CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

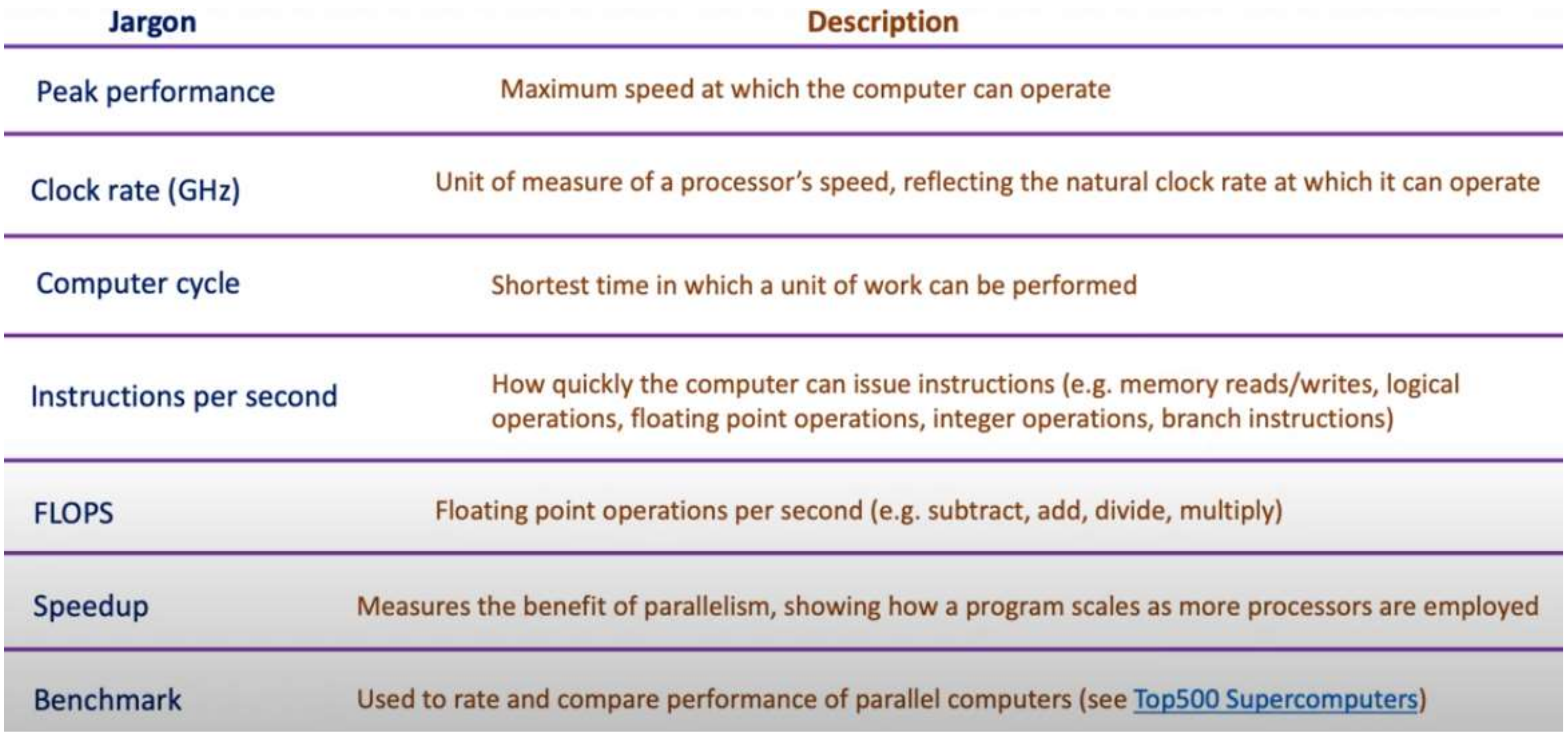
Prepared by
Dr Komarasamy G
Senior Associate Professor
School of Computing Science and Engineering
VIT Bhopal University

Unit-1

Parallelism Fundamentals – Key Concepts and Challenges – Overview of Parallel computing – Flynn's Taxonomy – Multi-Core Processors – Shared vs Distributed memory.

Performance of Parallel Computers, Performance Metrics for Processors, Parallel Programming Models, Parallel Algorithms.

