Fall 2022 CSYE7105

High Performance Parallel Machine Learning & AI

By Prof. Handan Liu

NIKITA GAURIHAR - 002980962

Masters – Information Systems, Northeastern University

ASSIGNMNENT 1

1. **What are the FLOPS? and usage in parallel computing?**

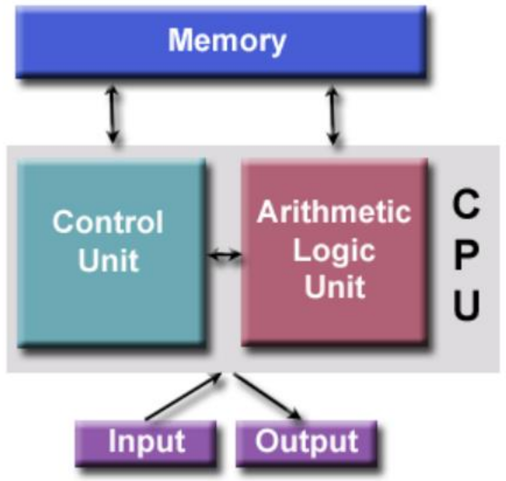
   Solution: FLOPS and its usage in Parallel Computing is as follows -

   - In parallel computing, FLOPS is a measure of computer performance.
   - It stands for Floating Point Operations per second.
   - It is majorly used in the fields of scientific computational research or real time applications which essentially need floating point calculations.
   - In parallel computing, FLOPS are mainly useful in the applications related to Science, Engineering, Industrial, Commercial, and global applications

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

   Solution: The Von-Neumann Computer Architecture is as follows -

   - The Von -Neuman architecture was derived by the well-known mathematician, John Neumann in 1945.
   - The Von-Neuman architecture is known as stored program computer where both program instructions and data are kept in electronic memory, but it differs from earlier computers which were programmed through "hard wiring"



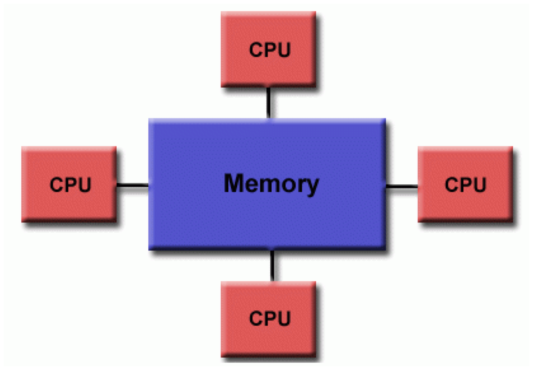
   - This architecture is comprised of four main components:
     1. **Memory –** Read/write random access memory is used to store both program instructions and data.
     2. **Control Unit –** It encodes instructions or data from the memory and decodes them followed by sequentially coordinating the operations to successfully carry out the programmed task.
     3. **Arithmetic Logic Unit –** This essentially performs the arithmetic operations
     4. **Input/Output –** It is the interface to the human operator

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

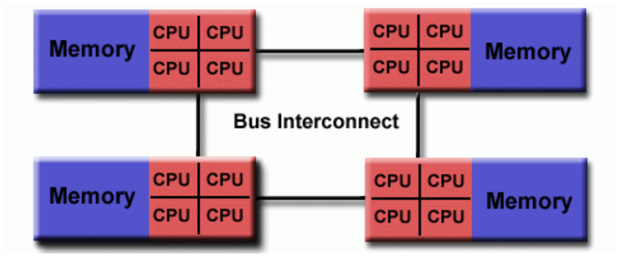
Solution:

❖ **Shared Memory Architecture:**

- As the name suggests, in a shared memory model, parallel processes asynchronously share a global address space that can be read or written to, so that the processors can communicate with each other.
- These shared memory machines have been classified as UMA (Uniform memory access) and NUMA (Non uniform memory access), based upon memory access times.
- UMA: It is represented mostly by Symmetric multiprocessor (SMP) machines where every processor is identical. It has equal access and access times to memory
- The basic shared memory architecture model for UMA is shown in the figure below –



