Name: **Peters Ayomide**

Matric No: **DU0319**

Department: **Computer science.**

**Parallel Programming Model**

A parallel programming model describes an abstract parallel machine by its basic operations (such as arithmetic operations, spawning of tasks, reading from and writing to shared memory, or sending and receiving messages), their effects on the state of the computation.

Parallel programming models are ways to structure programs so that multiple processors can execute them simultaneously.

**How Parallel Programming Works**

In parallel programming, a program is broken down into smaller chunks so that each processor can work on a separate chunk. This allows many calculations or processes to be carried out at the same time.

**Types Of Parallel Programming.**

1. Shared memory

2. Distributed memory

3. CUDA

* **Shared memory:** A model that uses shared memory for fast communication between processes. Examples include OpenMP, OmpsS, XiTAO, and Pthreads. Synchronization mechanisms are used to regulate access to shared data to prevent race conditions.
* **Distributed memory:** A model where processes run on different machines and communicate using message passing. Suitable for applications where data is distributed across different machines.
* **CUDA:** A parallel computing platform and programming model developed by NVIDIA. Allows developers to speed up computing applications by using graphical processing units (GPUs).

**The Concept of Parallel Programming**

A parallel programming model refers to a set of program abstractions that allow for the parallel execution of tasks on parallel hardware. It includes different layers such as applications, programming languages, compilers, libraries, network communication, and I/O systems.

Parallel programming involves breaking up code into smaller tasks or chunks that can be run simultaneously. Images sourced from Lawrence Livermore National Laboratory's Introduction to Parallel Computing Tutorial.

1. **Message passing interface (MPI):** A "message passing interface" (MPI) in parallel programming models refers to a standardized library that enables processes within a distributed memory system to communicate with each other by explicitly sending and receiving messages, allowing for parallel computation across multiple processors or nodes in a cluster, where each process has its own local memory space; essentially, it's the primary method for coordinating parallel tasks in a distributed computing environment by exchanging data through messages.

**Key points about MPI:**

**Function:**

MPI provides a set of functions that programmers can use to send and receive data between different processes, allowing for coordinated parallel operations across multiple nodes.

**Important concepts in MPI:**

\* Processes: Individual computational units within a parallel program that can send and receive messages.

\* Communicator: A group of processes that can communicate with each other.

\* Rank: A unique identifier assigned to each process within a communicator, used to identify which process to send a message to.

\* Point-to-point communication: Sending messages directly between two specific processes.

\* Collective communication: Operations that involve all processes in a communicator, like broadcasting a value to all processes or reducing data across all processes.

1. **Open MP (open multi-processing):** Open Multi-Processing" (OpenMP) refers to a parallel programming model that enables developers to write code that utilizes multiple cores on a shared-memory system, allowing for simultaneous execution of tasks and improved performance; essentially, it's a way to leverage the power of multiple processors within a single machine by adding special compiler directives to your C, C++, or Fortran code to indicate where parallelism should occur.

**Key points about OpenMP:**

**Shared-memory paradigm:**

OpenMP is primarily designed for shared-memory systems, where multiple threads can directly access the same memory space.

**Ease of use:** Considered a relatively simple and accessible way to introduce parallelism into existing code compared to other parallel programming models.

**Concept of OpenMpi:** OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared-memory multiprocessing programming in C, C++, and Fortran, on many platforms, instruction-set architectures and operating systems, including Solaris, AIX, FreeBSD, HP-UX, Linux, macOS, and Windows.

1. **MapReduce:** MapReduce is a parallel programming model presented by Google Company, mainly for large-scale data processing. MapReduce achieves the reliability calculation based on key/value pairs and distributes big data to each node in the network. MapReduce processing" refers to a parallel programming model where large datasets are distributed across multiple computing nodes and processed in parallel using two primary operations: "map" which splits the data into smaller chunks and performs operations on each chunk independently, and "reduce" which combines the intermediate results from the map phase to produce a final result, allowing for efficient processing of massive data volumes on distributed systems like Hadoop clusters.

**Key points about MapReduce:**

**Parallelism:** The core strength of MapReduce is its ability to parallelize data processing by distributing tasks across multiple nodes, significantly speeding up computations on large datasets.

**Map Function:** This function takes an input data chunk and transforms it into a set of key-value pairs, where the key is used to group related data during the reduce phase.

**Reduce Function:** After the map stage, the reduce function aggregates the intermediate key-value pairs with the same key, performing operations like summation, counting, or averaging based on the specific task.

1. **Open computing language (openCL):** Open Computing Language" (OpenCL) refers to a standard programming framework that enables parallel processing across different types of computing devices like CPUs and GPUs, allowing developers to write code that can run on diverse hardware platforms, utilizing both task-based and data-based parallel programming models; essentially, it provides a way to write parallel programs that can leverage the processing power of various computing architectures within a single application.

**Key points about OpenCL and its parallel programming models:**

**Heterogeneous Computing:** OpenCL's primary strength is its ability to target different types of processors (like CPUs, GPUs, and DSPs) within a single program, making it ideal for applications needing high computational power across diverse hardware.

**C-based Language:** OpenCL utilizes a C-like syntax for writing kernels (functions executed on the compute devices), making it familiar to programmers with C experience.

**Data Parallelism:** This model involves dividing large datasets into smaller chunks and processing them concurrently on multiple compute units, which is particularly efficient for tasks like image processing or scientific simulations.

**Task Parallelism:** This model involves breaking down a complex task into smaller, independent subtasks that can be executed in parallel on different compute unit.

**How OpenCL works:**

**Host and Device:** The program is executed on a "host" (typically the CPU) which manages the computation and sends instructions to "devices" (like GPUs) where the actual parallel processing occurs.

**OpenCL Kernels:** These are the core functions that are executed on the device in parallel, and they access data stored in the device's memory.

**CUDA (Compute Unified Device Architecture) Programming Models:** CUDA stands for Compute Unified Device Architecture and is a new hardware and software architecture for issuing and managing computations on the GPU as a data-parallel computing device without the need of mapping them to a graphics API. It is available for the GeForce 8 Series, Quadro FX 5600/4600, and Tesla solutions. The operating system’s multitasking mechanism is responsible for use.