CSYE7105 HW1
Instructor: Dr. Handan Liu
2020-09-27

---

1. **Give several reasons to use parallel computing. [4 pts]**

Sol: *The reasons to use parallel computing are:*

- Parallel Computing is much better suited for modeling ,simulating, and understanding complex, real world phenomena.
- Solves larger complex problems such as processing millions of transactions every second.
- It can take advantage of non-local resources when the local resources are finite.
- Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.
- It provides Concurrency as multiple compute resources can do many things simultaneously

---

2. **On the CPU architecture, how many parallel programming models are there based on memory access methods? List them and explain briefly. [4 pts]**

Sol: *There are several parallel programming models in common use:*

- Shared Memory (without threads)
- Threads
- Distributed Memory / Message Passing
- Data Parallel
- Hybrid
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)

Shared Memory Model (without threads)

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 are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks.
3. This is perhaps the *simplest parallel programming model*.

An Advantage of this model : the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks. All processes see and have equal access to shared memory. Program development can often be simplified.

A Disadvantage of this model: In terms of performance is that it becomes more difficult to understand and manage *data locality* (*Keeping data local to the process that works on it conserves memory accesses, cache refreshes and bus traffic that occurs when multiple processes use the same data.)*

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.
3. A thread's work may best be described as a subroutine within the main program. Any thread can execute any subroutine at the same time as other threads.
4. Threads communicate with each other through global memory (updating address locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time.
5. Threads can come and go, but a.out remains present to provide the necessary shared resources until the application has completed.

Unrelated standardization efforts have resulted in two very different implementations of threads: ***POSIX Threads*** *and* ***OpenMP****.*

POSIX Threads

- Part of Unix/Linux operating systems
- Library based
- Commonly referred to as Pthreads.
- Very explicit parallelism; requires significant programmer attention to detail.

OpenMP

- Industry standard jointly defined and endorsed by a group of major computer hardware and software vendors, organizations and individuals.
- Compiler directive based
- Portable / multi-platform, including Unix and Windows platforms
- Available in C/C++ and Fortran implementations
- Can be very easy and simple to use - provides for "incremental parallelism". Can begin with serial code.

Distributed Memory / Message Passing Model

1. 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.
2. Tasks exchange data through communications by sending and receiving messages.
3. Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.

Data Parallel Model *(May also be referred to as the Partitioned Global Address Space (PGAS) model)*

1. Address space is treated globally.
2. Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube.
3. A set of tasks work collectively on the same data structure; however, each task works on a different partition of the same data structure.
4. Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".
5. On shared memory architectures, all tasks may have access to the data structure through global memory.
6. On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.

Hybrid Model

1. A hybrid model combines more than one of the previously described programming models.
2. Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with the threads model (OpenMP).
    - Threads perform computationally intensive kernels using local, on-node data
    - Communications between processes on different nodes occurs over the network using MPI
3. This hybrid model lends itself well to the most popular hardware environment of clustered multi/many-core machines.
4. Another similar and increasingly popular example of a hybrid model is using MPI with CPU-GPU (Graphics Processing Unit) programming.
    - MPI tasks run on CPUs using local memory and communicating with each other over a network.
    - Computationally intensive kernels are off-loaded to GPUs on-node.
    - Data exchange between node-local memory and GPUs uses CUDA (or something equivalent).

Single Program Multiple Data (SPMD): *(using message passing or hybrid programming, is probably the most commonly used parallel programming model for multi-node clusters)*

1. SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
2. *SINGLE PROGRAM:* All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid.
3. *MULTIPLE DATA*: All tasks may use different data
4. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program - perhaps only a portion of it.

Multiple Program Multiple Data (MPMD):

1. Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
2. *MULTIPLE PROGRAM*: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid.
3. *MULTIPLE DATA*: All tasks may use different data
4. MPMD applications are not as common as SPMD applications but may be better suited for certain types of problems, particularly those that lend themselves better to functional decomposition than domain decomposition.

---

3. What is Flynn's Taxonomy? Please explain it. [4 pts]

Sol: Flynn's taxonomy is a specific classification of parallel computer architectures that are based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture.

- The sequence of instructions read from memory constitutes an *instruction stream*.
- The operations performed on the data in the processor constitute a *data stream*.

The four categories in Flynn's taxonomy are the following:

1. **<u>(SISD) single instruction, single data</u>**
    - *Single Instructions:* Only 1 instruction stream is being acted on by CPU during any 1 clock cycle.
    - *Single Data*: Only 1 data stream is being used as input during any 1 clock cycle.
    - A serial (non-parallel) computer.
    - Example : single processor/core PCs., older generation mainframes

2. **<u>(MISD) multiple instruction, single data</u>**
    - *Multiple Instructions*: Each processing unit operates on the data independently via separate instruction streams.
    - *Single Data:* A single data stream is fed into multiple processing unit.
    - A type of parallel computer.
    - Some conceivable uses might be:
        - ➢ multiple frequency filters operating on a single signal stream
        - ➢ multiple cryptography algorithms attempting to crack a single coded message.

