Parallel Hardware

Parallel Computing – COSC 3P93

Course Outline

- Introduction
- Parallel Hardware
- Parallel Software
- Performance
- Parallel Programming Models
- Patterns
- SMP with Threads and OpenMP
- Distributed-Memory Programming with MPI
- Algorithms
- Tools
- Parallel Program Design

Today Class

- Parallel Hardware
  - Flynn's Taxonomy
    - SIMD systems
    - MIMD systems

Parallel Hardware

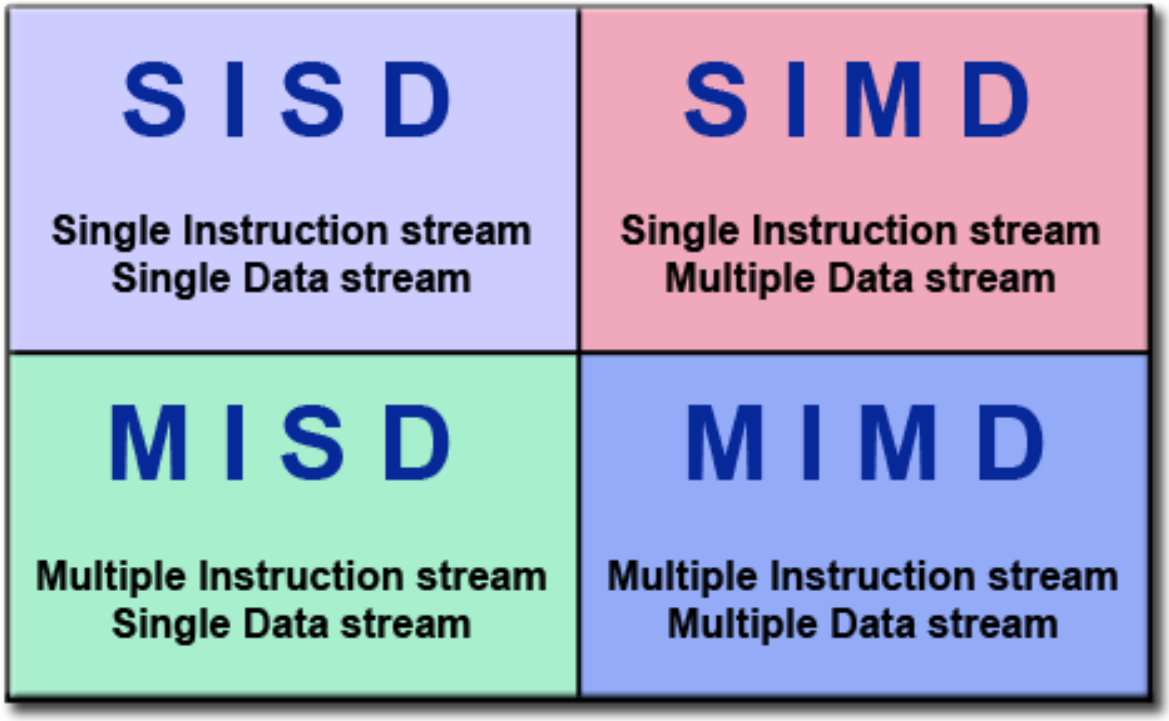
- Multiple issue and pipelining
  - Replication of functional units
  - Both
    - Extensions of the von Neumann model
    - Visible to programmers
      - Aware of them for better program design

Flynn's Classical Taxonomy

- There are different ways to classify parallel computers
- Flynn's Taxonomy → 1966
  - One of the more widely used classifications
- It 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
- Classification of computer architectures
  - Simultaneous coordination
    - # of instruction streams vs # of data streams

Flynn's Classical Taxonomy

- A matrix defining the 4 possible classifications according to Flynn



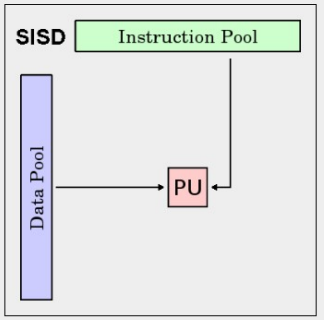
Single Instruction, Single Data (SISD)

