

Introduction to Parallel Processing

- **Parallel Computer Architecture: Definition & Broad issues involved**
 - A Generic Parallel Computer Architecture
- **The Need And Feasibility of Parallel Computing** Why?
 - 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**

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). i.e Parallel Processing

- Broad issues involved:

Task = Computation done on one processor

- 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 and synchronization.
 - 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.

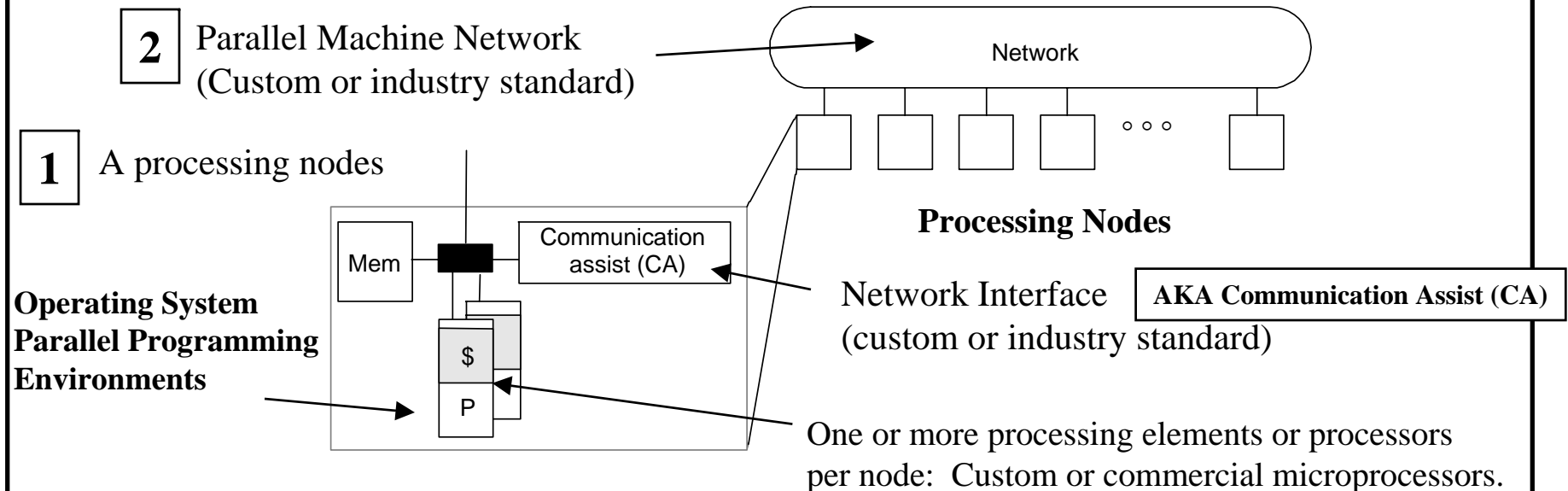
Processor = Programmable computing element that runs stored programs written using pre-defined instruction set

Processing Elements = PEs = Processors

EECC756 - Shaaban

#2 lec # 1 Spring 2010 3-9-2010

A Generic Parallel Computer Architecture



1 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)

2 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.

Parallel Computer = Multiple Processor System

EECC756 - Shaaban

The Need And Feasibility of Parallel Computing

• Application demands: More computing cycles/memory needed

Driving
Force

- *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:

Moore's Law still alive

- 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:

+ multi-tasking (multiple independent programs)

- 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)

Multi-core
Processors

• 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

EECC756 - Shaaban

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
-

Traditional Driving Force For HPC/Parallel Processing

Such applications have very high 1- computational and 2- 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

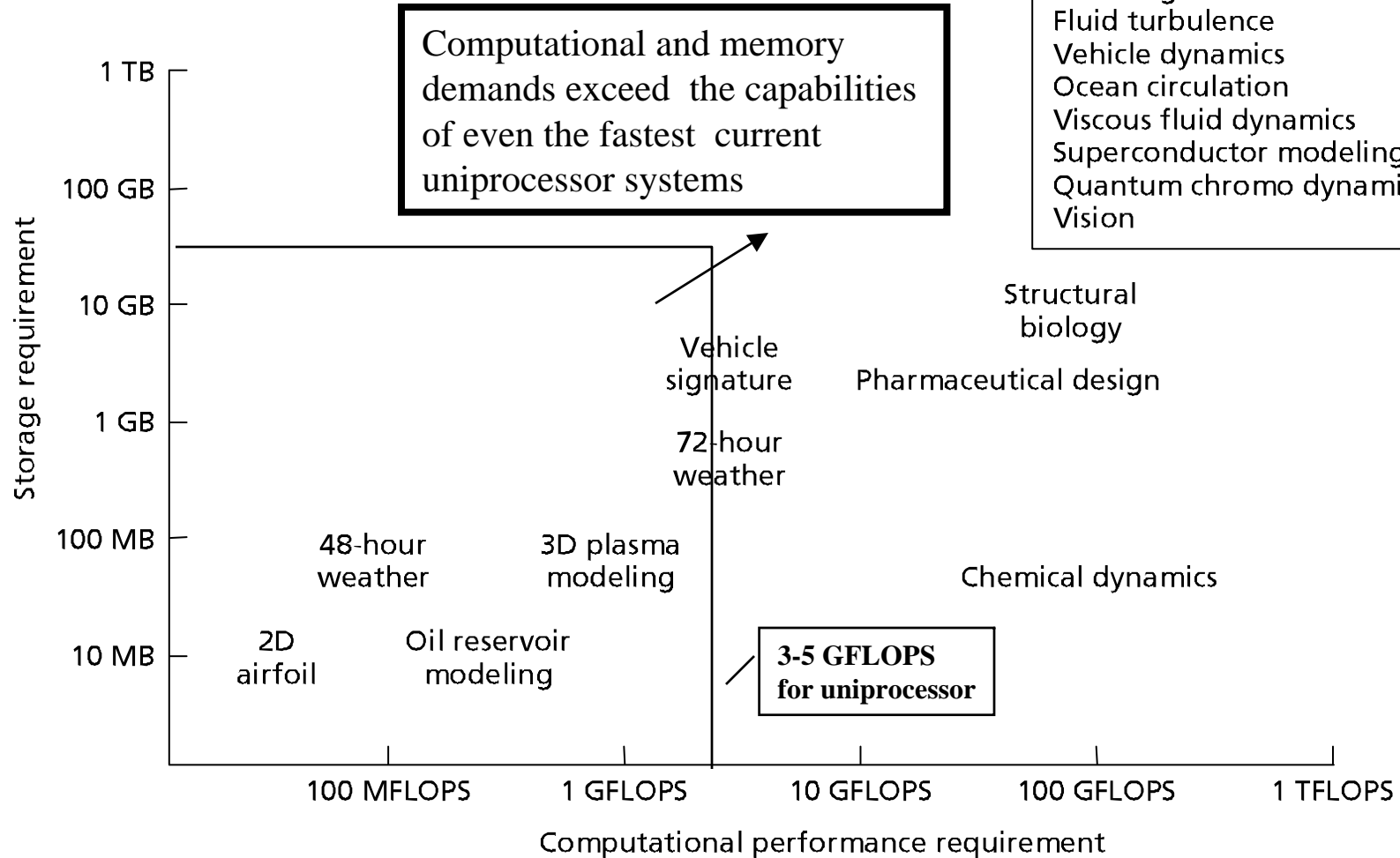
EECC756 - Shaaban

Why is Parallel Processing Needed? Scientific Computing Demands

Driving force for HPC and multiple processor system development

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

(Memory Requirement)



EECC756 - Shaaban

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 1970s through the 1980s.
 - 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, 6-12 cores 2010?)

Enabled with high transistor density/chip

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

As shown in last slide

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

EECC756 - Shaaban

Uniprocessor Performance Evaluation

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

- **Total CPU time = $T = TC / f = TC \times C = I \times CPI \times C$**
 $= I \times (CPI_{\text{execution}} + M \times k) \times C$ (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 = $I / (T \times 10^6) = f / (CPI \times 10^6) = f \times I / (TC \times 10^6)$**
(in million instructions per second)

- **Throughput Rate: $W_p = 1/T = f / (I \times CPI) = (MIPS) \times 10^6 / I$**
(in programs per second)

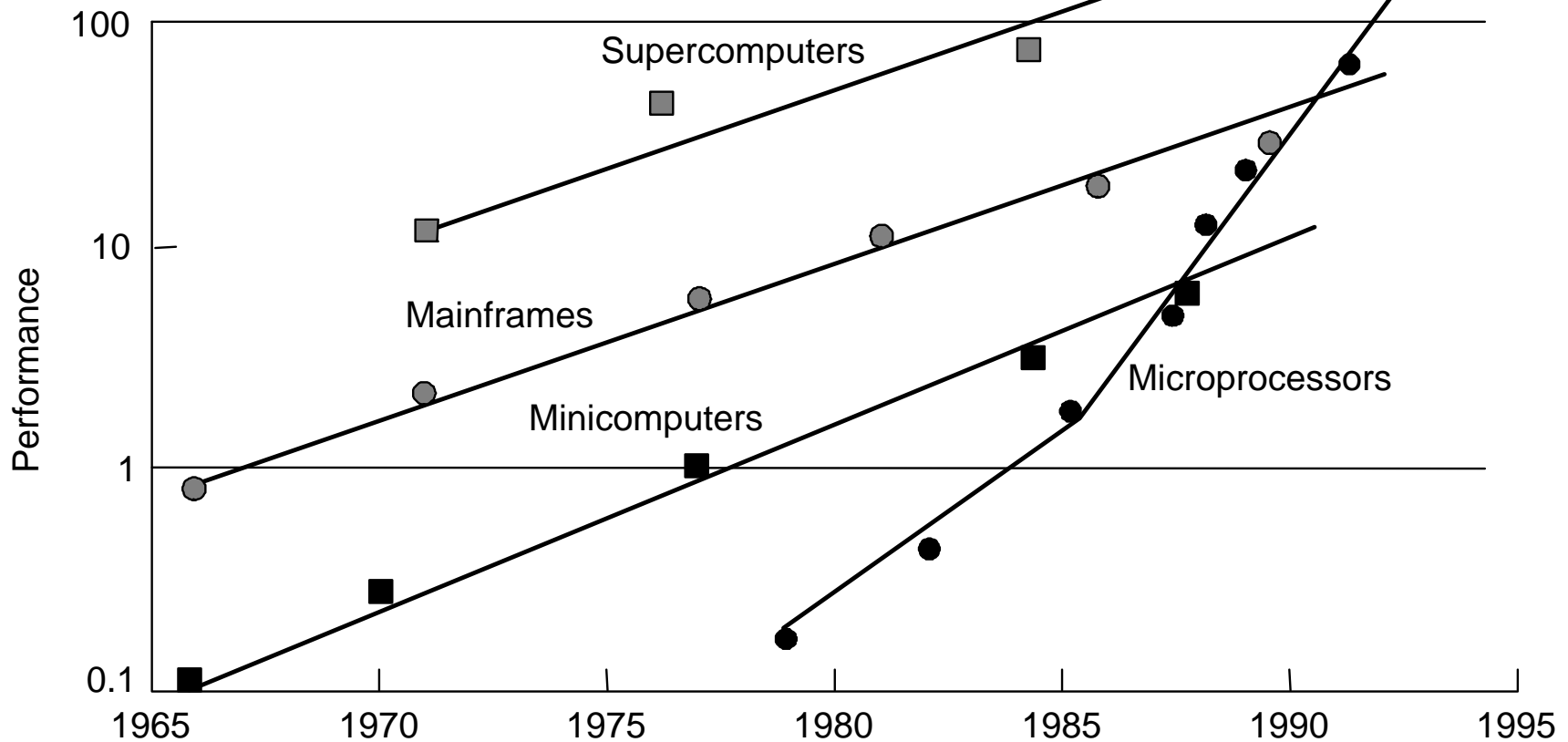
- Performance factors: (I, $CPI_{\text{execution}}$, m, k, C) are influenced by: instruction-set architecture (ISA), compiler design, CPU micro-architecture, implementation and control, cache and memory hierarchy, program access locality, and program instruction mix and instruction dependencies.

