Parallel Programming Systems and Models

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

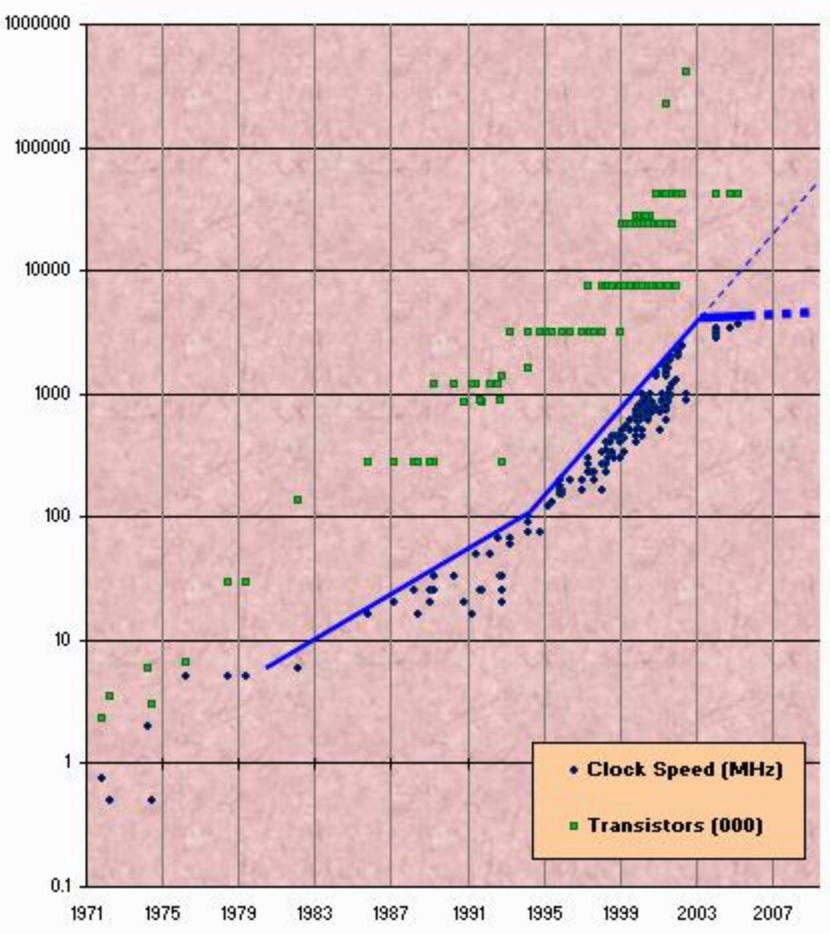
CS 595: Data-Intensive Computing
September 26th, 2011

Introduction to Parallel Computing

- Moore's Law
  - The number of transistors that can be placed inexpensively on an integrated circuit will double approximately every 18 months.
  - Self-fulfilling prophecy
    - Computer architect goal
    - Software developer assumption

Introduction to Parallel Computing

- Impediments to Moore's Law
  - Theoretical Limit
  - What to do with all that die
  - Design complexity
  - How do you meet the exp increase?

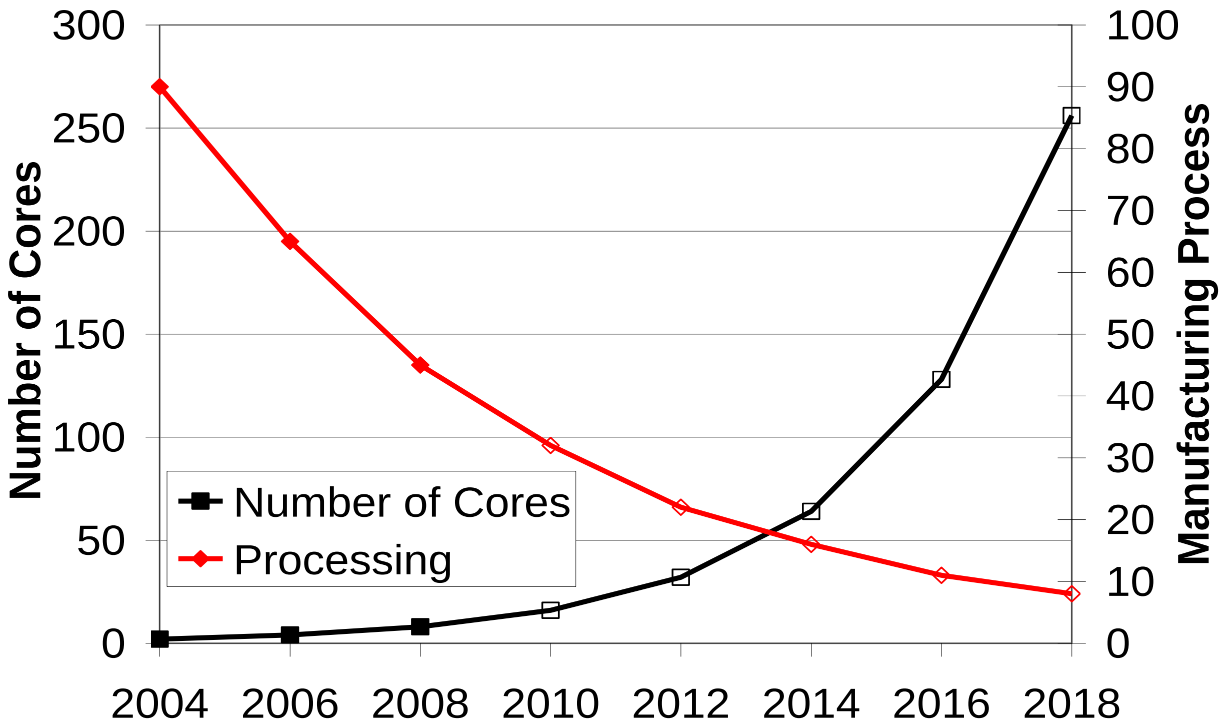


Introduction to Parallel Computing

- von Neumann model
  - Execute a stream of instructions (machine code)
  - Instructions can specify
    - Arithmetic operations
    - Data addresses
    - Next instruction to execute
  - Complexity
    - Track billions of data locations and millions of instructions
    - Manage with:
      - Modular design
      - High-level programming languages

Introduction to Parallel Computing

- Parallelism
  - Continue to increase performance via parallelism.

