integral:

$$I = \int_a^b f(x) \cdot dx$$

The divide and Conquer strategy is commonly used in numerical integration.

The idea is to divide the interval [a, b] into smaller subintervals, approximate the area in each

subinterval and then combine these approximation to estimate the total area.

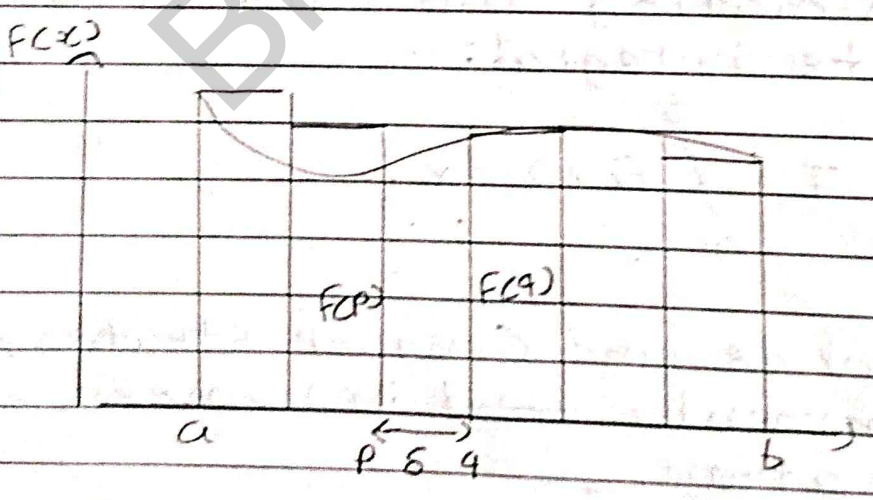
=> Methods:

1 Recangle Approximation:

The interval [a, b] is divided into smaller subintervals of width δ.

Each subinterval is approximated as a rectangle:

Area of Rectangle = f(p) · δ



2 Static Assignment

In this, we have to use Trapezoidal method.

SMVS
Page No.
Date : / /

Instead of rectangles, each subinterval is approximated using a trapezoid.

Area of Trapezoid = $\frac{1}{2}(f(p)+f(q)) \cdot \delta$

This method ensures that the error because as the number of subintervals n increases.

6 Explain N-Body Problem.

=> The N-Body problem deals with calculating the motion of N bodies influenced by mutual gravitational forces.

-> Gravitational Force:

The Force between two bodies of masses m_a and m_b, separated by a distance r,

$$F = \frac{G m_a m_b}{r^2}$$

-> Newton's Second Law:

For a body of mass m experiencing

a Force F, the resulting acceleration is,

$$F = ma$$

→ Three-Dimensional Space:

(i) Distance Between Two Bodies

If body a is at (x_a, y_a, z_a) and body b is at (x_b, y_b, z_b),

$$\text{Distance } r = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}$$

(ii) Gravitational Force Components:

The Gravitional Force is resolved into x, y and z components:

$$F_x = \frac{G m_a m_b}{r^2} \cdot \frac{x_b - x_a}{r}$$

$$F_y = \frac{G m_a m_b}{r^2} \cdot \frac{y_b - y_a}{r}$$

$$F_z = \frac{G m_a m_b}{r^2} \cdot \frac{z_b - z_a}{r}$$

7 Barnes-Hut Algorithm:

=) The Barnes-Hut Algorithm, introduced in 1986 by Barnes and Hut.

It is a divide-and-conquer method for solving the N-body problem efficiently.

-> Steps:

1 Tree Construction:

The simulation space is divided recursively into smaller regions until each region contains at most one body.

2 Mass and Center of Mass Calculation:

For each node in the tree, the total mass and center of mass of all bodies within that region are calculated recursively.

3 Force Calculation:

For each body, the tree is traversed starting from the root.

Page No.
Date : / /

clustering approximation is applied to compute the force from that region.

4 Position and Velocity Update:

Forces computed for each body are used to update their positions and velocities.

```
For(t=0; t<tmax; t++)
{
    Build_Octtree();
    Tot_mass_Center();
    Comp_Force();
    Update();
}
```

8 Explain Tree construction using divide and conquer.

=> Tree Construction is a visualization of the divide-and-conquer approach applied to solving problems by recursively breaking them into smaller subproblems.

SMVS
Page No.
Date : / /

→ Steps :

1 Root Node :

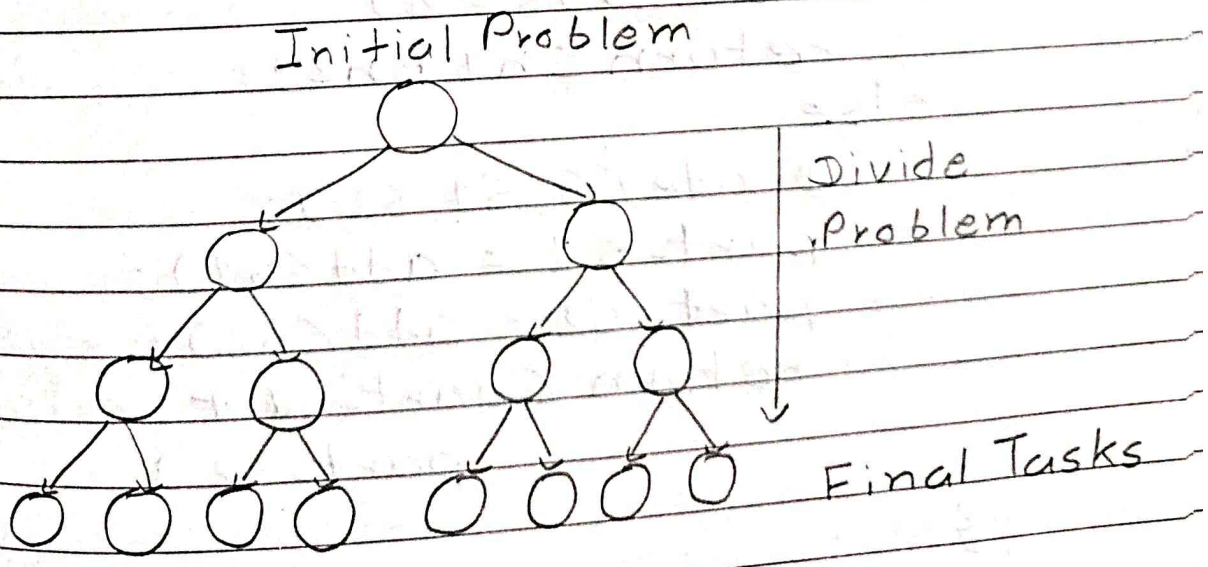
The process begins at the root of the tree, which represents the original problem.

The root divides the problem into two smaller subproblems.

2 Recursive Division :

Each subproblem further divides into smaller subproblem.

The recursive division continues until the problems cannot be divided further.



Page No.
Date :

3 Combining Results:

Once the computation at the leaves are complete, the results are passed back up the tree.

At each level, the result from the two childrean nodes are combined to produce the result for their parent node.

This continues upward until the root node has the final solution.

Ex. Adding a List of numbers (Sequential)

```
int add(int *s)
{
  if(num(s)<=2)
     return(n1+n2);
  else
     Divide(S, S1, S2);
     part_S1 = add(S1);
     part_S2 = add(S2);
     return(part_S1 +
                part_S2);

}
```