Introduction to Distributed and Parallel Computing
CS-401

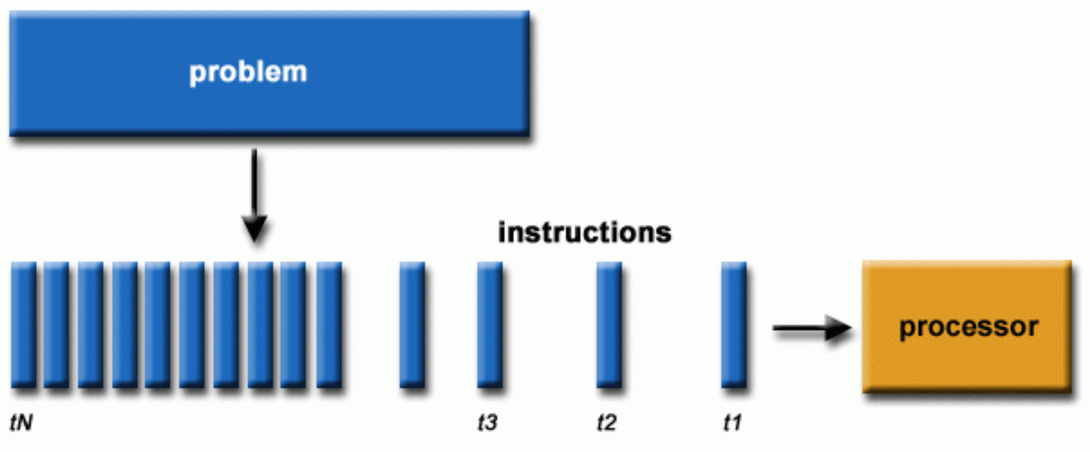
Dr. Sanjay Saxena
Visiting Faculty, CSE, IIIT Vadodara
Assistant Professor, CSE, IIIT Bhubaneswar
Post doc – University of Pennsylvania, USA
PhD – Indian Institute of Technology(BHU), Varanasi

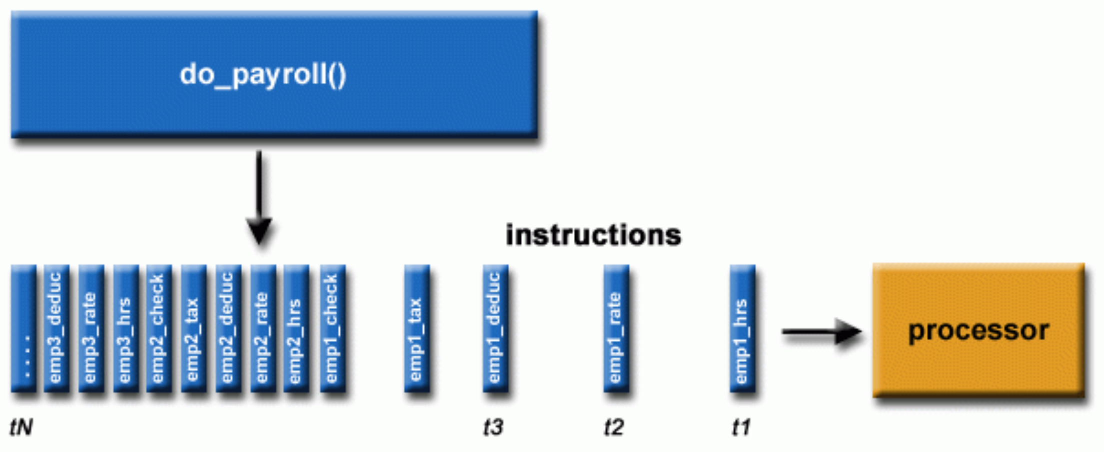
What is Parallel Computing?

Parallel computing refers to the process of executing several processors an application or computation simultaneously. Generally, it is a kind of computing architecture where the large problems break into independent, smaller, usually similar parts that can be processed in one go.

Traditionally, software has been written for ***serial*** computation:

- ➢ A problem is broken into a discrete series of instructions
- ➢ Instructions are executed sequentially one after another
- ➢ Executed on a single processor
- ➢ Only one instruction may execute at any moment in time