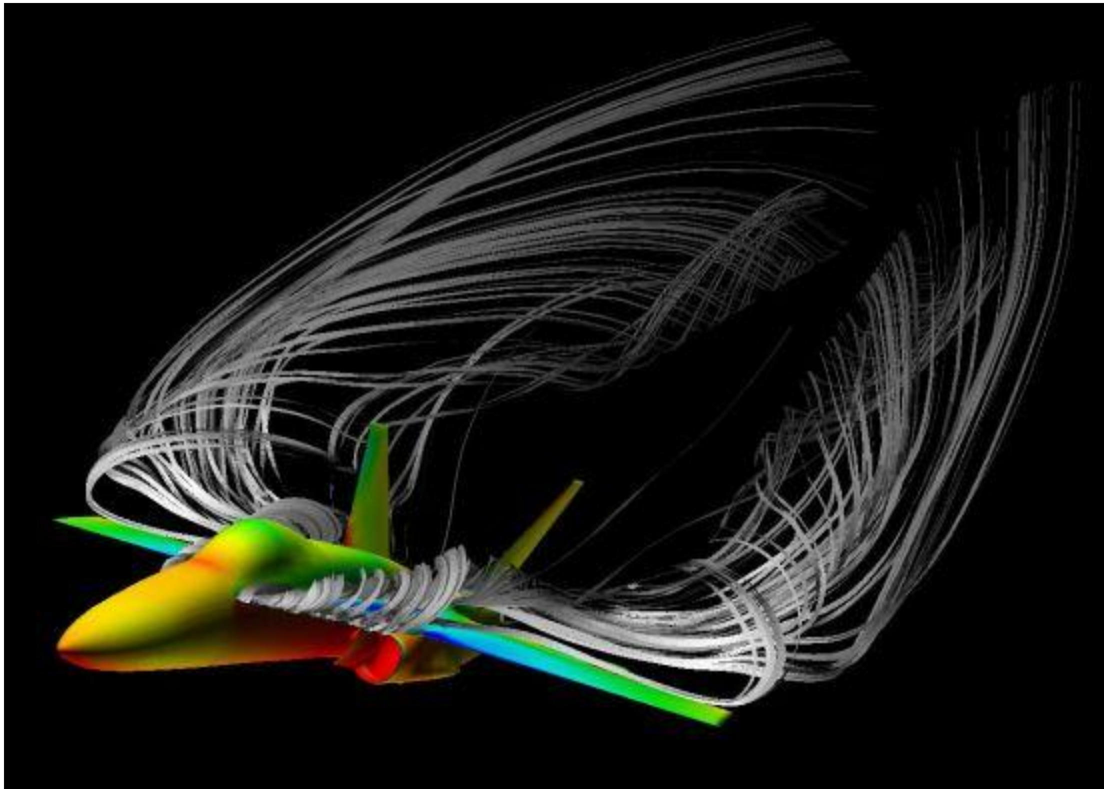


Introduction to Parallel Computing

- From a software point-of-view, need to solve demanding problems
  - Engineering Simulations
  - Scientific Applications
  - Commercial Applications
- Need the performance, resource gains afforded by parallelism