$$T = I \times CPI \times C$$

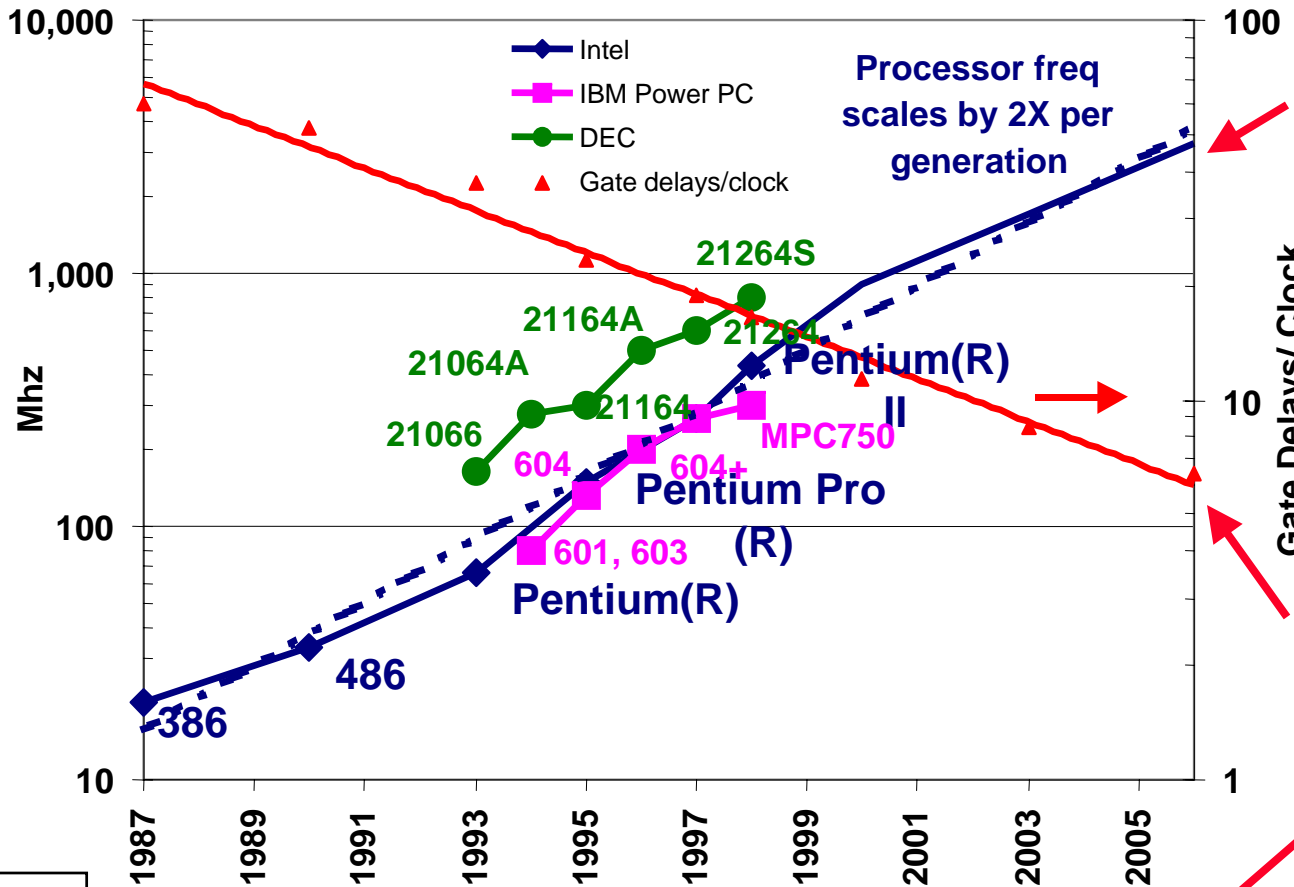
EECC756 - Shaaban

Single CPU Performance Trends

- 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



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

Why?
 1- Power leakage
 2- Clock distribution delays

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

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

No longer the case

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)

$$T = I \times CPI \times C$$

EECC756 - Shaaban

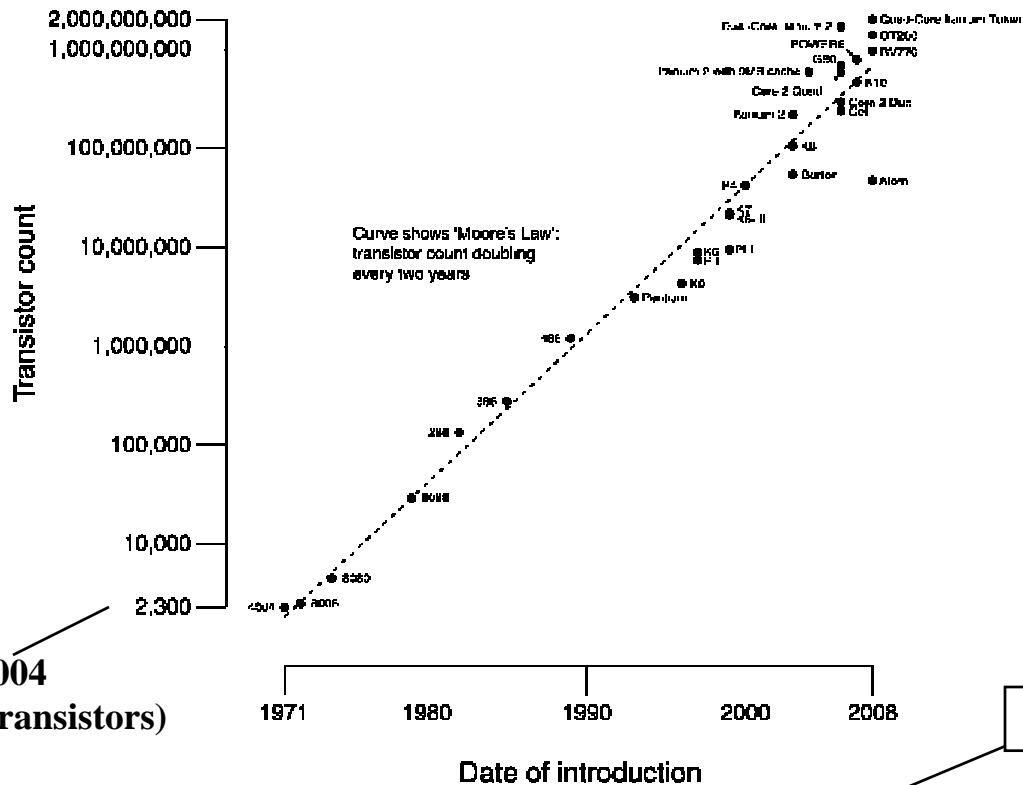
Transistor Count Growth Rate

Enabling Technology for Chip-Level Thread-Level Parallelism (TLP)

CPU Transistor Counts 1971-2008 & Moore's Law

~ 800,000x transistor density increase in the last 38 years

Currently ~ 2 Billion



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

Intel 4004
(2300 transistors)

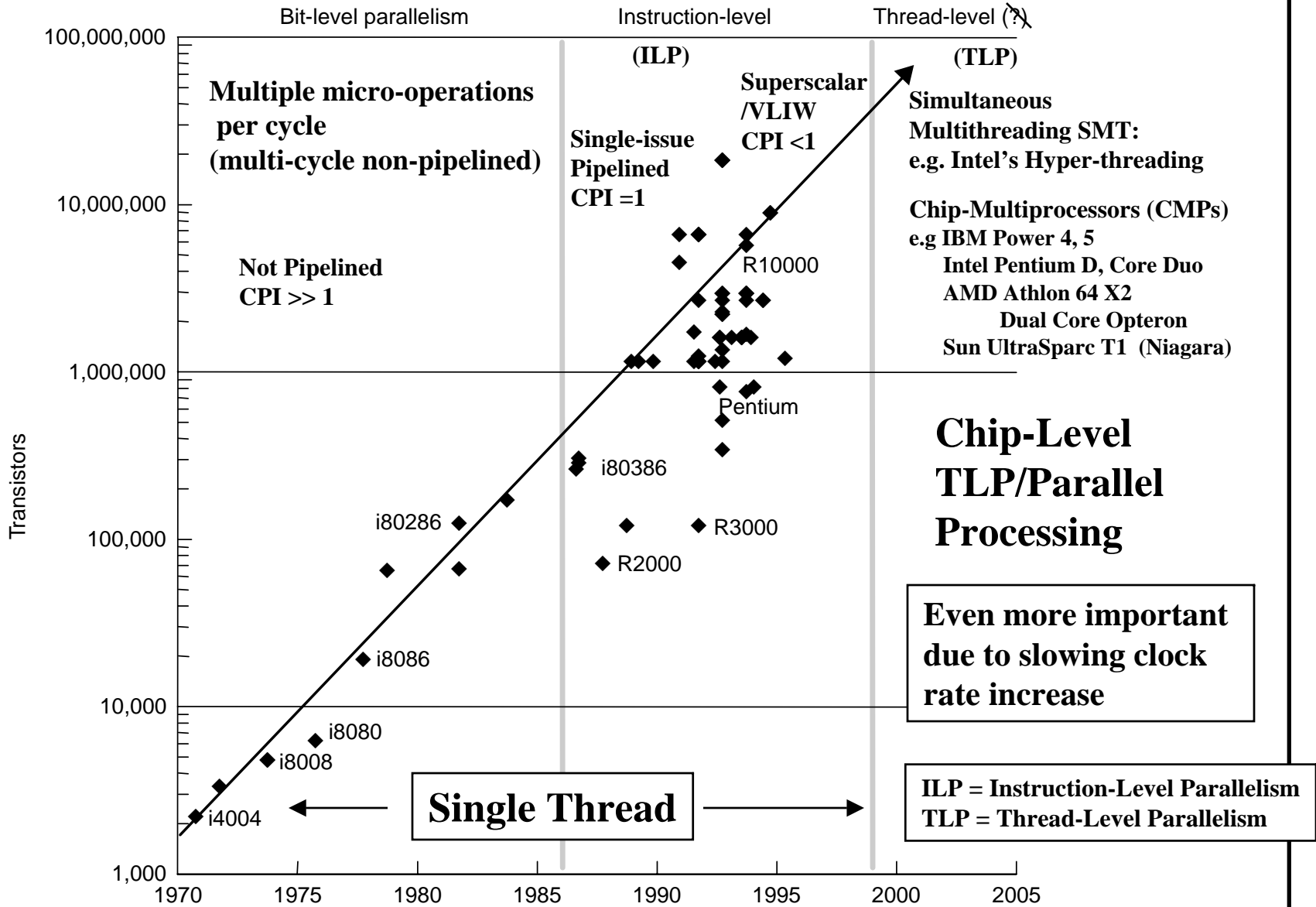
Solution

- One billion transistors/chip reached in 2005, two billion in 2008-9
- 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

EECC756 - Shaaban

Parallelism in Microprocessor VLSI Generations

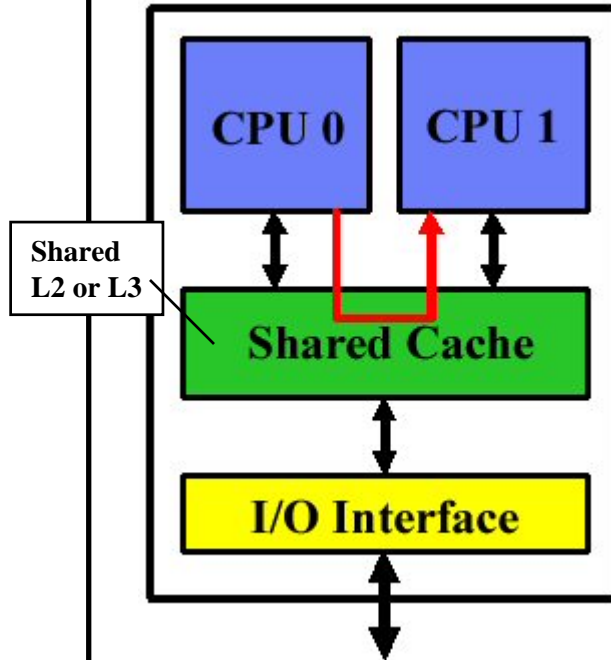


Improving microprocessor generation performance by exploiting more levels of parallelism