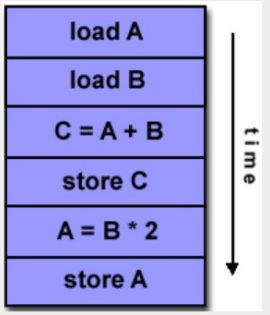
- A serial (non-parallel) computer
- **Single Instruction**
  - Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data**
  - Only one data stream is being used as input during any one clock cycle
- Deterministic execution

Single Instruction, Single Data (SISD)

- This is the **oldest** type of computer
  - **von Neumann system**
    - **Uniprogramming** model
      - Single instruction at a time
      - Fetching and storing one data item at a time
- Examples
  - Older generation mainframes
  - Minicomputers
  - Workstations
  - Single processor/core PCs

Single Instruction, Single Data (SISD)

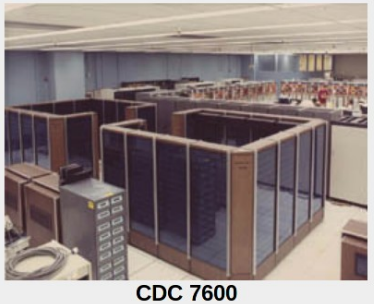





UNIVAC1


IBM 360


CRAY1


CDC 7600


PDP1


Dell Laptop

Single Instruction, Multiple Data (SIMD)

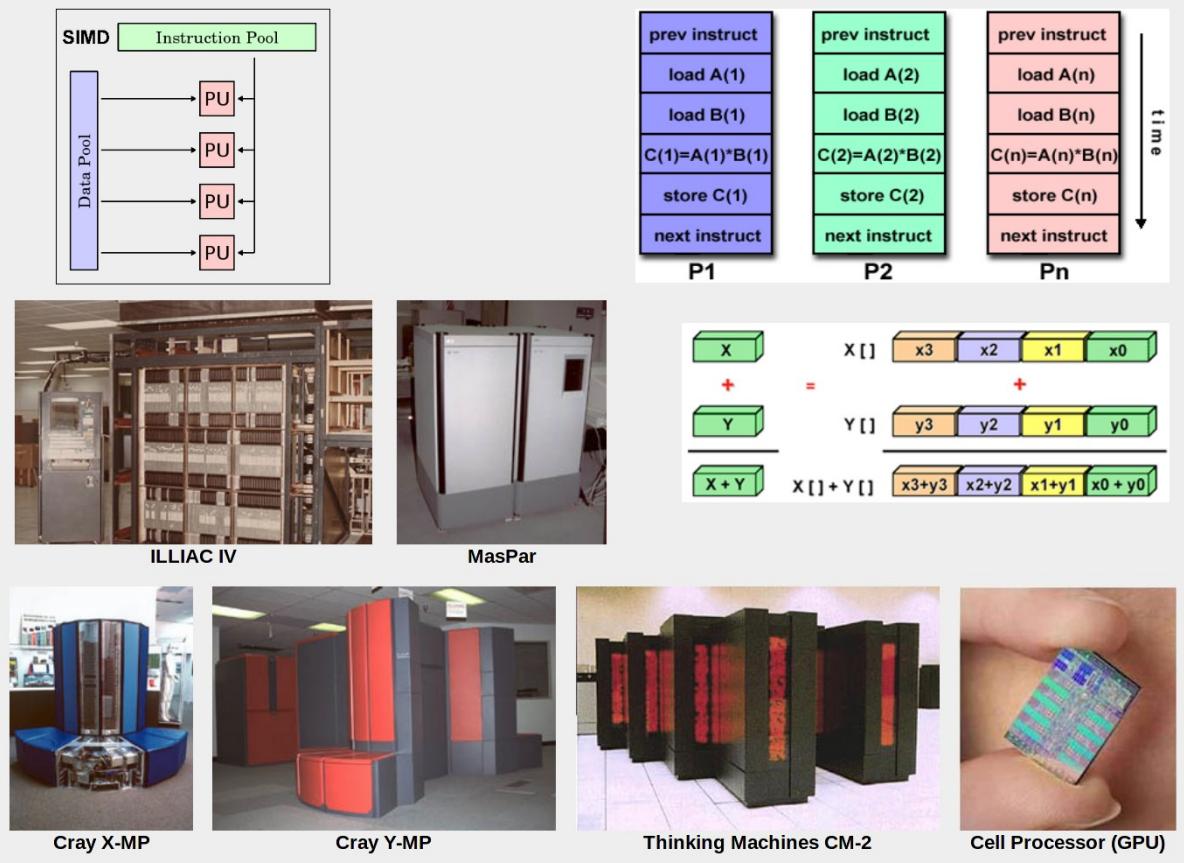
- A type of parallel computer
- **Single Instruction**
  - All processing units execute the same instruction at any given clock cycle
- **Multiple Data**
  - Each processing unit can operate on a different data element
- Same instruction to multiple data items
  - Abstraction
    - System having single control unit and multiple ALUs
- Best suited for specialized problems characterized by a high degree of regularity
  - Graphics/image processing