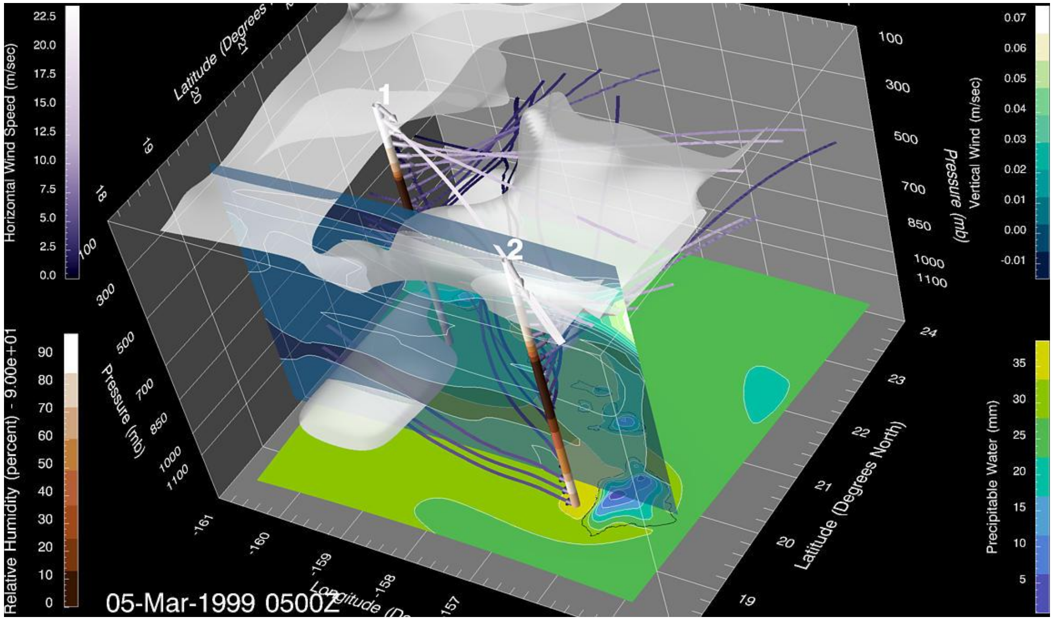
Introduction to Parallel Computing

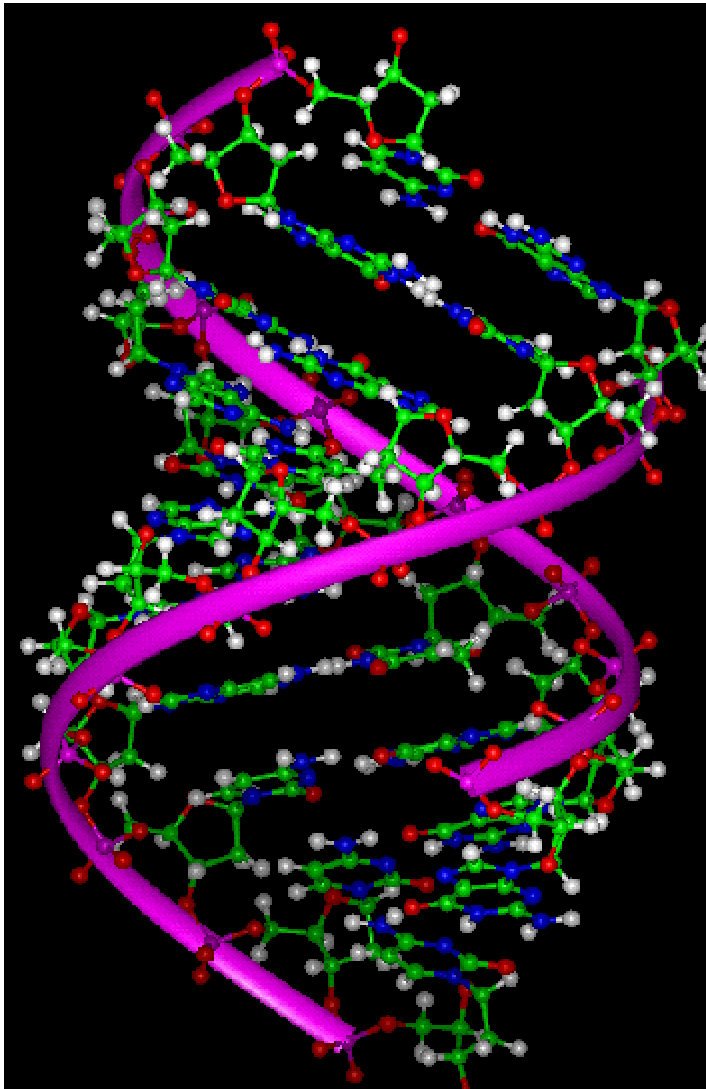
- Engineering Simulations
  - Aerodynamics
  - Engine efficiency



Introduction to Parallel Computing

- Scientific Applications
  - Bioinformatics
  - Thermonuclear processes
  - Weather modeling





Introduction to Parallel Computing

- Commercial Applications
  - Financial transaction processing
  - Data mining
  - Web Indexing

Introduction to Parallel Computing

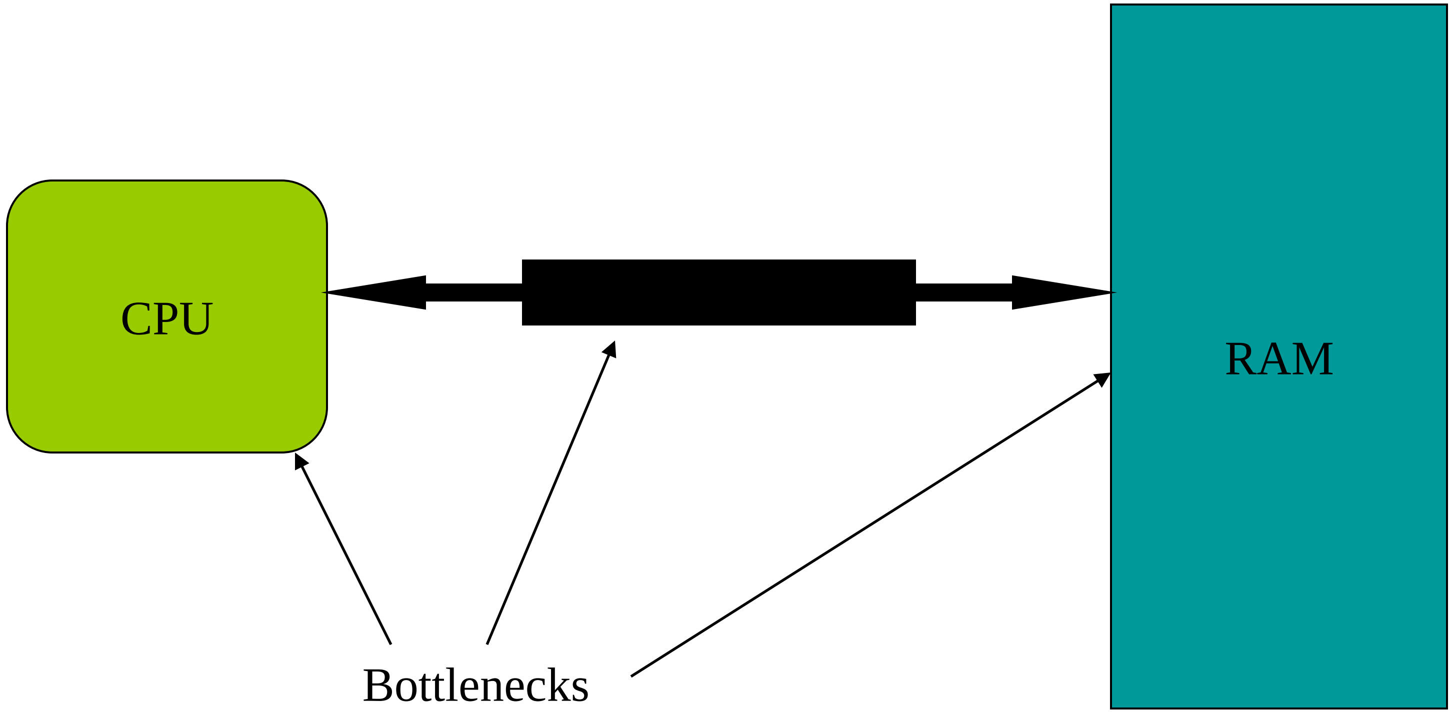
- Unfortunately, greatly increases coding complexity
  - Coordinating concurrent tasks
  - Parallelizing algorithms
  - Lack of standard environments and support

Introduction to Parallel Computing

- The challenge
  - Provide the abstractions, programming paradigms, and algorithms needed to effectively design, implement, and maintain applications that exploit the parallelism provided by the underlying hardware in order to solve modern problems.