EECC756 - Shaaban

Current Dual-Core Chip-Multiprocessor Architectures

Single Die
Shared L2 Cache

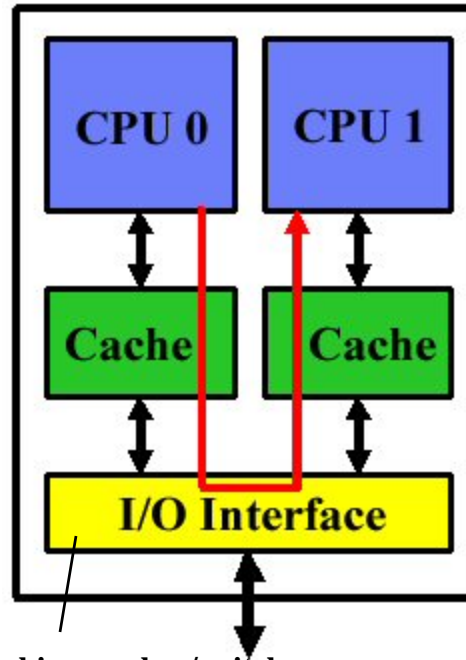


Cores communicate using shared cache
(Lowest communication latency)

Examples:

IBM POWER4/5
Intel Pentium Core Duo (Yonah), Conroe (Core 2), i7, Sun UltraSparc T1 (Niagara)
AMD Phenom

Single Die
Private Caches
Shared System Interface



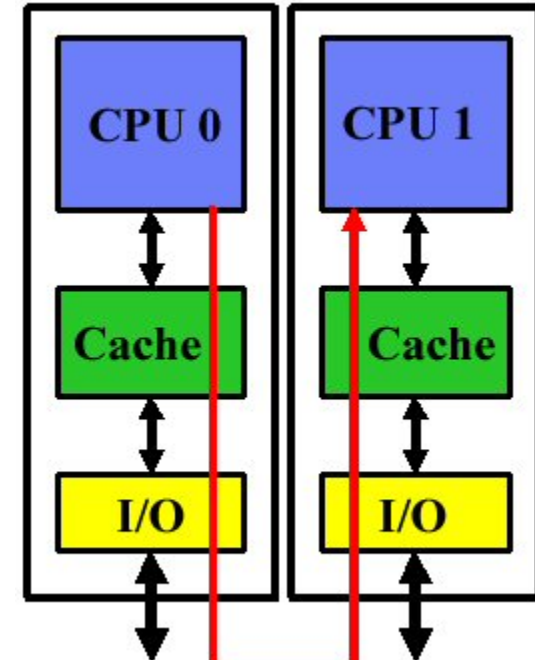
On-chip crossbar/switch

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



FSB

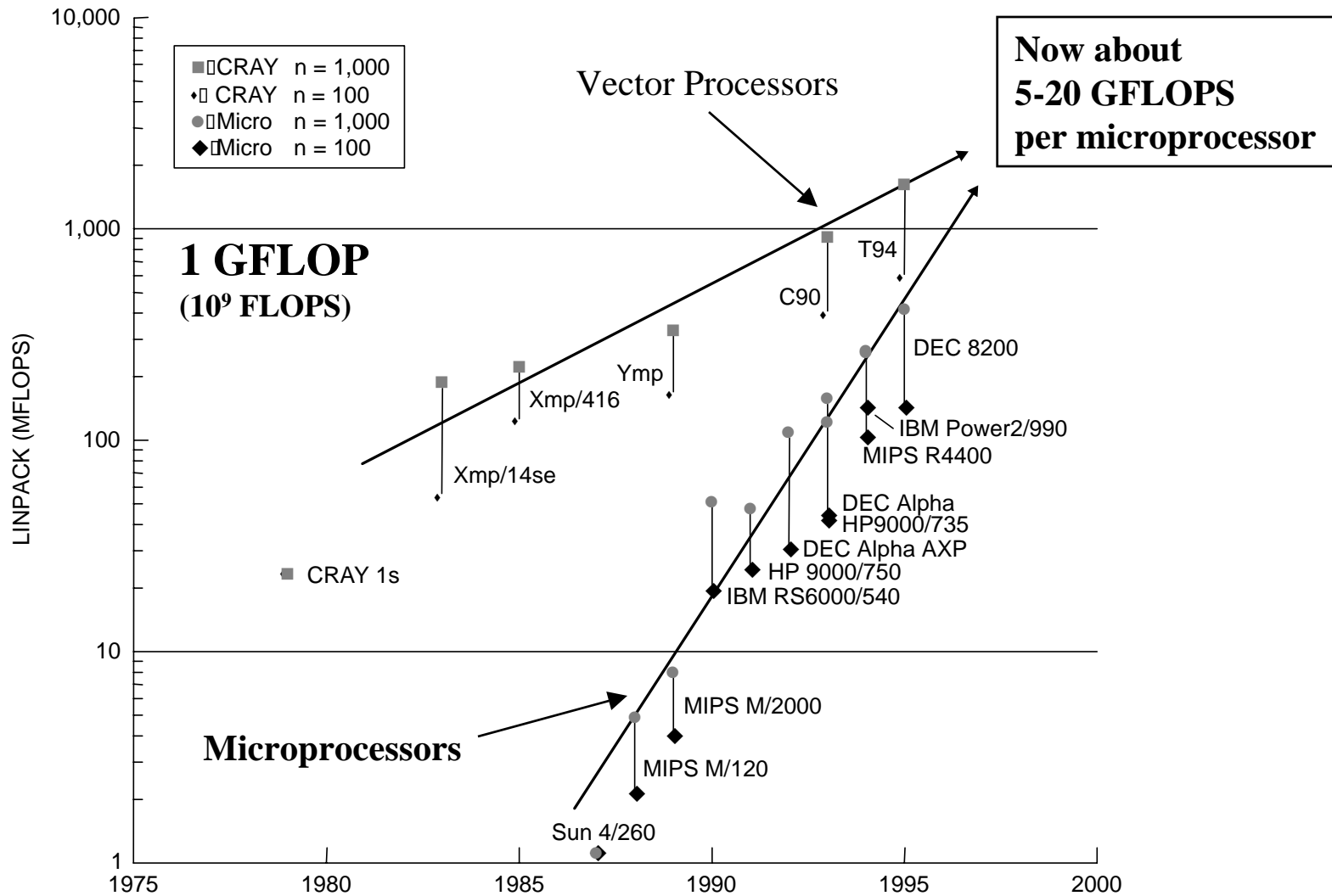
Cores communicate over external Front Side Bus (FSB)
(Highest communication latency)

Examples:

Intel Pentium D,
Intel Quad core (two dual-core chips)

Microprocessors Vs. Vector Processors

Uniprocessor Performance: LINPACK



EECC756 - Shaaban

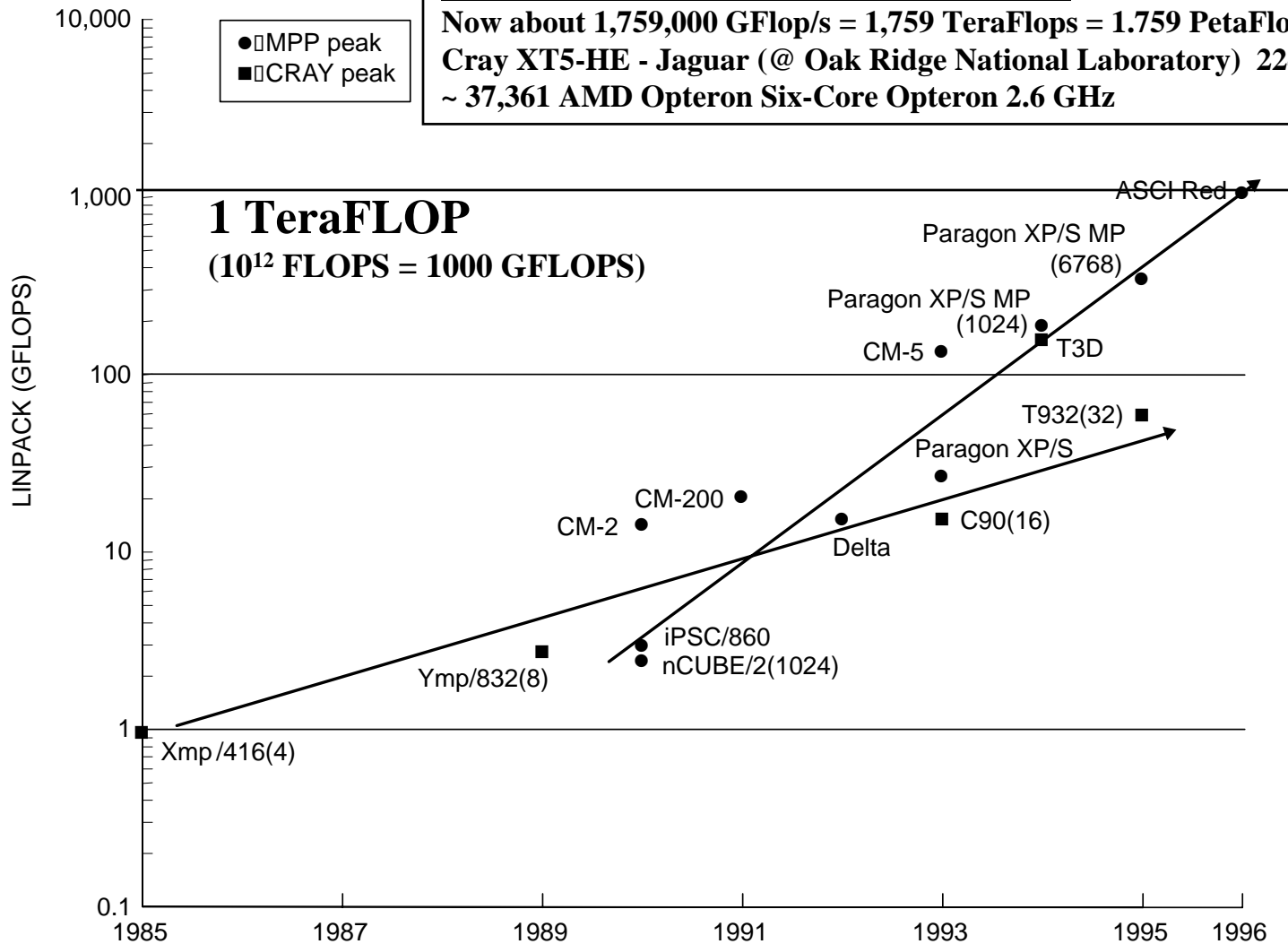
Parallel Performance: LINPACK

Since ~ Nov. 2009

Current Top LINPACK Performance:

Now about 1,759,000 GFlop/s = 1,759 TeraFlops = 1.759 PetaFlops

Cray XT5-HE - Jaguar (@ Oak Ridge National Laboratory) 224,162 processor cores:
~ 37,361 AMD Opteron Six-Core Opteron 2.6 GHz