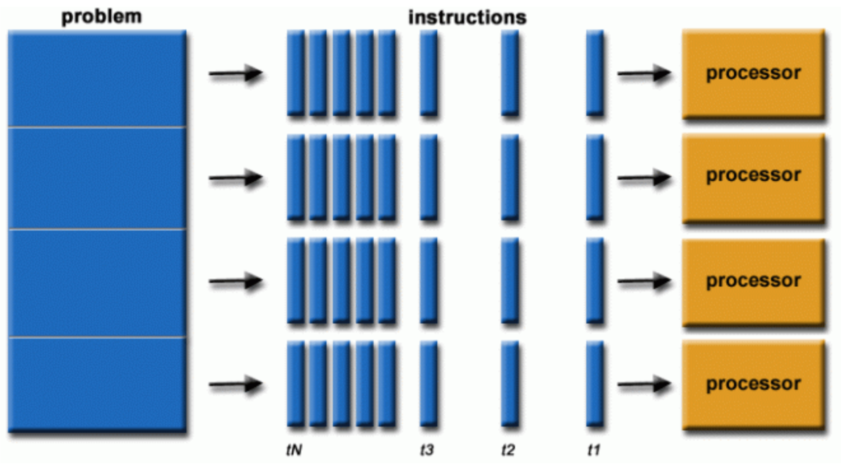


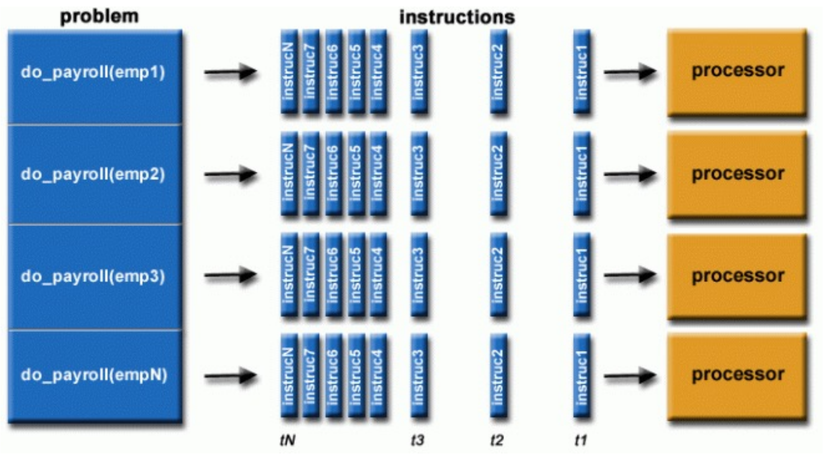


Contd..

In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem:

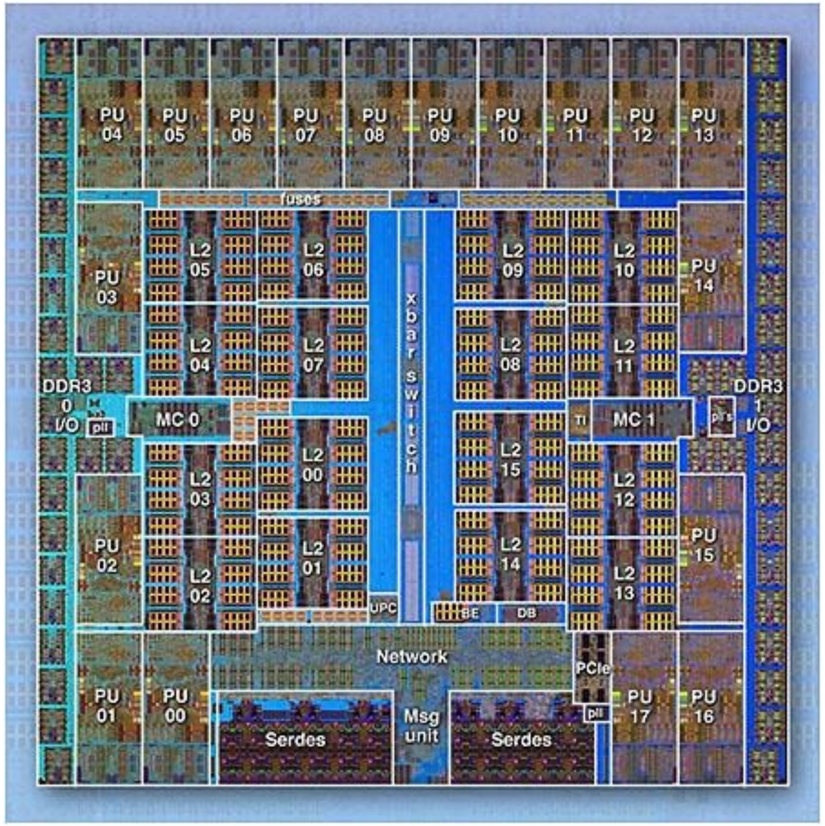
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed
- The computational problem should be able to:
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple compute resources than with a single compute resource.
- The compute resources are typically:
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network





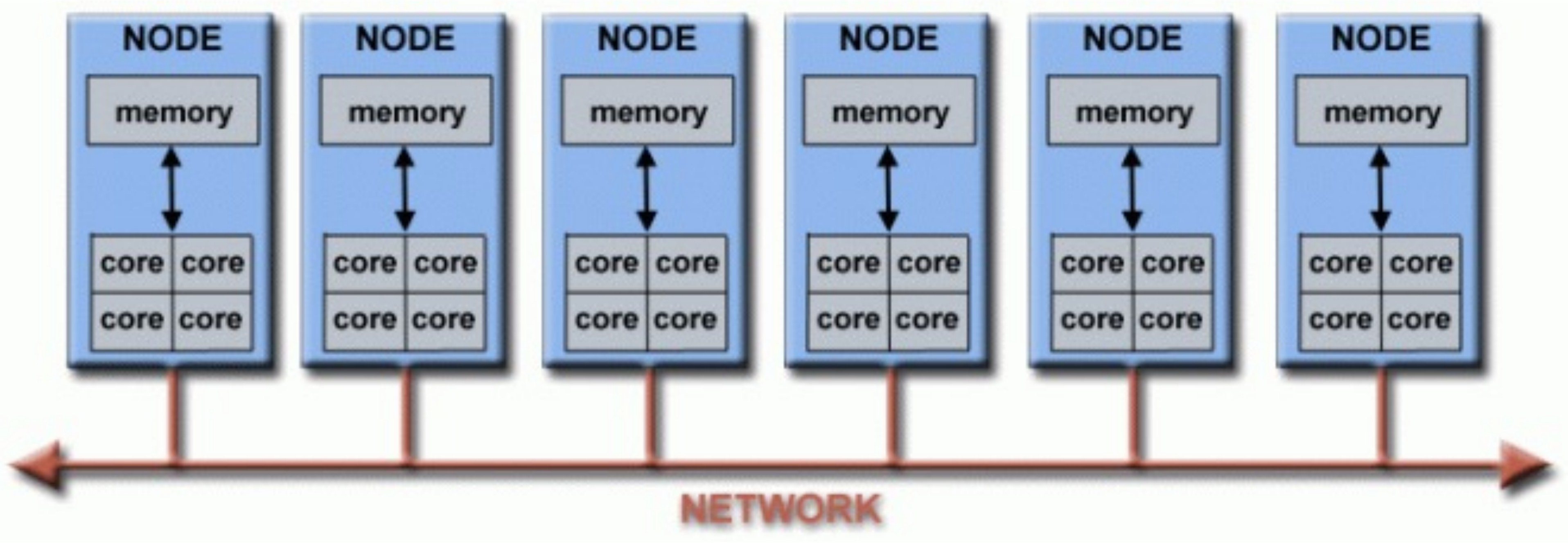
Parallel Computers

- Virtually all stand-alone computers today are parallel from a hardware perspective:
  - Multiple functional units (L1 cache, L2 cache, branch, prefetch, decode, floating-point, graphics processing (GPU), integer, etc.)
  - Multiple execution units/cores
  - Multiple hardware threads



