CS3006 Parallel and Distributed Computing

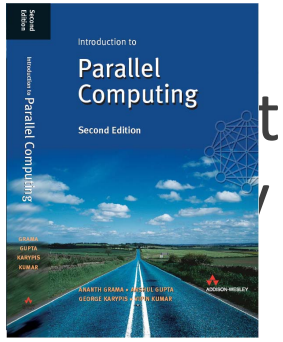
FALL 2022

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Instructor

Muhammad Danish Khan
Lecturer, Department of Computer Science
FAST NUCES Karachi
m.danish@nu.edu.pk

Recommended Text Book



troduction to Parallel Computing, Second Edition
Ananth Grama

Marks Distribution and Deadlines

Assessment	Weightage	Schedule/Deadlines/ Remarks
Mid-term I Examinations	15	6th Week
Mid-term II Examinations	15	12th Week
Quizzes/Assignments	10	**3 Quizzes** + **1**-**2** Assignment Tasks per Week with Plagiarism Checking
Semester Project	10	Project idea submission: **7**th week, Final Submission due in **15**th Week of the semester
Final Examinations	50	Comprehensively from all the covered and assigned topics.

Pre-Requisite

- Operating System Concepts
- Algorithms

Semester Plan

- **Week 1**: Introduction to parallel computing
- **Week 2**: Parallel Programming Platforms
- **Week 3**: Principles of Parallel Algorithm Design
- **Week 4**: Programming Shared Address Space (MPI) , **Assignment-1**
- **Week 5**: Programming Shared Address Space, **Quiz-1**
- **Week 6**: Mid Term-1 Examinations

Semester Plan

- **Week 7**: Programming Using the Message Passing Paradigm**, Project Proposal Submission**
- **Week 8**: Programming Using the Message Passing Paradigm
- **Week 9**: Distributed File System (HDFS)
- **Week 10**: Map Reduce Framework, **Assignment 2, Quiz-2**
- **Week 11**: **Mid Term-2 Examinations**

Semester Plan

- **Week 12:** Map Reduce Framework
- **Week 13**: Introduction to Data Parallelism and CUDA C
- **Week 14**: Data-Parallel Execution Model
- **Week 15**: Data-Parallel Execution Model, **Assignment-3, Quiz-3, Project Evaluations**
- **Week 16**: **Project Evaluations**

LMS: **Google Class Room**

- Section 5B Class Code: **bjc6ek5**

- Section 5D Class Code*:* ***tty4qde***

Some Relevant Concepts

Program

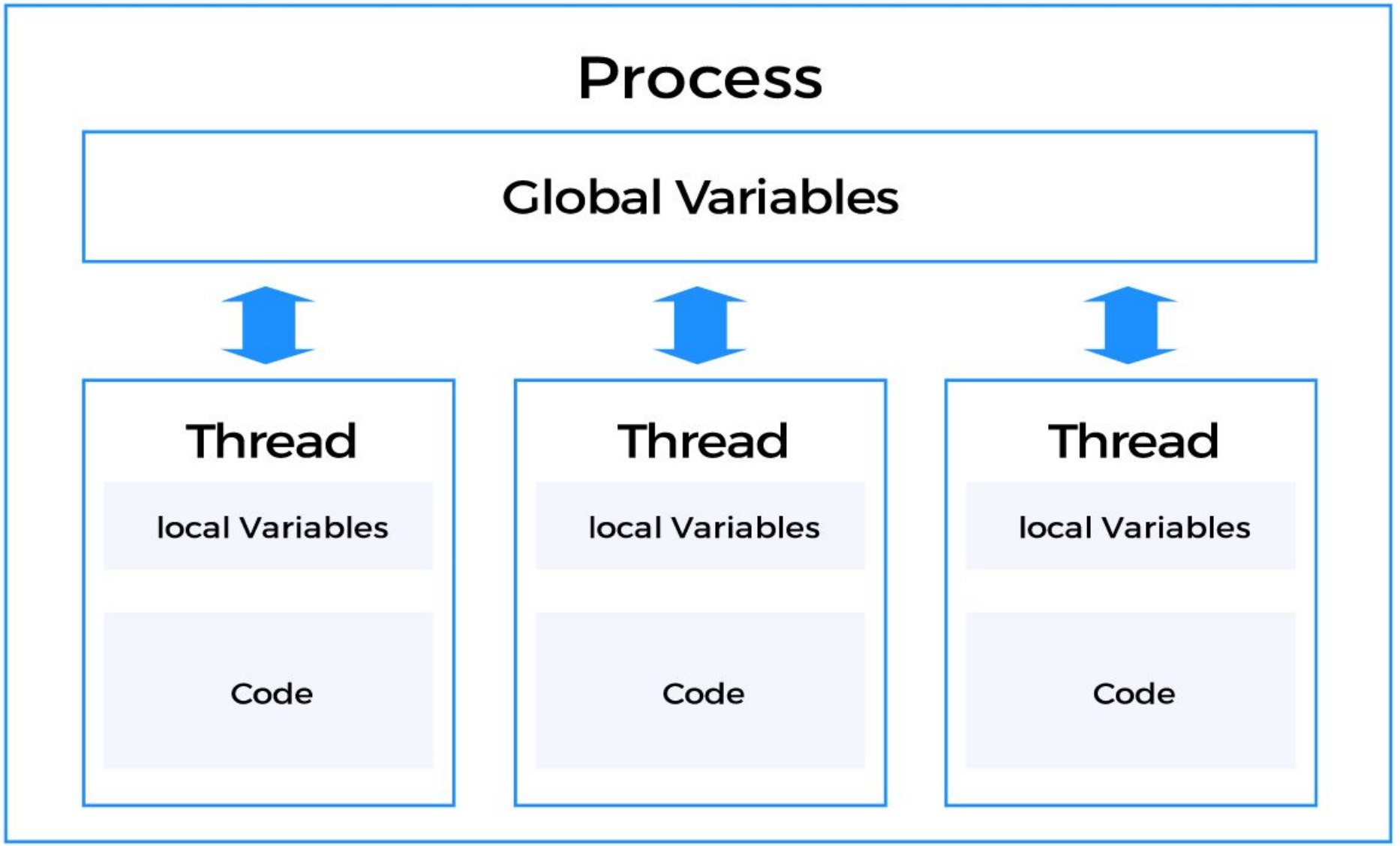
- Set of instructions and associated data
- resides on the **disk** and is **loaded** by the operating system to perform some task.
- E.g. An executable file or a python script file.

