**CSYE7105 13969 Parallel Machine Learning & AI SEC 01 Fall 2024**

Submitted by: Anshul Chaudhary

NEU ID: 002820096

Professor: Handan Liu

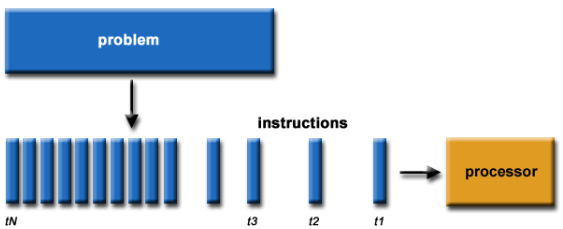
**Part 2:**

**1. Explain the differences between serial and parallel code. [4 points]**

**A:** Serial and parallel code differ primarily in how tasks are executed and how computational resources are utilized

|  |  |
| --- | --- |
| Serial Code | Parallel Code |
| Code, tasks are executed one after another in a linear order | Tasks are divided into smaller sub-tasks that can be executed simultaneously across multiple processing units |
| Each task is dependent on the previous one being completed | The tasks or sub-tasks are independent to run concurrently |
| All the instructions are executed on a single processor | The tasks are executed on a multiple processor |
| Only makes use of a single resource at a time | Make better use of underlying parallel hardware |

**Serial:**



**Parallel:**

A diagram of instructions and instructions

Description automatically generated

**2. What is the FLOPS and its usage in parallel computing? [2 points]**

**A:** FLOPS stands for Floating Point Operations Per Second. It is a measure of computational performance

* The ratio of Floating-point Operations per Second (FLOPS, flops, or flop/s) to the peak possible performance is a common way to report overall efficiency for parallel code.
* The peak possible performance is calculated with the assumption that each processor core performs every possible floating-point operation during each clock cycle.
* In computing, FLOPS (flops or flop/s) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. For such cases, it is a more accurate measure than measuring instructions per second.

**3. Simply explain the computer architecture of von Neumann. [4 points]**

**A:** The von Neumann architecture is a design model for computers that describes computers working at a basic level. It has become the foundation for most modern computers. The architecture is characterized by the separation of program and data, as well as the use of a single memory space for both instructions and data

1. CPU: The CPU is responsible for executing instructions. It consists of the Arithmetic and Logic Unit (ALU) for performing mathematical and logical operations. The Control Unit for managing the execution of instructions.
2. Memory: It uses a single memory unit to store both program instructions and data. This contrasts with earlier architectures, where program instructions and data were stored in separate memory units.
3. Input/Output (I/O): External devices, such as keyboards, displays, and storage devices, are used for input and output operations. These devices are connected to the CPU through input and output channels.

A diagram of a computer component

Description automatically generated

**4. Simply explain: what is shared memory architecture and what is distributed memory architecture? [4 points]**

**A:**

**Shared Memory Architecture:**

* In a shared memory architecture, multiple processors share a common, centralized memory space.
* Each processor can access any location in memory as part of a unified address space.
* Processors communicate by reading and writing to this shared memory.
* Historically, shared memory machines have been classified as UMA (Uniform Memory Access) and NUMA (Non-Uniform Memory Access), based upon memory access times.

A diagram of a computer processor

Description automatically generated

**Distributed Memory Architecture:**

* In a distributed memory architecture, each processor has its own private memory.
* Processors communicate via message passing, explicitly sending and receiving messages.
* There is no global address space; each processor operates independently with local memory.
* Communication is achieved through explicit message passing between processors.
* This architecture is common in parallel and distributed computing systems, like clusters or grid computing environments.

A diagram of a computer network

Description automatically generated

**5. According to the lectures of this course, list three factors that can cause parallel overhead. [3 points]**

**A:** In parallel computing, parallel overhead refers to the extra time or resources consumed in managing parallel tasks. Key factors include:

1. Task: This occurs when dividing a problem into smaller tasks and distributing them across processors adds computational cost. Time spent in task decomposition, assigning tasks, and combining results can contribute to overhead, especially if tasks are unevenly distributed.
2. Synchronization: This arises when parallel tasks need to coordinate their execution. While synchronization ensures correct results, excessive waiting between tasks can create performance bottlenecks and reduce the benefits of parallelism.
3. Data Communication: This happens when tasks exchange or share data, introducing delays in message passing or data transfer. Frequent or inefficient communication increases overhead, making it crucial to minimize unnecessary transfers and optimize communication patterns.

