Name: Adeleke Asimiyu. A

Matric. No: DU0372

Course Code: CSC 421

Date: 05/02/2025.

Question:

Write on the topic PARALLEL PROGRAMMING MODELS covering definition, types, and

concepts like Message Passing Interface (MPI), OpenMP (Open Multi-Processing),

MapReduce, OpenCL (Open Computing Language), and CUDA (Compute Unified Device

Architecture) programming model.

# PARALLEL PROGRAMMING

Parallel programming, in simple terms, is the process of decomposing a problem into smaller tasks that can be executed at the same time using multiple computer resources. Computer software was written conventionally for serial computing. This means that to solve a problem, an algorithm divides the problem into smaller instructions. These discrete instructions are then executed on the Central Processing Unit of a computer one by one. Only after one instruction is finished, next one starts.

The term parallel programming may be used interchangeably with parallel processing or in conjunction with parallel computing, which refers to the systems that enable the high efficiency of parallel programming. It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource that has been applied to work is working at the same time.

The main benefit of parallel computing is that it ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of the hardware was used and the rest rendered idle.

# PARALLEL PROGRAMMING MODELS

A parallel programming model is a set of program abstractions for fitting parallel activities from the application to the underlying parallel hardware. It spans over different layers: applications, programming languages, compilers, libraries, network communication, and I/O systems. Classifications of parallel programming models can be divided broadly into two areas: **process interaction** and **problem decomposition**.

* **Process Interaction**

Process interaction relates to the mechanisms by which parallel processes are able to communicate with each other. The most common forms of interaction are shared memory and message passing, but interaction can also be implicit (invisible to the programmer). A suitable combination of two previous models is sometimes appropriate. Processors can directly access memory on another processor. This is achieved via message passing, but what the programmer actually sees is shared-memory model.

* **Shared memory**: is an efficient means of passing data between processes. In a shared-memory model, parallel processes share a global address space that they read and write to asynchronously. Mechanisms such as locks, semaphores and monitors can be used to avoid race conditions.
* **Message passing**: In a message-passing model, parallel processes exchange data through passing messages to one another. These communications can be asynchronous, where a message can be sent before the receiver is ready, or synchronous, where the receiver must be ready.
* **Partitioned global address space**: PGAS models provide a middle ground between shared memory and message passing. PGAS provides a global memory address space abstraction that is logically partitioned, where a portion is local to each process.
* **Implicit interaction**: here, no process interaction is visible to the programmer and instead the compiler and/or runtime is responsible for performing it.
* **Problem decomposition**

A parallel program is composed of simultaneously executing processes. Problem decomposition relates to the way in which the constituent processes are formulated.

* **Task parallelism**: A task-parallel model focuses on processes, or threads of execution. These processes will often be behaviourally distinct, which emphasises the need for communication. In Flynn's taxonomy, task parallelism is usually classified as MIMD or MISD.
* **Data parallelism**: A data-parallel model focuses on performing operations on a data set, typically a regularly structured array. A set of tasks will operate on this data, but independently on disjoint partitions. In Flynn's taxonomy, data parallelism is usually classified as MIMD or SIMD.
* **Stream Parallelism**: Stream parallelism, also known as pipeline parallelism, focuses on dividing a computation into a sequence of stages, where each stage processes a portion of the input data. Each stage operates independently and concurrently, and the output of one stage serves as the input to the next stage.
* **Implicit parallelism**: As with implicit process interaction, an implicit model of parallelism reveals nothing to the programmer as the compiler, the runtime or the hardware is responsible.

# CONCEPTS IN PARALLEL PROGRAMMING MODELS

1. **MPI (Message Passing Interface)**: The Message Passing Interface (MPI) is an Application Program Interface that defines a model of parallel computing where each parallel process has its own local memory, and data must be explicitly shared by passing messages between processes. This model was created in the early 1990s by Jack Dongarra, Tony Hey, and David W. Walker, with the first standardized version, MPI-1, released in 1994. It was designed to enable efficient communication in distributed memory systems, such as clusters and supercomputers, supporting both point-to-point and collective operations. MPI is known for its portability, scalability, and flexibility, making it suitable for high-performance computing (HPC) applications. However, its low-level nature can complicate programming and debugging, and communication overhead may introduce latency, impacting performance.

Functions of MPI include barrier synchronization across all group members, broadcasting from one member to all members of a group, gathering data from all group members to one member, and scatter data from one member to all members of a group.

1. **OpenMP**: Open Multi-Processing is a directive-based Application Programming Interface (API) for developing parallel programs on shared memory architectures. It was established in 1997 by companies like Intel and IBM to simplify parallel programming for shared memory systems, such as multi-core CPUs. It provides a high-level API with simple compiler directives, facilitating the conversion of serial code into parallel code with minimal changes.

While OpenMP offers ease of use and portability, it is mainly suited for shared memory architectures and may struggle with distributed systems. Additionally, the overhead from thread creation and synchronization can impact performance in some cases.

The main difference between MPI and OpenMP is that, with MPI, each process has its own memory space and executes independently from the other processes and it supports parallel computation for distributed-memory and shared-memory systems. With OpenMP, threads share the same resources and access shared memory, also, it only supports parallel computation for shared-memory systems.

1. **MapReduce**: MapReduce is a programming model that uses parallel processing to process large amounts of data in a distributed way. It's a key part of the Hadoop framework, which is a data analytics engine for Big Data. MapReduce was introduced by Google in 2004 by Jeffrey Dean and Sanjay Ghemawat, transformed large dataset processing with a parallel model across distributed systems. It simplifies complexities like data partitioning and fault tolerance, consisting of two phases: Map, which creates key-value pairs, and Reduce, which aggregates them for the final output. This allows high scalability for processing petabytes of data efficiently.

The limitations of MapReduce include high latency that makes it unsuitable for real-time applications and a limited expressiveness compared to other frameworks, restricting its use for complex algorithms.

1. **OpenCL (Open Computing Language)**: OpenCL, or Open Computing Language, is an open standard for writing programs that run across multiple computing platforms. It's used in many applications, including scientific and medical software, vision processing, and neural network training. OpenCL was developed by the Khronos Group and released in 2009, is a framework designed for parallel programming across a variety of hardware platforms, including CPUs, GPUs, and FPGAs. It aims to address the complexities of creating portable and efficient code that can run on diverse architectures, allowing developers to harness the computational power of these accelerators effectively. OpenCL supports both data parallelism and task parallelism, making it a flexible choice for high-performance computing tasks.

However, The low-level nature of the framework can make writing and optimizing code difficult for developers. Additionally, achieving optimal performance often necessitates specific tuning for the hardware being used, which can complicate the development process further.

1. **CUDA (Compute Unified Device Architecture)**: CUDA, developed by NVIDIA and released in 2006, is a parallel computing platform and programming model that allows developers to harness the massive parallel processing capabilities of NVIDIA GPUs for general-purpose computing. It includes a C/C++ extension, enabling the creation of GPU-accelerated applications and simplifying the offloading of computationally intensive tasks to GPUs. This framework is particularly effective for parallel workloads, excelling in areas such as deep learning and scientific computing due to its exceptional performance.

CUDA is a vendor lock-in model, which means it is restricted to NVIDIA GPUs, which can hinder portability across different hardware platforms. Additionally, while CUDA provides a high-level programming interface and a robust ecosystem of libraries and tools, achieving optimal performance often requires a deep understanding of GPU architecture, making it complex for some users.