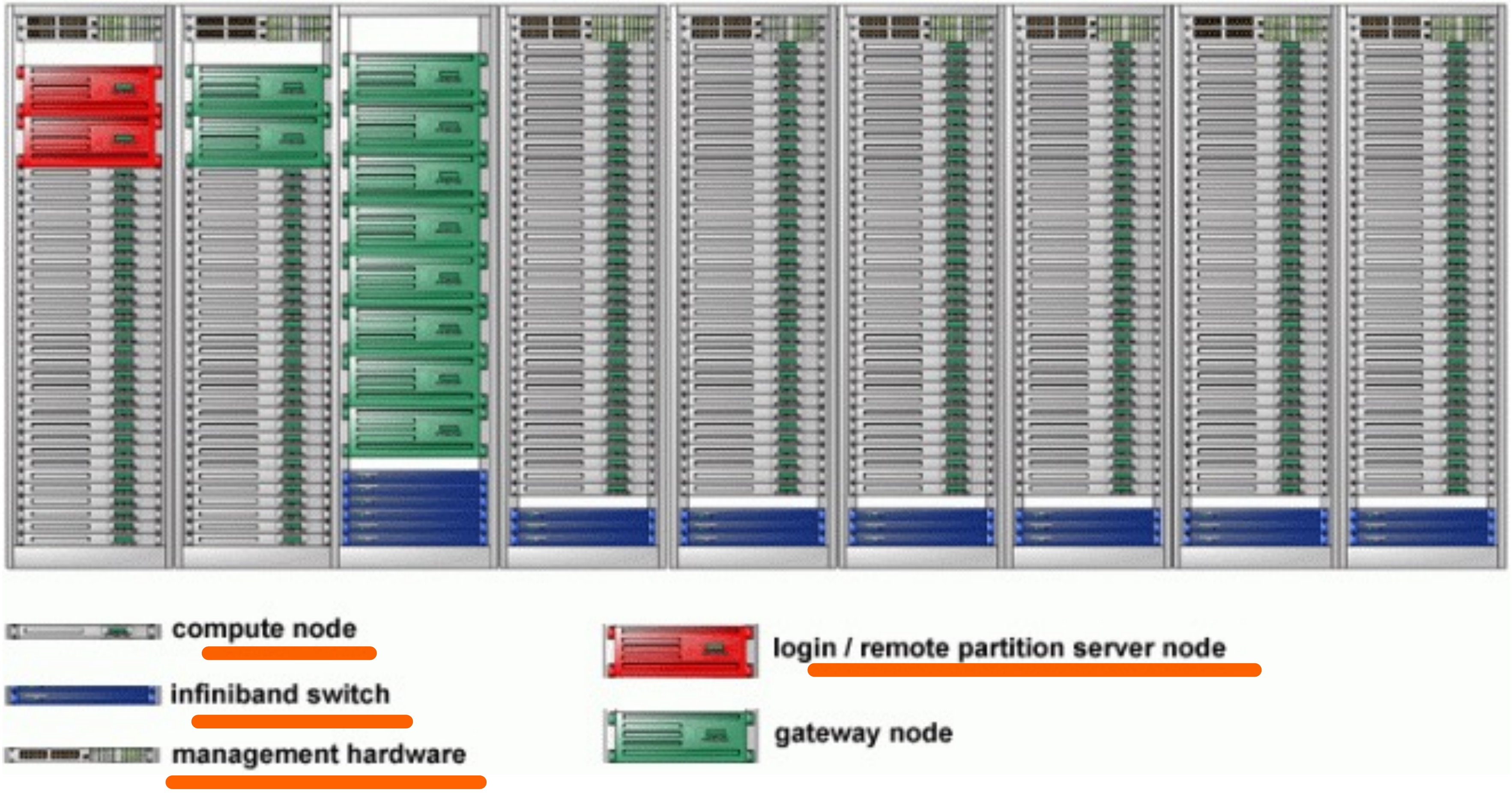
IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.



Network connections

- For example, the schematic below shows a typical LLNL parallel computer cluster:
  - Each compute node is a multi-processor parallel computer in itself
  - Multiple compute nodes are networked together with an Infiniband network
  - Special purpose nodes, also multi-processor, are used for other purposes



Benefits of Parallel Programming

➢ **SAVE TIME AND/OR MONEY**

- In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
- Parallel computers can be built from cheap, commodity components.

Working in parallel shortens completion time

➢ **SOLVE LARGER / MORE COMPLEX PROBLEMS**

• Many problems are so large and/or complex that it is impractical or impossible to solve them using a serial program, especially given limited computer memory.

• Example: "Grand Challenge Problems" (en.wikipedia.org/wiki/Grand_Challenge) requiring petaflops and petabytes of computing resources.

• Example: Web search engines/databases processing millions of transactions every second

Parallel computing can solve increasingly complex problems

➢ PROVIDE CONCURRENCY

• A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously.

• Example: Collaborative Networks provide a global venue where people from around the world can meet and conduct work "virtually."