Process

- A program in execution.
- In order to run a program, the **operating system's kernel** is first asked to create a new **process**, which is an environment in which a program executes.
- consists of instructions, user-data, and system-data segments, CPU, memory, address-space, disk acquired at runtime

Thread

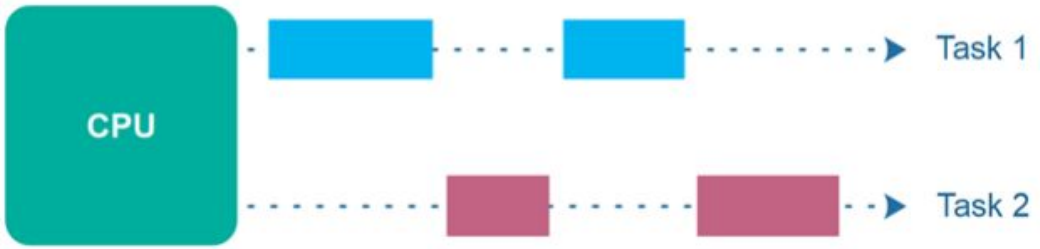
- the smallest unit of execution in a *process*.
- A thread simply executes instructions serially.
- A process can have multiple threads running as part of it.
- **Processes don't share any resources amongst themselves whereas threads of a process can share the resources allocated to that particular process, including memory address space.**


Process
Global Variables
Thread
local Variables
Code
Thread
local Variables
Code
Thread
local Variables
Code

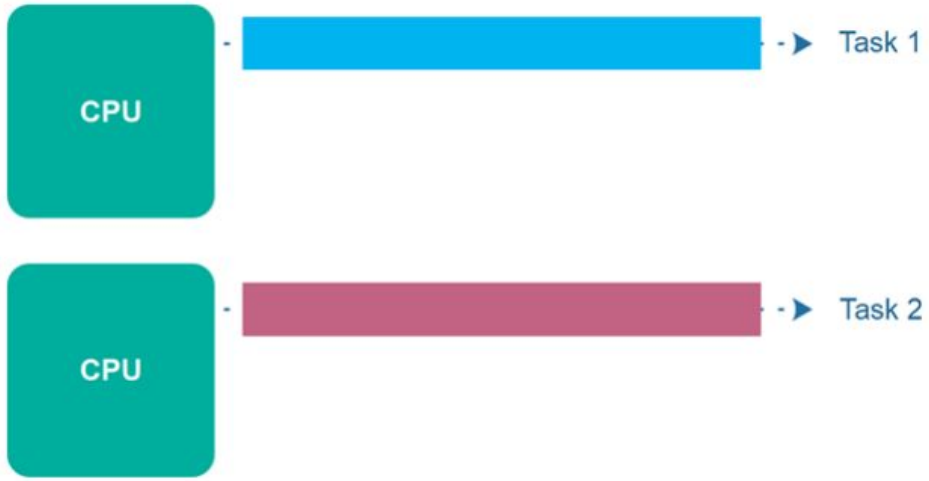
- “**Multiprocessing** systems”
  - where multiple processes get scheduled on more than one CPU.
  - Usually, this requires hardware support where a single system comes with multiple cores
  - or the execution takes place in a *cluster* of machines.