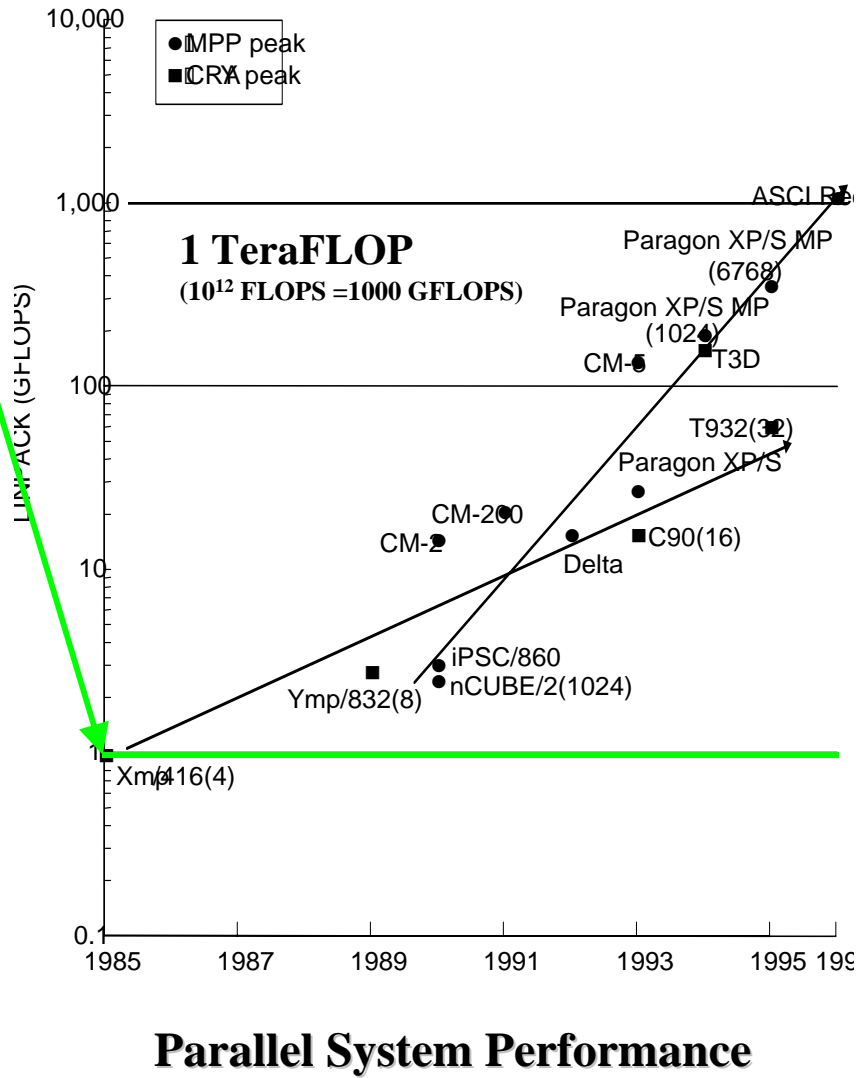
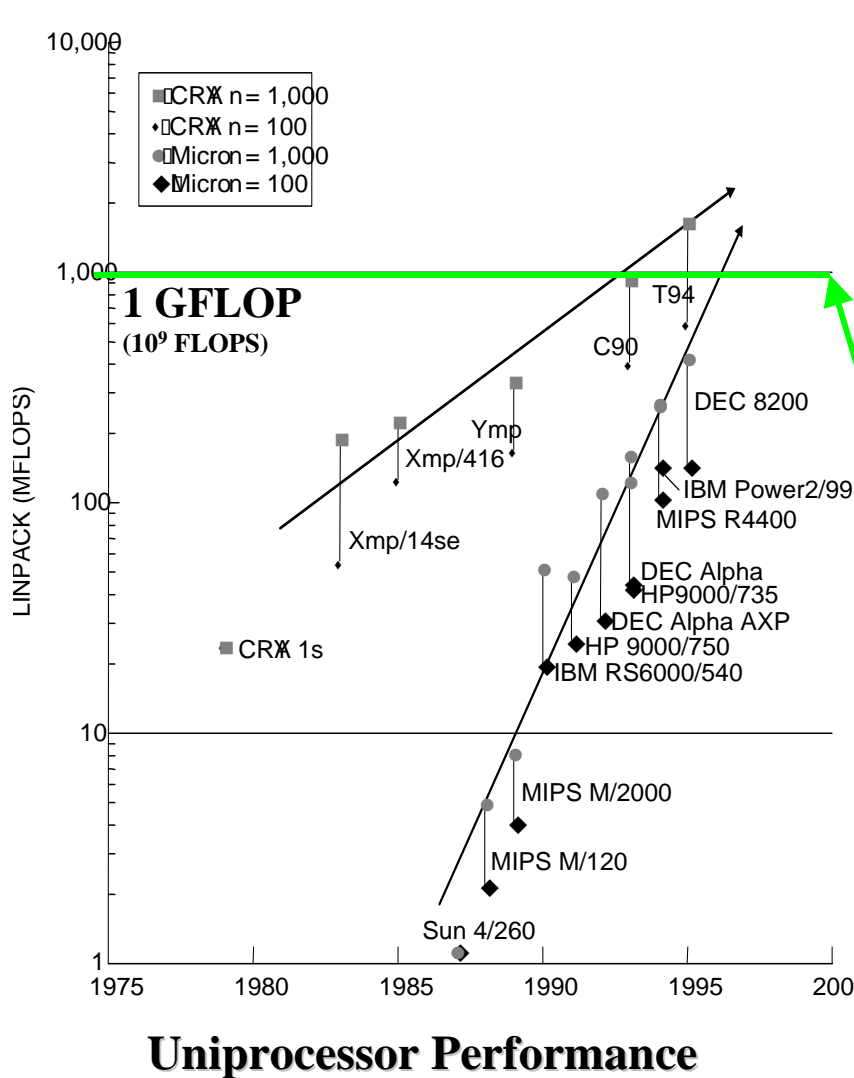


Current ranking of top 500 parallel supercomputers in the world is found at: www.top500.org

EECC756 - Shaaban

Why is Parallel Processing Needed?

LINPAK Performance Trends



EECC756 - Shaaban

Computer System Peak FLOP Rating History

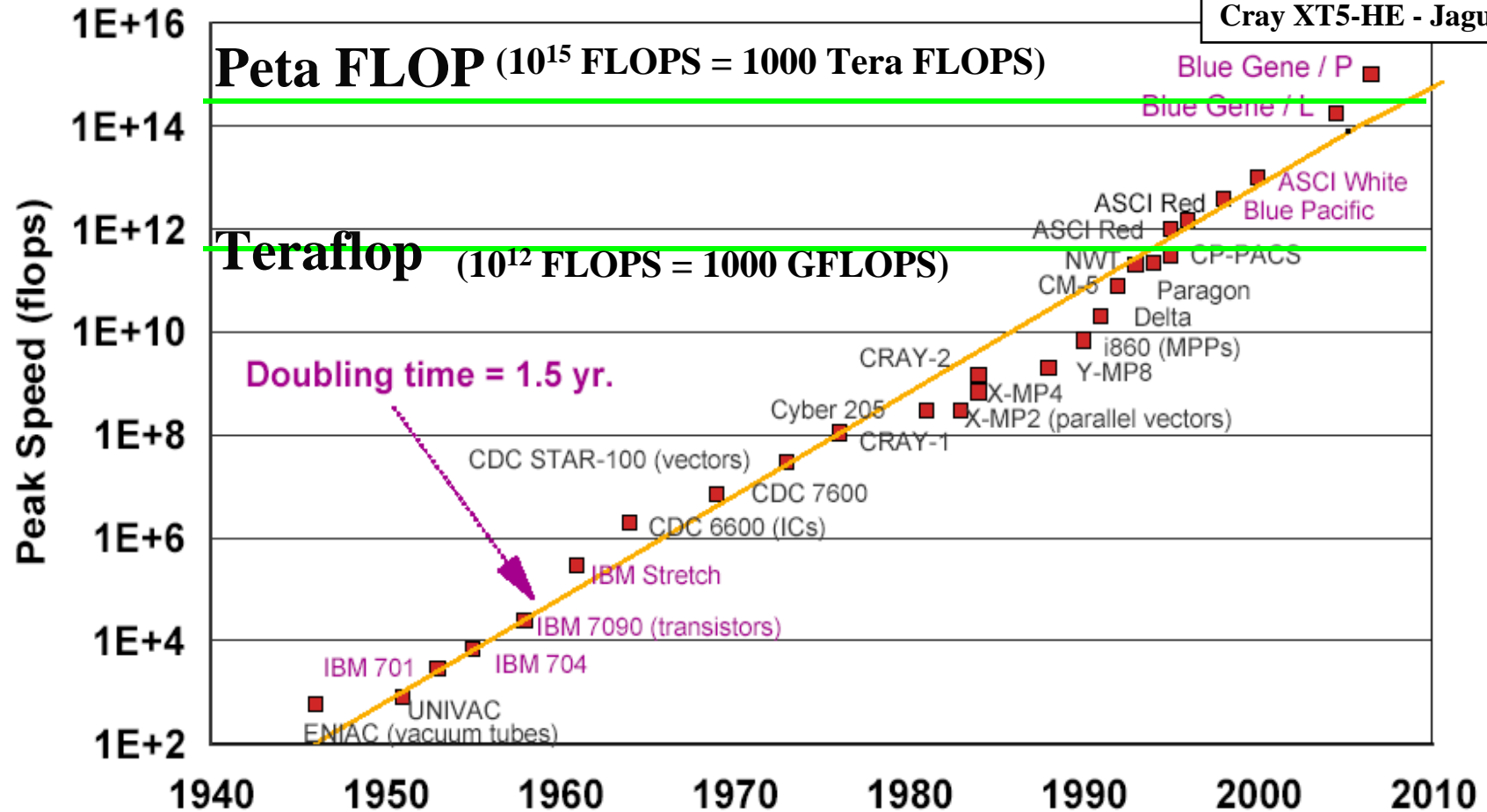
Current Top Peak FP Performance:

Now about 1,759,000 GFlop/s = 1,759 TeraFlops = 1.759 PetaFlops

Cray XT5-HE - Jaguar (@ Oak Ridge National Laboratory) 224,162 processor cores:

~ 37,361 AMD Opteron Six-Core Opteron 2.6 GHz

Since ~ Nov. 2009



Current ranking of top 500 parallel supercomputers in the world is found at: www.top500.org

EECC756 - Shaaban



26th List: The TOP10

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

Source (and for current list): www.top500.org

EECC756 - Shaaban

TOP500 Supercomputers**32nd List (November 2008): The Top 10**

Rank	Site	Computer/Year Vendor	Cores	R _{max}	R _{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2008 IBM	129600	1105.00	1456.70	2483.47
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT5 QC 2.3 GHz / 2008 Cray Inc.	150152	1059.00	1381.40	6950.60
3	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008 SGI	51200	487.01	608.83	2090.00
4	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
5	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	450.30	557.06	1260.00
6	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008 Sun Microsystems	62976	433.20	579.38	2000.00
7	NERSC/LBNL United States	Franklin - Cray XT4 QuadCore 2.3 GHz / 2008 Cray Inc.	38642	266.30	355.51	1150.00
8	Oak Ridge National Laboratory United States	Jaguar - Cray XT4 QuadCore 2.1 GHz / 2008 Cray Inc.	30976	205.00	260.20	1580.71
9	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, XT3/4, 2.4/2.2 GHz dual/quad core / 2008 Cray Inc.	38208	204.20	284.00	2506.00
10	Shanghai Supercomputer Center China	Dawning 5000A - Dawning 5000A, QC Opteron 1.9 Ghz, Infiniband, Windows HPC 2008 / 2008 Dawning	30720	180.60	233.47	

Source (and for current list): www.top500.org**EECC756 - Shaaban**

TOP500 Supercomputers**34nd List (November 2009): The Top 10**

Rank	Site	Computer/Year Vendor	Cores	R _{max}	R _{peak}	Power
1	Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
2	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50
3	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	98928	831.70	1028.85	
4	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70	2268.00
5	National SuperComputer Center in Tianjin/NUDT China	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband / 2009 NUDT	71680	563.10	1206.19	
6	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0 GHz/Nehalem EP 2.93 Ghz / 2009 SGI	56320	544.30	673.26	2348.00
7	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
8	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	458.61	557.06	1260.00
9	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008 Sun Microsystems	62976	433.20	579.38	2000.00
10	Sandia National Laboratories / National Renewable Energy Laboratory United States	Red Sky - Sun Blade x6275, Xeon X55xx 2.93 Ghz, Infiniband / 2009 Sun Microsystems	41616	423.90	487.74	

Source (and for current list): www.top500.org**EECC756 - Shaaban**

The Goal of Parallel Processing

- Goal of applications in using parallel machines:

Maximize Speedup over single processor performance

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

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

Fixed Problem Size Parallel Speedup

Parallel Speedup, Speedup_p

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

- Ideal speedup = number of processors = p

Very hard to achieve

+ load imbalance

Due to parallelization overheads: communication cost, dependencies ...