Performance of Parallel Computers



Jargon	Description
Peak performance	Maximum speed at which the computer can operate
Clock rate (GHz)	Unit of measure of a processor's speed, reflecting the natural clock rate at which it can operate
Computer cycle	Shortest time in which a unit of work can be performed
Instructions per second	How quickly the computer can issue instructions (e.g. memory reads/writes, logical operations, floating point operations, integer operations, branch instructions)
FLOPS	Floating point operations per second (e.g. subtract, add, divide, multiply)
Speedup	Measures the benefit of parallelism, showing how a program scales as more processors are employed
Benchmark	Used to rate and compare performance of parallel computers (see Top500 Supercomputers)

Performance Metrics for Processors

- **Processor speed:** The speed of the processor, measured in GHz (gigahertz), determines how quickly the computer can execute instructions and process data.
- **Memory:** The amount and speed of the memory, including RAM (random access memory) and cache memory, can impact how quickly data can be accessed and processed by the computer.
- **Storage:** The speed and capacity of the storage devices, including hard drives and solid-state drives (SSDs), can impact the speed at which data can be stored and retrieved.
- **I/O devices:** The speed and efficiency of input/output devices, such as keyboards, mice, and displays, can impact the overall performance of the system.
- **Software optimization:** The efficiency of the software running on the system, including operating systems and applications, can impact how quickly tasks can be completed.

Parallel Programming Models

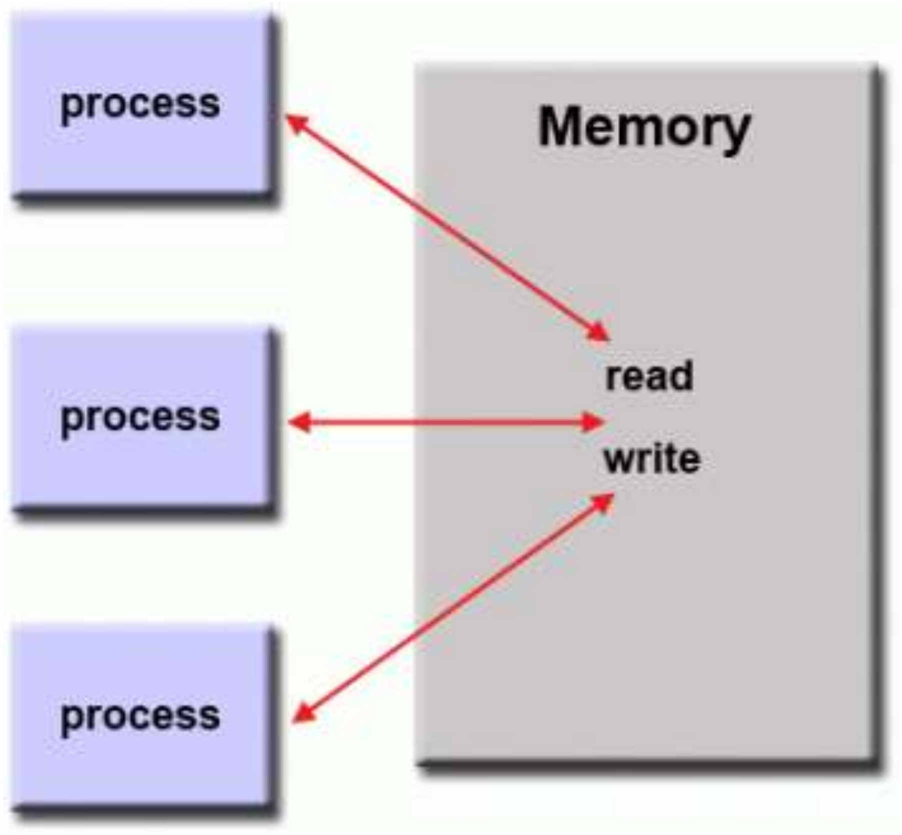
- **There are several parallel programming models in common use:**
  1. Shared Memory (without threads)
  2. Threads
  3. Distributed Memory / Message Passing
  4. Data Parallel
  5. Hybrid
  6. Single Program Multiple Data (SPMD)
  7. Multiple Program Multiple Data (MPMD)

Parallel Programming Models

1. Shared Memory Model (without threads)

- processes/tasks share a common address space, which they read and write to asynchronously.
- Various mechanisms such as locks / semaphores are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks.
- This is perhaps the simplest parallel programming model.
- **Advantage** of this model from the programmer's point of view is that 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.

