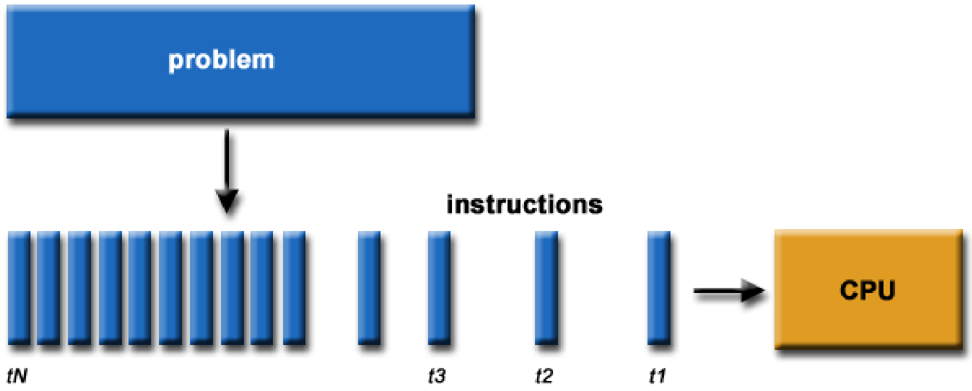
Introduction to Parallel Computing

Module 1 [Abstract Idea]

- This presentation covers the basics of parallel computing. Beginning with a brief overview and some concepts and terminology associated with parallel computing, the topics of parallel memory architectures and programming models are then explored.
- These topics are followed by a discussion on a number of issues related to designing parallel programs.
- The last portion of the presentation is spent examining how to parallelize several different types of serial programs.

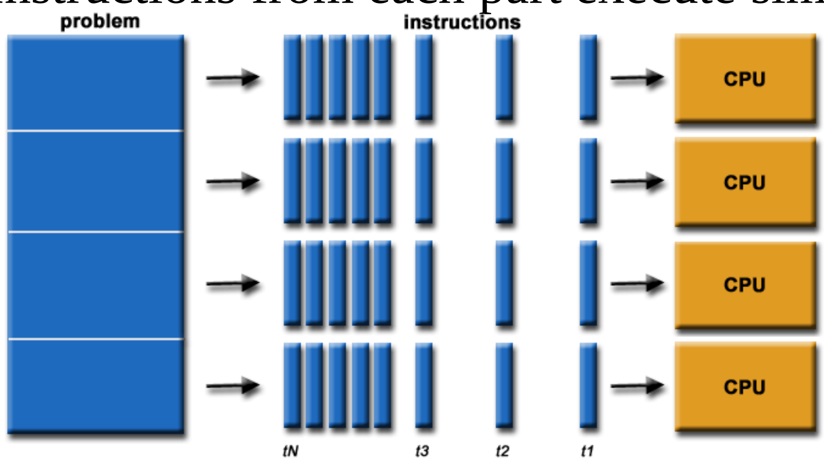
What is Parallel Computing?

- 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.



What is Parallel Computing?

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute 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;
  - 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.

Parallel Computing: What for?

- Parallel computing is an evolution of serial computing that attempts to emulate what has always been the state of affairs in the natural world: many complex, interrelated events happening at the same time, yet within a sequence.
- Some examples:
  - Planetary and galactic orbits; Weather and ocean patterns
  - Tectonic plate drift; Rush hour traffic in Paris
  - Automobile assembly line; Daily operations within a business
  - Building a shopping mall; Ordering a hamburger at the drive through.

Parallel Computing: What for?

- Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:
  - weather and climate; chemical and nuclear reactions
  - biological, human genome; geological, seismic activity
  - mechanical devices - from prosthetics to spacecraft
  - electronic circuits; manufacturing processes

Parallel Computing: What for?

- Today, commercial applications are providing an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. Example applications include:
  - parallel databases, data mining
  - oil exploration
  - web search engines, web based business services
  - computer-aided diagnosis in medicine
  - management of national and multi-national corporations
  - advanced graphics and virtual reality, particularly in the entertainment industry
  - networked video and multi-media technologies
  - collaborative work environments
- Ultimately, parallel computing is an attempt to maximize the infinite but seemingly scarce commodity called time.

Why Parallel Computing?

- This is a legitime question!

  [Parallel computing is complex on any aspect!]

- The primary reasons for using parallel computing:
  - Save time - wall clock time
  - Solve larger problems
  - Provide concurrency (do multiple things at the same time)

Why Parallel Computing?

- Other reasons might include:
  - Taking advantage of non-local resources - using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
  - Cost savings - using multiple "cheap" computing resources instead of paying for time on a supercomputer.
  - Overcoming memory constraints - single computers have very finite memory resources. For large problems, using the memories of multiple computers may overcome this obstacle.

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.
- Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip.

  However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- 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.

The Future Direction

- In the past 10 years, the trends have indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that ***parallelism is the future of computing***.
- It will be multi-forms, mixing general purpose solutions (your PC…) and very speciliazed solutions as IBM Cells, ClearSpeed, GPGPU from NVidia …

Who and What?

- Top500.org provides statistics on parallel computing users - the charts below are just a sample. Some things to note:
  - Sectors may overlap - for example, research may be classified research. Respondents have to choose between the two.
- "Not Specified" is by far the largest application - probably means multiple applications.