Parallel Architectures

- Standard sequential architecture

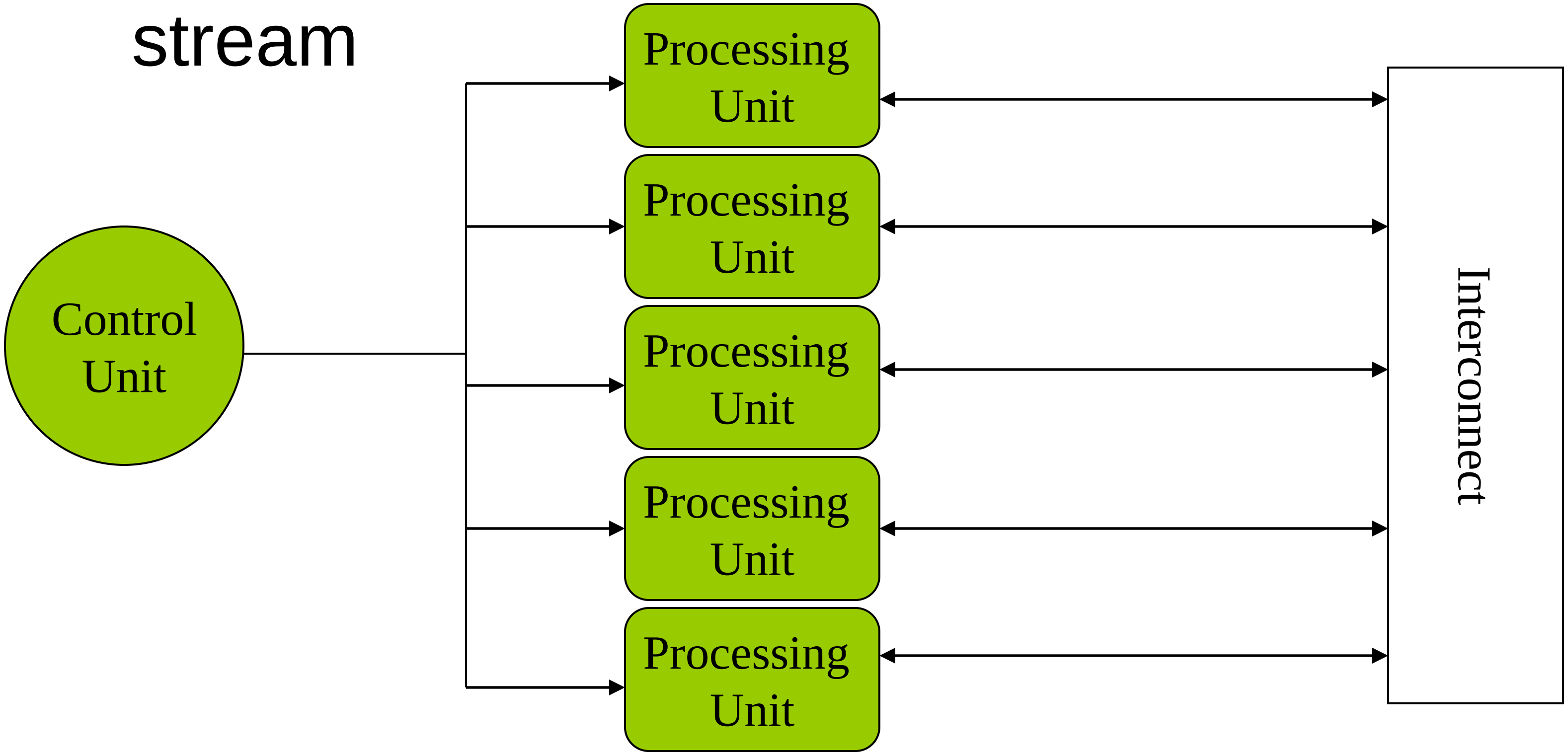


Parallel Architectures

- Use multiple
  - Datapaths
  - Memory units
  - Processing units

Parallel Architectures

- SIMD
  - Single instruction stream, multiple data stream

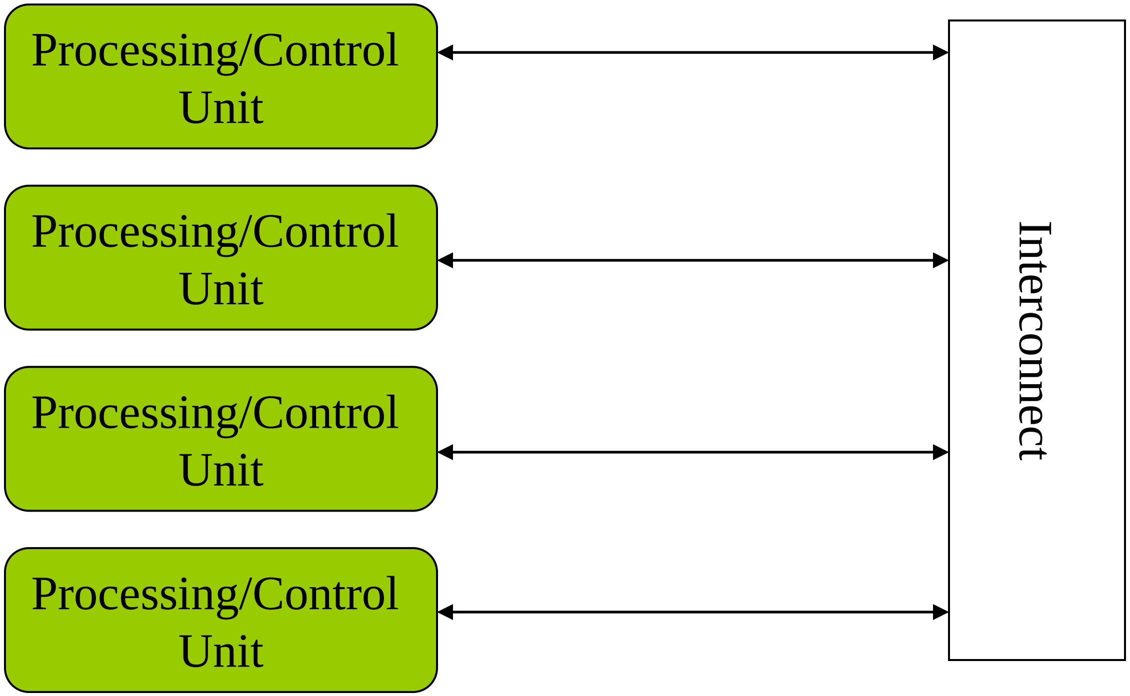


Parallel Architectures

- SIMD
  - Advantages
    - Performs vector/matrix operations well
      - EX: Intel’s MMX chip
  - Disadvantages
    - Too dependent on type of computation
      - EX: Graphics
    - Performance/resource utilization suffers if computations aren’t “embarrasingly parallel”.

Parallel Architectures

- MIMD
  - Multiple instruction stream, multiple data stream



Parallel Architectures

- MIMD
  - Advantages
    - Can be built with off-the-shelf components
    - Better suited to irregular data access patterns
  - Disadvantages
    - Requires more hardware (!sharing control unit)
    - Store program/OS at each processor

- Ex: Typical commodity SMP machines we see today.

Parallel Architectures

- Task Communication
  - Shared address space
    - Use common memory to exchange data
    - Communication and replication are implicit
  - Message passing
    - Use send()/receive() primitives to exchange data
    - Communication and replication are explicit

Parallel Architectures

- Shared address space
  - Uniform memory access (UMA)
    - Access to a memory location is independent of which processing unit makes the request.