EECC756 - Shaaban

The Goal of Parallel Processing

- Parallel processing goal is to maximize parallel speedup:

Fixed Problem Size Parallel Speedup

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

Or time
← Parallelization overheads →

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.

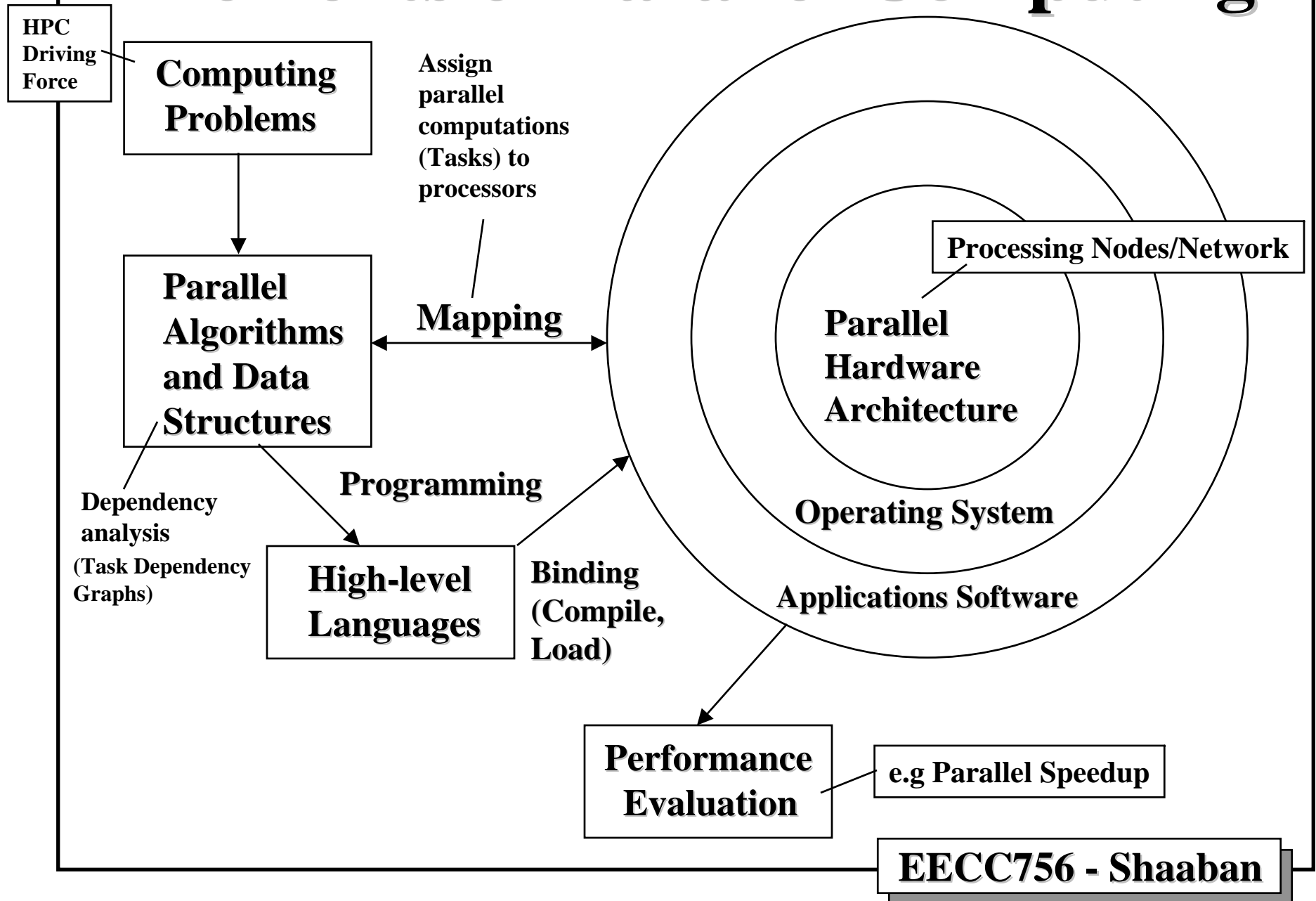
- Maximize parallel speedup by:

- ➔ 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



Elements of Parallel Computing

1 Computing Problems:

Driving
Force

- **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

Computing power

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

4 Operating Systems

Communication/connectivity

- 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/libraries
- User interfaces and tools.

6 Compiler Support

Approaches to parallel programming

(a) – Implicit Parallelism Approach

- Parallelizing compiler: Can automatically detect parallelism in sequential source code and transforms it into parallel constructs/code.
- Source code written in conventional sequential languages

(b) – Explicit Parallelism Approach:

- Programmer explicitly specifies parallelism using:
 - Sequential compiler (conventional sequential HLL) and low-level library of the target parallel computer , or ..
 - Concurrent (parallel) HLL .
- Concurrency Preserving Compiler: The compiler in this case preserves the parallelism explicitly specified by the programmer. It may perform some program flow analysis, dependence checking, limited optimizations for parallelism detection.

Illustrated next →

EECC756 - Shaaban

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

Compiler automatically detects parallelism in sequential source code and transforms it into parallel constructs/code

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

Programmer explicitly specifies parallelism using parallel constructs

EECC756 - Shaaban

Factors Affecting Parallel System Performance

- **Parallel Algorithm Related:**

i.e Inherent
Parallelism

- Available concurrency and profile, grain size, uniformity, patterns.
 - Dependencies between computations represented by dependency graph
- Type of parallelism present: Functional and/or data parallelism.
- Required communication/synchronization, uniformity and patterns.
- Data size requirements.
- Communication to computation ratio (C-to-C ratio, lower is better).

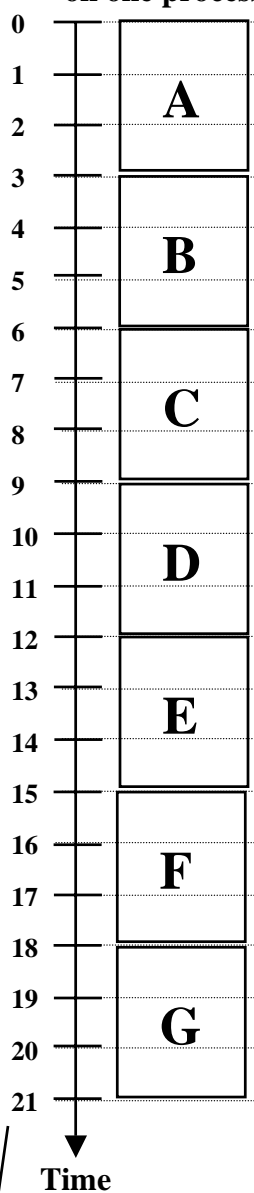
- **Parallel program Related:**

- Programming model used.
- Resulting data/code memory requirements, locality and working set characteristics.
- Parallel task grain size.
- Assignment (mapping) of tasks to processors: Dynamic or static.
- Cost of communication/synchronization primitives.

- **Hardware/Architecture related:**

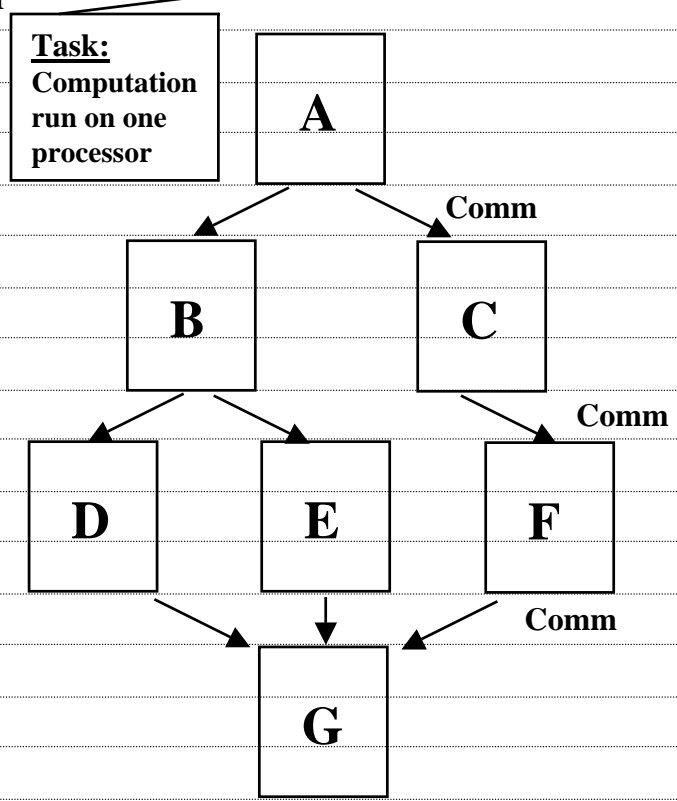
- Total CPU computational power available.
- Types of computation modes supported.
- Shared address space Vs. message passing.
- Communication network characteristics (topology, bandwidth, latency)
- Memory hierarchy properties.