Concepts and Terminology

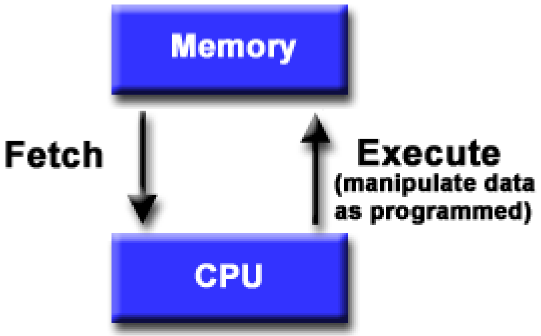
Von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. It is named after the Hungarian mathematician John von Neumann.

- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

Basic Design

- Basic design
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then ***sequentially*** performs them.



Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. 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*** and ***Data***.
- Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.

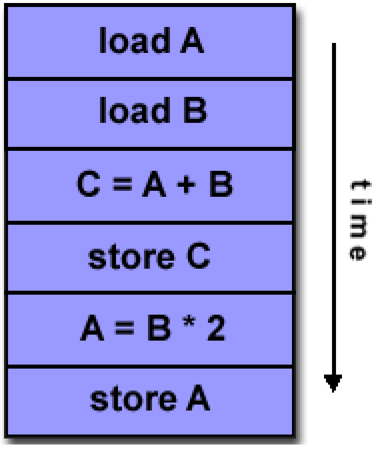
Flynn Matrix

- The matrix below defines the 4 possible classifications according to Flynn

S I S D **Single Instruction, Single Data**	**S I M D** **Single Instruction, Multiple Data**
M I S D **Multiple Instruction, Single Data**	**M I M D** **Multiple Instruction, Multiple Data**

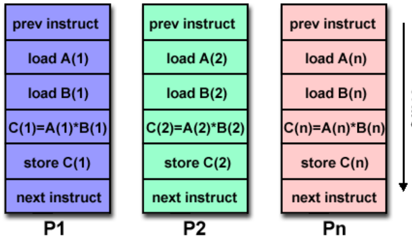
Single Instruction, Single Data (SISD)

- 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 and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



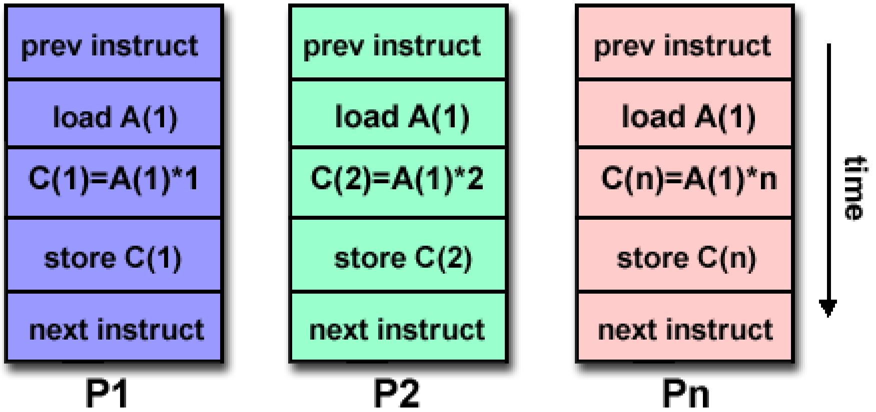
Single Instruction, Multiple Data (SIMD)

- 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
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity,such as image processing.
- Synchronous (lockstep) and deterministic execution
- 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



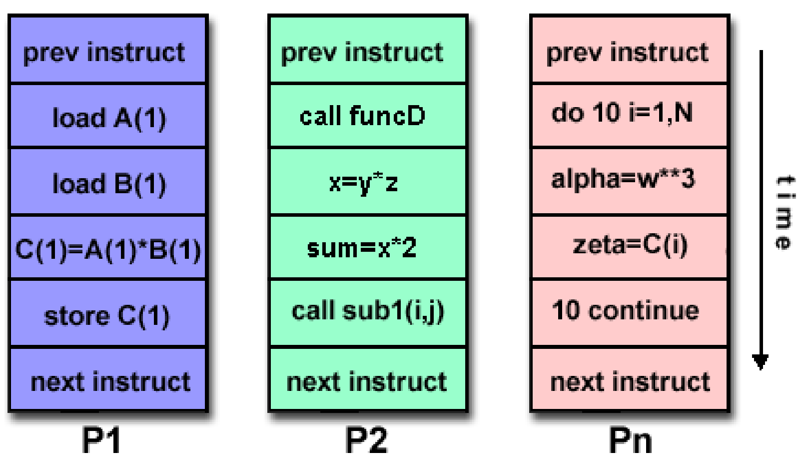
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- 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)

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



Parallel Terminologies

Like everything else, parallel computing has its own "jargon". Some of the more commonly used terms associated with parallel computing are listed below. Most of these will be discussed in more detail later.

- **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.
- **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.

Parallel Terminologies

- **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.

Parallel Terminologies

- **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.

Parallel Terminologies

- **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:

$$\frac{Wall\ Clock\ of\ Serial\ Execution}{Wall\ Clock\ Time\ of\ Parallel\ Execution}$$

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

Parallel Terminologies

- **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.

Parallel Terminologies

- **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 scalability 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