Introduction to Parallel Processing

- Parallel Computer Architecture: Definition & Broad issues involved
  - A Generic Parallel Computer Architecture
- The Need And Feasibility of Parallel Computing
  - Scientific Supercomputing Trends
  - CPU Performance and Technology Trends, Parallelism in Microprocessor Generations
  - Computer System Peak FLOP Rating History/Near Future
- The Goal of Parallel Processing
- Elements of Parallel Computing
- Factors Affecting Parallel System Performance
- Parallel Architectures History
  - Parallel Programming Models
  - Flynn’s 1972 Classification of Computer Architecture
- Current Trends In Parallel Architectures
  - Modern Parallel Architecture Layered Framework
- Shared Address Space Parallel Architectures
- Message-Passing Multicomputers: Message-Passing Programming Tools
- Data Parallel Systems
- Dataflow Architectures
- Systolic Architectures: Matrix Multiplication Systolic Array Example

(PCA Chapter 1.1, 1.2)
Parallel Computer Architecture

A parallel computer (or multiple processor system) is a collection of communicating processing elements (processors) that cooperate to solve large computational problems fast by dividing such problems into parallel tasks, exploiting Thread-Level Parallelism (TLP).

**Broad issues involved:**

- The concurrency and communication characteristics of parallel algorithms for a given computational problem (represented by dependency graphs)
- Computing Resources and Computation Allocation:
  - The number of processing elements (PEs), computing power of each element and amount/organization of physical memory used.
  - What portions of the computation and data are allocated or mapped to each PE.
- Data access, Communication and Synchronization
  - How the processing elements cooperate and communicate.
  - How data is shared/transmitted between processors.
  - Abstractions and primitives for cooperation/communication.
  - The characteristics and performance of parallel system network (System interconnects).
- Parallel Processing Performance and Scalability Goals:
  - Maximize performance enhancement of parallelism: Maximize Speedup.
    - By minimizing parallelization overheads and balancing workload on processors.
  - Scalability of performance to larger systems/problems.

**Processor** = Programmable computing element that runs stored programs written using pre-defined instruction set

**Processing Elements** = PEs = Processors
A Generic Parallel Computer Architecture

Parallel Machine Network (Custom or industry standard)

A processing node

Operating System
Parallel Programming Environments

Processing Nodes:
Each processing node contains one or more processing elements (PEs) or processor(s), memory system, plus communication assist: (Network interface and communication controller)

Parallel machine network (System Interconnects).
Function of a parallel machine network is to efficiently (reduce communication cost) transfer information (data, results .. ) from source node to destination node as needed to allow cooperation among parallel processing nodes to solve large computational problems divided into a number parallel computational tasks.
The Need And Feasibility of Parallel Computing

- **Application demands:** More computing cycles/memory needed
  - *Scientific/Engineering computing:* CFD, Biology, Chemistry, Physics, ...
  - *General-purpose computing:* Video, Graphics, CAD, Databases, Transaction Processing, Gaming...
  - Mainstream multithreaded programs, are similar to parallel programs

- **Technology Trends:**
  - Number of transistors on chip growing rapidly. Clock rates expected to continue to go up but only slowly. Actual performance returns diminishing due to deeper pipelines.
  - Increased transistor density allows integrating multiple processor cores per creating Chip-Multiprocessors (CMPs) even for mainstream computing applications (desktop/laptop..).

- **Architecture Trends:**
  - Instruction-level parallelism (ILP) is valuable (superscalar, VLIW) but limited.
  - Increased clock rates require deeper pipelines with longer latencies and higher CPIs.
  - Coarser-level parallelism (at the task or thread level, TLP), utilized in multiprocessor systems is the most viable approach to further improve performance.
    - Main motivation for development of chip-multiprocessors (CMPs)

- **Economics:**
  - The increased utilization of commodity of-the-shelf (COTS) components in high performance parallel computing systems instead of costly custom components used in traditional supercomputers leading to much lower parallel system cost.
    - Today’s microprocessors offer high-performance and have multiprocessor support eliminating the need for designing expensive custom Pes.
    - Commercial System Area Networks (SANs) offer an alternative to custom more costly networks
Why is Parallel Processing Needed?