  - Non-uniform memory access (NUMA)
    - Access to a memory location depends on the location of the processing unit relative to the memory accessed.

Parallel Architectures

- Message passing
  - Each processing unit has its own private memory
  - Exchange of messages used to pass data
  - APIs
    - Message Passing Interface (MPI)
    - Parallel Virtual Machine (PVM)

Parallel Algorithms

- Algorithm
  - a sequence of finite instructions, often used for calculation and data processing.
- Parallel Algorithm
  - An algorithm that which can be executed a piece at a time on many different processing devices, and then put back together again at the end to get the correct result

Parallel Algorithms

- Challenges
  - Identifying work that can be done concurrently.
  - Mapping work to processing units.
  - Distributing the work
  - Managing access to shared data
  - Synchronizing various stages of execution.

Parallel Algorithms

- Models
  - A way to structure a parallel algorithm by selecting decomposition and mapping techniques in a manner to minimize interactions.

Parallel Algorithms

- Models
  - Data-parallel
  - Task graph
  - Work pool
  - Master-slave
  - Pipeline
  - Hybrid

Parallel Algorithms

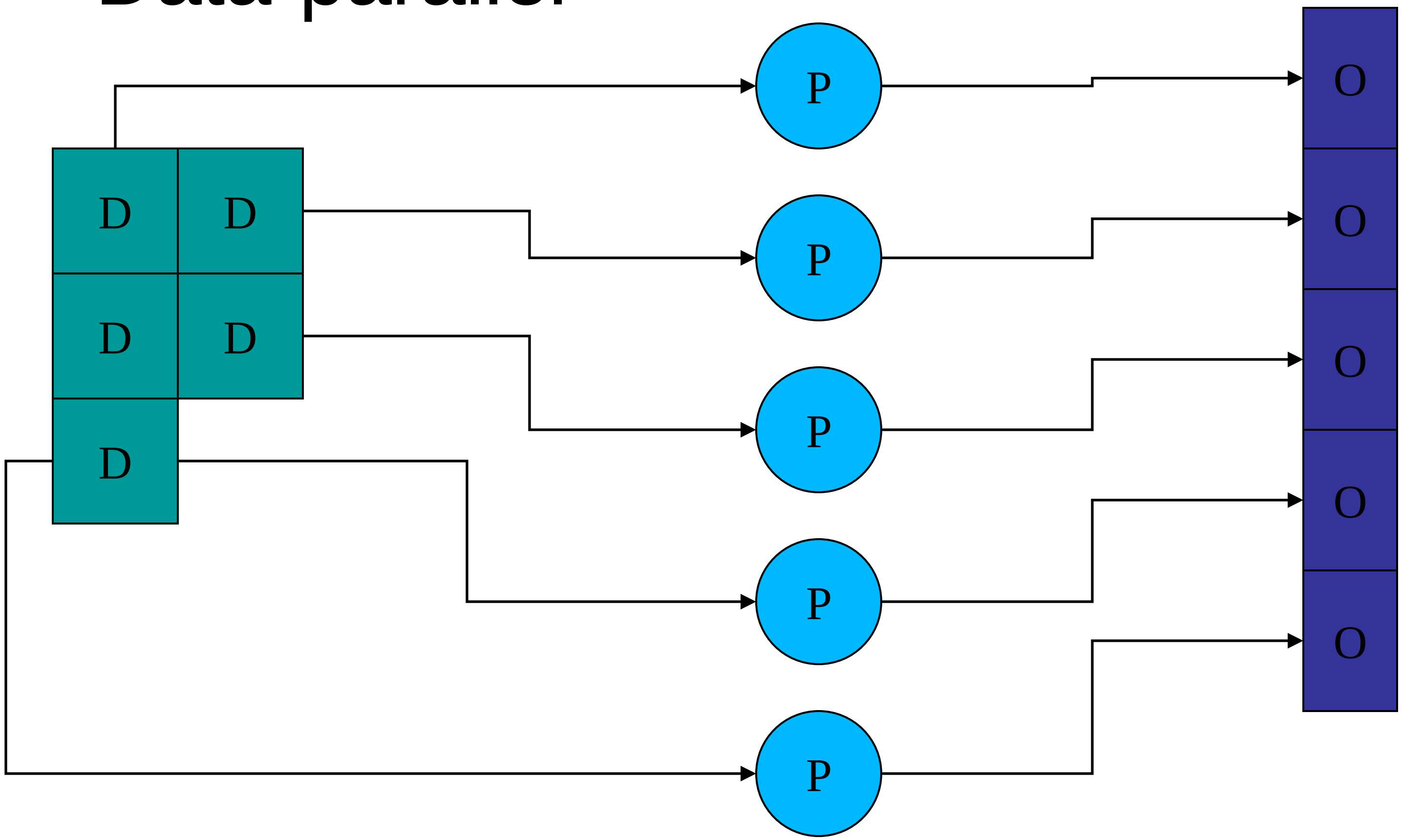
- Data-parallel
  - Mapping of Work
    - Static
    - Tasks -> Processes
  - Mapping of Data
    - Independent data items assigned to processes (Data Parallelism)