- NUMA is often made by physically linking 2 or more SMPs where one SMP can directly access memory of another SMP by physically linking them together.
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA
- The basic shared memory architecture model for NUMA is shown in the figure below –



Advantages:

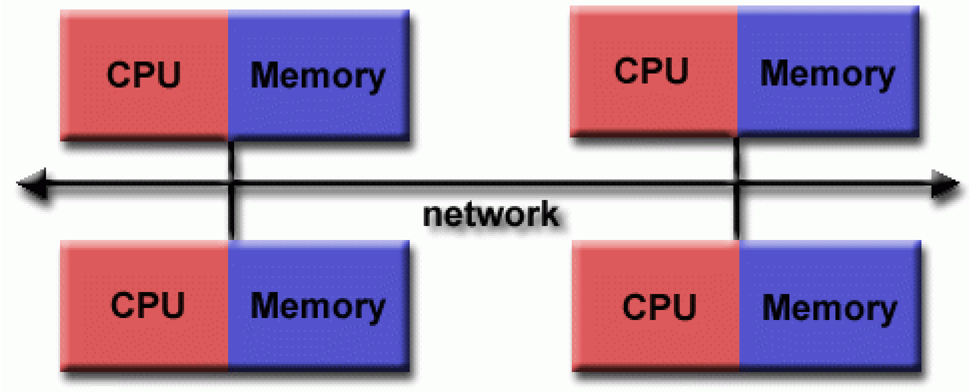
- Global address space provides a user-friendly programming perspective to memory.
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- 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.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

❖ **Distributed Memory Architechture:**

- Distributed memory systems require a communication network to connect inter-processor memory.
- There is no concept of global address space across all processors. Memory addresses in one processor do not map to another processor as they have their own local memory.
- Hence, it operates independently as each processor has its own local memory . The concept of cache coherency does not apply as changes it makes to its local memory have no effect on the memory of other processors.
- Synchronization between tasks is likewise the programmer's responsibility.



✓ **Advantages**

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

✓ **Disadvantages**

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

4. List three factors that can cause parallel overhead and simply explain.

Solution: In parallel computing, as the no. of processors increases, the performance metrics becomes challenging to achieve the satisfactory results. Listed down are various other costs we have to overcome when dealing with parallelism -

- Task(Thread or Process) start up and termination cost.
- Synchronization problems
- Cost of communication among multiple tasks.
- Software overhead of libraries, parallel compiler, and supportive OS.
- Complexity of designing, coding, debugging and maintenance.

5. Give the formula of Amdahl's Law including the number of processors.

Solution: Amdahl's law is used in parallel computing to predict the theoretical speedup when multiple processors are being used. The formula for Amdahl's Law is as follows -

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

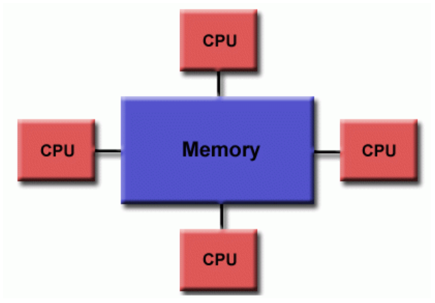
where P = parallel fraction, N = number of processors and S = serial fraction.

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

Solution: In parallel computing, there are four types of parallel programming models:

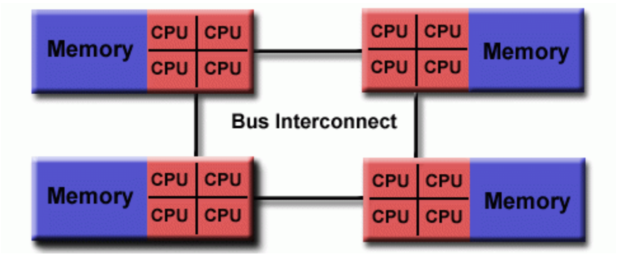
1. **Shared memory model**:

- As the name suggests, in a shared memory model, parallel processes asynchronously share a global address space that can be read or written to, so that the processors can communicate with each other.
- These shared memory machines have been classified as UMA (Uniform memory access) and NUMA (Non uniform memory access), based upon memory access times.