**6. Please give the parallel programming models in common use and simply explain. [4 points]**

**A:** Parallel programming models exist as an abstraction above hardware and memory architectures

There are several parallel programming models in common use:

1. Shared Memory Model:
   1. In this programming model, processes/tasks share a common address space, which they read and write to asynchronously.
   2. Various mechanisms such as locks / semaphores are used to control access to the shared memory, resolve contentions, and prevent race conditions and deadlocks.
   3. This is perhaps the simplest parallel programming model.

A diagram of process and process

Description automatically generated

1. Threads Model:
   1. This programming model is a type of shared memory programming.
   2. In the threads model of parallel programming, a single "heavy weight" process can have multiple "light weight", concurrent execution paths.

A diagram of a model

Description automatically generated

1. Distributed Memory / Message Passing Model:
   1. A set of tasks that use their local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
   2. Tasks exchange data through communications by sending and receiving messages.
   3. Data transfer usually requires cooperative operations to be performed by each process.

A computer screen shot of a computer

Description automatically generated

1. Hybrid Model:

A diagram of a computer hardware system

Description automatically generated

**7. According to the lectures, how to evaluate your parallel performance practically? List the calculating formulas and explain. [6 points]**

**A:**

* A common way to assess the efficiency of a parallel program is through its speedup.
* Speedup is defined as the ratio of the wall-clock time for a serial program to the wall-clock time for the parallel program that accomplishes the same work.
* **Speedup**: Speedup measures how much faster a parallel program runs compared to its sequential counterpart.

Speedup =

* Usually, the speedup cannot be greater than the number of parallel resources on which the program is running.
* **Efficiency:** Efficiency measures the utilization of the processors in a parallel system.

Efficiency =

Speedup and efficiency are key metrics for evaluating the effectiveness of parallelization. High values indicate optimal resource utilization in the parallel program, while lower values may suggest scalability issues or inefficiencies in the implementation.

**8. Briefly explain the hybrid parallel programming model on current supercomputers (HPC clusters). [4 points]**

**A:** Hybrid parallel programming model:

* Hybrid parallel programming combines distributed memory parallelism (across multiple nodes) and shared memory parallelism (within individual nodes).
* Distributed memory typically uses MPI (Message Passing Interface) for communication between nodes.
* Shared memory uses techniques like OpenMP or threading for parallelism within a single node.
* This model enables the scaling of applications across large numbers of nodes while efficiently utilizing the processing resources within each node.
* It addresses the challenges of balancing workload between multiple processors and across memory spaces.
* The hybrid model is ideal for large-scale scientific simulations and complex computations common in supercomputers and HPC clusters.
* It provides flexibility and scalability, making it suitable for handling large data sets and massive parallelism.
* Widely used in HPC-related applications like scientific research, engineering, and computational modeling.

**9. Explain the elements of Flynn's taxonomy. [4 points]**

**A:** The essence of computing is instructions operating on data. Parallelization can be achieved by using multiple instruction streams and/or multiple data streams. This insight is formalized in Flynn's taxonomy (1966), which classifies different types of parallel computing architectures.

Flynn's taxonomy is a two-by-two table where the rows represent the type of instruction stream, and the columns represent the type of data stream. Each kind of stream can be classified as single or multiple

* SISD (Single Instruction, Single Data): A single processor executes one instruction at a time on a single data stream. This is a traditional serial computing model, like most basic CPUs.
* SIMD (Single Instruction, Multiple Data): One instruction is executed on multiple data points simultaneously. This is common in vector processors or GPUs, which process large datasets in parallel.
* MISD (Multiple Instruction, Single Data): Multiple instructions operate on the same data stream. This model is rare and not widely used in practical systems.
* MIMD (Multiple Instruction, Multiple Data): Multiple processors execute different instructions on different data streams concurrently. This is typical in multi-core processors and distributed systems

A white rectangular box with black text

Description automatically generated

**10. Explain the terms of nodes and cluster. [4 points]**

**A:**

**Nodes:**

* A node is a standalone physical computer unit with a network connection that typically runs its own copy of the operating system.
* Supercomputer clusters are composed of nodes connected by a communications network.

**Clusters:**

* A cluster is a collection of machines (each of which is called a node) that function in some way as a single resource.
* They may be administered as a unit, provide a uniform environment for tasks running on the cluster, or work together to provide fault-tolerant access to file storage.

A computer components diagram

Description automatically generated with medium confidence

**11. What is data parallelism? [2 points]**

**A:** Data Parallelism:

* The application involves applying the same operations to a large collection of relatively independent data.
* In data parallel design, each parallel worker (thread) applies the same operations to a different segment of the data.
  + Each worker does the same work on a unique piece of data.
  + Data placement is an essential part of a data-parallel algorithm.