- Multiple Processors vs Multiple Cores

Concurrency

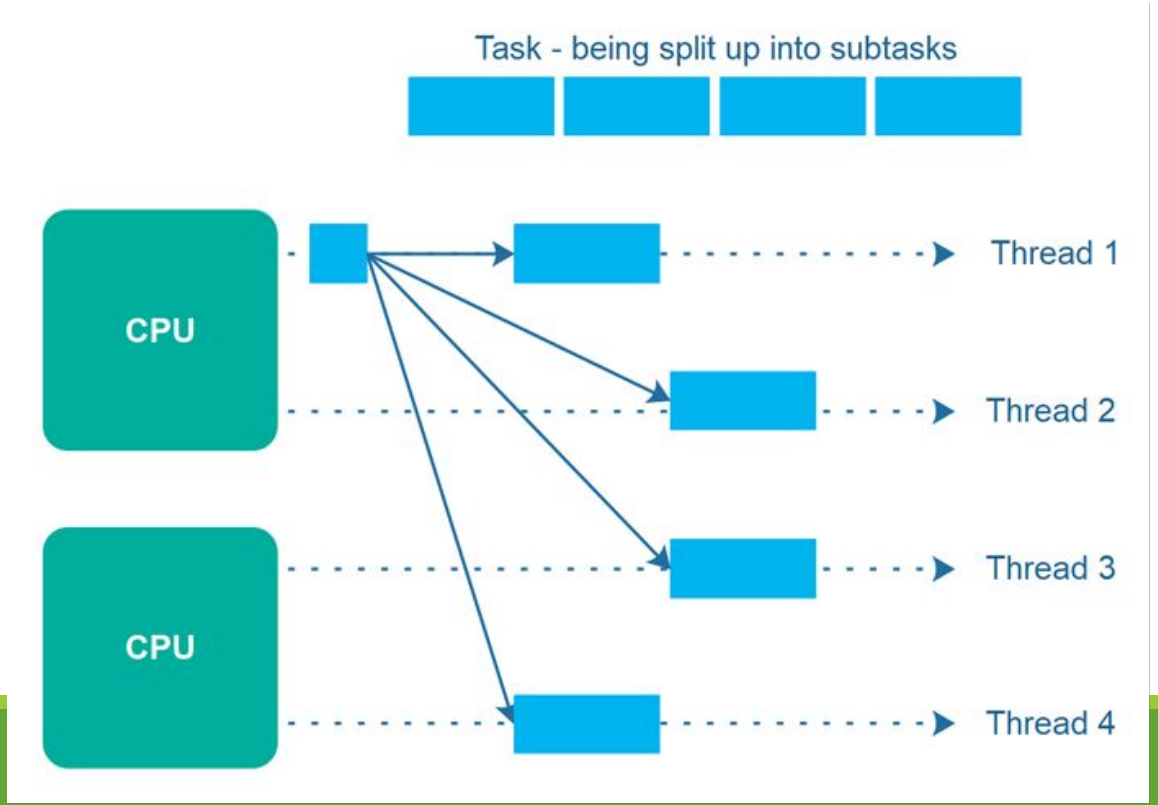


Parallel Execution



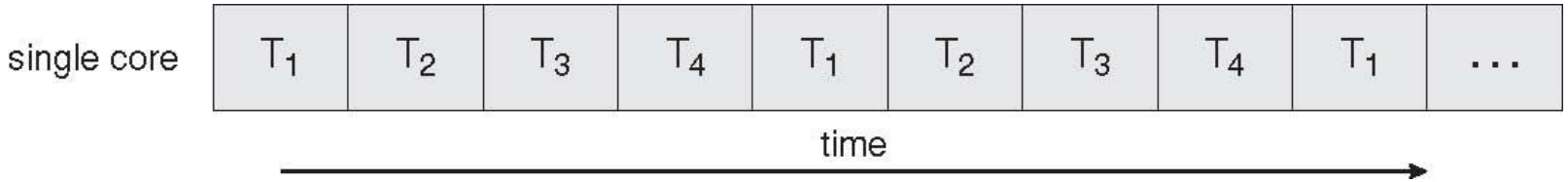
Parallelism

- The term *parallelism* means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time.