Sequential Execution on one processor



$T_1 = 21$

Task Dependency Graph

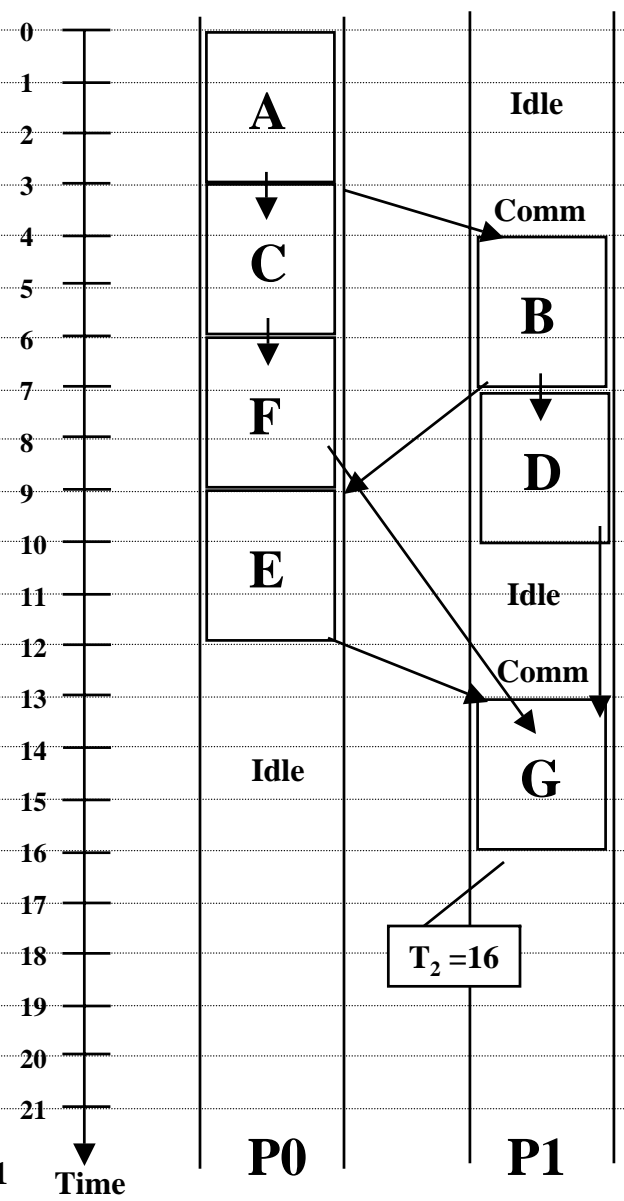


Task:
Computation run on one processor

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 = $T_1/T_2 = 21/16 = 1.3$

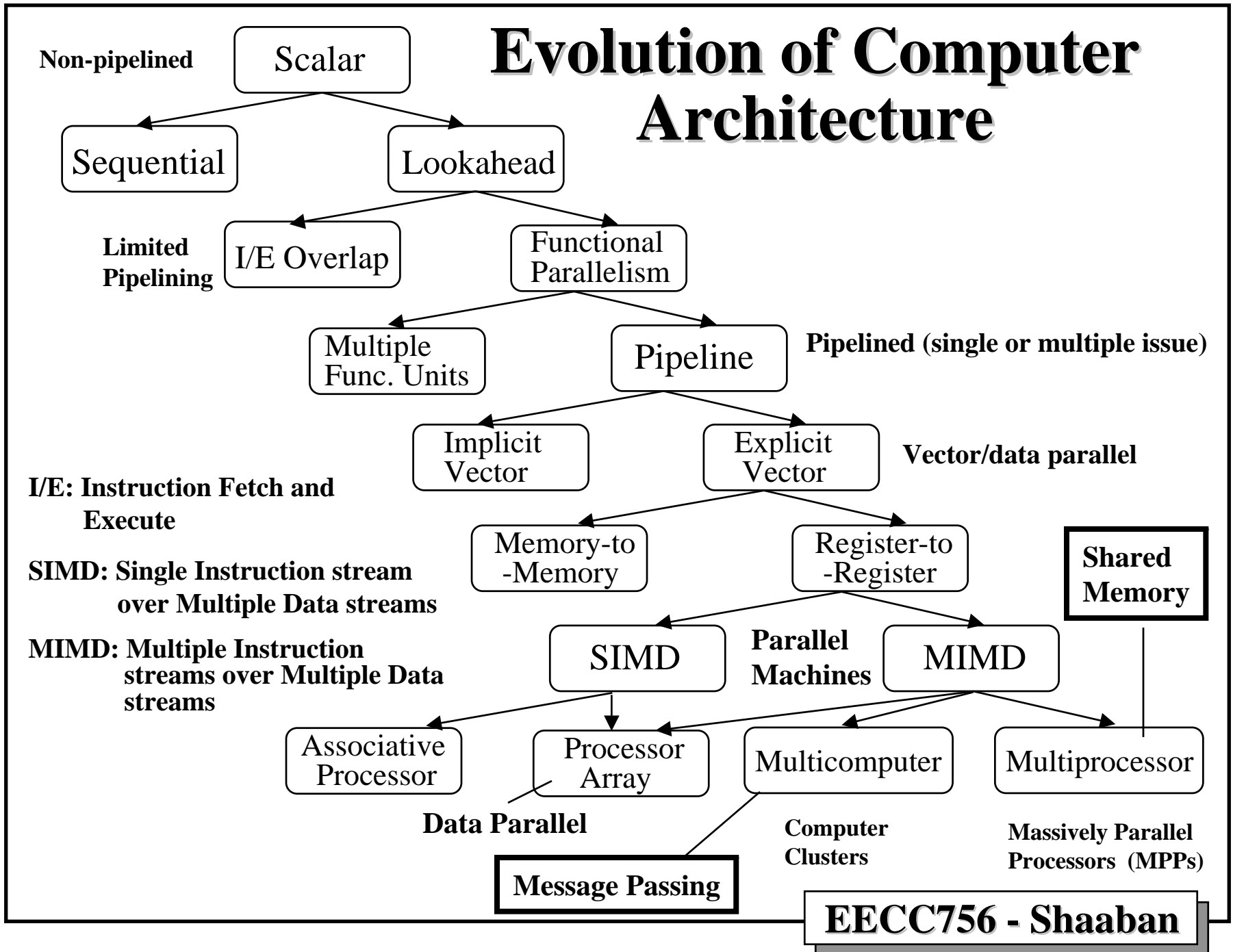
Possible Parallel Execution Schedule on Two Processors P0, P1



$T_2 = 16$

A simple parallel execution example

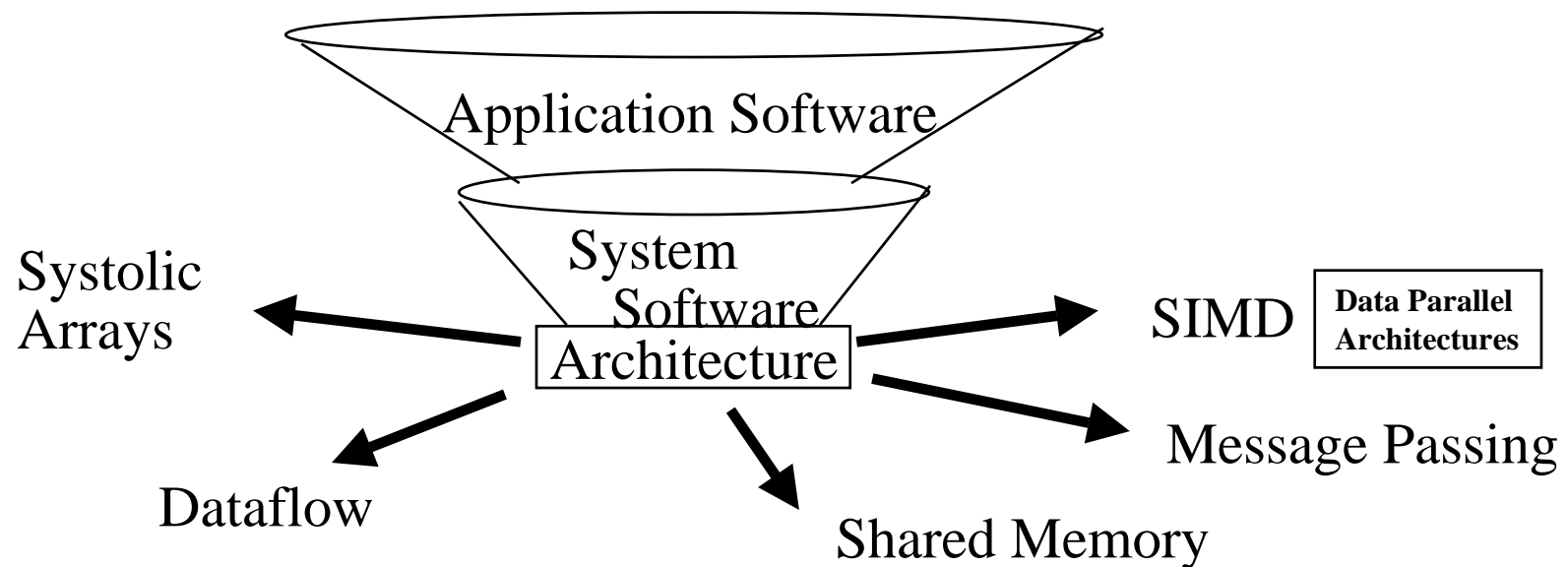
Evolution of Computer Architecture



Parallel Architectures History

Historically, parallel architectures were tied to parallel programming models:

- Divergent architectures, with no predictable pattern of growth.



More on this next lecture

EECC756 - Shaaban

Parallel Programming Models

- Programming methodology used in coding parallel applications
- Specifies: 1- communication and 2- synchronization

- Examples:

However, a good way to utilize multi-core processors for the masses!

- 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 implicitly using a shared memory address space (shared data in memory).
- Message passing:
Explicit point to point communication (via send/receive pairs) 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

(Taxonomy)

Instruction Stream = Thread of Control or Hardware Context

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

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

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

(d) **Multiple Instruction streams over Multiple Data streams (MIMD):** Parallel computers:

Tightly coupled processors

- Shared memory multiprocessors.

Loosely coupled processors

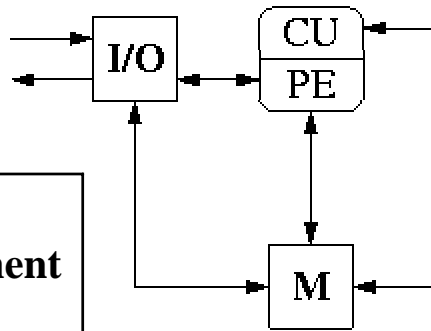
- Multicomputers: Unshared distributed memory, message-passing used instead (e.g clusters)

Classified according to number of instruction streams (threads) and number of data streams in architecture

EECC756 - Shaaban

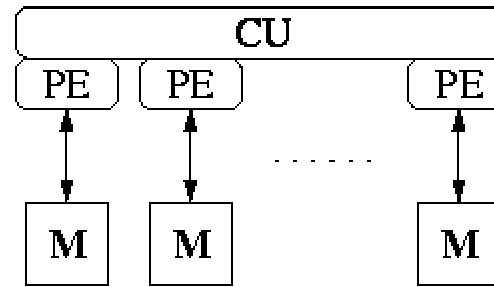
Flynn's Classification of Computer Architecture (Taxonomy)

Uniprocessor



(a) SISD

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

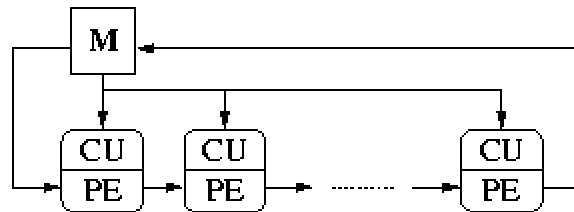


(b) SIMD

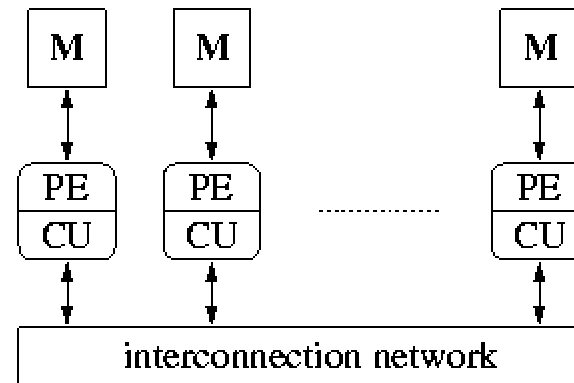
Shown here:
array of synchronized
processing elements

CU = Control Unit
PE = Processing Element
M = Memory

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



(c) MISD



(d) MIMD

Parallel computers
or multiprocessor systems

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

Classified according to number of instruction streams
(threads) and number of data streams in architecture

EECC756 - Shaaban

Current Trends In Parallel Architectures

Conventional or sequential

- The extension of “computer architecture” to support communication and cooperation:

- OLD: Instruction Set Architecture (ISA)
- NEW: *Communication Architecture*

- Defines:

- 1 – Critical abstractions, boundaries, and primitives (interfaces).
- 2 – Organizational structures that implement interfaces (hardware or software)