Parallel Programming Models



Shared memory model

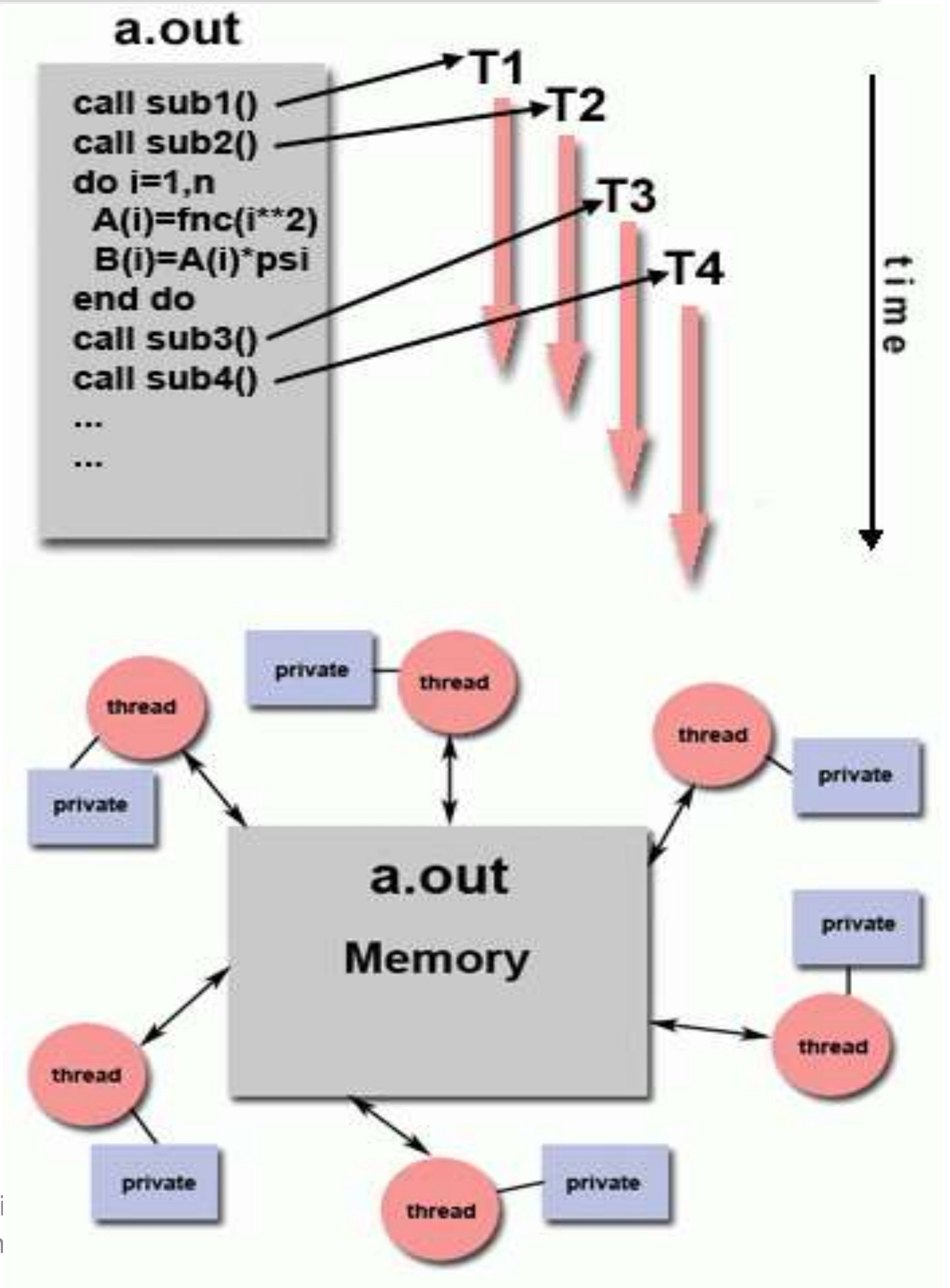
Parallel Programming Models

- **Disadvantage** 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.
  - Unfortunately, controlling data locality is hard to understand and may be beyond the control of the average user.