Challenging Applications in Applied Science/Engineering

• Astrophysics
• Atmospheric and Ocean Modeling
• Bioinformatics
• Biomolecular simulation: Protein folding
• Computational Chemistry
• Computational Fluid Dynamics (CFD)
• Computational Physics
• Computer vision and image understanding
• Data Mining and Data-intensive Computing
• Engineering analysis (CAD/CAM)
• Global climate modeling and forecasting
• Material Sciences
• Military applications
• Quantum chemistry
• VLSI design
• ....

Such applications have very high computational and memory requirements that cannot be met with single-processor architectures.

Many applications contain a large degree of computational parallelism

Driving force for High Performance Computing (HPC) and multiple processor system development
Why is Parallel Processing Needed?

**Scientific Computing Demands**

- Computational and memory demands exceed the capabilities of even the fastest current uniprocessor systems.

**Grand Challenge problems**
- Global change
- Human genome
- Fluid turbulence
- Vehicle dynamics
- Ocean circulation
- Viscous fluid dynamics
- Superconductor modeling
- Quantum chromo dynamics
- Vision

**Computational performance requirement**

- Grand Challenge problems
- Vehicle signature
- Structural biology
- Pharmaceutical design
- 72-hour weather
- Chemical dynamics
- 3-5 GFLOPS for uniprocessor
- 48-hour weather
- 3D plasma modeling
- 2D airfoil
- Oil reservoir modeling

**Storage requirement**
- 1 TB
- 100 GB
- 10 GB
- 1 GB
- 100 MB
- 10 MB
Scientific Supercomputing Trends

• Proving ground and driver for innovative architecture and advanced high performance computing (HPC) techniques:
  – Market is much smaller relative to commercial (desktop/server) segment.
  – Dominated by costly vector machines starting in the 70s through the 80s.
  – Microprocessors have made huge gains in the performance needed for such applications:
    • High clock rates.  (Bad: Higher CPI?)
    • Multiple pipelined floating point units.
    • Instruction-level parallelism.
    • Effective use of caches.
    • Multiple processor cores/chip (2 cores 2002-2005, 4 end of 2006, 8 cores 2007?)

However even the fastest current single microprocessor systems still cannot meet the needed computational demands.

• Currently: Large-scale microprocessor-based multiprocessor systems and computer clusters are replacing (replaced?) vector supercomputers that utilize custom processors.
Uniprocessor Performance Evaluation

• CPU Performance benchmarking is heavily program-mix dependent.
• Ideal performance requires a perfect machine/program match.

Performance measures:
  – Total CPU time = \( T = \frac{TC}{f} = TC \times C = I \times CPI \times C \)
    \[ = I \times (CPI_{\text{execution}} + m \times k) \times C \]  
    (in seconds)

  TC = Total program execution clock cycles
  f = clock rate  \( C = \) CPU clock cycle time = \( 1/f \)  \( I = \) Instructions executed count
  CPI = Cycles per instruction  \( CPI_{\text{execution}} = \) CPI with ideal memory
  m = Memory stall cycles per memory access
  k = Memory accesses per instruction

  – MIPS Rating = \( \frac{I}{(T \times 10^6)} = \frac{f}{(CPI \times 10^6)} = \frac{f \times I}{(TC \times 10^6)} \)  
    (in million instructions per second)

  – Throughput Rate:  \( W_p = \frac{1}{T} = \frac{f}{(I \times CPI)} = (MIPS) \times 10^6 /I \)  
    (in programs per second)

Performance factors: \( (I, CPI_{\text{execution}}, m, k, C) \) are influenced by: instruction-set architecture (ISA), compiler design, CPU micro-architecture, implementation and control, cache and memory hierarchy, program access locality, and program instruction mix and instruction dependencies.

\[ T = I \times CPI \times C \]
The microprocessor is currently the most natural building block for multiprocessor systems in terms of cost and performance. This is even more true with the development of cost-effective multi-core microprocessors that support TLP at the chip level.
Microprocessor Frequency Trend

Result:
- Deeper Pipelines
- Longer stalls
- Higher CPI (lowers effective performance per cycle)

Why?
1. Power leakage
2. Clock distribution delays

Solution:
Exploit TLP at the chip level, Chip-multiprocessor (CMPs)

Frequency doubles each generation?
Number of gates/clock reduce by 25%
Leads to deeper pipelines with more stages
(e.g. Intel Pentium 4E has 30+ pipeline stages)

T = I x CPI x C

Realty Check:
Clock frequency scaling is slowing down!
(Did silicone finally hit the wall?)

