CSN-254
Software Engineering
Assignment-2

Group-4

Name	Roll No	Contribution
Meet Sindhav	22114053	Common discussion and articulation of Q1
Mohammed Haaziq	22114055	Common discussion and articulation of Q3
Aditya Mundada	22114058	Common discussion and suggesting changes
Nayan Kakade	22114060	Common discussion and adding insights
Polasa Hariharan	22114068	Common discussion and articulation of Q2
Roopam Taneja	22125030	Common discussion, reviewing and final document preparation

Q1. Discuss parallel and distributed computing from the following perspective: System, Programming, and Architecture with appropriate examples.

A computer system may perform tasks according to human instructions. A single processor can execute only one task in the computer system, which is not an effective way. There are several computation types which include ***parallel computing*** and ***distributed computing.***

Parallel Computing:

Parallel computing is the process of performing computational tasks across multiple processors at once to improve computing speed and efficiency. It divides tasks into sub-tasks and executes them simultaneously through different processors.

Parallel computing provides numerous advantages. Parallel computing helps to increase CPU utilization and improve performance because several processors work simultaneously. Moreover, the failure of one CPU has no impact on the other CPUs' functionality.

Distributed Computing:

Distributed computing is the process of connecting multiple computers via a local network or wide area network so that they can act together as a single ultra-powerful computer capable of performing computations that no single computer within the network would be able to perform on its own.

A distributed system can be made up of any number of different configurations, such as mainframes, PCs, workstations, and minicomputers. Each device within the system is referred to as a 'node', with a group of nodes known as a 'cluster'. Nodes in the system communicate with one another by passing messages through the network. During distributed computing, the various steps in a process will be distributed to the most efficient machine in the network.

SYSTEM PERSPECTIVE:

Parallel:
Multi-core processing uses parallel computing on multi-core processors which involves simultaneous execution of multiple tasks on separate cores within a single processor. Cores can execute independent instructions concurrently, enhancing the overall throughput of the system. **Example:** Intel Core i7 processors with multiple cores executing parallel threads.

Distributed:
Distributed computing involves multiple interconnected systems (nodes) that collaborate to achieve a common goal. Nodes communicate and share resources over a network to accomplish tasks collectively. Message passing is also required which involves communication between distributed nodes through the exchange of messages. **Example:** Clusters of servers connected over the internet or an intranet forming a distributed system. Message Passing Interface (MPI) is used for communication in high-performance computing clusters.

PROGRAMMING PERSPECTIVE:

Parallel:
Parallel programming models provide abstractions and APIs (Application Programming Interfaces) to express parallelism in code. Developers use these models to design algorithms that can be executed concurrently, managing tasks, synchronization, and communication between parallel threads or processes. Parallel libraries can also be used which provide pre-built functions and routines optimized for parallel execution. Developers leverage these libraries to parallelize common operations without delving into low-level details. **Example:** OpenMP (Open Multi-Processing) allows developers to add parallelism to C, C++, and Fortran code using directives to define parallel regions. Intel Math Kernel Library (MKL) provides parallelized mathematical functions. CUDA libraries allow parallel programming on NVIDIA GPUs.

Distributed:

Distributed programming models offer abstractions for writing code that runs on multiple nodes in a distributed system. Developers use these models to manage communication, data consistency, and fault tolerance across distributed nodes. **Example:** Apache Hadoop's MapReduce paradigm for processing large datasets in a distributed environment.

RPC (**Remote Procedure Calls**) enables invoking procedures or functions on remote nodes as if they were local. Developers use RPC to facilitate communication and invoke functions across distributed components. **Example:** gRPC, a modern RPC framework developed by Google.

ARCHITECTURAL PERSPECTIVE:

Parallel:
A shared memory or distributed memory system can be used to assist in parallel computing. All CPUs in shared memory systems share the memory. Memory is shared between the processors in distributed memory systems.

SIMD (Single Instruction, Multiple Data) architectures allow a single instruction to operate on multiple data elements simultaneously. SIMD enhances parallelism by performing the same operation on multiple data elements in parallel. **Example:** Graphics Processing Units (GPUs) often use SIMD architectures for parallel processing of pixel data.

Distributed:
Distributed computing involves multiple interconnected computers that work together to achieve a common goal. Networked architectures enable communication and coordination between distributed nodes.

Example: The Internet or an intranet connecting servers in a cloud computing environment.