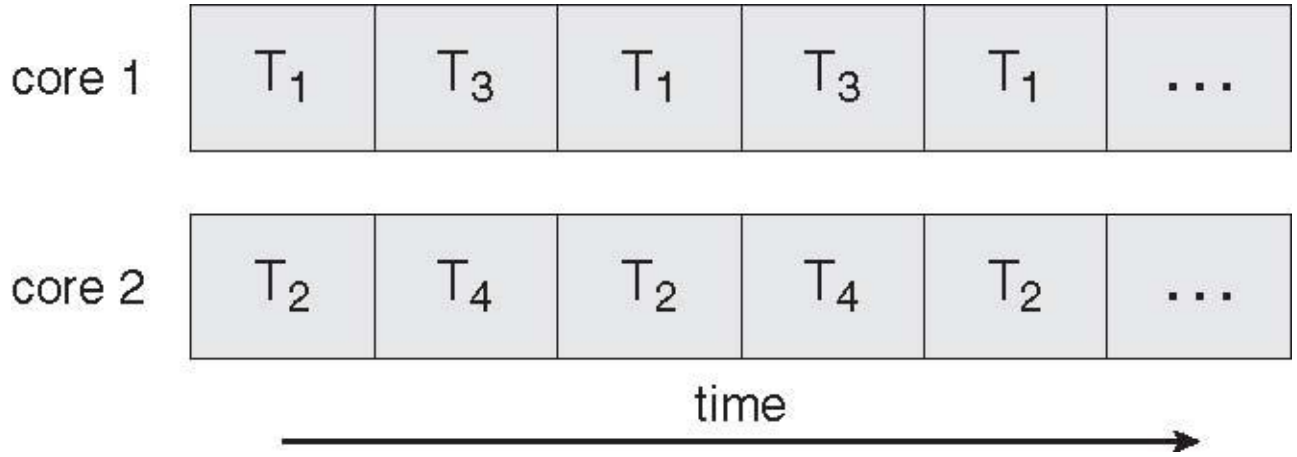


Serial Execution vs. Parallel Execution

Concurrent Execution on a Single-core System



Parallel Execution on a Multicore System



Limitations of Serial Computing

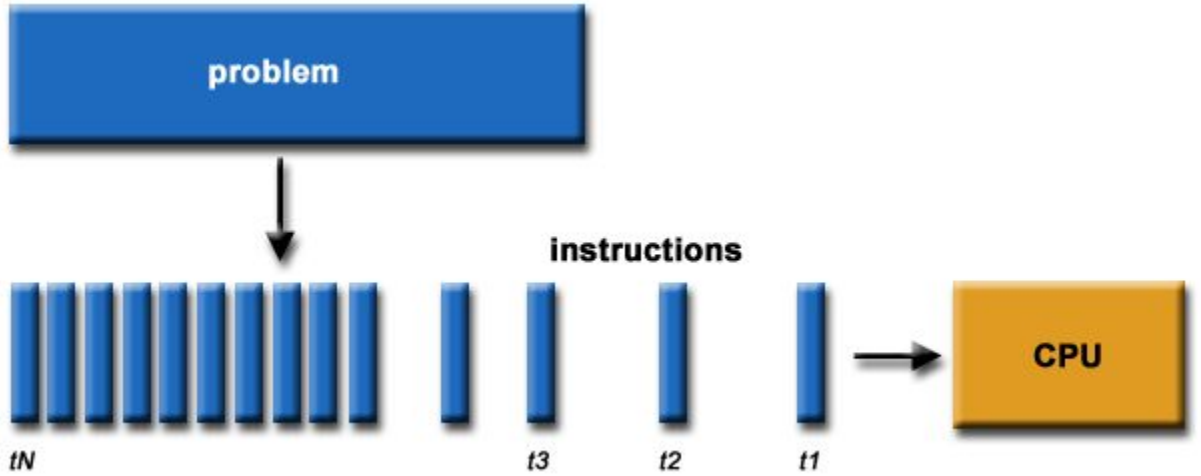
- Limits to serial computing - both physical and practical reasons pose significant constraints to simply building ever faster serial computers.

- Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware.
  - Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.

- Economic limitations - it is increasingly expensive to make a single processor faster.
  - Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