Processor freq scales by 2X per generation

Gate delays/clock
Moore’s Law:
- 2X transistors/Chip
- Every 1.5 years (circa 1970) still holds

Enables Thread-Level Parallelism (TLP) at the chip level:
- Chip-Multiprocessors (CMPs) + Simultaneous Multithreaded (SMT) processors

- One billion transistors/chip reached in 2005
- Transistor count grows faster than clock rate: Currently ~ 40% per year
- Single-threaded uniprocessors do not efficiently utilize the increased transistor count.

Limited ILP, increased size of cache
Improving microprocessor generation performance by exploiting more levels of parallelism.
Current Dual-Core Chip-Multiprocessor Architectures

**Single Die**
- Shared L2 Cache

**Cores communicate using shared cache**
(*Lowest communication latency*)

**Examples:**
- IBM POWER4/5
- Intel Pentium Core Duo (Yonah), Conroe
- Sun UltraSparc T1 (Niagara)

---

**Single Die**
- Private Caches
- Shared System Interface

**Cores communicate using on-chip Interconnects (shared system interface)**

**Examples:**
- AMD Dual Core Opteron,
- Athlon 64 X2
- Intel Itanium2 (Montecito)

---

**Two Dice – Shared Package**
- Private Caches
- Private System Interface

**Cores communicate over external Front Side Bus (FSB)**
(*Highest communication latency*)

**Example:**
- Intel Pentium D

Source: Real World Technologies,
http://www.realworldtech.com/page.cfm?ArticleID=RW101405234615
Microprocessors Vs. Vector Processors
Uniprocessor Performance: LINPACK

Now about 3-5 GFLOPS per microprocessor

1 GFLOP
(10⁹ FLOPS)

Microprocessors

Vector Processors

Linpack (MFLOPS)

CRAY
n = 1,000

CRAY
n = 100

Micro
n = 1,000

Micro
n = 100

Microprocessors

Vector Processors


CRAY 1s

DEC 8200

IBM Power2/990

MIPS R4400

DEC Alpha

HP 9000/735

DEC Alpha AXP

HP 9000/750

IBM RS6000/540

MIPS M/2000

MIPS M/120

Sun 4/260

Microprocessors

Vector Processors

1 GFLOP
(10⁹ FLOPS)
**Parallel Performance: LINPACK**

**Current Top LINPACK Performance:**
Now about 280,600 GFLOPS = 280.6 TeraFLOPS
IBM BlueGene/L
131072 (128K) processors (0.7 GHz PowerPC 440)

**1 TeraFLOP**
(10^{12} FLOPS = 1000 GFLOPS)

- CRAY peak
- MPP peak
- Xmp/416(4)
- Ymp/832(8)
- Paragon XP/S MP (6768)
- CM-2
- CM-200
- CM-5
- Delta
- IPSC/860 nCUBE/2 (1024)
- Paragon XP/S MP (1024)
- Paragon XP/S MP (1024)
- T3D
- T932(32)
- ASCI Red
- C90(16)

Current ranking of top 500 parallel supercomputers in the world is found at: www.top500.org
Why is Parallel Processing Needed?  
LINPAK Performance Trends

Uniprocessor Performance  
Parallel System Performance

1 TeraFLOP  
(10^{12} FLOPS = 1000 GFLOPS)

1 GFLOP  
(10^9 FLOPS)
Current Top Peak FP Performance:
Now about 367,000 GFLOPS = 367 TeraFLOPS = 0.367 Peta FLOP
IBM BlueGene/L
131072 (128K) processors (0.7 GHz PowerPC 440)
The Goal of Parallel Processing