Single Instruction, Multiple Data (SIMD)

- **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

Single Instruction, Multiple Data (SIMD)

- Most modern computers
  - Particularly those with graphics processor units (GPUs) employ **SIMD** instructions and execution units.



SIMD Example

- Arithmetic operations on two arrays of n elements
  - **for (i = 0; i < n ; i ++)**
    - **x [i] += y [i];**
- SIMD system has n ALUs
  - x[i] and y[i] in ith ALU → storing result in x[i]
- *m* ALUs, where m < n
  - Additions in blocks of m elements at a time
    - Ex → m= 4 and n = 15
- Idling ALUs degrades peformance
  - **for (i = 0; i < n ; i ++)**
    - **if (y [i] > 0.0) x [i] += y [i];**
  - Each element of y must be loaded in an ALU for testing before proceeding with the additions

Classical SIMD Systems

- Each ALU operates synchronously
  - Each unit must wait for the next instruction to be broadcast before proceeding
- ALUs have not instruction storage
  - Instruction cannot be stored for later execution
- SIMD systems → ideal for parallelizing simple loops
  - Large data arrays → split among many processors
    - Data-parallelism
  - Very efficient with large data parallel problems
  - Not very efficient with other parallel problems
- In early 1990s → **vector processors**
- Nowadays → GPUs

Vector Processors

- Vectors of data
  - CPUs operating on individual data elements (**scalars**)
- Characteristics
  - **Vector registers**
    - Storing a vector of operands → simultaneous operations
    - Fixed length → 4-128 64-bit elements
  - **Vectorized** and **pipelined functional units**
    - The same over data → vector oprations are **SIMD**
  - **Vector instructions**
    - Instead of scalars → operations on vectors (one single load)
  - **Interleaved memory**
    - Little to no delay in loading/storing successive elements stored over distributed data banks
  - **Strided memory access** and **hardware scatter/gather**
    - Access in fixed intervals (strides) → Ex: 1^{st}, 5^{th}, 9^{th} ... elements
    - Writing (scatter) or reading (gather) in irregular intervals
    - For both → special hardware to accelerate these operations

Vector Processors

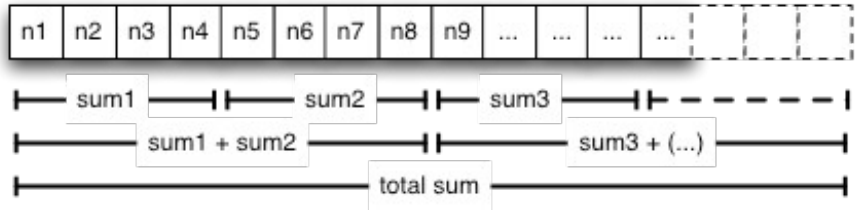
- Virtue
  - Very fast and easy to use for many applications
- Vectorizing compilers
  - Efficient in identifying code for parallelization
    - Determining loops that cannot be parallelized
- Vector systems → high memory bandwidth
  - Every loaded data item is used → not so much caching
  - No handling of irregular data structures
    - Limiting factor for their **scalability** → handling larger problems
    - Limits of operations over longer vectors
      - Commodity systems → very short vectors
      - Custom manufactured → long vectors