Parallel Algorithms

- Data-parallel
  - Computation
    - Tasks process data, synchronize to get new data or exchange results, continue until all data processed
  - Load Balancing
    - Uniform partitioning of data
  - Synchronization
    - Minimal or barrier needed at end of a phase
  - Examples
    - Ray Tracing

Parallel Algorithms

- Data-parallel



Parallel Algorithms

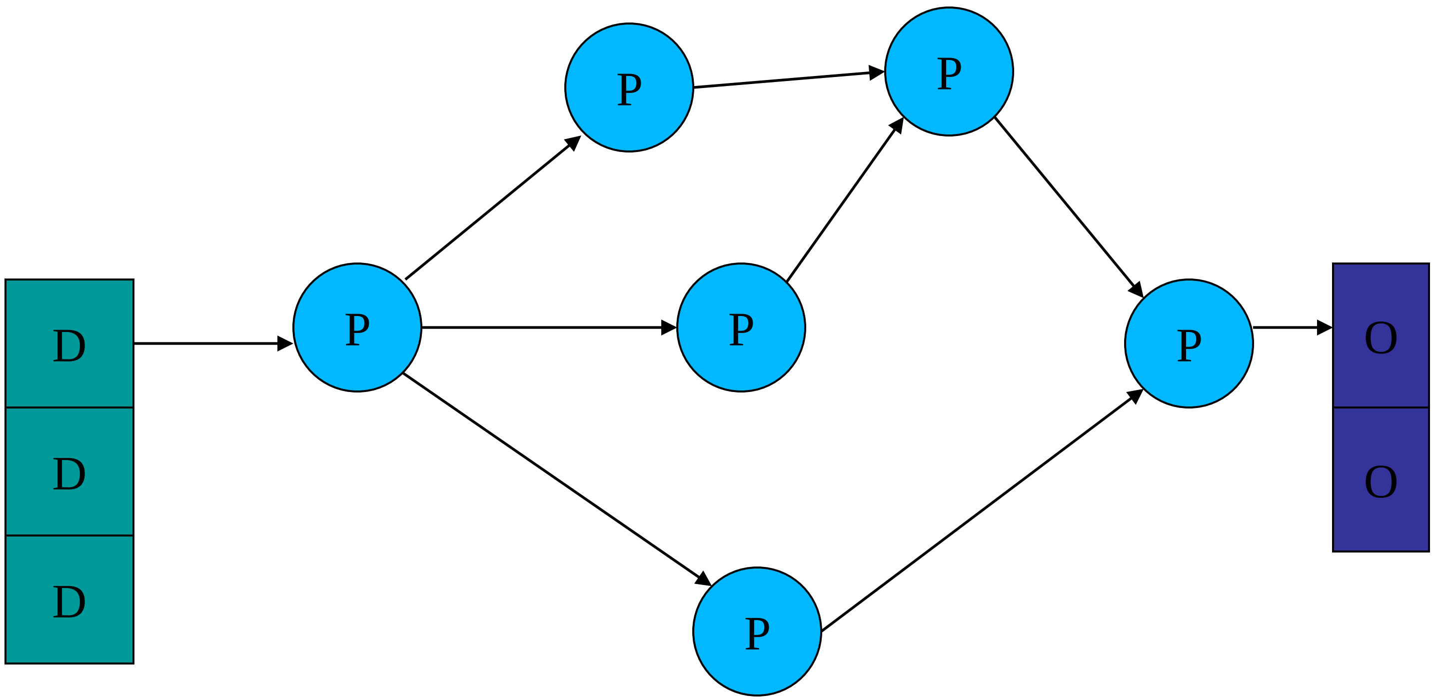
- Task graph
  - Mapping of Work
    - Static
    - Tasks are mapped to nodes in a data dependency task dependency graph (Task parallelism)
  - Mapping of Data
    - Data moves through graph (Source to Sink)

Parallel Algorithms

- Task graph
  - Computation
    - Each node processes input from previous node(s) and send output to next node(s) in the graph
  - Load Balancing
    - Assign more processes to a given task
    - Eliminate graph bottlenecks
  - Synchronization
    - Node data exchange
  - Examples
    - Parallel Quicksort, Divide and Conquer approaches
    - Scientific Applications that can be expressed in workflows (e.g. DAGs)