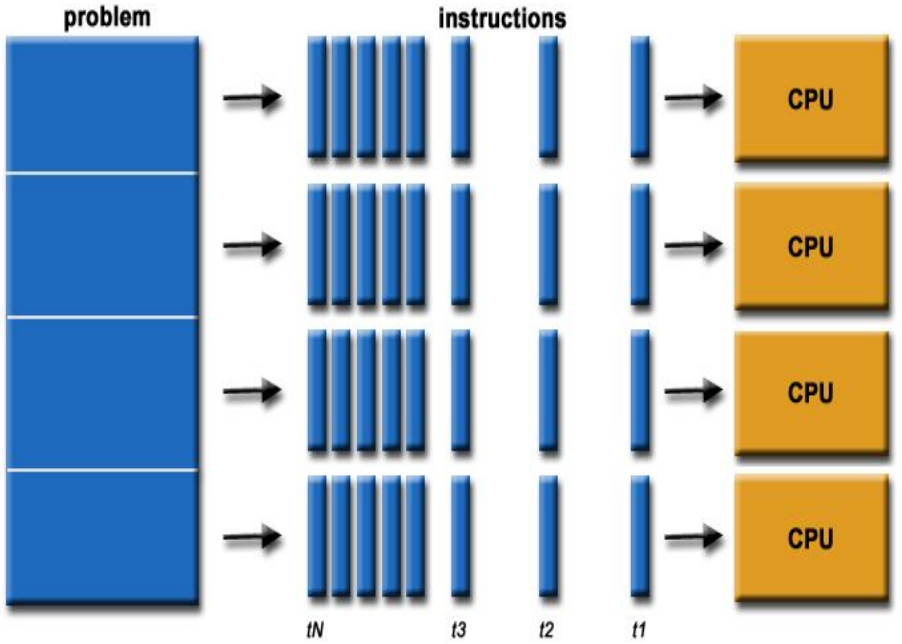
Parallel Computing

- Traditionally, software has been written for ***serial*** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.



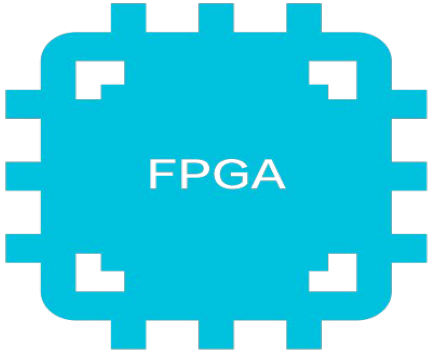
Parallel Computing

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple **computing resources** to solve a computational problem.
  - To be run using multiple CPUs
  - 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 CPUs



Parallel Computing: Resources

- The compute resources can include:
  - A single computer with multiple processors;
  - A single computer with (multiple) processor(s) and some specialized computer resources (GPU, FPGA …)
  - An arbitrary number of computers connected by a network (Cluster)
  - A combination of both.



Parallel Computing: The computational problem

- The computational problem usually demonstrates characteristics such as the ability to be:
  - Broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Solved in less time with multiple compute resources than with a single compute resource.

```
LD   $12, (100)
ADD      $11, $12
SUB $10, $11
INC  $10
SW  $13, ($10)
```

```
int    sample1
{
        X = sample2()
        Return x;
}
floatsample3
{
        Pi=3.14
        Return pi
}
Int    sample2()
{
        Cin>>I
        Return I;
}
```

Parallel Computing: what for?

- Example applications include:
  - Parallel Databases, Data Mining
  - Web Search Engines, Web Based Business Services
  - Computer-aided diagnosis in medicine
  - advanced graphics and virtual reality, particularly in the entertainment industry
  - networked video and multi-media technologies

Why Parallel Computing?

- Save time
- Solve larger problems
- Provide parallelism (do multiple things at the same time)
- … ..

Flynn Taxonomy

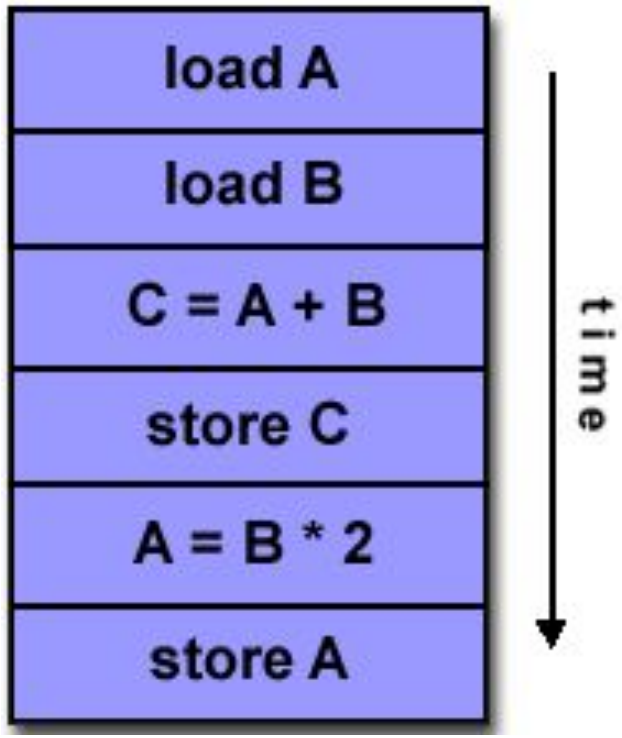
- Based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture

- The four initial classifications defined by Flynn are based upon the number of concurrent instruction (or control) streams and data streams available in the architecture

SISD **Single Instruction, Single Data**	**SIMD** **Single Instruction, Multiple Data**
MISD **Multiple Instruction, Single Data**	**MIMD** **Multiple Instruction, Multiple Data**