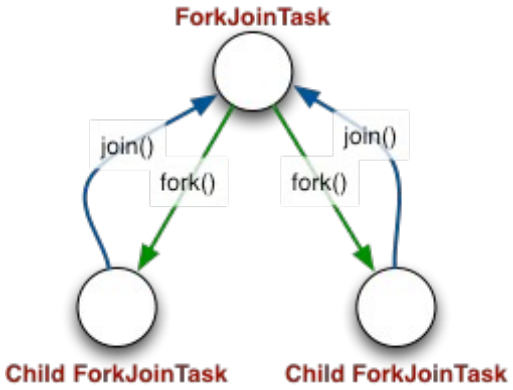
Graphics Processing Units

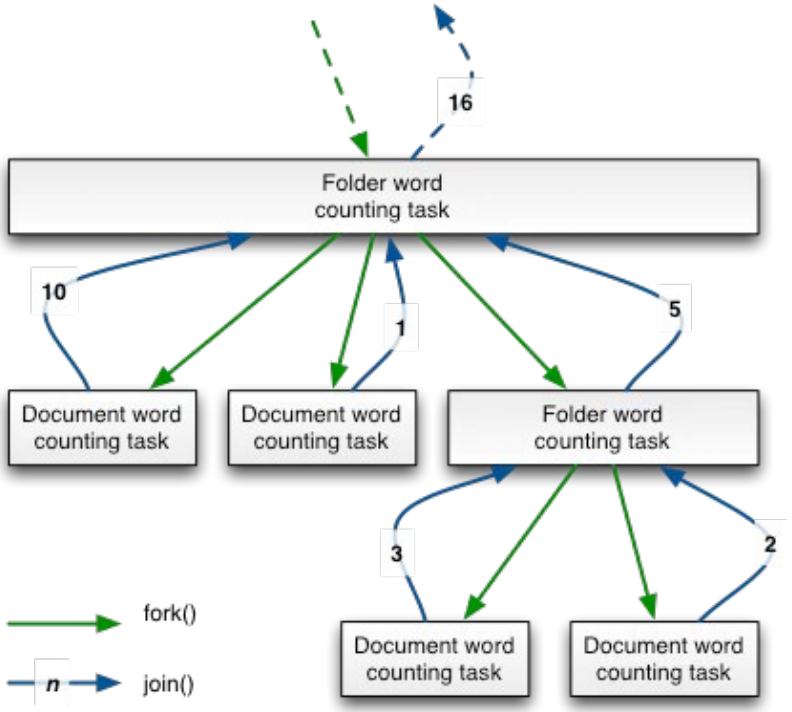
- Real-time graphics application programming interfaces
- Graphics processing pipeline
  - Conversion of internal representation into array of pixels
  - Some stages of this pipeline are programmable
    - Behaviour defined by functions
      - Shader functions → short (a few lines of C code)
        - Implicitly parallel
- GPU → **SIMD parallelism**
  - Large number of ALUs in each GPU core
- **Very high rates of data movement**
  - Large amounts of data (images)
  - To avoid stalls on memory access
- **Heavily relying on hardware multithreading**
  - # of threads relates to registers needed by the shader func.
  - GPU → relatively poor performance on small problems
- **GPU not pure SIMD** → multiple cores
  - So independent instruction streams