Parallel Algorithms

- Task graph



Parallel Algorithms

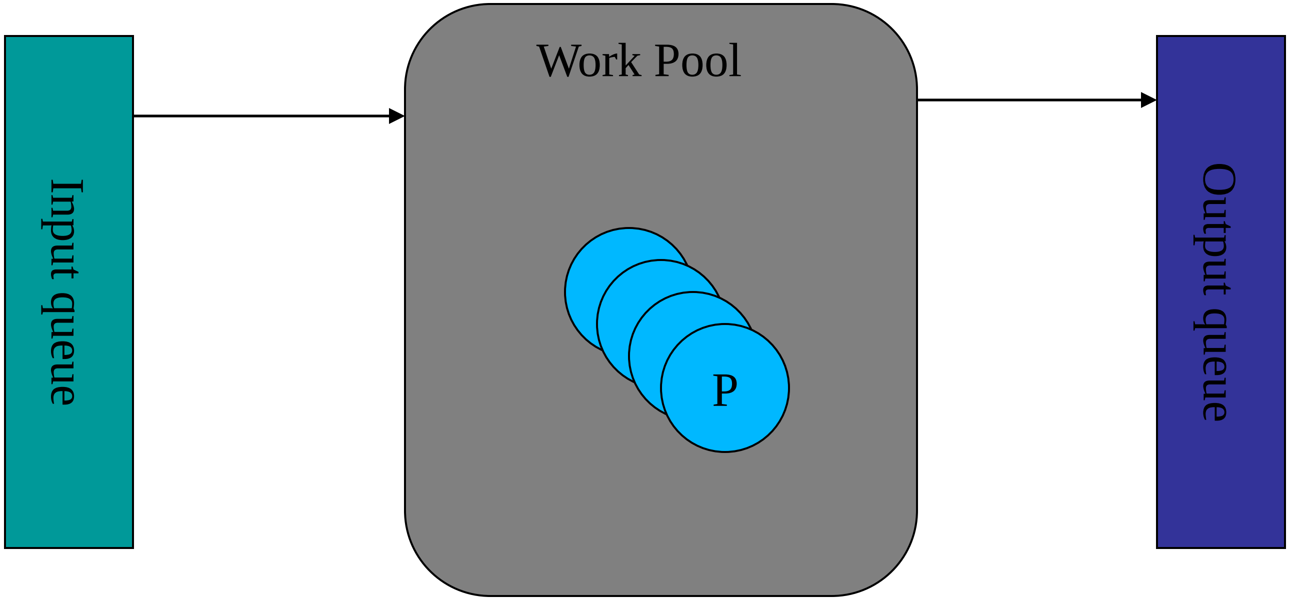
- Work pool
  - Mapping of Work/Data
    - No desired pre-mapping
    - Any task performed by any process
    - Pull-model oriented
  - Computation
    - Processes work as data becomes available (or requests arrive)

Parallel Algorithms

- Work pool
  - Load Balancing
    - Dynamic mapping of tasks to processes
  - Synchronization
    - Adding/removing work from input queue
  - Examples
    - Web Server
    - Bag-of-tasks

Parallel Algorithms

- Work pool



Parallel Algorithms

- Master-slave
  - Modification to Worker Pool Model
    - One or more Master processes generate and assign work to worker processes\
    - Push-model oriented
  - Load Balancing
    - A Master process can better distribute load to worker processes

Parallel Algorithms

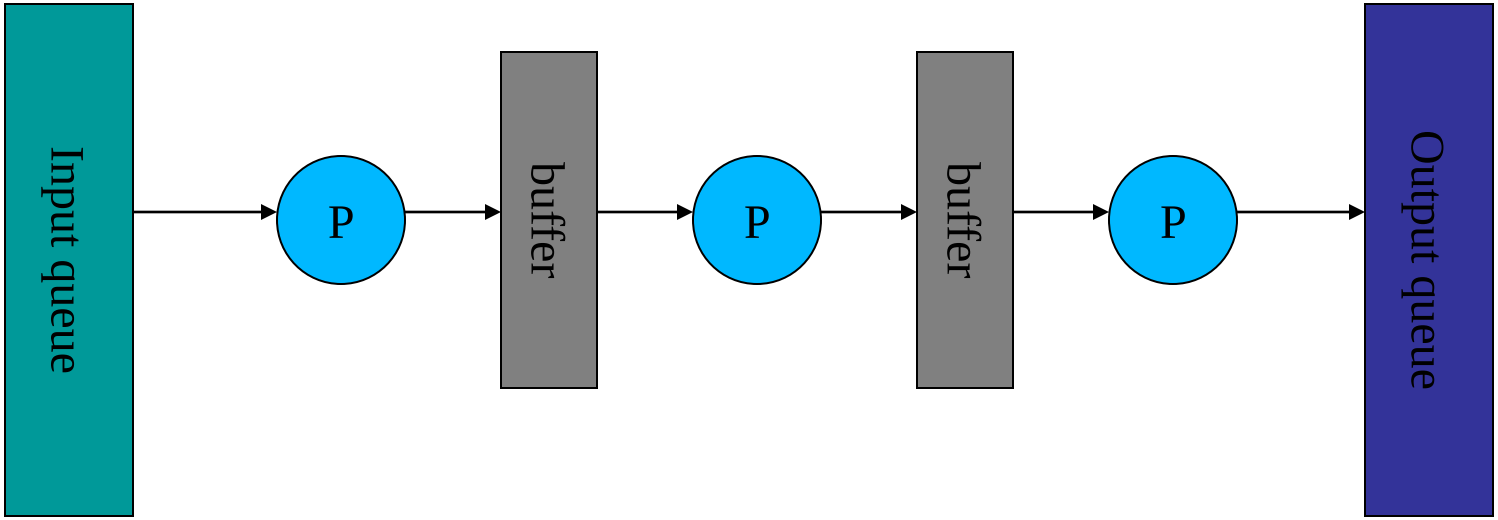
- Pipeline
  - Mapping of work
    - Processes are assigned tasks that correspond to stages in the pipeline
    - Static
  - Mapping of Data
    - Data processed in FIFO order
      - Stream parallelism

Parallel Algorithms

- Pipeline
  - Computation
    - Data is passed through a succession of processes, each of which will perform some task on it
  - Load Balancing
    - Insure all stages of the pipeline are balanced (contain the same amount of work)
  - Synchronization
    - Producer/Consumer buffers between stages
  - Ex: Processor pipeline, graphics pipeline

