PART 2

**Q1. What is the FLOPS? and usage in parallel computing?**

**Ans**. FLOPS stands for Floating-Point Operations Per Second. It is a way to gauge how well a computer performs, particularly when performing mathematical and scientific tasks that include lots of floating-point calculations. FLOPS quantifies how many floating-point arithmetic operations a computer or program can perform in one second.

In parallel computing, FLOPS are used to measure the computational performance of parallel systems, including multi-core processors, GPUs (Graphics Processing Units), and parallel computing clusters. Parallel computing often involves distributing the workload across multiple processing units to speed up computations. FLOPS can be used to measure the efficiency of floating-point operations carried out by parallel systems while assessing their performance.

**Q2. Simply explain the computer architecture of von Neumann.**

**Ans**. Von Neumann architecture is an early, influential type of computing structure. It primarily consists of memory chips that are able to both hold and process data. Each chip has the ability to perform different tasks, depending on how it is affected by the operation executed before it. In this architecture, each computer would have memory, mechanisms for output and input, a central control, a place for central arithmetic, and external storage.

The Von-Neuman Architecture is based on the principle of:

* Fetch an Instruction
* Decode the Instruction
* Execute the Instruction

A diagram of a computer

Description automatically generated

**Q3. Simply explain: what is shared memory architecture and what is distributed memory architecture?**

**Ans**. **Shared Memory Architecture**

All of the processors or nodes in a system with a shared memory architecture have access to the same memory pool. This entails that they can share data through shared variables and read and write to the same memory locations. This architecture has the benefit of simplifying the programming model since the programmers do not have to explicitly control data transport or processor synchronization. As the processors compete for memory access and bandwidth and the memory size restricts the number of processors, the drawback is that it might lead to scalability and contention problems.

**A diagram of process and process

Description automatically generated**

**Distributed Memory Architecture**

A distributed-memory architecture is a system where each processor or node has its own local memory, and they communicate with each other through message passing. As a result, they are only able to access their own memory locations and communicate via message sending and receiving. As a result of the processors not interfering with each other's memory access and bandwidth and the memory capacity not imposing a restriction on the number of processors, this architecture has the advantage of avoiding the contention and scalability issues of the shared-memory architecture. The drawback is that it makes the programming model more challenging because the programmers must actively control data distribution and interprocessor communication.

**A diagram of a computer network

Description automatically generated**

**Q4. List three factors that can cause parallel overhead according to the lecture and simply explain.**

**Ans**.

1. **Task start-up time**: When the workload is distributed unevenly, Task start-up time overhead develops. While some processing units may complete their tasks fast and sit idle, others may still be working on calculations. Uneven data distributions, fluctuating workloads, or ineffective job scheduling methods can all cause load-balancing problems. To reduce this overhead and enhance parallel performance, it is critical to efficiently distribute the workload among the processing units. Extensive additional calculations increase start-up time.
2. **Synchronizations**: Multiple threads or processes frequently need to coordinate their operations in parallel computing to prevent interference. Thread execution is coordinated through synchronization procedures such as locks, barriers, and semaphores. Waiting for synchronization points, however, can add overhead, particularly when threads have to wait a long time for others to do their work. As threads spend more time waiting than working on parallel tasks, excessive synchronization can prevent the parallel speedup. Additional resources are required for the synchronization of processes (threads).
3. **Data communications**: To cooperate on tasks in parallel systems, various threads or processes must talk to one another and exchange information. Sending messages, changing shared variables, or moving data across various components of a parallel program are all examples of this communication. When a lot of time is spent on various communication activities, communication overhead results. Mainly determined by latency and bandwidth. Efficient communication strategies and minimizing unnecessary data transfers are crucial to reduce this overhead.

**Q5. Give the formula of Amdahl’s Law including the number of processors.**

**Ans**. speedup = 1/(P/N+S)

where P = parallel fraction,

N= number of processors and,

S = serial fraction

**Speedup**: Represents the potential improvement in the overall execution time of the program when parallelized. It is the ratio of the execution time of the entire task without parallelization to the execution time with parallelization.