Parallel Programming Models

2. Threads Model

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



```
call sub1()
call sub2()
do i=1,n
  A(i)=fnc(i**2)
  B(i)=A(i)*psi
end do
call sub3()
call sub4()
...
...
```

Parallel Programming Models

- **For example:**
  - The main program **a.out** is scheduled to run by the native operating system. **a.out** loads and acquires all of the necessary system and user resources to run. This is the "heavy weight" process.
  - **a.out** performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently.
  - Each thread has local data, but also, shares the entire resources of **a.out**. This saves the overhead associated with replicating a program's resources for each thread ("light weight"). Each thread also benefits from a global memory view because it shares the memory space of **a.out**.

Parallel Programming Models

- 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.
- 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.
- Threads can come and go, but **a.out** remains present to provide the necessary shared resources until the application has completed.

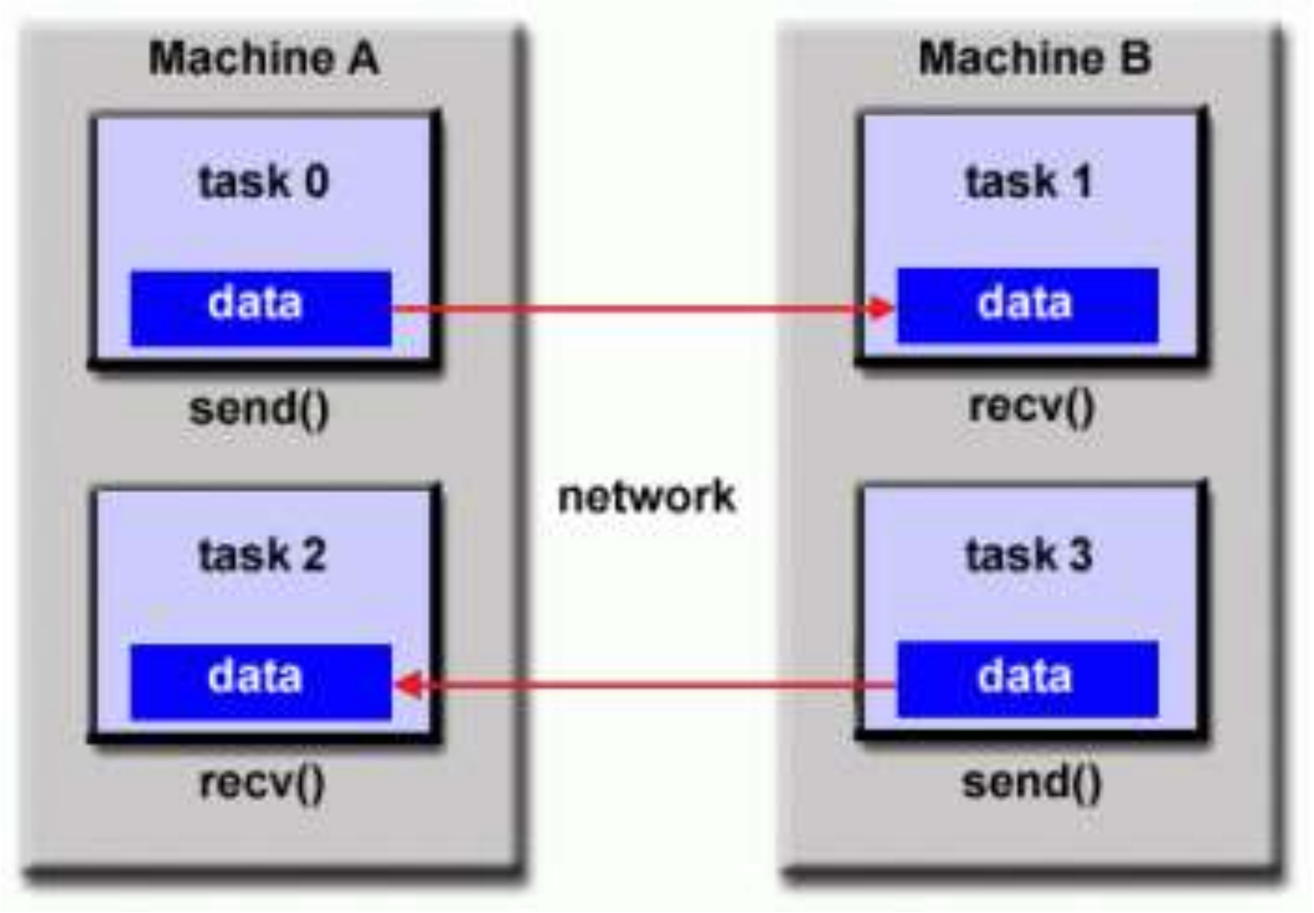
Parallel Programming Models

3. Distributed Memory / Message Passing Model

- This model demonstrates the **following characteristics:**
- **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.
- Tasks exchange data through communications by sending and receiving messages.
- Data transfer usually requires cooperative operations to be performed by each process.
- For example, a send operation must have a matching receive operation.

Parallel Programming Models

3. Distributed Memory / Message Passing Model



Parallel Programming Models

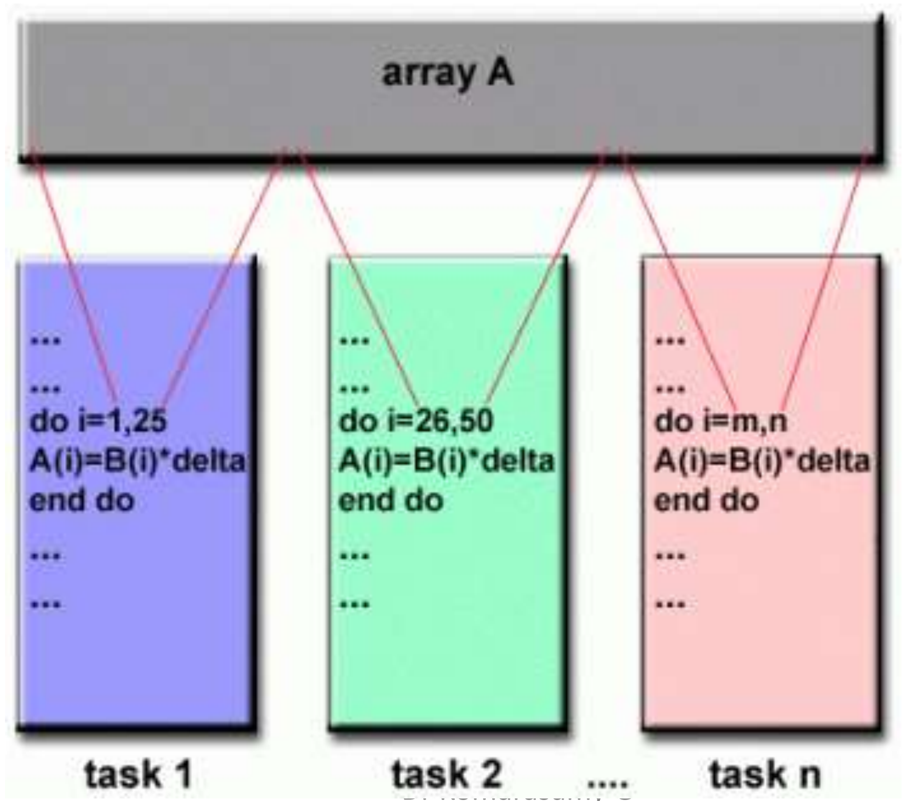
4. Data Parallel Model

- May also be referred to as the **Partitioned Global Address Space (PGAS)** model.
- The data parallel model demonstrates the following characteristics:
  - **Address space is treated globally**
  - 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.
  - A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.
  - Tasks perform the same operation on their partition of work, for example, "add 4 to every array element“.