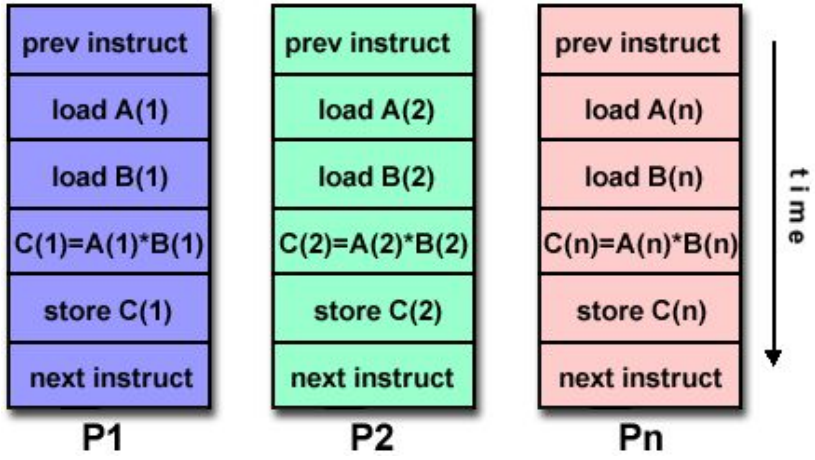
Single Instruction, Single Data (SISD)

- It represents the organization of a single computer containing a control unit, processor unit and a memory unit.
- **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
- A single processor executes a single instruction stream, to operate on data stored in a single memory.
- This is the oldest and until recently, the most prevalent form of computer
  - Examples: most PCs, single CPU workstations and mainframes



Single Instruction, Multiple Data (SIMD)

- 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
- The processing units are made to operate under the control of a **common control unit**, thus providing a single instruction stream and multiple data streams.
  - Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820



50 X 50

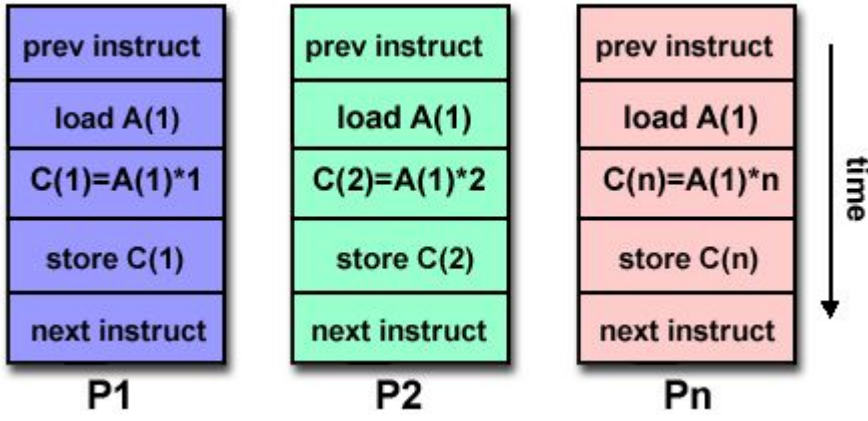
{1,2,3,4,5,6,7,8,9,10}

⇒ TWICE (*2) Every Element (2500)

⇒ MUL 2

Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- It consists of a single computer containing multiple processors connected with multiple control units and a common memory unit
- Each processing unit operates on the data independently via independent instruction streams.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream



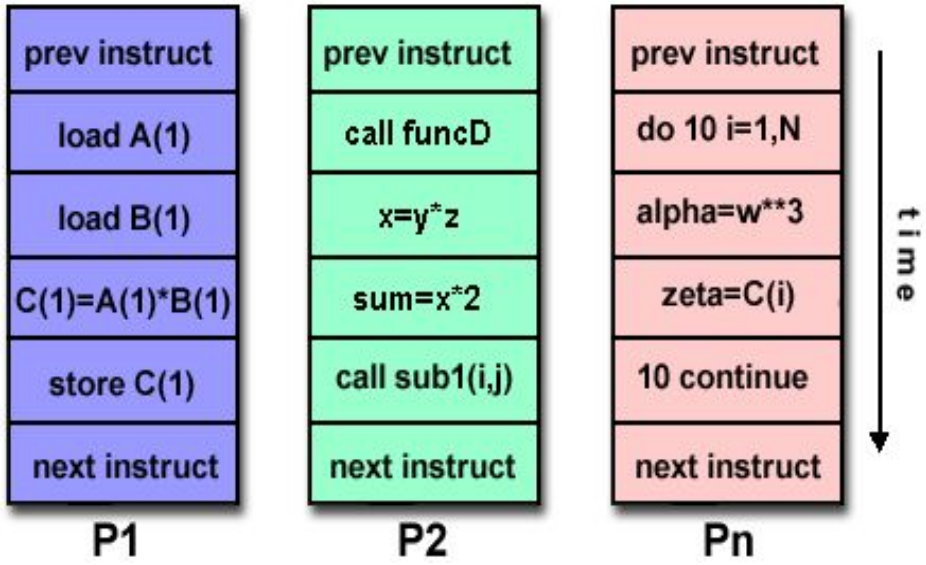
LD $3, ($12)

ADD $4, $12

OR $7, $12

Multiple Instruction, Multiple Data (MIMD)

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- It represents the organization which is capable of processing several programs at same time.
- 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
  - Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



Some General Parallel Terminology

Task/Process

- 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.