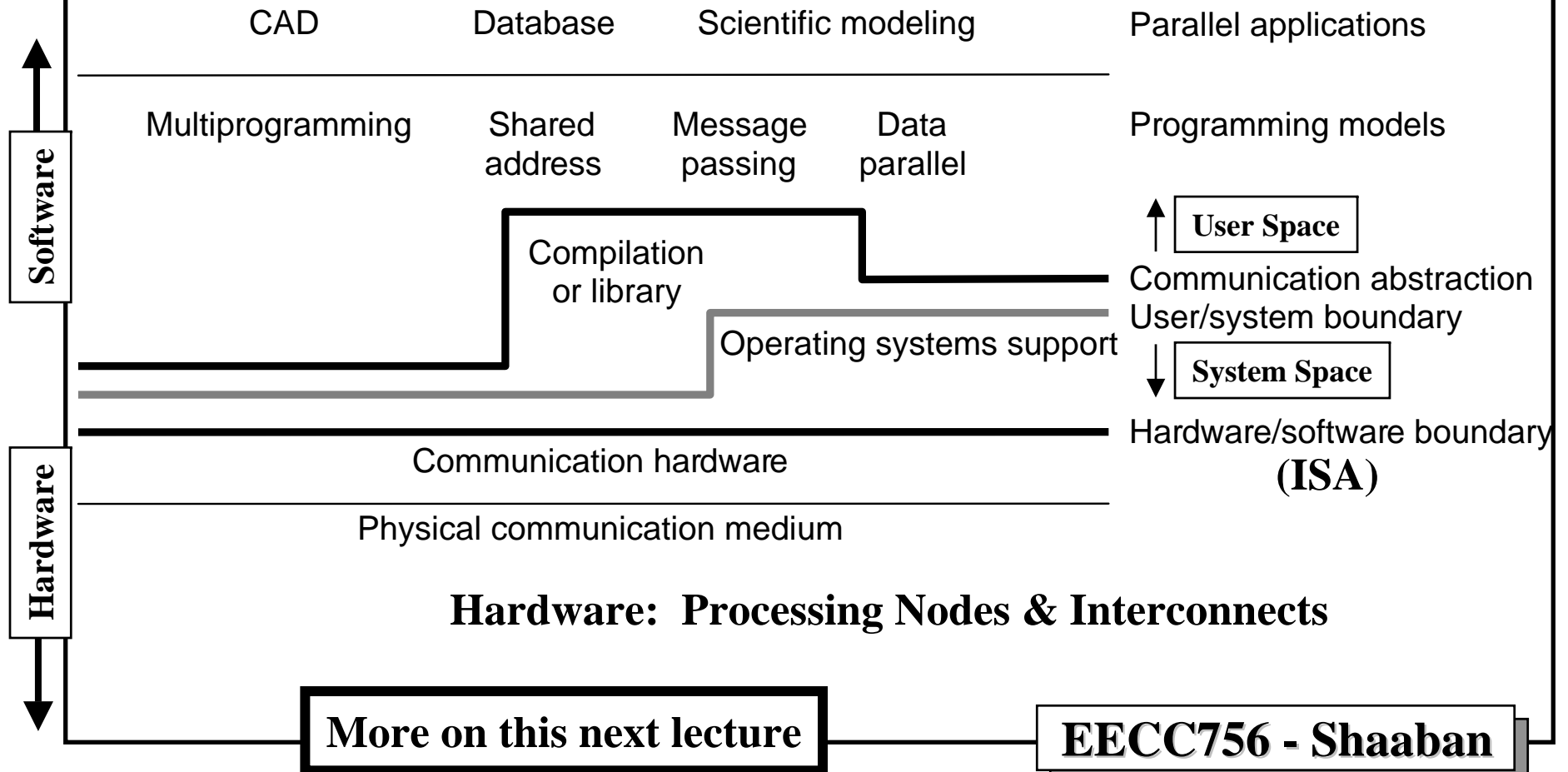
Implementation of Interfaces

- Compilers, libraries and OS are important bridges today

More on this next lecture

EECC756 - Shaaban

Modern Parallel Architecture Layered Framework



Shared Address Space (SAS) Parallel Architectures

(in shared address space)

- Any processor can directly reference any memory location
 - Communication occurs implicitly as result of loads and stores
- Convenient: Communication is implicit via loads/stores
 - Location transparency
 - Similar programming model to time-sharing in uniprocessors
 - Except processes run on different processors
 - Good throughput on multiprogrammed workloads i.e multi-tasking
- 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

EECC756 - Shaaban

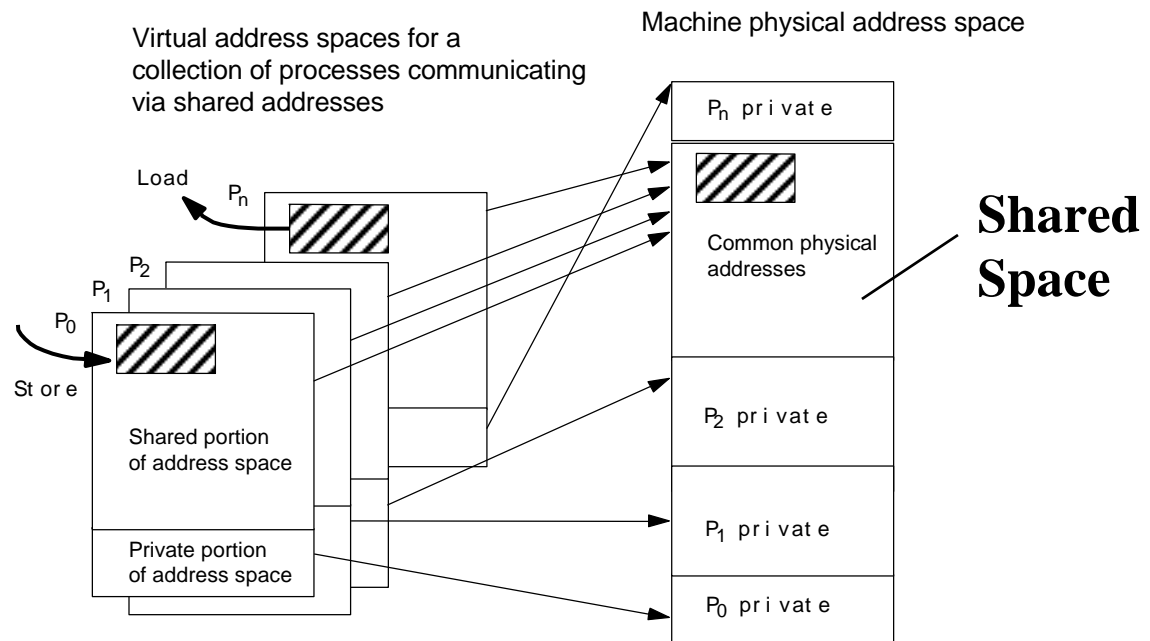
Shared Address Space (SAS) Parallel Programming Model

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

In SAS:

Communication is implicit
via loads/stores.

Ordering/Synchronization
is explicit using synchronization
Primitives.



- Writes to shared address visible to other threads (in other processes too)

- Natural extension of the uniprocessor model:

Thus communication is implicit via loads/stores

- Conventional memory operations used for communication

- Special atomic operations needed for synchronization:

i.e for event ordering and mutual exclusion

- Using Locks, Semaphores etc.

Thus synchronization is explicit

- OS uses shared memory to coordinate processes.

EECC756 - Shaaban

Models of Shared-Memory Multiprocessors

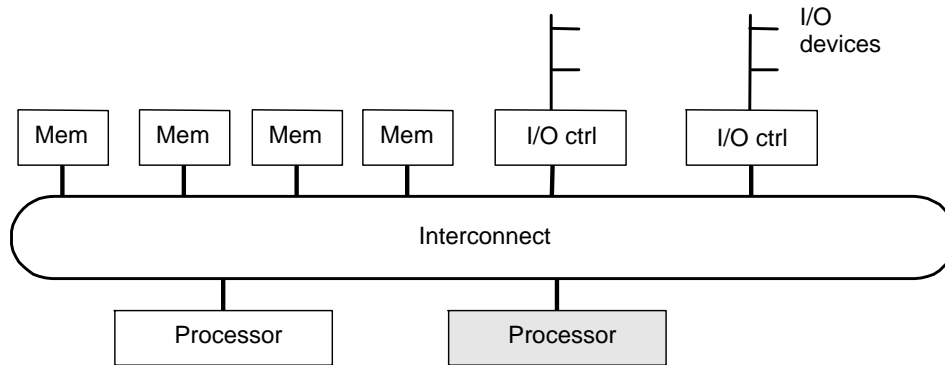
- 1 • **The Uniform Memory Access (UMA) Model:**
 - All physical memory is shared by all processors.
 - All processors have equal access (i.e equal memory bandwidth and access latency) to all memory addresses.
 - Also referred to as Symmetric Memory Processors (SMPs).
- 2 • **Distributed memory or Non-uniform Memory Access (NUMA) Model:**
 - Shared memory is physically distributed locally among processors. Access latency to remote memory is higher.
- 3 • **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

1

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

UMA



Interconnect:

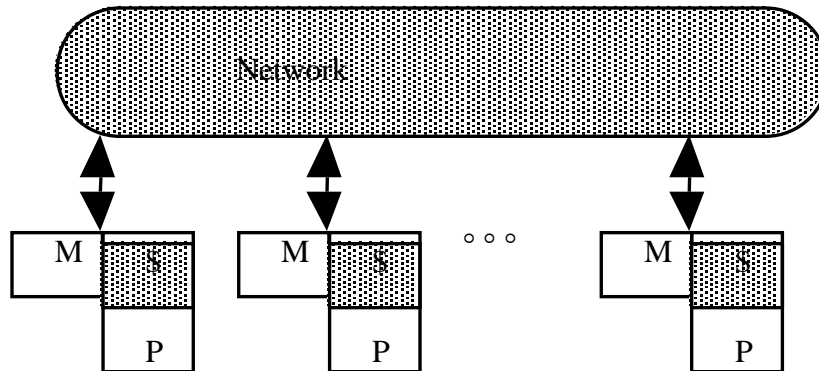
Bus, Crossbar, Multistage network

P: Processor

M or Mem: Memory

C: Cache

D: Cache directory

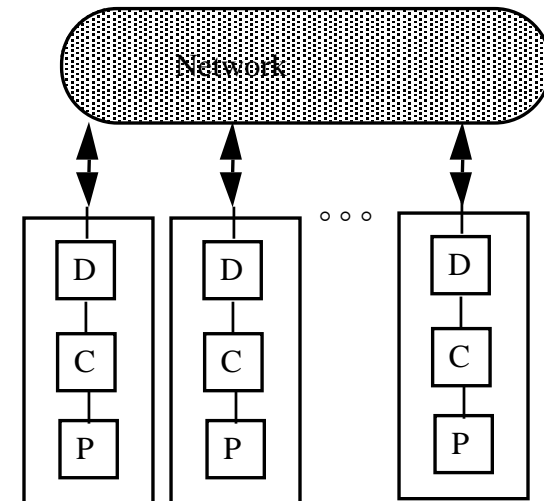


NUMA

2

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

3



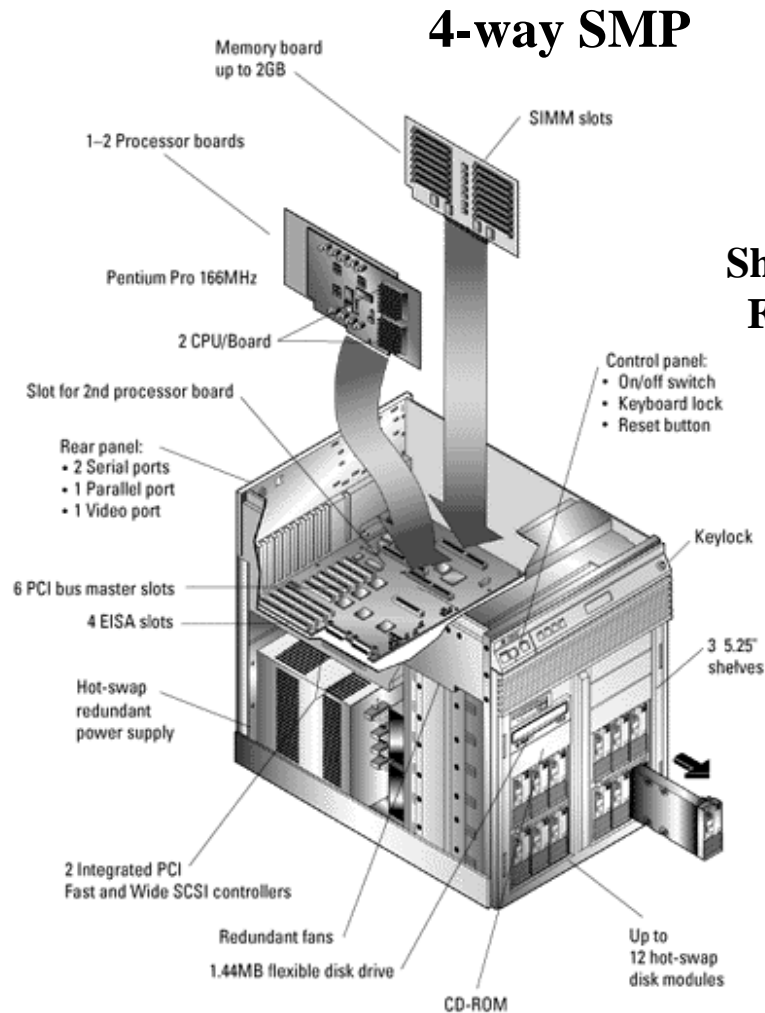
Cache-Only Memory Architecture (COMA)

