Name: A. Shiva Surya Saran
Roll No.: BE-B 20

High Performance Computing

Assignment 2

Unit 2: Parallel Programming

1. Explain the Principles of Parallel Algorithm Design.

An algorithm provides step by step solution of the given problem. An algorithm accepts input from the user and based upon the input, output is given after performing the defined computations.

The basic steps in the design of parallel algorithm are:

- Partitioning of overall computation into smaller computation and
- Assignment of these smaller computation into different processors.
- Identify portions of work that can be performed concurrently
- Map concurrent portions of work onto multiple processes running in parallel
- Distribute a program's input, output, and intermediate data
- Manage accesses to shared data: avoid conflicts
- Synchronize the processes at stages of the parallel program execution

2. Explain Decomposition, Tasks and Dependency Graphs.

Decomposition:
Divides a computation into smaller parts, which can be executed concurrently.

Task:
Programmer-defined units of computation.

Task-dependency graph:
A decomposition can be illustrated in the form of a directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called a task dependency graph. Node represents task. Directed edge represents control dependence.

3. What is Mapping Techniques for Load Balancing?

- The mapping technique used to map task in 2 processes so that parallel execution can be performed.
- The overall computation associated in problem to be solved is hey divided into number of tasks.
- These tasks are mapped in 2 processes with the goal of completing execution at minimum amount of time.

The techniques are as follows:

Static Mapping

In this technique, the mapping of task in 2 processes is performed before execution of algorithm. This indicates that tasks are distributed among available processes prior to the execution of algorithm.
Good mapping in this case depends on:

i. size of data associated with tasks
ii. characteristics of inter-task interactions
iii. parallel programming paradigm.
iv. Knowledge of task sizes

Schemes for Static Mapping:

1. Mappings Based on Data Partitioning
   Array Distribution Schemes
   i. Block Distributions
   ii. Cyclic and Block-Cyclic Distributions
   iii. Randomized Block Distributions
   Graph Partitioning
2. Mappings Based on Task Partitioning
3. Hierarchical Mappings

Dynamic Mapping

In this technique mapping of task onto the processes is performed during the execution of the algorithm. This indicates that the task are distributed among available processors when algorithm is actually executing.

These are basic situations with dynamic mapping supplied. The first cases if tasks are generated dynamically then this mapping techniques are used. Dynamic technique is applied in situations where large amount of data is associated with tasks. In this situation dynamic mapping most the data among available processors.

Schemes for Dynamic Mapping:

1. Centralized Schemes

--Master Process
--Slave Processes

2. Distributed Schemes

Critical parameters of a distributed load balancing scheme are as follows:

1. How are the sending and receiving processes paired together?
2. Is the work transfer initiated by the sender or the receiver?

3. How much work is transferred in each exchange?
4. When is the work transfer performed?

4. Write a short note on Data-decomposition and Recursive decomposition.

Data Decomposition

- Large data sets involved in the problem can be divided into small parts and pieces independently by several computers concurrently.
- Data decomposition is common technique used for concurrent processing of data.
- There are basically 2 steps in this technique the first step is the overall input data is required for performing the defined computation is divided into multiple parts.
- In the second step, it is based upon the division of overall computation into number of tasks handled on the partitioned data.
- The actual operations are implemented on the tasks a similar but data upon which these operations work are different.

Example:
Matrix multiplication:
consider the problem of multiplying 2 N x N matrices. A and B to yield matrix C. The output matrix C can be partitioned into 4 tasks given below. Here each task computes one element of result matrix.

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

$$\begin{aligned} &\text{Task 1: } C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ &\text{Task 2: } C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ &\text{Task 3: } C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ &\text{Task 4: } C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

(b)

Recursive decomposition

- As we know, divide and conquer is one of the strategies of solving the computational problem. It is based on 2 ideas first approach is to solve problem directly if it is minimal and second approach decompose the problem into smaller problems keep it can't be solved as it is and solve the smaller portions.
- The recursive decomposition is based on providing concurrency in solving problems that can be handled in divide and conquer strategy.
- The problem to be solved using recursion is divided into multiple sub problems provided that each of the sub problems is independent of each subproblem which can be solved individually by dividing further into other sub problems and solve them using recursion method.

- In programming a function or procedure calls itself is called as recursion.

Example:

```
main()
{
        int i;
        i = 10;
        demo(i);
}

demo(int count)
{
        count--;
        print("The value of the count is {count}")
        if(count >0)
        {
                demo(count);
                print("Count is {count}");
        }
}
```

5. Write a short note on GPU computing.

GPU computing is the use of a GPU (graphics processing unit) as a co-processor to accelerate CPUs for general-purpose scientific and engineering computing.

The GPU accelerates applications running on the CPU by offloading some of the compute-intensive and time-consuming portions of the code. The rest of the application still runs on the CPU. From a user's perspective, the application runs faster because it's using the massively parallel processing power of the GPU to boost performance. This is known as "heterogeneous" or "hybrid" computing.

A CPU consists of four to eight CPU cores, while the GPU consists of hundreds of smaller cores. Together, they operate to crunch through the data in the application. This massively parallel architecture is what gives the GPU its high compute performance. There are a number of GPU-accelerated applications that provide an easy way to access high-performance computing (HPC).

Core comparison between a CPU and a GPU

Application developers harness the performance of the parallel GPU architecture using a parallel programming model invented by NVIDIA called "CUDA." All NVIDIA GPUs - GeForce®, Quadro®, and Tesla® - support the NVIDIA® CUDA® parallel-programming model.

Tesla GPUs are designed as computational accelerators or companion processors optimized for scientific and technical computing applications. The latest Tesla 20-series GPUs are based on the latest implementation of the CUDA platform called the "Fermi architecture". Fermi has key computing features such as 500+ gigaflops of IEEE standard double-precision floating-point hardware support, L1 and L2 caches, ECC memory error

protection, local user-managed data caches in the form of shared memory dispersed throughout the GPU, coalesced memory accesses, and more.

6. Write a short note on Granularity, Concurrency and Task-Interaction.

Granularity

- The number and size of tasks into which a problem is decomposed determines the granularity of the decomposition.
- A decomposition into many small tasks is called fine-grained.
- A decomposition into a small number of large tasks is called coarse-grained.

Example:

- First decomposition for matrix-vector multiplication is fine-grained.
- Second decomposition is a coarse-grained decomposition of the same problem into four tasks.

Concurrency

Executing multiple tasks simultaneously, is called concurrency. The number of tasks that can be executed simultaneously in a parallel program at any given time is known as its degree of concurrency.

Task Interaction

- The nodes in a task-interaction graph represent tasks and the edges connect tasks that interact with each other.
- Captures the pattern of interaction among tasks.
- The nodes in a task-interaction graph represent tasks and the edges connect tasks that interact with each other.
- The nodes and edges of a task-interaction graph can be assigned weights proportional to the amount of computation a task performs and the amount of interaction that occurs along an edge if this information is known.
- The edges in a task-interaction graph are usually undirected but directed edges can be used to indicate the direction of flow of data if it is unidirectional.
- The edge-set of a task-interaction graph is usually a superset of the edge-set of the task-dependency graph.