• Goal of applications in using parallel machines:
  Maximize Speedup over single processor performance

\[
\text{Speedup (p processors)} = \frac{\text{Performance (p processors)}}{\text{Performance (1 processor)}}
\]

• For a fixed problem size (input data set),
  performance = 1/time

\[
\text{Speedup}_{\text{fixed problem}} (p \text{ processors}) = \frac{\text{Time (1 processor)}}{\text{Time (p processors)}}
\]

• Ideal speedup = number of processors = p

Very hard to achieve

Due to parallelization overheads: communication cost, dependencies ...
The Goal of Parallel Processing

- Parallel processing goal is to maximize parallel speedup:

\[
\text{Speedup} = \frac{\text{Time}(1)}{\text{Time}(p)} \leq \frac{\text{Sequential Work on one processor}}{\text{Max} (\text{Work + Synch Wait Time + Comm Cost + Extra Work})}
\]

i.e the processor with maximum execution time

- Ideal Speedup = \( p \) number of processors
  - Very hard to achieve: Implies no parallelization overheads and perfect load balance among all processors.
- Maximize parallel speedup by:
  1. Balancing computations on processors (every processor does the same amount of work) and the same amount of overheads.
  2. Minimizing communication cost and other overheads associated with each step of parallel program creation and execution.

- Performance Scalability:
  Achieve a good speedup for the parallel application on the parallel architecture as problem size and machine size (number of processors) are increased.
Elements of Parallel Computing

- Computing Problems
- Algorithms and Data Structures
- Operating System
- Applications Software
- Hardware Architecture
- High-level Languages
- Performance Evaluation
- Dependency analysis
- Mapping
- Programming
- Binding (Compile, Load)
- e.g. Speedup

Assign parallel computations to processors
Elements of Parallel Computing

1 Computing Problems:

- Numerical Computing: Science and engineering numerical problems demand intensive integer and floating point computations.
- Logical Reasoning: Artificial intelligence (AI) demand logic inferences and symbolic manipulations and large space searches.

2 Parallel Algorithms and Data Structures

- Special algorithms and data structures are needed to specify the computations and communication present in computing problems (from dependency analysis).
- Most numerical algorithms are deterministic using regular data structures.
- Symbolic processing may use heuristics or non-deterministic searches.
- Parallel algorithm development requires interdisciplinary interaction.
Elements of Parallel Computing

3 Hardware Resources
- Processors, memory, and peripheral devices (processing nodes) form the hardware core of a computer system.
- Processor connectivity (system interconnects, network), memory organization, influence the system architecture.

4 Operating Systems
- Manages the allocation of resources to running processes.
- Mapping to match algorithmic structures with hardware architecture and vice versa: processor scheduling, memory mapping, interprocessor communication.
- Parallelism exploitation possible at: algorithm design, program writing, compilation, and run time.
Elements of Parallel Computing

5 System Software Support
   – Needed for the development of efficient programs in high-level languages (HLLs.)
   – Assemblers, loaders.
   – Portable parallel programming languages
   – User interfaces and tools.

6 Compiler Support
   – Preprocessor compiler: Sequential compiler and low-level library of the target parallel computer.
   – Precompiler: Some program flow analysis, dependence checking, limited optimizations for parallelism detection.
   – Parallelizing compiler: Can automatically detect parallelism in source code and transform sequential code into parallel constructs.
Approaches to Parallel Programming

Programmer

Source code written in sequential languages C, C++, FORTRAN, LISP ..

Parallelizing compiler

Parallel object code

Execution by runtime system

(a) Implicit Parallelism

Programmer

Source code written in concurrent dialects of C, C++, FORTRAN, LISP ..

Concurrency preserving compiler

Concurrent object code

Execution by runtime system

(b) Explicit Parallelism
Factors Affecting Parallel System Performance

• **Parallel Algorithm Related:**
  – Available concurrency and profile, grain size, uniformity, patterns.
  – Dependencies between computations represented by *dependency graph*
  – Required communication/synchronization, uniformity and patterns.
  – Data size requirements.
  – Communication to computation ratio (C-to-C ratio, lower is better).

• **Parallel program Related:**
  – Programming model used.
  – Resulting data/code memory requirements, locality and working set characteristics.
  – Parallel task grain size.
  – Assignment of tasks to processors: Dynamic or static.
  – Cost of communication/synchronization primitives.