SIMD Example Code - Java

- Searching words in file tree







https://www.oracle.com/technetwork/articles/java/fork-join-422606.html

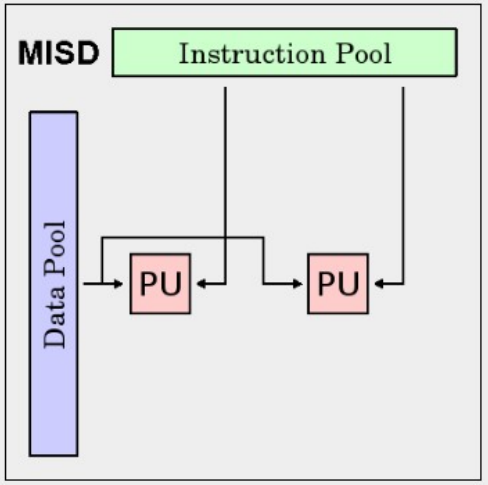
SIMD Example Code - C++

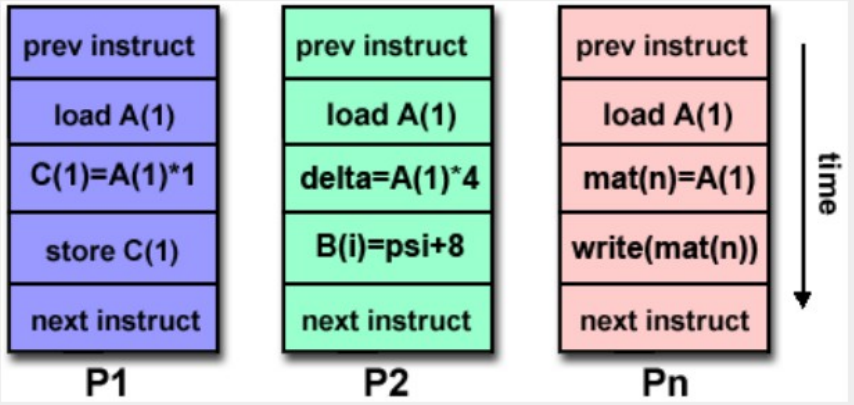
- C/C++ → pthreads (old)
- C++11 / C++14
-
-
-
-
-
-
-
-
-
-
- image_processing

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 Instruction, Single Data (MISD)





MISD Example

- Cracking an encrypted code
  - Multiple cryptography algorithms attempting to crack a single coded message
- Running multiple different instances of code over the same data input
  - Testing and verification
  - Experimental analysis → statistical validation

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

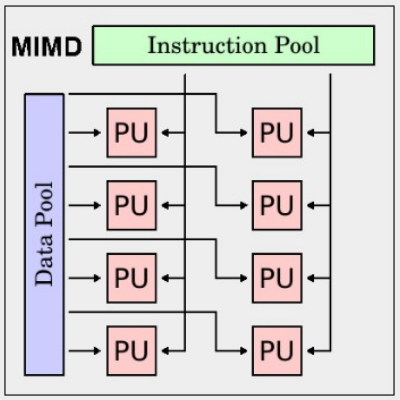
MIMD Systems

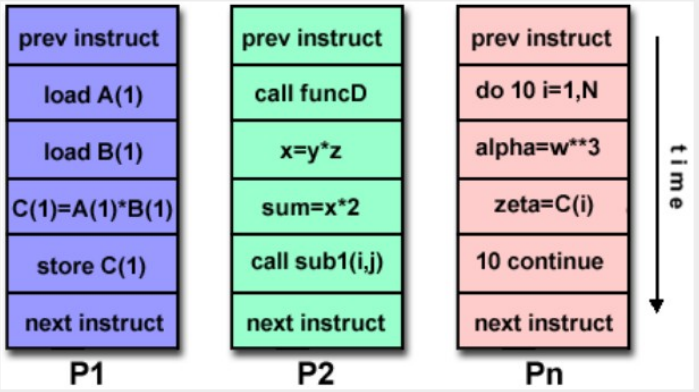
- A collection of fully **independent processing units** or cores
  - Each unit with its own control unit and ALU
- MIMD systems are usually **asynchronous**
  - Each processor operating at its own pace
- **There is no global clock**
  - No relation between system times on two different processors
    - Unless the programmer imposes certain synchronization

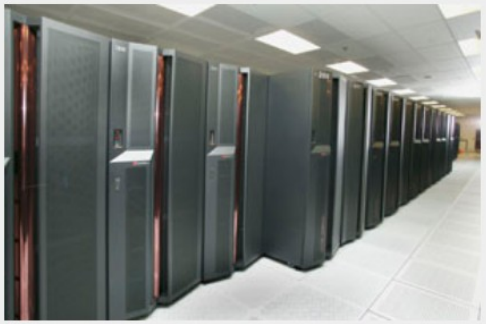
Multiple Instruction, Multiple Data (MIMD)

- 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
- **Note**
  - Many MIMD architectures also include SIMD execution sub-components

Multiple Instruction, Multiple Data (MIMD)







IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

IBM BG/L

Class Recap

- Parallel Hardware
  - Flynn's Taxonomy
    - SIMD systems
    - MIMD systems

Next Class

- Parallel Hardware
  - Interconnection networks
  - Cache coherence
  - Shared-memory versus distributed-memory