**Parallel Programming Skills**

**Foundation:**

1. **Define the following**:
2. **Task**:

The interpretation and execution of pseudocode, as derived from discrete mathematical expressions. In other words, a task is a program with a set of instructions that is executed by a processor. In parallel programming, several sets of instructions are executed by multiple processors at the same time.

1. **Pipelining:**

Is the process by which multiple overlapping instructions are stored and executed in an orderly process. A pipeline has two main categories, Arithmetic and Instruction with each operating in segments with instructions entering from one end and exiting from the other end. Each stage of the pipeline consists of an input register that holds the data and a combinational circuit that performs operations on the data. The output from the combinational circuit is then applied to the input register of the next segment. Pipelining increases the efficiency of the CPU by reducing the cycle time of the processor as well as increasing the amount of data that can be processed simultaneously.

1. **Shared Memory**:

In computer architecture, it simply means that in a multiprocessor computer system all the central processing units (CPU’s) have individual buses that gives direct access to random access memory (RAM). With regards to parallel programming it describes a model where programs are able to exchange data by creating a cache of memory that can be accessed by all programs. This conserves memory space by removing the need for programs to create copies of data when executing instructions. This also increases processor speeds as programs can directly access the same logical memory locations regardless of where the physical memory actually exists.

1. **Communications**:

Is the exchange of data from processor to processor or processor to memory, through a shared memory bus or over an interconnected network.

1. **Synchronization**:

The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a point within an application where a task may not proceed further until another task reaches the same equivalent point. Synchronization usually involves waiting by at least one task and can therefore cause a parallel application's wall clock execution time to increase.

1. **Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them**.

Flynn's taxonomy was developed in 1966 by Michael Flynn and is a way of categorizing different forms of parallel computer architectures based on the processing streams of data and instructions. He defined four categories of computer architecture:

1. Single Instruction, Single Data (SISD)
2. Single Instruction, Multiple Data, (SIMD)
3. Multiple Instruction and Single Data stream (MISD)
4. Multiple Instruction, Multiple Data streams (MIMD).
5. **Single Instruction, Single Data (SISD)**:

An SISD computing system is the oldest type of computer architecture and is found in computers with only one processor. Therefore, SISD machines are conventional serial computers that process only one stream of instructions and one stream of data meaning only one instruction stream is being acted on by the CPU during any one clock cycle. Examples of SISD’s are older generation mainframes, minicomputers, workstations and single processor/core PCs

1. **Single Instruction, Multiple Data (SIMD)**:

In a SIMD computing system, multiple processing elements work under the control of a single control unit. It has one instruction set and multiple data streams. All the processing elements receive the same instruction broadcast from the control unit. Main memory can also be divided into modules for generating multiple data streams acting as a distributed memory. The processing elements simultaneously execute the same instruction at any given clock cycle. Each processor has its own memory and separate data streams. Every processor must be allowed to complete its instruction before the next instruction is taken for execution. Therefore, the execution of instructions all occur at the same time. An example of SIMD organization is Cray’s vector processing machine.

1. **Multiple Instruction and Single Data stream (MISD**):

In a MISD computing system, each processing unit operates on shared input data independently via separate instruction streams. In this case, multiple processing units are working on only a single data stream at a time. The only known example of a computer capable of MISD operation is the C.mmp built by Carnegie-Mellon University.

1. **Multiple Instruction, Multiple Data (MIMD**):

In a MIMD computing system, multiple processing elements and multiple control units are organized as in MISD. But the difference is that this system, allows multiple instruction streams to operate on multiple data streams. The processors work on their own data with their own instructions. Tasks executed by different processors can start or finish at different times. They are not lock-stepped, as in SIMD computers, but run asynchronously. This classification actually recognizes the parallel computer. That means in the real sense MIMD organization is said to be a Parallel computer. All multiprocessor systems fall under this classification. Examples include; Burroughs D825, Cray-2, S1, Cray X-MP, IBM 370/168 MP, Univac

1. **What are the Parallel Programming Models?**
2. **Shared Memory (without threads)**:

Shared Memory is an OS provided abstraction which allows a memory region to be simultaneously accessed by multiple programs with an intent to provide communication among them. One process will create an area in RAM which other processes can access. Normally the OS prevents processes from accessing the memory of another process, but the Shared Memory features in the OS can allow data to be shared. Since both processes can access the shared memory area

like regular working memory, this is a very fast way of communication. On the other hand, it is less powerful, as for example the communicating processes must be running on the same machine (whereas other IPC methods can use a computer network), and care must be taken to avoid issues if processes sharing memory are running simultaneously and may try to edit the shared buffer at the same time

1. **Threads**:

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. From a programming perspective, threads implementations commonly comprise, a library of subroutines that are called from within parallel source code and a set of compiler directives imbedded in either serial or parallel source code.

1. **Distributed Memory / Message Passing**:

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.

1. **Hybrid**:

A hybrid model combines more than one of the previously described programming models. 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 hardware environment of clustered multi/many-core machines. 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.

1. **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. All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid. All tasks may use different data.

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.

1. **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. Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid. 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.

1. **List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?**

OpenMP (Open Multi-Processing) is an Application Program Interface (API) that was designed for parallelism within a node or computer system. It works entirely differently, in that the programmer specifies various parallel directives through compiler pragmas. The compiler then attempts to automatically split the problem into N parts, according to the number of available processor cores. OpenMP uses the Shared Memory model of Parallel Computer Memory Architecture because it facilitates the ability of processors to access memory from connected networks or from a global address space.