Parallel Programming Models

4. Data Parallel Model

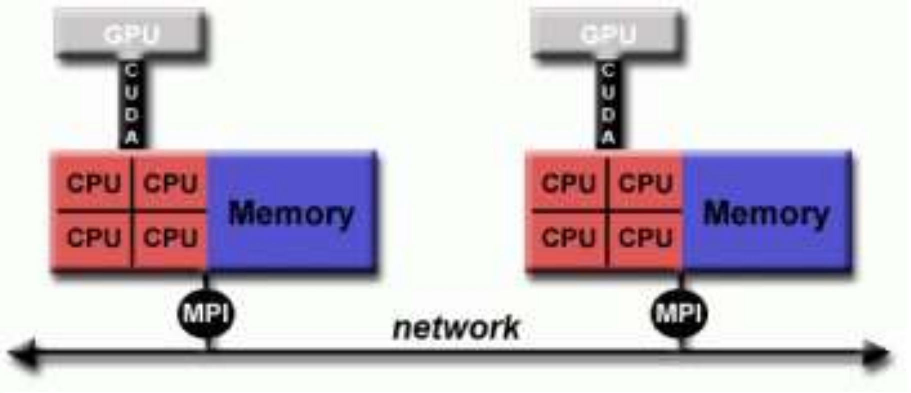
- On shared memory architectures, all tasks may have access to the data structure through global memory.
- On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.



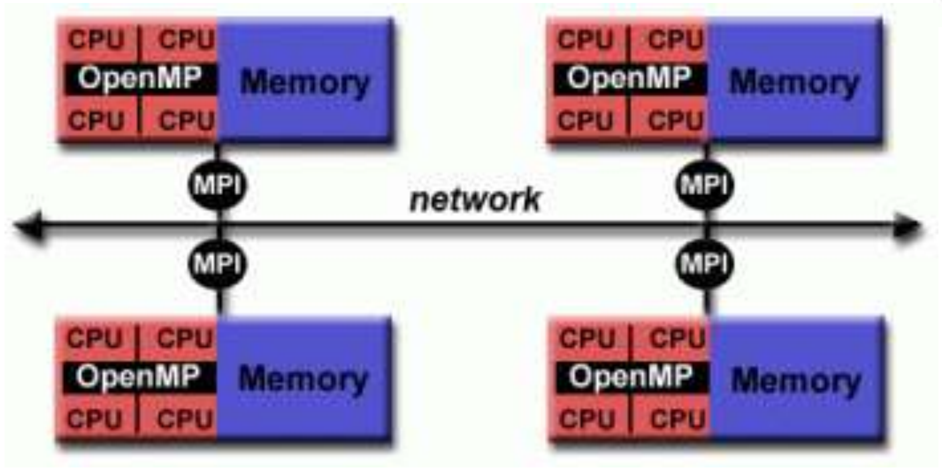
Parallel Programming Models

5. Hybrid Model

- A hybrid model combines more than one of the previously described programming models.



Hybrid model with MPI and CUDA



Hybrid model with MPI and OpenMP

Parallel Programming Models

5. Hybrid Model

- 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
- This hybrid model lends itself well to the most popular (currently) hardware environment of clustered multi/many-core machines.

Parallel Programming Models

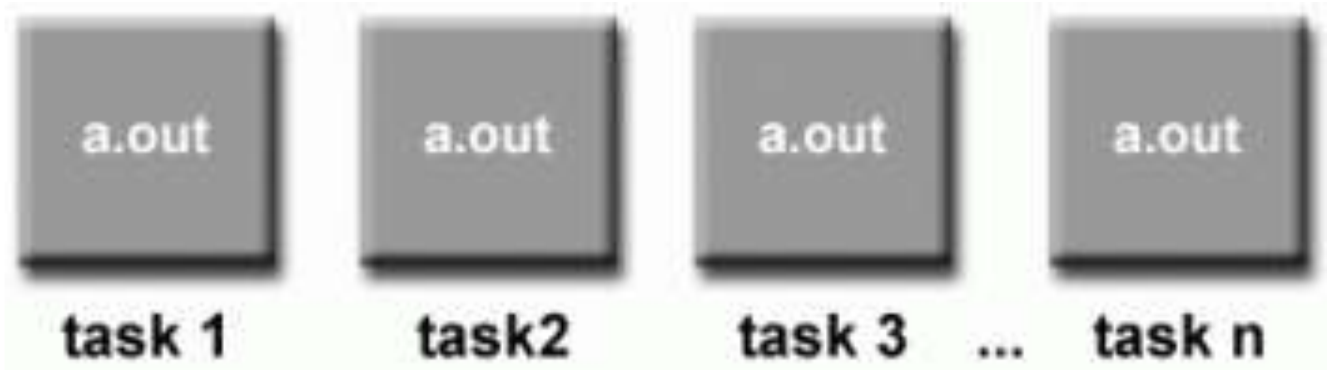
5. Hybrid Model

- 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).
- Other hybrid models are common:
  - MPI with Pthreads
  - MPI with non-GPU accelerators