• **Hardware/Architecture related:**
  – Total CPU computational power available.
  – Types of computation modes supported.
  – Shared address space Vs. message passing.
  – Communication network characteristics (topology, bandwidth, latency)
  – Memory hierarchy properties.

Concurrency = Parallelism
A simple parallel execution example

Sequential Execution on one processor

Dependency Graph

Possible Parallel Execution Schedule on Two Processors P0, P1

What would the speed be with 3 processors? 4 processors? 5 … ?

Assume computation time for each task A-G = 3
Assume communication time between parallel tasks = 1
Assume communication can overlap with computation
Speedup on two processors = \( \frac{T_1}{T_2} = \frac{21}{16} = 1.3 \)

A simple parallel execution example
Evolution of Computer Architecture

- Scalar
  - Sequential
  - Lookahead
    - I/E Overlap
    - Functional Parallelism
      - Multiple Func. Units
      - Pipeline
        - Implicit Vector
        - Explicit Vector
          - Memory-to-Memory
          - Register-to-Register

- SIMD
  - Single Instruction stream over Multiple Data streams
- MIMD
  - Multiple Instruction streams over Multiple Data streams

I/E: Instruction Fetch and Execute
SIMD: Single Instruction stream over Multiple Data streams
MIMD: Multiple Instruction streams over Multiple Data streams

Data Parallel
Message Passing

Shared Memory

EECC756 - Shaaban
Historically, parallel architectures were tied to programming models:

- Divergent architectures, with no predictable pattern of growth.

More on this next lecture.
Parallel Programming Models

• Programming methodology used in coding applications (parallel)
• Specifies communication and synchronization

• Examples:
  – **Multiprogramming**: or Multi-tasking (not true parallel processing!)
    No communication or synchronization at program level. A number of independent programs running on different processors in the system.
  – **Shared memory address space (SAS)**:
    Parallel program threads or tasks communicate using a shared memory address space (shared data in memory).
  – **Message passing**:
    Explicit point to point communication is used between parallel program tasks using messages.
  – **Data parallel**:
    More regimented, global actions on data (i.e the same operations over all elements on an array or vector)
    – Can be implemented with shared address space or message passing
Flynn’s 1972 Classification of Computer Architecture

- Single Instruction stream over a Single Data stream (SISD): Conventional sequential machines or uniprocessors.
- Single Instruction stream over Multiple Data streams (SIMD): Vector computers, array of synchronized processing elements.
- Multiple Instruction streams and a Single Data stream (MISD): Systolic arrays for pipelined execution.
- Multiple Instruction streams over Multiple Data streams (MIMD): Parallel computers:
  - Shared memory multiprocessors.
  - Multicomputers: Unshared distributed memory, message-passing used instead (e.g. clusters)
Flynn’s Classification of Computer Architecture  
(Taxonomy)

Uniprocessor

Single Instruction stream over a Single Data stream (SISD):  
Conventional sequential machines or uniprocessors.

Single Instruction stream over Multiple Data streams (SIMD):  
Vector computers, array of synchronized processing elements.

Multiple Instruction streams and a Single Data stream (MISD):  
Systolic arrays for pipelined execution.

Multiple Instruction streams over Multiple Data streams (MIMD):  
Parallel computers: Distributed memory multiprocessor system shown

CU = Control Unit
PE = Processing Element
M = Memory

Shown here: array of synchronized processing elements

Classified according to number of instruction streams (threads) and number of data streams in architecture
Current Trends In Parallel Architectures

• The extension of “computer architecture” to support communication and cooperation:
  
  – OLD: Instruction Set Architecture (ISA)
  – NEW: Communication Architecture

• Defines:
  
  – Critical abstractions, boundaries, and primitives (interfaces)
  – Organizational structures that implement interfaces (hardware or software)

• Compilers, libraries and OS are important bridges today

More on this next lecture
Modern Parallel Architecture
Layered Framework

- CAD
- Database
- Scientific modeling
- Parallel applications
  - Programming models
  - Communication abstraction
    - User/system boundary
  - Hardware/software boundary
    - (ISA)