Collaborative networks

➢ TAKE ADVANTAGE OF NON-LOCAL RESOURCES

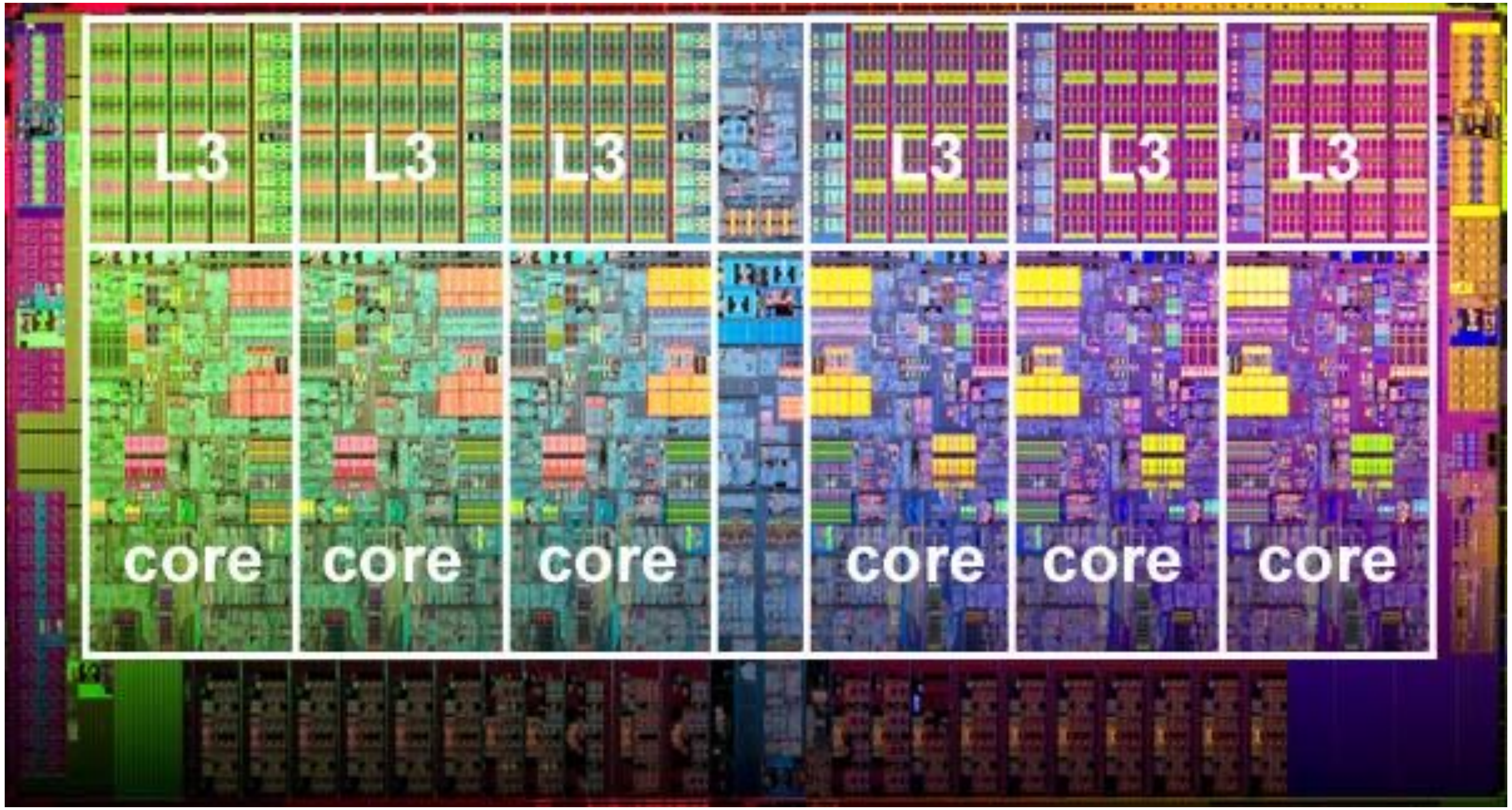
• Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.

• Example: SETI@home (setiathome.berkeley.edu) has over 1.7 million users in nearly every country in the world (May, 2018).

SETI has a large worldwide user base

➢ MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE

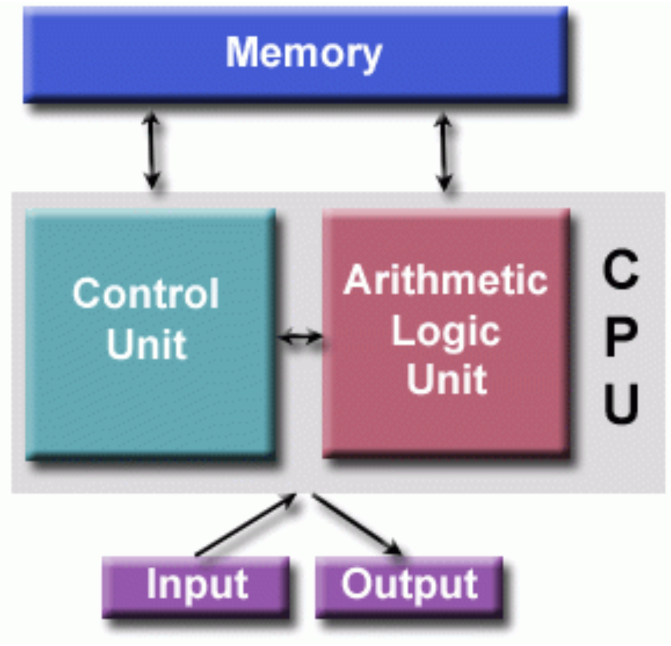
- Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
- Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
- In most cases, serial programs run on modern computers "waste" potential computing power.