Parallel Algorithms

- Pipeline



Common Parallel Programming Models

- Message-Passing
- Shared Address Space

Common Parallel Programming Models

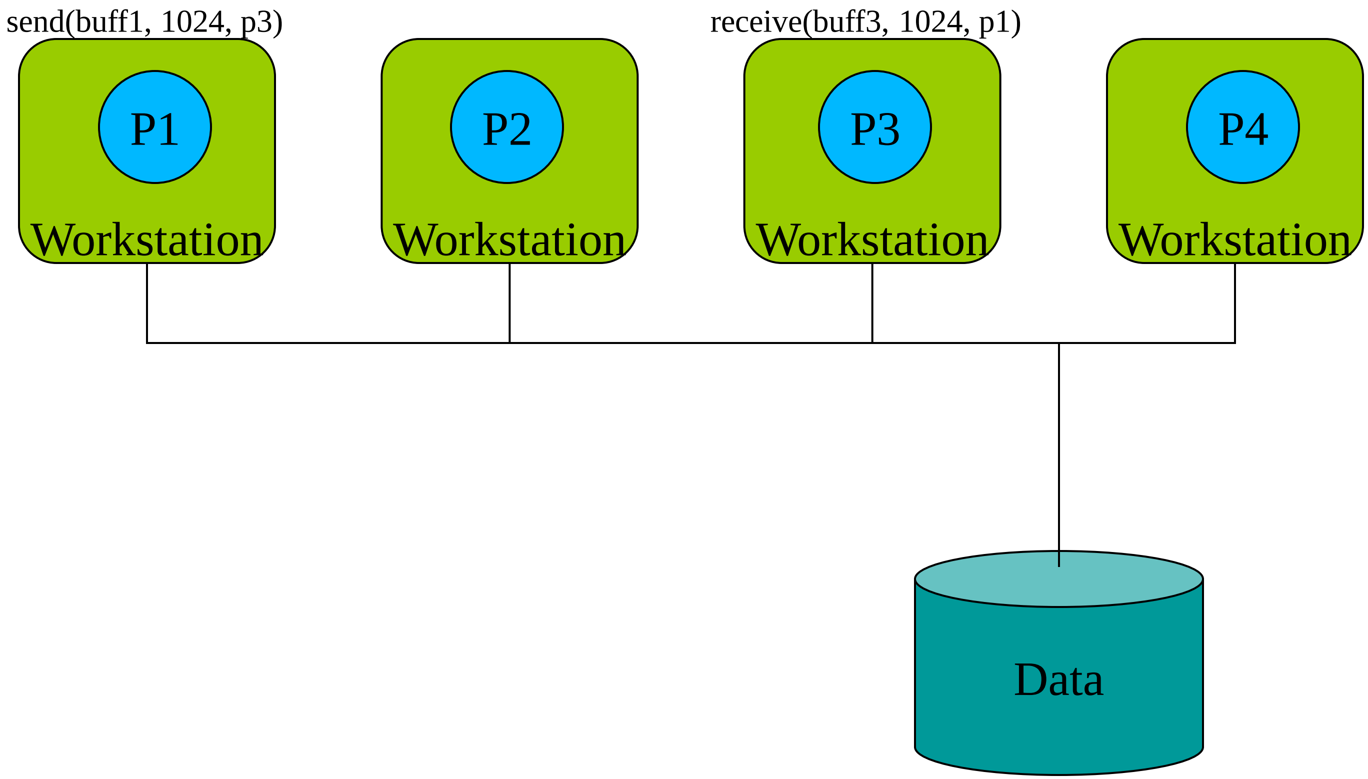
- Message-Passing
  - Most widely used for programming parallel computers (clusters of workstations)
  - Key attributes:
    - Partitioned address space
    - Explicit parallelization
  - Process interactions
    - Send and receive data

Common Parallel Programming Models

- Message-Passing
  - Communications
    - Sending and receiving messages
    - Primitives
      - send(buff, size, destination)
      - receive(buff, size, source)
      - Blocking vs non-blocking
      - Buffered vs non-buffered
    - Message Passing Interface (MPI)
      - Popular message passing library
      - ~125 functions

Common Parallel Programming Models

- Message-Passing



Common Parallel Programming Models

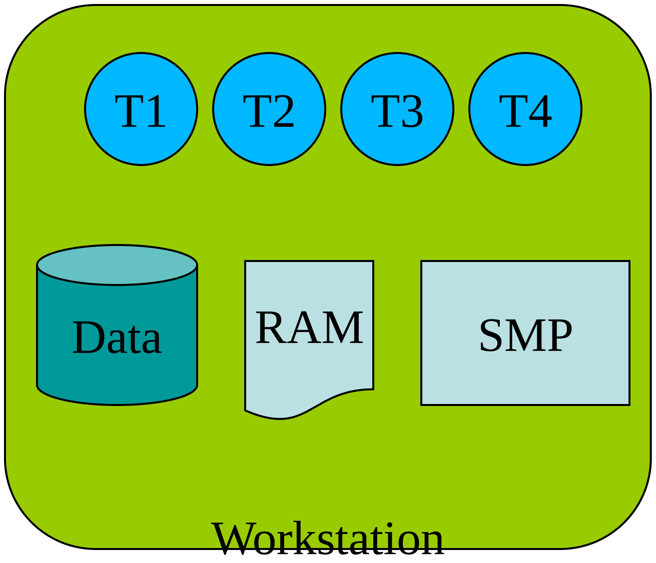
- Shared Address Space
  - Mostly used for programming SMP machines (multicore chips)
  - Key attributes
    - Shared address space
      - Threads
      - Shmget/shmat UNIX operations
    - Implicit parallelization
  - Process/Thread communication
    - Memory reads/stores

Common Parallel Programming Models

- Shared Address Space
  - Communication
    - Read/write memory
      - EX: x++;
  - Posix Thread API
    - Popular thread API
    - Operations
      - Creation/deletion of threads
      - Synchronization (mutexes, semaphores)
      - Thread management

Common Parallel Programming Models

- Shared Address Space



Parallel Programming Pitfalls

- Synchronization
  - Deadlock
  - Livelock
  - Fairness
- Efficiency
  - Maximize parallelism
- Reliability
  - Correctness
  - Debugging

Questions

?