- Multiprogramming
- Shared address
- Message passing
- Data parallel

- Compilation or library
- Operating systems support

- Communication hardware
- Physical communication medium

More on this next lecture
Shared Address Space Parallel Architectures (SAS)

- Any processor can directly reference any memory location
  - Communication occurs implicitly as result of loads and stores

- Convenient:
  - Location transparency
  - Similar programming model to time-sharing in uniprocessors
    - Except processes run on different processors
    - Good throughput on multiprogrammed workloads

- Naturally provided on a wide range of platforms
  - Wide range of scale: few to hundreds of processors

- Popularly known as *shared memory* machines or model
  - *Ambiguous:* Memory may be physically distributed among processing nodes.
    - i.e Distributed shared memory multiprocessors

Sometimes called Tightly-Coupled Parallel Computers
Shared Address Space (SAS) Parallel Programming Model

- **Process**: virtual address space plus one or more threads of control
- Portions of address spaces of processes are shared

- Writes to shared address visible to other threads (in other processes too)
- Natural extension of the uniprocessor model:
  - Conventional memory operations used for communication
  - Special atomic operations needed for synchronization
- OS uses shared memory to coordinate processes
Models of Shared-Memory Multiprocessors

• **The Uniform Memory Access (UMA) Model:**
  – The physical memory is shared by all processors.
  – All processors have equal access to all memory addresses.
  – Also referred to as Symmetric Memory Processors (SMPs).

• **Distributed memory or Non-uniform Memory Access (NUMA) Model:**
  – Shared memory is physically distributed locally among processors. Access to remote memory is higher.

• **The Cache-Only Memory Architecture (COMA) Model:**
  – A special case of a NUMA machine where all distributed main memory is converted to caches.
  – No memory hierarchy at each processor.
Models of Shared-Memory Multiprocessors

Uniform Memory Access (UMA) Model
or Symmetric Memory Processors (SMPs).

Interconnect:
Bus, Crossbar, Multistage network
P: Processor
M: Memory
C: Cache
D: Cache directory

Distributed memory or
Non-uniform Memory Access (NUMA) Model

Cache-Only Memory Architecture (COMA)
Uniform Memory Access Example: Intel Pentium Pro Quad

- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume

Bus-Based Symmetric Memory Processors (SMPs).
A single Front Side Bus (FSB) is shared among processors
This severely limits scalability to only 2-4 processors

EECC756 - Shaaban

#38  lec #1  Spring 2006  3-14-2006
Non-Uniform Memory Access (NUMA)

Example: AMD 8-way Opteron Server Node

Dedicated point-to-point interconnects (HyperTransport links) used to connect processors alleviating the traditional limitations of FSB-based SMP systems. Each processor has two integrated DDR memory channel controllers: memory bandwidth scales up with number of processors. NUMA architecture since a processor can access its own memory at a lower latency than access to remote memory directly connected to other processors in the system.

Total 16 processor cores when dual core Opteron processors used.
Uniform Memory Access Example: SUN Enterprise

- 16 cards of either type: processors + memory, or I/O
- All memory accessed over bus, so symmetric
- Higher bandwidth, higher latency bus
Distributed Shared-Memory Multiprocessor System Example: Cray T3E

NUMA Example

Circa 1995-1999

- Scale up to 2048 processors, DEC Alpha EV6 microprocessor (COTS)
- Custom 3D Torus point-to-point network, 480MB/s links
- Memory controller generates communication requests for non-local references
- No hardware mechanism for coherence (SGI Origin etc. provide this)

Example of Non-uniform Memory Access (NUMA)
Message-Passing Multicomputers

- Comprised of multiple autonomous computers (nodes) connected via a suitable network. (Industry standard System Area Network (SAN) or proprietary network)
- Each node consists of one or more processors, local memory, attached storage and I/O peripherals.
- Local memory is only accessible by local processors in a node (no shared memory among nodes).
- Inter-node communication is carried out by message passing through the connection network.
- Process communication achieved using a message-passing programming environment (e.g. PVM, MPI).
  - Programming model more removed from basic hardware operations
- Include:
  - A number of commercial Massively Parallel Processor systems (MPPs).
  - Computer clusters that utilize commodity of-the-shelf (COTS) components.