Parallel computing cores

Basic Terminology

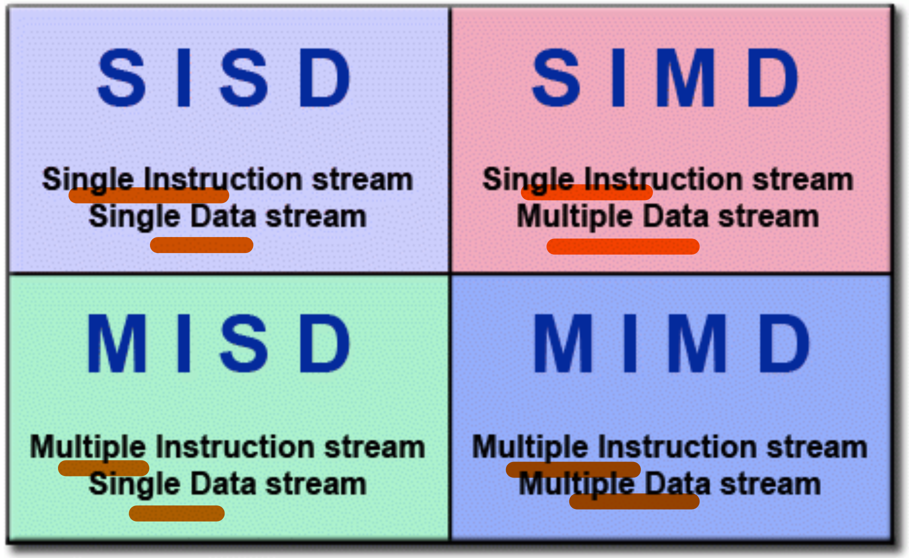
von Neumann Computer Architecture



- Named after the Hungarian mathematician John von Neumann who first authored the general requirements for an electronic computer in his 1945 papers.
- Also known as "stored-program computer" - both program instructions and data are kept in electronic memory. Differs from earlier computers which were programmed through "hard wiring".
- Since then, virtually all computers have followed this basic design:
- Comprised of four main components: Memory, Control Unit, Arithmetic Logic Unit, Input/Output
- Read/write, random access memory is used to store both program instructions and data
- Program instructions are coded data which tell the computer to do something
- Control unit fetches instructions/data from memory, decodes the instructions and then ***sequentially*** coordinates operations to accomplish the programmed task.
- Arithmetic Unit performs basic arithmetic operations
- Input/Output is the interface to the human operator

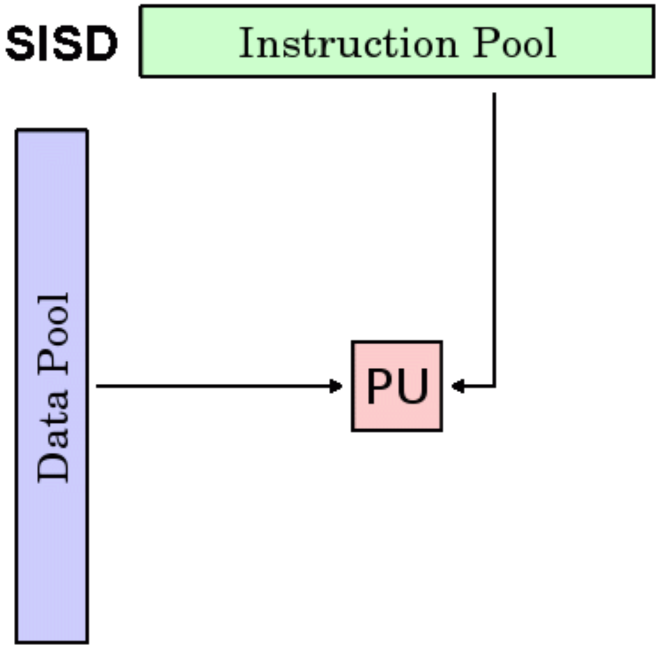
Flynn's Classical Taxonomy

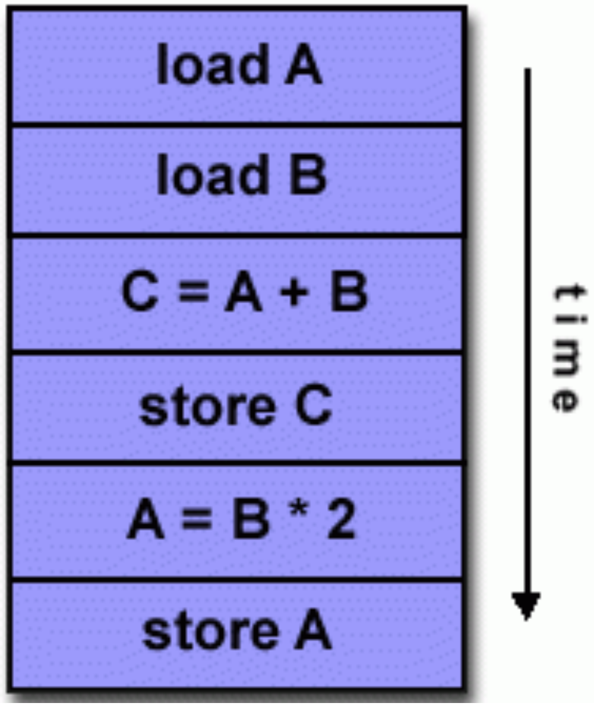
- There are a number of different ways to classify parallel computers. Examples are available in the references.
- One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction Stream*** and ***Data Stream***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.
- The matrix below defines the 4 possible classifications according to Flynn:



Single Instruction, Single Data (SISD)

- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest type of computer
- Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.