**12. Please list the three primary API components in OpenMP and give simple examples. [4 points]**

**A:** OpenMP (Open Multi-Processing) is an API that facilitates multi-platform shared-memory multiprocessing through a set of compiler directives, library routines, and environment variables designed for parallel programming. The three main components of OpenMP are:

1. **Compiler Directives:** These are instructions for the compiler on how to parallelize your program. They are specific comments or annotations added to the code, guiding the compiler in parallelizing tasks. The most common directive is #pragma omp, followed by a keyword that indicates the desired parallel behavior.

Example - Parallelizing a loop:

#include <stdio.h>

#include <omp.h>

int main() {

int i, n = 10;

int array[n];

#pragma omp parallel

for (i = 0; i < n; i++) {

array[i] = i \* i;

}

for (i = 0; i < n; i++) {

printf("%d ", array[i]);

}

return 0;

}

1. **Runtime Library Routines:** These are calls to functions within the OpenMP library that allow a program to manipulate and query the execution environment. These functions serve various purposes, such as setting and querying the number of threads, retrieving a thread's unique identity (thread ID), and determining if a thread is in a parallel region. A commonly used function is omp\_get\_thread\_num(), which returns the thread number.

Example - Using OpenMP runtime functions:  
#include <stdio.h>

#include <omp.h>

int main() {

#pragma omp parallel

{

int thread\_id = omp\_get\_thread\_num();

printf("Hello World %d\n", thread\_id);

}

return 0;

}

1. **Environment Variables:** These variables control the behavior of the runtime system. OpenMP allows the execution environment to be adjusted via these variables, which can be set before executing the application to manage aspects such as the number of threads utilized.

Example - Setting the number of threads:

#include <stdio.h>

#include <omp.h>

int main() {

omp\_set\_num\_threads(4);

#pragma omp parallel

{

int thread\_id = omp\_get\_thread\_num();

printf("Hello World %d\n", thread\_id);

}

return 0;

}

**13. Please write the steps to use gcc compiler to compile a C file “hello\_omp.c” with OpenMP flag and get the executable file “hello\_omp”. On Linux bash shell system, set 4 threads by using the environment variable; then run this executable. [3 points]**

**A:** Navigate to the directory where the hello\_omp.c file is located

1. gcc -fopenmp hello\_omp.c -o hello\_omp
2. export OMP\_NUM\_THREADS=4
3. ./hello\_omp

**14. In HPC, there are two main ways of scaling a program. Please explain them. [4 points]**

**A:** In High-Performance Computing (HPC), the two main ways of scaling a program are:

1. Strong Scaling:
   * + Definition: Strong scaling refers to increasing the number of parallel tasks (workers) while keeping the problem size constant.
     + Goal: The goal is to reduce the overall time to solution as more resources (processors/cores) are added.
     + Challenge: As more tasks are added, communication and synchronization overhead can limit performance improvements, especially if the problem size remains small.
2. Weak Scaling:
   * Definition: Weak scaling involves increasing both the problem size and the number of parallel tasks simultaneously, such that the amount of work per task remains constant.
   * Goal: The objective is to maintain the same time-to-solution as the problem size grows, by adding more resources in proportion to the increased workload.
   * Benefit: This approach is often more efficient when scaling large problems, as it avoids the diminishing returns seen in strong scaling.

**15. When you run your serial code on multiple cores (CPUs), you can employ two different approaches. Please explain these two approaches [2 points]. Two theoretical results (laws) describe the conditions for the success of each approach. Please explain the two corresponding laws [2 points].**

**A:** Two Approaches to Running Serial Code on Multiple Cores

Parallelization: In this approach, the serial code is modified to run concurrently on multiple cores by dividing the workload into smaller tasks that can be executed in parallel. This is done using parallel programming techniques such as OpenMP or MPI.

Multithreading: Another approach is to use multithreading, where a single process is split into multiple threads that can run concurrently on different cores. Multithreading doesn't necessarily require changing the core algorithm but instead leverages the system's ability to run different threads in parallel.

**Amdahl's Law (Parallelization Approach):** Amdahl’s Law focuses on the limitations of parallelization. It states that the speedup of a parallel program is limited by the portion of the code that cannot be parallelized (the serial part). No matter how many cores are used, the serial portion will constrain the maximum speedup.

**Gustafson’s Law (Multithreading/Scalability Approach):** Gustafson’s Law suggests that, unlike Amdahl’s Law, the overall performance gain is not limited by the serial portion if the problem size increases with more cores. It assumes that as you add more processors, you also increase the workload, allowing better scalability.