Parallel Task

- A task that can be executed by multiple processors safely (yields correct results)

Serial Execution

- Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

Parallel Execution

- Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

Shared Memory

- From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) 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.

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, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

Synchronization

- The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically 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.

Granularity

- In parallel computing, granularity is a 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

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

$$\frac{wallclock\ time\ of\ serial\ execution}{wallclock\ time\ of\ parallel\ execution}$$

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

Parallel Overhead

- The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
  - Task termination time

Massively Parallel

- Refers to the hardware that comprises a given parallel system - having many processors. The meaning of many keeps increasing, but currently BG/L* pushes this number to 6 digits.

*Blue Gene is an IBM project aimed at designing supercomputers that can reach operating speeds in the petaFLOPS (PFLOPS) range, with low power consumption.

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 processors.

- Factors that contribute to scalabilty include:
  - Hardware - particularly memory-cpu bandwidths and network communications
  - Application Algorithm
  - Parallel overhead related
  - Characteristics of your specific application and coding

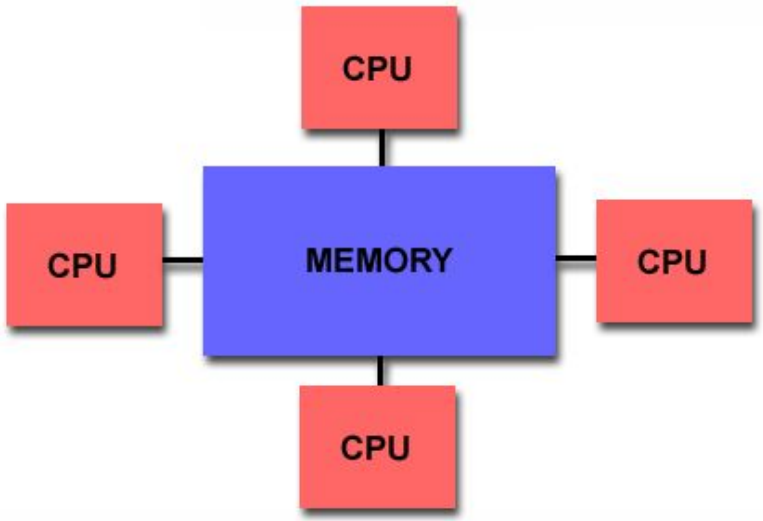
Parallel Computer Memory Architectures

Memory architectures

- Shared Memory
- Distributed Memory
- Hybrid Distributed-Shared Memory

Shared Memory

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.