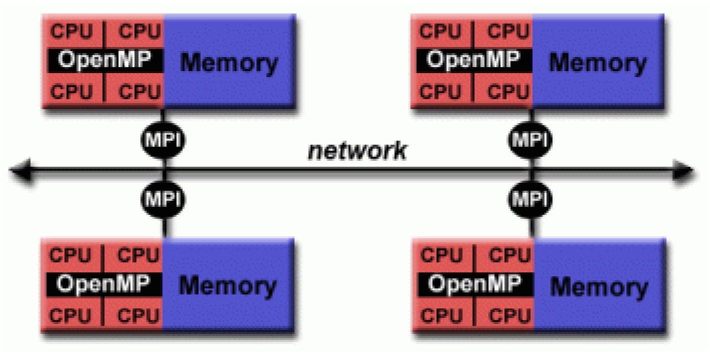
2. **Distributed memory model:**

- Distributed memory systems require a communication network to connect inter-processor memory.
- There is no concept of global address space across all processors. Memory addresses in one processor do not map to another processor as they have their own local memory.
- Hence, it operates independently as each processor has its own local memory . The concept of cache coherency does not apply as changes it makes to its local memory have no effect on the memory of other processors.

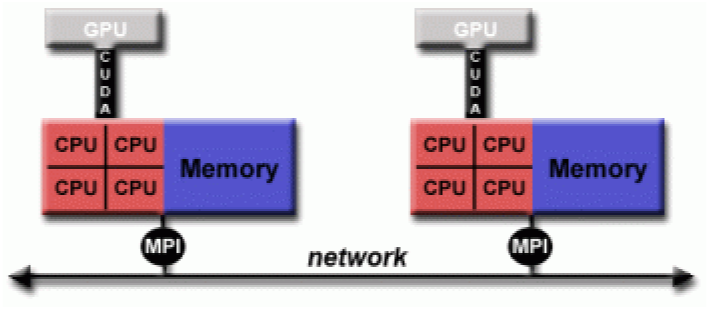


3. **Hybrid Memory Model:**

- A hybrid model combines more than one of the previously described programming models.
- Combination of the message passing model (MPI) with the threads model (OpenMP) is the best example of Hybrid model.



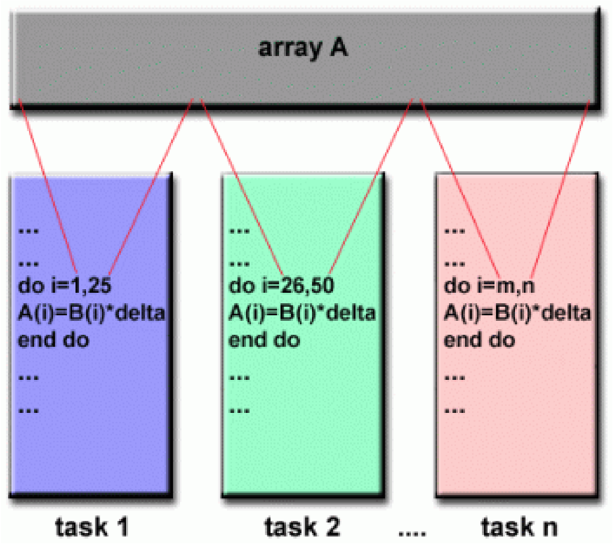
Hybrid model with MPI and OpenMP



Hybrid model with MPI and CUDA

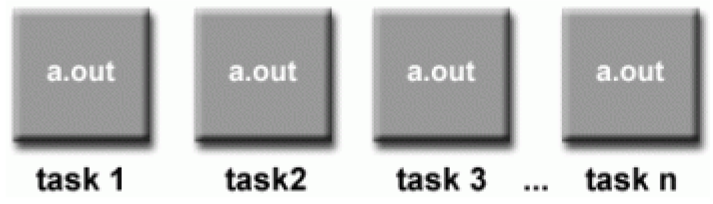
4. **Data parallel model.**

- The data parallel model demonstrates the following characteristics:
    - They have a global address space.
    - As most of the parallel work is focused on performing operations on a data set, it is organized into an array or cube-like structure.
    - A set of tasks work collectively on the same data structure, but each task works on a varied partition of the same data structure.
- Tasks are performed by the same operation on their partition of work.
- All tasks have access to the data structure through global memory in case of shared memory architectures whereas the global data structure can be split up logically and/or physically across tasks in case of distributed memory architectures