Also called Loosely-Coupled Parallel Computers
Message-Passing Abstraction

- Send specifies buffer to be transmitted and receiving process
- Receive specifies sending process and application storage to receive into
- Memory to memory copy possible, but need to name processes
- Optional tag on send and matching rule on receive
- User process names local data and entities in process/tag space too
- In simplest form, the send/receive match achieves pairwise synchronization event
  - Ordering of computations according to dependencies
- Many possible overheads: copying, buffer management, protection...

 Sender

\[
\text{Send (X, Q, t)}
\]

Recipient

\[
\text{Receive (Y, P, t)}
\]

 Local process address space

<table>
<thead>
<tr>
<th>Address X</th>
<th>Address Y</th>
</tr>
</thead>
</table>

Tag

Data

Recipient

Tag

Data

Sender

Process P

Process Q
Message-Passing Example: Intel Paragon

Sandia’s Intel Paragon XP/S-based Supercomputer

2D grid network with processing node attached to every switch

2D grid point to point network

Circa 1983
Message-Passing Example: IBM SP-2

- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bandwidth limited by I/O bus)

General interconnection network formed from 8-port switches

Circa 1994-1998
Message-Passing MPP Example:
IBM Blue Gene/L

(2 processors/chip) • (2 chips/compute card) • (16 compute cards/node board) • (32 node boards/tower) • (64 tower) = 128k = 131072 (0.7 GHz PowerPC 440) processors (64k nodes)

System Location: Lawrence Livermore National Laboratory

Networks:
3D Torus point-to-point network
Global tree 3D point-to-point network (both proprietary)

Current Top Ranking LINPACK Performance:
280,600 GFLOPS = 280.6 TeraFLOPS = 0.2806 Peta FLOP

Current Top Peak FP Performance:
Now about 367,000 GFLOPS = 367 TeraFLOPS = 0.367 Peta FLOP
Message-Passing Programming Tools

- Message-passing programming environments include:
  - **Message Passing Interface (MPI):**
    - Provides a standard for writing concurrent message-passing programs.
    - MPI implementations include parallel libraries used by existing programming languages (C, C++).
  - **Parallel Virtual Machine (PVM):**
    - Enables a collection of heterogeneous computers to used as a coherent and flexible concurrent computational resource.
    - PVM support software executes on each machine in a user-configurable pool, and provides a computational environment of concurrent applications.
    - User programs written for example in C, Fortran or Java are provided access to PVM through the use of calls to PVM library routines.
Data Parallel Systems

- Programming model (Data Parallel)
  - Operations performed in parallel on each element of data structure
  - Logically single thread of control, performs sequential or parallel steps
  - Conceptually, a processor is associated with each data element

- Architectural model
  - Array of many simple, cheap processors each with little memory
    - Processors don’t sequence through instructions
  - Attached to a control processor that issues instructions
  - Specialized and general communication, cheap global synchronization

- Example machines:
  - Thinking Machines CM-1, CM-2 (and CM-5)
  - Maspar MP-1 and MP-2,

PE = Processing Element
Dataflow Architectures

- Represent computation as a graph of essential data dependencies
  - Non-Von Neumann Architecture (Not PC-based)
  - Logical processor at each node, activated by availability of operands
  - Message (tokens) carrying tag of next instruction sent to next processor
  - Tag compared with others in matching store; match fires execution

Research Dataflow machine prototypes include:
- The MIT Tagged Architecture
- The Manchester Dataflow Machine

The Tomasulo approach of dynamic instruction execution utilizes dataflow driven execution engine:
- The data dependency graph for a small window of instructions is constructed dynamically when instructions are issued in order of the program.
- The execution of an issued instruction is triggered by the availability of its operands (data it needs) over the CDB.