Parallel Programming Models

6. Single Program Multiple Data (SPMD)

- SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
- **SINGLE PROGRAM:** All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid.
- **MULTIPLE DATA:** All tasks may use different data



Parallel Programming Models

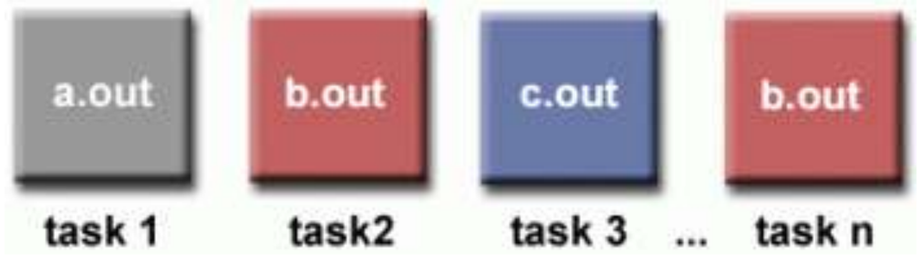
6. Single Program Multiple Data (SPMD)

- 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.
- The SPMD model, using message passing or hybrid programming, is probably the most commonly used parallel programming model for multi-node clusters.

Parallel Programming Models

7. Multiple Program Multiple Data (MPMD)

- Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.
- **MULTIPLE PROGRAM:** Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid.
- **MULTIPLE DATA**: All tasks may use different data
- 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 (discussed later under Partitioning).



https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial##Examples

Parallel Algorithms

- Parallel algorithms are methods for organizing the computational work of a given application such that multiple parts of the workload can be **performed concurrently to reduce the time to solution and increase performance.**
- An **algorithm** is a sequence of steps that take inputs from the user and after some computation, produces an output.
- A **parallel algorithm** is an **algorithm that can execute several instructions simultaneously** on different processing devices and then combine all the individual outputs to produce the final result.

Concurrent Processing

- The easy availability of computers along with the growth of Internet has changed the way we store and process data.
- We are living in a day and age where data is available in abundance.

Parallel Algorithms

- Every day we deal with huge volumes of data that require complex computing and that too, in quick time. Sometimes, we need to fetch data from similar or interrelated events that occur simultaneously.
- This is where we require **concurrent processing** that can divide a complex task and process it multiple systems to produce the output in quick time.
- Concurrent processing is essential where the task involves processing a huge bulk of complex data.
- Examples include − accessing large databases, aircraft testing, astronomical calculations, atomic and nuclear physics, biomedical analysis, economic planning, image processing, robotics, weather forecasting, web-based services, etc.

Parallel Algorithms

- **What is Parallelism?**
- **Parallelism** is the process of processing several set of instructions simultaneously.
- It reduces the total computational time. Parallelism can be implemented by using **parallel computers,** i.e. a computer with many processors.
- Parallel computers require parallel algorithm, programming languages, compilers and operating system that support multitasking.
- In this tutorial, we will discuss only about **parallel algorithms**. Before moving further, let us first discuss about algorithms and their types.

Parallel Algorithms

- **Depending on the architecture of computers, we have two types of algorithms**
- **Sequential Algorithm** − An algorithm in which some consecutive steps of instructions are executed in a chronological order to solve a problem.
- **Parallel Algorithm** − The problem is divided into sub-problems and are executed in parallel to get individual outputs. Later on, these individual outputs are combined together to get the final desired output.

Parallel Algorithms

- **Model of Computation**
- Both sequential and parallel computers operate on a set (stream) of instructions called algorithms. These set of instructions (algorithm) instruct the computer about what it has to do in each step.
- Depending on the instruction stream and data stream, computers can be classified into four categories −

1. **Single Instruction stream, Single Data stream (SISD) computers**
2. **Single Instruction stream, Multiple Data stream (SIMD) computers**
3. **Multiple Instruction stream, Single Data stream (MISD) computers**
4. **Multiple Instruction stream, Multiple Data stream (MIMD) computers**

https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_quick_guide.htm