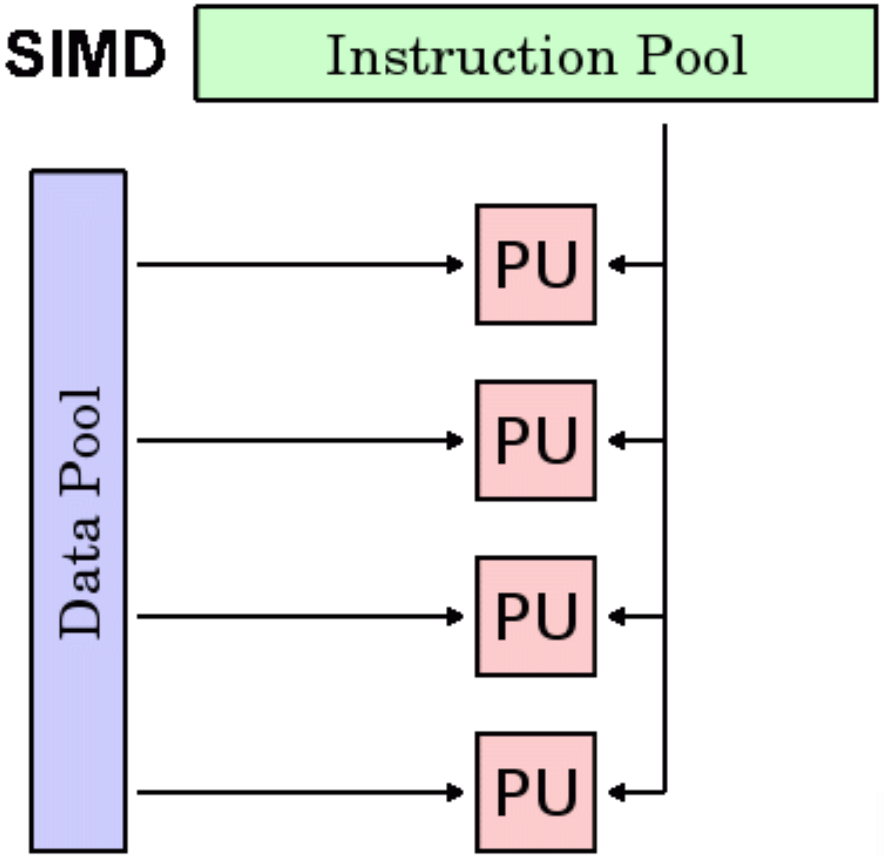




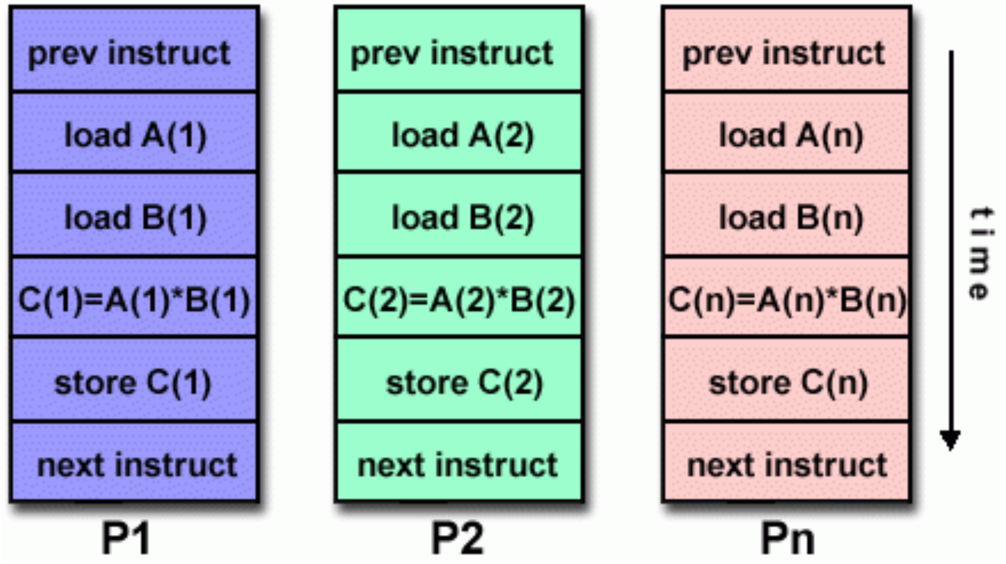
Single Instruction, Multiple Data (SIMD)

- A type of parallel computer
- **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.
- Synchronous (lockstep) and deterministic execution
- 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
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