Tokens = Copies of computation results
Systolic Architectures

- Replace single processor with an array of regular processing elements
- Orchestrate data flow for high throughput with less memory access

- Different from pipelining
  - Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory
- Different from SIMD: each PE may do something different
- Initial motivation: VLSI Application-Specific Integrated Circuits (ASICs)
- Represent algorithms directly by chips connected in regular pattern

A possible example of MISD in Flynn’s Classification of Computer Architecture
Systolic Array Example:

3x3 Systolic Array Matrix Multiplication

• Processors arranged in a 2-D grid
• Each processor accumulates one element of the product

Alignments in time

Rows of A

a0,2  a0,1  a0,0

a1,2  a1,1  a1,0

a2,2  a2,1  a2,0

T = 0

Columns of B

C = A X B

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

T = 2

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

\[ T = 5 \]

Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

\[ T = 6 \]

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

Alignments in time

\[
\begin{align*}
a_{0,0}b_{0,0} &+ a_{0,1}b_{1,0} + a_{0,2}b_{2,0} \\
a_{1,0}b_{0,0} &+ a_{1,1}b_{1,0} + a_{1,2}b_{2,0} \\
a_{2,0}b_{0,0} &+ a_{2,1}b_{1,0} + a_{2,2}b_{2,0} \\
a_{0,0}b_{0,1} &+ a_{0,1}b_{1,1} + a_{0,2}b_{2,1} \\
a_{1,0}b_{0,1} &+ a_{1,1}b_{1,1} + a_{1,2}b_{2,1} \\
a_{2,0}b_{0,1} &+ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} \\
a_{0,0}b_{0,2} &+ a_{0,1}b_{1,2} + a_{0,2}b_{2,2} \\
a_{1,0}b_{0,2} &+ a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\
a_{2,0}b_{0,2} &+ a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \\
\end{align*}
\]

\[T = 7\]

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/
### 26th List: The TOP10

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Computer</th>
<th>Rmax [TF/s]</th>
<th>Installation Site</th>
<th>Country</th>
<th>Year</th>
<th>#Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM BlueGene/L eServer Blue Gene</td>
<td>280.6</td>
<td>DOE/NNSA/LLNL</td>
<td>USA</td>
<td>2005</td>
<td>131072</td>
</tr>
<tr>
<td>2</td>
<td>IBM BGW eServer Blue Gene</td>
<td>91.29</td>
<td>IBM Thomas Watson</td>
<td>USA</td>
<td>2005</td>
<td>40960</td>
</tr>
<tr>
<td>3</td>
<td>IBM ASC Purple eServer pSeries p575</td>
<td>63.39</td>
<td>DOE/NNSA/LLNL</td>
<td>USA</td>
<td>2005</td>
<td>10240</td>
</tr>
<tr>
<td>4</td>
<td>SGI Columbia Altix, Infiniband</td>
<td>51.87</td>
<td>NASA Ames</td>
<td>USA</td>
<td>2004</td>
<td>10160</td>
</tr>
<tr>
<td>5</td>
<td>Dell Thunderbird</td>
<td>38.27</td>
<td>Sandia</td>
<td>USA</td>
<td>2005</td>
<td>8000</td>
</tr>
<tr>
<td>6</td>
<td>Cray Red Storm Cray XT3</td>
<td>36.19</td>
<td>Sandia</td>
<td>USA</td>
<td>2005</td>
<td>10880</td>
</tr>
<tr>
<td>7</td>
<td>NEC Earth-Simulator</td>
<td>35.86</td>
<td>Earth Simulator Center</td>
<td>Japan</td>
<td>2002</td>
<td>5120</td>
</tr>
<tr>
<td>8</td>
<td>IBM MareNostrum BladeCenter JS20, Myrinet</td>
<td>27.91</td>
<td>Barcelona Supercomputer Center</td>
<td>Spain</td>
<td>2005</td>
<td>4800</td>
</tr>
<tr>
<td>9</td>
<td>IBM eServer Blue Gene</td>
<td>27.45</td>
<td>ASTRON University Groningen</td>
<td>Netherlands</td>
<td>2005</td>
<td>12288</td>
</tr>
<tr>
<td>10</td>
<td>Cray Jaguar Cray XT3</td>
<td>20.53</td>
<td>Oak Ridge National Lab</td>
<td>USA</td>
<td>2005</td>
<td>5200</td>
</tr>
</tbody>
</table>

Source: www.top500.org