1. **Shared Memory**: OpenMP
   1. Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as a global address space.
   2. Multiple processors can operate independently but share the same memory resources.
   3. Changes in a memory location effected by one processor are visible to all other processors.
   4. Historically, shared memory machines have been classified as UMA and NUMA, based upon memory access times.

Uniform Memory Access (UMA):

* 1. Most commonly represented today by Symmetric Multiprocessor (SMP) machines
  2. Identical processors
  3. Equal access and access times to memory
  4. Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.

Non-Uniform Memory Access (NUMA):

* 1. Often made by physically linking two or more SMPs
  2. One SMP can directly access memory of another SMP
  3. Not all processors have equal access time to all memories
  4. Memory access across link is slower
  5. If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

Advantages:

* 1. Global address space provides a user-friendly programming perspective to memory
  2. Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
  3. Shared Memory (UMA)

Shared Memory (NUMA)

Disadvantages:

* 1. Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  2. Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

1. **Distributed Memory:**
2. Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.
3. Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
4. Because each processor has its own local memory, it operates independently.
5. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
6. When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
7. The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.

Advantages:

1. Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.
2. Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
3. Cost effectiveness: can use commodity, off-the-shelf processors and networking.

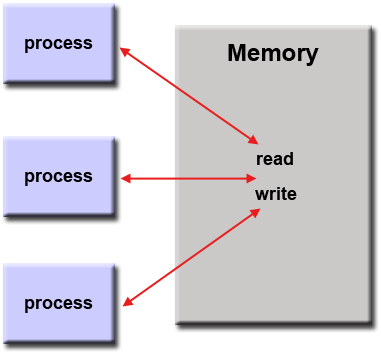
Disadvantages:

1. The programmer is responsible for many of the details associated with data communication between processors.
2. It may be difficult to map existing data structures, based on global memory, to this memory organization.
3. Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.
4. Parallel Computer Memory Architectures
5. **Hybrid Distributed-Shared Memory**:
6. The largest and fastest computers in the world today employ both shared and distributed memory architectures.
7. The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
8. The distributed memory component is the networking of multiple shared memory/GPU machines, which know only about their own memory - not the memory on another machine. Therefore, network communications are required to move data from one machine to another.
9. Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.

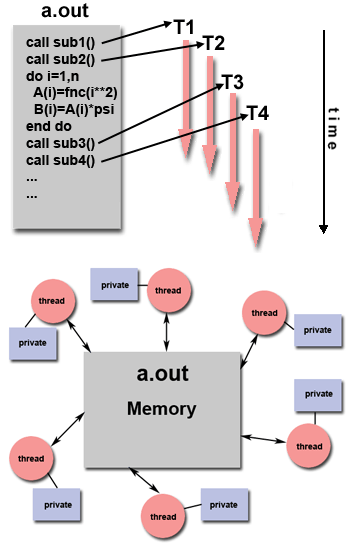
Advantages and Disadvantages:

1. Whatever is common to both shared and distributed memory architectures.
2. Increased scalability is an important advantage
3. Increased programmer complexity is an important disadvantage
4. **Compare Shared Memory Model with Threads Model? (in your own words and show pictures**

Shared Memory Model



Threads Model



A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The issue with shared memory systems is that many CPUs need fast access to memory and will likely cache memory, which has two complications: access time degradation: when several processors try to access the same memory location and a lack of data coherence: whenever one cache is updated with information that may be used by other processors, the change needs to be reflected to the other processors, otherwise the different processors will be working with incoherent data. In the threading model, all the resources belong to the same process. Each thread has its own address pointer and stack, yet they share a common address space and system resources. The common shared memory access makes it easy for a developer to divide up work, tasks, and data. The disadvantage is that because all resources are available to all threads, this allows for data races. A data race occurs when two or more threads access the same memory address and at least one of the threads alters the value in memory. The results of the computation can be altered depending on whether the writing thread completes its write before or after the reading thread reads the value.

**6. What is Parallel Programming? (in your own words)**

It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time. Parallel processing is a method in computing of running two or more processors (CPUs) to handle separate parts of an overall task. Breaking up different parts of a task among multiple processors will help reduce the amount of time to run a program. Any system that has more than one CPU can perform parallel processing, as well as multi-core processors which are commonly found in computers today.

7. **What is system on chip (SoC)? Does Raspberry PI use system on SoC?**

Yes, the Raspberry PI uses a SoC. A system on a chip or system on chip is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functions—all on a single chip substrate. Along with a CPU, an SoC usually contains a GPU (a graphics processor), memory, USB controller, power management circuits, and wireless radios(WIFI, 3G, 4G LTE, and so on). Whereas a CPU cannot function without dozens of other chips, it’s possible to build complete computers with just a single SoC. SoCs are very common in the mobile electronics market because of their low power consumption.

8. **Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM component**

The number one advantage of an SoC is its size: An SoC is only a little bit larger than a CPU. An SoC is only a little bit larger than a CPU, and yet it contains a lot more functionality. Using SoCs, we can put complete computers in smartphones and tablets, and still have plenty of space for batteries. If you use a CPU, it’s very hard to make a computer that’s smaller than 10cm (4 inches) squared, purely because of the number of individual chips that you need to squeeze in. Due to its very high level of integration and much shorter wiring, an SoC also uses considerably less power. Using SoCs, we can put complete computers in smartphones and tablets, and still have plenty of space for batteries. It’s much cheaper to build a computer using an SoC, too.