**P**: Represents the fraction of the program that can be parallelized. It ranges from 0 to 1, where 0 means the entire program is sequential (non-parallelizable), and 1 means the entire program can be parallelized.

**N**: Represents the number of processors or processing units available for parallel execution.

**Q6. Please give the parallel programming models in common use and simply explain.**

**Ans**.

* **Shared Memory Model**: Processes share a common address space, which they read and write to asynchronously.
* **Distributed Memory Model**: A set of tasks that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
* **Data Parallel Model**: Address space is treated globally. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.
* **Hybrid Model**: Combines more than one of the previously described programming models. It lends itself well to the most popular (currently) hardware environment of clustered multi/many-core machines.
* **Single Program Multiple Data(SPMD):** Single Program - All tasks execute their copy of the same program simultaneously. Multiple Data - All tasks may use different data.
* **Multiple Program Multiple Data(MPMD):** Multiple Program - Tasks may execute different programs simultaneously. Multiple Data - All tasks may use different data.

**Q7. Give 3 possibilities for poor parallel performance according to the lecture and simply explain.**

**Ans**.

* Load balance, where the computing resources aren't contributing equally to the problem.
* Network effects, such as latency and bandwidth, that impact performance and communication overhead.
* Impacts from use of RAM and virtual memory, such as cache misses and page faults.

**Q8. Briefly explain two popular hybrid parallel programming models on current supercomputers and HPC clusters.**

**Ans**.

1. **MPI + OpenMP:**

Explanation: This hybrid model combines MPI (Message Passing Interface) for distributed memory parallelism with OpenMP for shared memory parallelism. MPI is used for communication and coordination between different nodes or processors in a cluster, allowing for data exchange and synchronization. OpenMP, on the other hand, is utilized within each node or processor to exploit multicore parallelism, where multiple threads can work on shared memory within a node. This combination allows for efficient utilization of both distributed and shared memory resources, making it suitable for large-scale parallel applications on clusters with multicore nodes. Developers can parallelize across nodes using MPI and exploit multicore processors within nodes using OpenMP, optimizing both inter-node and intra-node parallelism.

2. **MPI + CUDA/OpenACC:**

Explanation: This hybrid model combines MPI for inter-node communication with CUDA or OpenACC for GPU (Graphics Processing Unit) parallelism. MPI handles communication between nodes in a cluster, allowing for distributed memory parallelism. On each node, GPU parallelism is exploited using CUDA (for NVIDIA GPUs) or OpenACC directives, which allow developers to offload specific parts of the code to the GPU. CUDA provides a lower-level, more manual approach to GPU programming, whereas OpenACC offers a higher-level, directive-based approach, simplifying GPU acceleration for developers. This combination enables highly parallel applications to leverage the massive parallel processing capabilities of GPUs, enhancing overall performance in HPC environments.

**Q9. Please list the three primary API components in OpenMP and give simple examples.**

**Ans.**

* Compiler Directives: #pragma omp parallel default(shared) private(beta,pi)
* Runtime Library Routines:

#include <omp.h> int omp\_get\_num\_threads(void)

* Environment Variables: export OMP\_NUM\_THREADS=8

**Q 10. Please write the steps to use gcc compiler to compile a C file “hello.c” with OpenMP flag; set the environment variable of 4 threads on the bash-shell Linux operating system; and run this executable.**

**Ans.**

* gcc -fopenmp hello.c -o hello
* export OMP\_NUM\_THREADS=4

Step 1: Create the "hello.c" File

Step 2: Compile "hello.c" with OpenMP Flag

**gcc -fopenmp -o hello hello.c**

Step 3: Set Environment Variable for 4 Threads

**export OMP\_NUM\_THREADS=4**

Step 4: Run the Executable

**./hello**