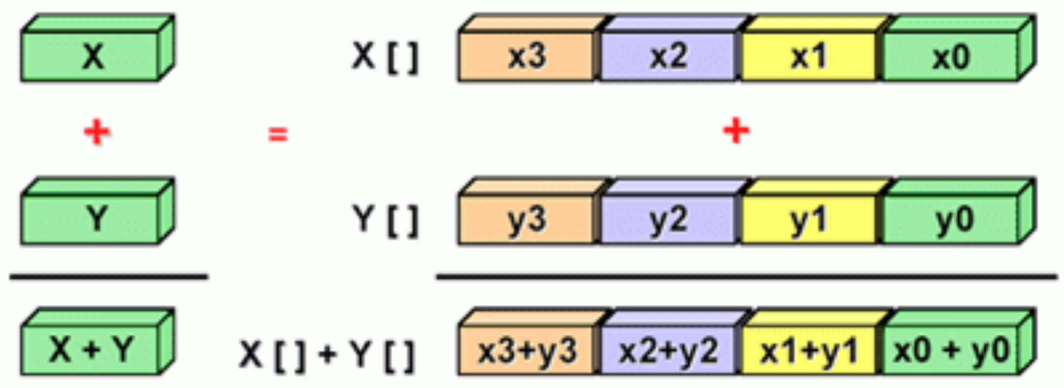
SIMD



Instruction Pool
Data Pool
PU
PU
PU
PU



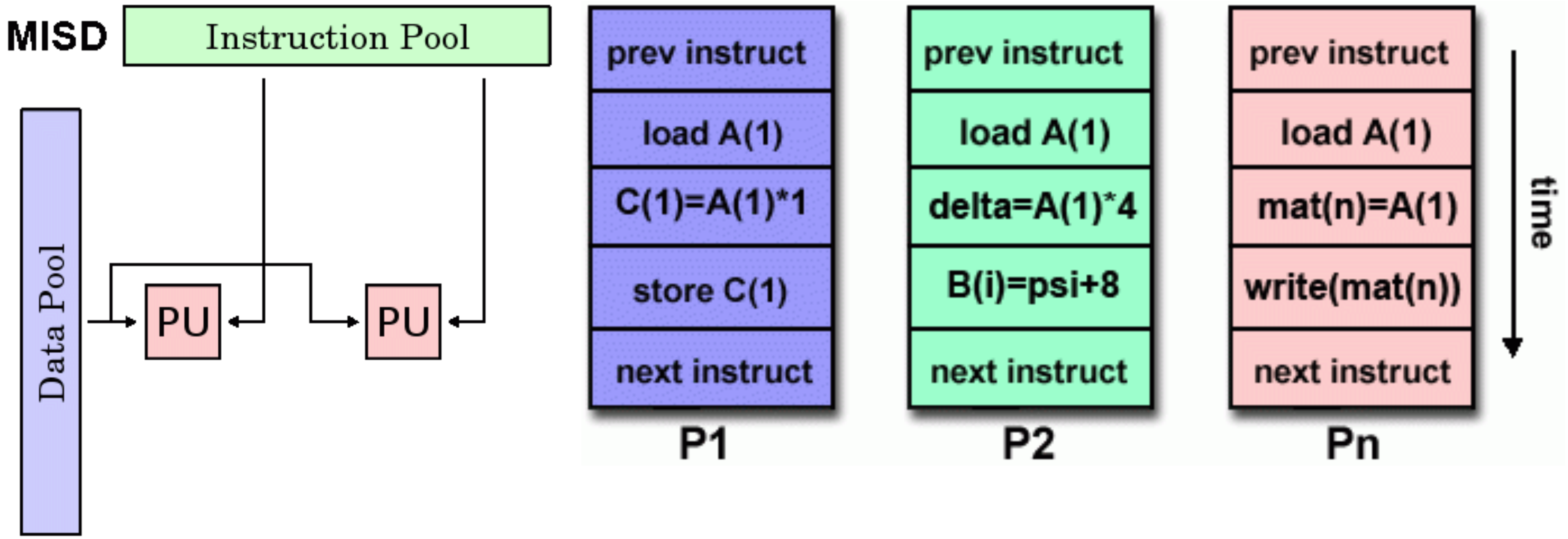
prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time



X
+
Y
X + Y
=
X [ ]
x3
x2
x1
x0
+
Y [ ]
y3
y2
y1
y0
X [ ] + Y [ ]
x3+y3
x2+y2
x1+y1
x0 + y0

Multiple Instruction, Single Data (MISD)

- A type of parallel computer
- **Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.
- Few (if any) actual examples of this class of parallel computer have ever existed.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.



Multiple Instruction, Multiple Data (MIMD)

•A type of parallel computer

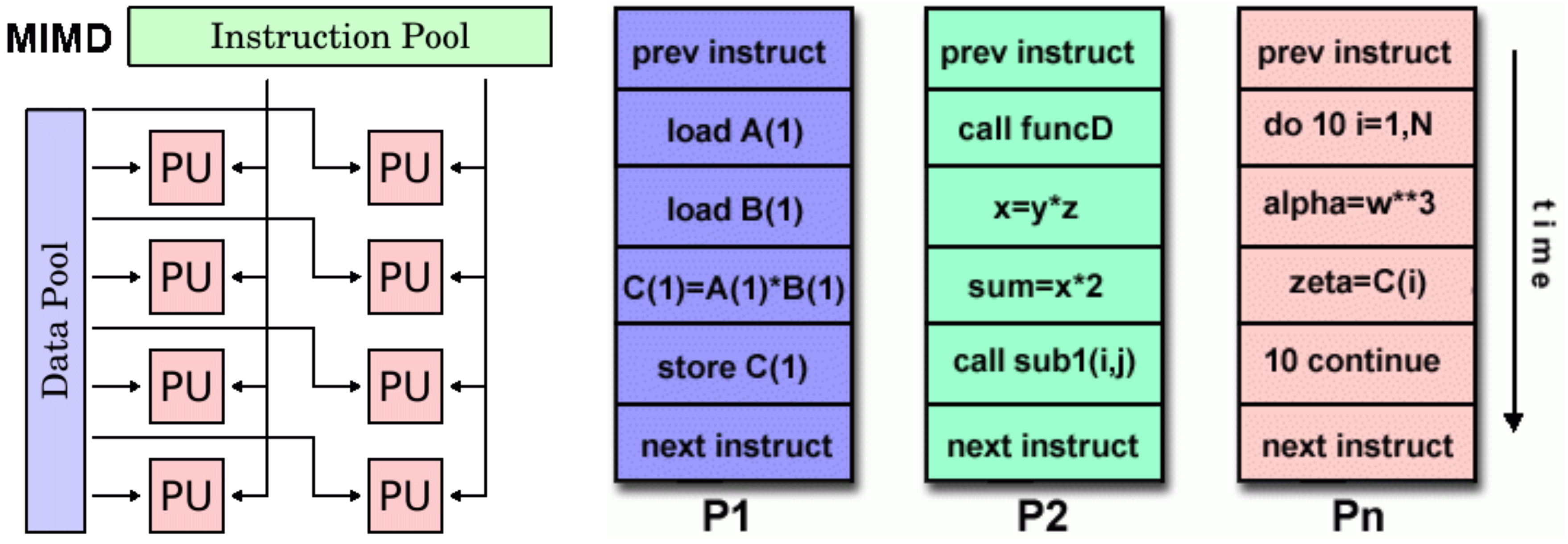
•**Multiple Instruction:** Every processor may be executing a different instruction stream

•**Multiple Data:** Every processor may be working with a different data stream

•Execution can be synchronous or asynchronous, deterministic or non-deterministic

•Currently, the most common type of parallel computer - most modern supercomputers fall into this category.

•Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.



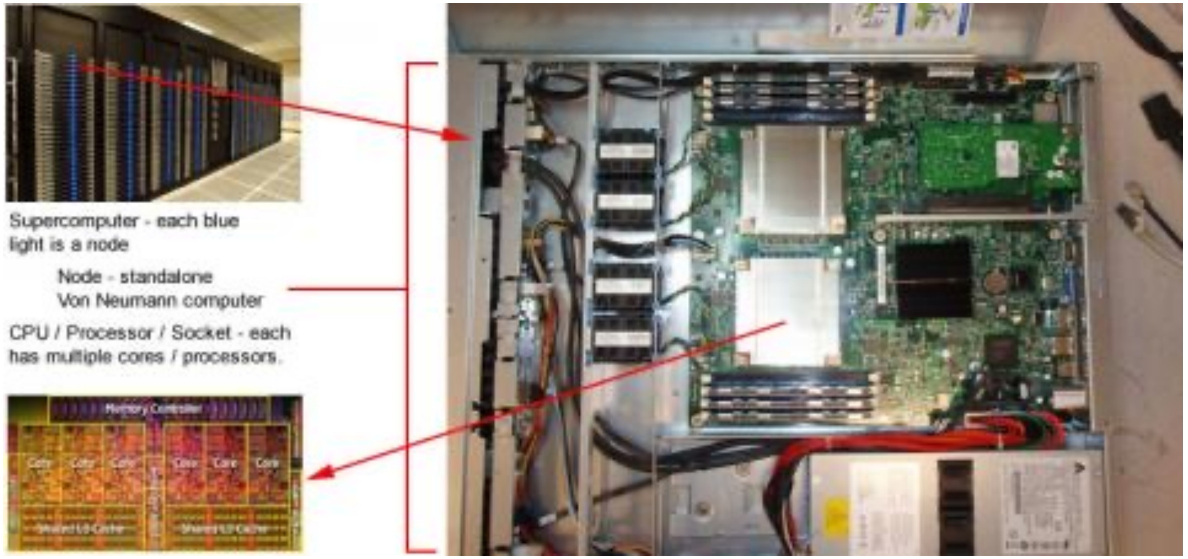
General Parallel Computing Terminology

- **CPU**

Contemporary CPUs consist of one or more cores - a distinct execution unit with its own instruction stream. Cores with a CPU may be organized into one or more sockets - each socket with its own distinct memory . When a CPU consists of two or more sockets, usually hardware infrastructure supports memory sharing across sockets.

- **Node**

A standalone "computer in a box." Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.



Nodes in a supercomputer

- **Task**

A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor. A parallel program consists of multiple tasks running on multiple processors.

- **Pipelining**

Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

- **Shared Memory**

Describes a computer architecture where all processors have direct access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

- **Symmetric Multi-Processor (SMP)**

Shared memory hardware architecture where multiple processors share a single address space and have equal access to all resources - memory, disk, etc.

➢ Distributed Memory

In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

➢ Communications

Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network.

➢ Synchronization

The coordination of parallel tasks in real time, very often associated with communications.

Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

➢ Computational Granularity

In parallel computing, granularity is a quantitative or qualitative measure of the ratio of computation to communication.

•***Coarse:*** relatively large amounts of computational work are done between communication events

•***Fine:*** relatively small amounts of computational work are done between communication events

- **Observed Speedup**

wall-clock time of serial execution/wall-clock time of parallel execution

One of the simplest and most widely used indicators for a parallel program's performance.

- **Parallel Overhead**

Required execution time that is unique to parallel tasks, as opposed to that for doing useful work. Parallel overhead can include factors such as:

•Task start-up time

•Synchronizations

•Data communications

•Software overhead imposed by parallel languages, libraries, operating system, etc.

•Task termination time

- **Massively Parallel**

Refers to the hardware that comprises a given parallel system - having many processing elements. The meaning of "many" keeps increasing, but currently, the largest parallel computers are comprised of processing elements numbering in the hundreds of thousands to millions.

- **Embarrassingly (IDEALY) Parallel**

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

- **Scalability**

Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources. Factors that contribute to scalability include:

• Hardware - particularly memory-cpu bandwidths and network communication properties

• Application algorithm

• Parallel overhead related

• Characteristics of your specific application

Types of Parallelism:

1.Bit-level parallelism –

It is the form of parallel computing which is based on the increasing processor’s size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.

Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.

2. Instruction-level parallelism –

A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

3. Task Parallelism –

Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform the execution of sub-tasks concurrently.

4. Data-level parallelism (DLP) –

Instructions from a single stream operate concurrently on several data – Limited by non-regular data manipulation patterns and by memory bandwidth

Limitations of Parallel Computing:

- It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
- The algorithms must be managed in such a way that they can be handled in a parallel mechanism.
- The algorithms or programs must have low coupling and high cohesion. But it’s difficult to create such programs.
- More technically skilled and expert programmers can code a parallelism-based program well.

Thanks & Cheers!!

Small aim is a crime; have great aim.

Bharat-Ratan A. P. J. Abdul Kalam