3. **<u>(SIMD) single instruction, multiple data</u>**
    - *Single Instruction*: All processing units execute the same instruction at any given clock cycle
    - *Multiple Data:* Each processing unit can operate on a different data element.
    - Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
    - A type of parallel computer
    - Two varieties: Processor Arrays and Vector Pipelines
    - Examples:
        - ➢ Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV

- Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

4. **<u>(MIMD) multiple instruction, multiple data (</u>***the most common type of parallel computer - most modern supercomputers fall into this category.)*
    - *Multiple Instruction*: Every processor may be executing a different instruction stream
    - *Multiple Data*: Every processor may be working with a different data stream
    - Execution can be synchronous or asynchronous, deterministic or non-deterministic.
    - A type of parallel computer.
    - Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

---

4. What kind of classification in Flynn classification does GPU computing belong to? Please explain. [4 pts]

Sol:

- GPU belongs to SIMD category of Flynn's classification: same portion of code will be executed in parallel and applied to various elements of a data set.
- GPUs parallelize by continuing the SIMD approach and executing the same program multiple times
- Both by pure SIMD where a set of programs execute in lock step which is why branching is bad on a GPU, as both sides of an if statement must execute
- One result be thrown away so that the lock step programs proceed at the same rate.

---

5. What is the embarrassingly parallel? [4 pts]

Sol:

- Solving many similar, but independent tasks simultaneously; little to no need for coordination/communication between the tasks.

- Is one for which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case where there exists no dependency between those parallel tasks.
- They are easy to perform on server farms which do not have any of the special infrastructure used in a true supercomputer cluster.
- They are thus well suited to large, internet based distributed platforms such as BOINC.
- Example : graphics processing units for the task of 3D projection, where each pixel on the screen may be rendered independently.

---

6. Give 3 popular math libraries for high performance computing. [4 pts]
Sol: 3 popular math libraries are:

Open Source Math Libraries

1. BLAS: Basic Linear Algebra Subprograms
2. LAPACK: Linear Algebra PACKage
3. ScaLAPACK: Scalable Linear Algebra PACKage

Commercial Math Libraries

1. Intel's MKL: Intel Math Kernel Library
2. IBM's ESSL: Engineering and Scientific Subroutine library
3. AMD's AMCL: AMD Core Math Library

---

7. How can we evaluate the speedup of parallel computing comparing with serial computing? Please explain in detail. [4 pts]
Sol: The performance is achieved by analyzing and quantifying the number of threads and/or the number of processes used.
To analyze this, a few performance indexes are introduced: speedup, efficiency, and scaling.

- **Speedup** (*one of the simplest and most widely used indicators):* It is defined as the ratio of the time taken to solve a problem on a single processing element, T1, to the time required to solve the same problem on p identical processing elements, Tp.

- Speedup of a code which has been parallelized, defined as:

$$Speedup = \frac{Wall\ clock\ time\ of\ serial\ execution}{Wall\ clock\ time\ of\ parallel\ execution}$$

Amdahl's law is a widely used law used to design processors and parallel algorithms. It states that the maximum speedup that can be achieved is limited by the serial component of the program:

$$Speedup(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

Serial part of job = 1(100%) – Parallel Part

Parallel part is divided up by N workers

The bottom of the fraction represents the percent of the original time we will take - this equals the serial part plus one share of the parallel part. Dividing 1 by that gives us how many times faster we are working.

---

8. Provide 6 MPI necessary functions for MPI parallel computing. Only write the function name, no parameters. C is recommended here. [4 pts]

Sol: *According to me, the 6 MPI necessary functions are:*

1. MPI_Init()
2. MPI_Comm_size()
3. MPI_Comm_rank()
4. MPI_Wtime()
5. MPI_Finalize()
6. MPI_Send() / MPI_Recv()

---

9. **What are strong scaling and weak scaling? Or what difference between them. [4 pts]**

Sol:

- **Strong scaling:**
  - The total problem size stays fixed as more processors are added.
  - Goal is to run the same problem size faster
  - Perfect scaling means problem is solved in 1/P time (compared to serial)
  - A program is considered to scale *linearly* if the *speedup* (in terms of work units completed per unit time) is equal to the number of processing elements used (**N**)
  - If the amount of time to complete a work unit with 1 processing element is **t1**, and the amount of time to complete the same unit of work with N processing elements is **tN**, the strong scaling efficiency (as a percentage of linear) is given as:

$$\frac{\boldsymbol{t1}}{(\boldsymbol{N} * \boldsymbol{tN}) * \mathbf{100\%}}$$

- **Weak scaling:**
  - The problem size *per processor* stays fixed as more processors are added and the total problem size is proportional to the number of processors used.
  - Goal is to run larger problem in same amount of time
  - Perfect scaling means problem Px runs in same time as single processor run.
  - Linear scaling is achieved if the run time stays constant while the workload is increased in direct proportion to the number of processors
  - If the amount of time to complete a work unit with 1 processing element is **t1**, and the amount of time to complete **N** of the same work units with **N** processing elements is **tN**, the weak scaling efficiency (as a percentage of linear) is given as:

$$\left(\frac{\boldsymbol{t1}}{\boldsymbol{tN}}\right) * \mathbf{100}$$

10. Give several possible reasons why sometimes the computation time on multi-CPU on multiple nodes is slower than that of multi-CPU on a single node. [4 pts]

Sol: Adding more CPU's increases the speed of the computation, but also increases the time spent communicating between CPU's.

1. *Time Spent Communicating > Time Spent Computing* And, the time waiting to get the results increases.
2. *Scalability*: How well does a parallel code perform as we increase the number of computing units. . Parallel computing comprises numerical operations and non-productive work that takes the form of communications between CPU's. For a fixed problem size, as the number of cores increases, the work per core decreases (numerical operations) and the communications increases.
3. Making parallel code involves extra elements that sequential code does not have (control logic to distribute work, load balancing, sequential points that cannot be parallelized and synchronizations where processors must share information they have computed independently before proceeding to simulate a sequential behavior and make results coherent).
4. Wall clock time
5. Hardware
6. Memory issue