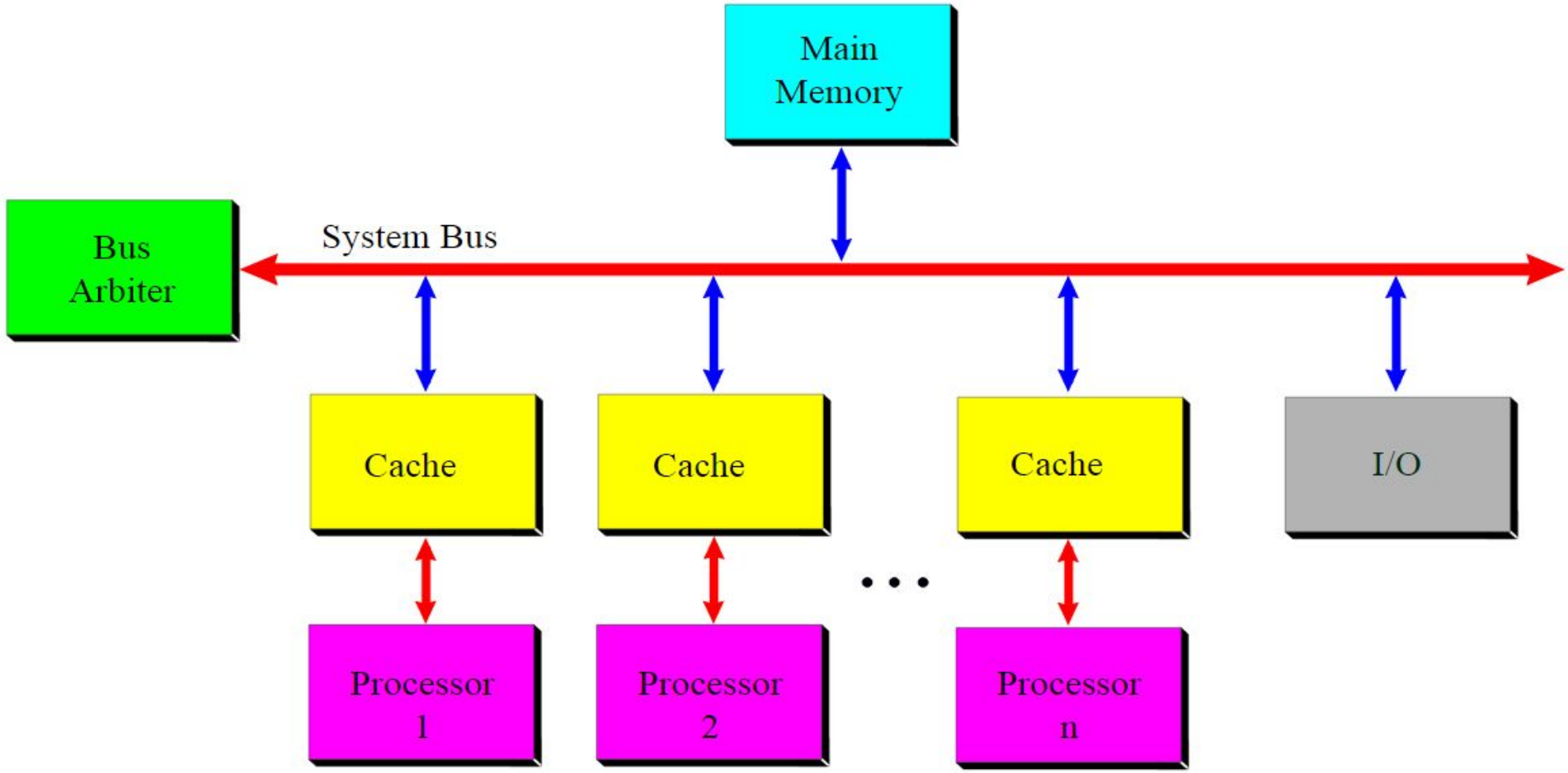
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** and ***NUMA***.

Shared Memory : UMA vs. NUMA

- Uniform Memory Access (UMA):
  - Most commonly represented today by Symmetric Multiprocessor (SMP) machines
  - Identical processors with equal access and access times to memory
  - Sometimes called CC-UMA - Cache Coherent UMA.

- Non-Uniform Memory Access (NUMA):
  - Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - Not all processors have equal access time to all memories

SMP - Symmetric Multiprocessor System

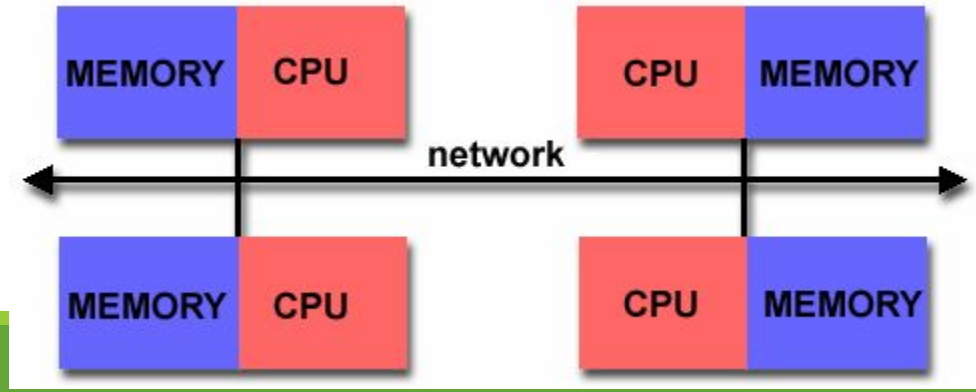


Shared Memory: Pro and Con

- 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 increases 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 insure "correct" access of global memory.
  - Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- 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.
- 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.
- Because each processor has its own local memory, it operates independently. 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.
- 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.
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet.



Distributed Memory: Pro and Con

- Advantages
  - Memory is scalable with 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 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 (NUMA) times

Hybrid Distributed-Shared Memory

Comparison of Shared and Distributed Memory Architectures			
Architecture	CC-UMA	CC-NUMA	Distributed
Examples	SMPs Sun Vexx DEC/Compaq SGI Challenge IBM POWER3	Bull NovaScale SGI Origin Sequent HP Exemplar DEC/Compaq IBM POWER4 (MCM)	Cray T3E Maspar IBM SP2 IBM BlueGene
Communications	MPI Threads OpenMP shmem	MPI Threads OpenMP shmem	MPI
Scalability	to 10s of processors	to 100s of processors	to 1000s of processors
Draw Backs	Memory-CPU bandwidth	Memory-CPU bandwidth Non-uniform access times	System administration Programming is hard to develop and maintain
Software Availability	many 1000s ISVs	many 1000s ISVs	100s ISVs