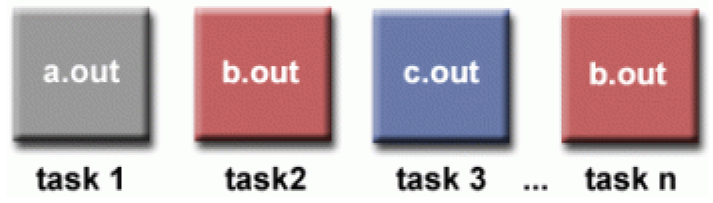
5. **SPMD - Single Program Multiple Data:**

- A "high level" programming model be created by combining any of the parallel programming paradigms
- For a Single Program, each task runs a separate instance of the same program concurrently. This program can run in parallel, use message passing, use threads, or be hybrid, whereas in case of Multiple Data, all tasks may use different data



6. **MPMD - Multiple Program Multiple Data:**

- It is a high-level programming model that can be derived using any combination of the previously mentioned parallel programming models.
- Unlike SMPD, in case of MPMP, tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid whereas in case of Multiple Data, all tasks may use different data



7. **Simply give 3 possibilities for poor parallel performance and simply explain.**

Solution: The 3 possibilities for poor parallel performance are –

1. False Cache-Line Sharing:

- A common problem that is faced by machines with multiple cores is when a value is read from the CPU it is stored in cache at memory, and it also saves the location of cache in memory. I
- f one core fails to validate a particular memory location, the version of that memory location cached by another core gets invalidated.
- Main problem is the two cores don't even have to be writing into the same memory location. The same problem occurs also if they are writing into two memory locations that are on the same cache line.

2. Locality Issues:

- Locality is affected negatively when a program is being modified to run in parallel.
- Two tasks can be created on a dual-core machine –
  1. to perform the operation on the elements with even indices
  2. to handle odd indices.
- The elements that a thread needs to access are interleaved with elements it does not care about, thus creating a negative consequence.
- Each cache line will contain only half as many elements as it did previously, which may mean that twice as many memory accesses will go all the way to the main memory.

3. Load Balancing:

- In Parallel computing, even though the parallelization CPU work exists, it needs to balance the chunkc of work evenly on all the cores and processors.
- These chunks of work vary a lot when it comes to executing the tasks required to successfully compile the programs.
- This is the common issue that we need to deal with when we work on our codes in parallel computing.

8. Briefly explain the hybrid parallel programming model on current supercomputers and HPC clusters.

Solution: Following is the brief idea about the Hybrid Parallel programming –

- As the name suggests, a combination of more than one of the previously described programming models is said to be the hybrid model
  For e.g.: combination of the message passing model (MPI) with the threads model (OpenMP
- Threads perform computationally intensive kernels using local, on-node data and 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.
- 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:
  1. MPI with Pthreads
  2. MPI with non-GPU accelerators

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

Solution: Three primary components of the API are: –

- **Compiler Directives** (c/c++) - for (…)
- **Runtime Library Routines** – LD_LIBRARY_PATH
- **Environment Variables** - OMP_NUM_THREADS

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 Linux operating system by using "export"; and run this executable.

Note: only give the commands.

Solution:

- **Setting the environment variable**
  ```
  export OMP_NUM_THREADS = 4
  ```
- **Command to compile the file**
  ```
  gcc -o omp_hello omp_hello.c -fopenmp
  ```
- **Command to run the file after successful compilation**
  ```
  .\omp_hello
  ```

References

1. https://devblogs.microsoft.com/pfxteam/most-common-performance-issues-in-parallel-programs/
2. https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial##Neumann
3. Canvas – Teaching Slides