Also, distributed file systems allow storage to be distributed across multiple nodes in a network. It provides a unified and scalable storage solution for distributed applications, enabling efficient data access and retrieval.

Example: Hadoop Distributed File System (HDFS) for storing and retrieving large datasets in a distributed environment.

Q2. What do you mean by control and data parallelism in PDC? Discuss with examples.

In order to increase overall performance and efficiency, **PDC** or **Parallel and Distributed Computing** involves the use of multiple processing elements (computers or machines). Control parallelism and data parallelism refers to various ways of breaking down and distributing computations across multiple processing elements.

Control Parallelism: (also referred to as task parallelism):

It refers to dividing large tasks into smaller independent subtasks that can be executed concurrently. In control parallelism each processing element works on different part of the problem, and the coordination among these elements ensures a coherent execution. This is particularly useful when the problem can be decomposed into multiple independent subproblems so that there is minimal need for communication or synchronization among the processing elements.

Example: Say we need to find sum of two elements and product of two other elements and then find the difference between them. Here we can perform the first two operations concurrently(but again the third operation requires the results from these).

Data Parallelism:

It refers to breaking down of problem such that same operation is performed on multiple pieces of data.

Example: If we consider the example of matrix multiplication, we can see that we can compute each element of new matrix independently. Here the operation is multiply and add and it is performed on different rows and columns of matrices. GPUs specialize in these kinds of operations.

Example: Consider summing the contents of an array of size N. For a single-core system, one thread would simply sum the elements 0 . . . N-1. For a dual-core system, however, thread A, running on core 0, could sum the elements 0 . . . N/2 - 1 and while thread B, running on core 1, could sum the elements N/2 . . . N - 1.

Based on the problem, one may be advantageous over other. So, the best approach will depend on the available resources(hardware) and the specific problem.

Q3. Study and analyze the effects of data and control parallelism on traditional software process models.

Parallelism relies on the fact that humans can handle multiple processes at a time and that will exploit human ability to the full extent. This is why these traditional software models stem from this basic human fact.

Data parallelism and control parallelism are two key concepts in parallel computing that can significantly impact traditional software process models. The effects of these parallelism types on common software development models:

Waterfall Model:

Data Parallelism: In waterfall models, stages are executed sequentially, and data parallelism may not be explicitly utilized. However, if large datasets are involved, parallel processing can be applied within each stage to enhance performance.

Control Parallelism: Being restricted to a sequential flow, parallelism cannot be introduced between stages. However tasks within each stage can be performed parallel if they are independent.

Iterative Waterfall Model:

Data Parallelism:Limited data parallelism, as each iteration follows a sequential flow. However, within each iteration, tasks like requirements gathering, design, and coding can be performed concurrently, improving efficiency within that iteration.

Control parallelism:Control parallelism is not a characteristic of the traditional Waterfall Model, including its iterative version. Each iteration follows a sequential flow, and the next iteration begins only if there is error in previous phases.

Prototyping Model:

Data Parallelism: In the prototyping model, parallelism may not be explicitly incorporated during the prototyping phase. However, parallel development can occur during the independent phases.

Control Parallelism: During the prototyping phase, multiple designs can be developed parallelly, i.e. parallel prototyping is possible allowing for control parallelism.

Evolutionary Model:

Data Parallelism: Data parallelism can be leveraged in the Evolutionary Model during the creation and refinement of prototypes. Different teams or individuals can work on different aspects of the prototype concurrently, processing and analysing various data sets to enhance different functionalities.

Control Parallelism: Control parallelism is evident in the concurrent execution of prototyping phases. . It is useful for very large problems, where it is easier to find modules for incremental implementation. The Core modules can be developed parallelly with the non-core modules. This allows for a more dynamic and responsive development process.

Spiral Model:

Data Parallelism: At each iteration of the spiral model, data parallelism can be applied by analysing and developing different parts of the system concurrently.

Control Parallelism: Spiral model inherently involves multiple iterations, each with its set of tasks. Control parallelism can be achieved by executing tasks in different iterations concurrently. During the prototyping phase, multiple prototypes for different aspects of the system can be developed concurrently, each handling a specific set of data or functionality

In summary, the effects of data and control parallelism on traditional software process models depend on the inherent characteristics of each model. While some models are more amenable to parallelization due to their iterative and incremental nature, others, like the waterfall model, may require more thoughtful integration of parallel processing techniques. Incorporating parallelism can lead to improved efficiency, reduced development time, and better resource utilization in software development processes.