EECC756 - Shaaban

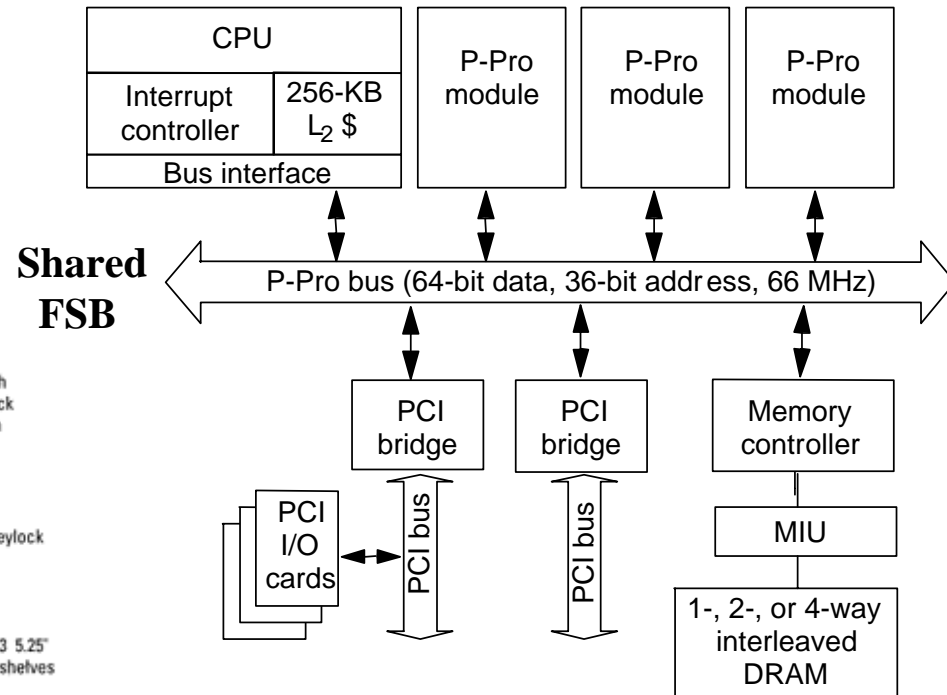
Uniform Memory Access (UMA)

Example: Intel Pentium Pro Quad

Circa 1997



4-way SMP



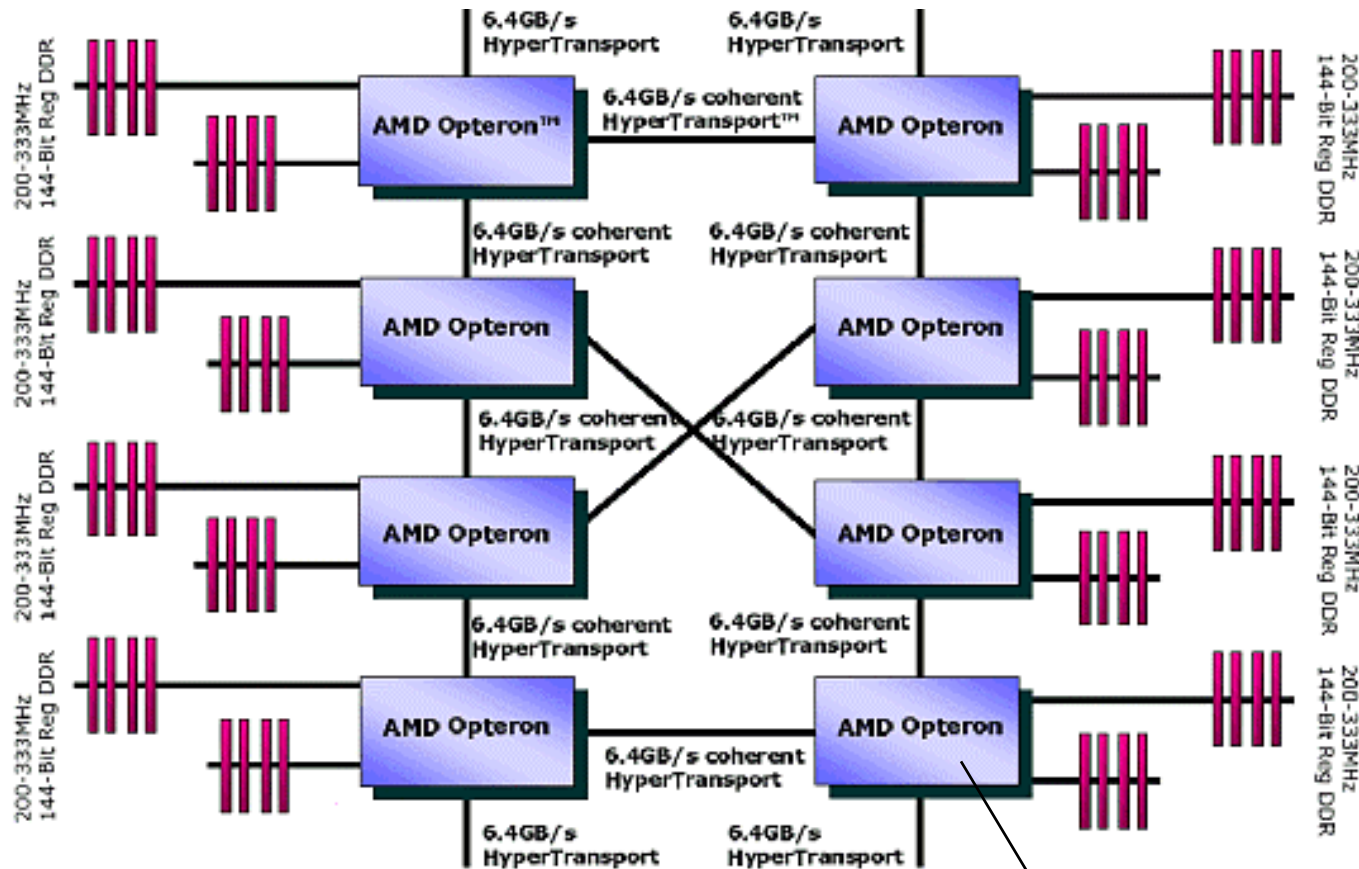
- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Computing node used in Intel's ASCI-Red MPP

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

Non-Uniform Memory Access (NUMA) Example: AMD 8-way Opteron Server Node



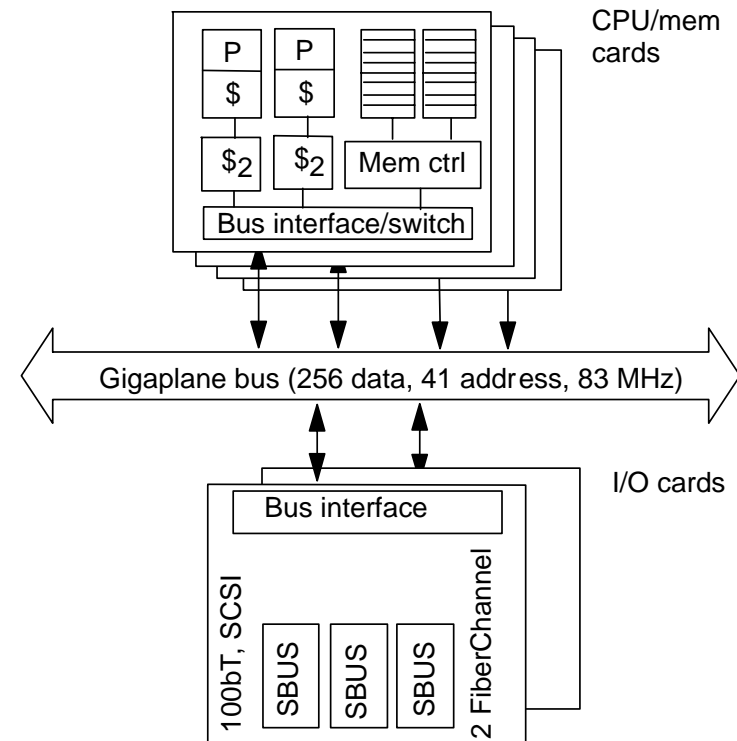
Circa 2003

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 (32 cores with quad core processors)

EECC756 - Shaaban

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:

NUMA MPP Example

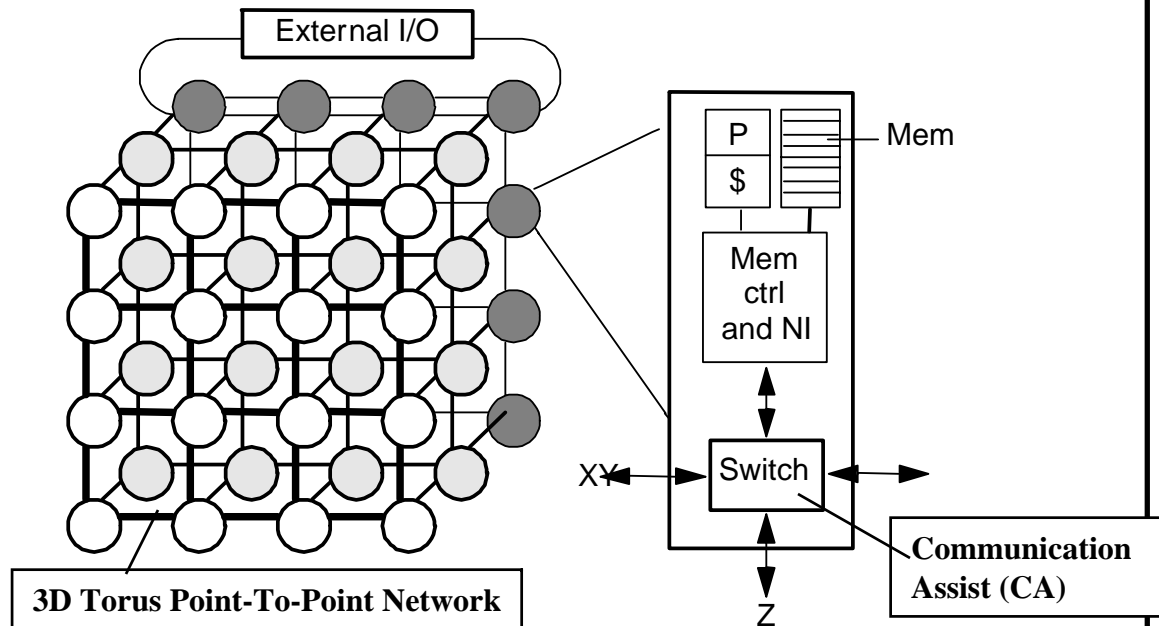
MPP = Massively Parallel Processor System



More recent Cray MPP Example:
Cray X1E Supercomputer

Cray T3E

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)

EECC756 - Shaaban

Message-Passing Multicomputers

- Comprised of multiple autonomous computers (computing 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 and Communication Assist (CA).
- Local memory is only accessible by local processors in a node (no shared memory among nodes).
- Inter-node communication is carried explicitly out by message passing through the connection network via send/receive operations.

Thus communication is explicit

- Process communication achieved using a message-passing programming environment (e.g. PVM, MPI).

Portable, platform-independent

- Programming model more removed or abstracted from basic hardware operations

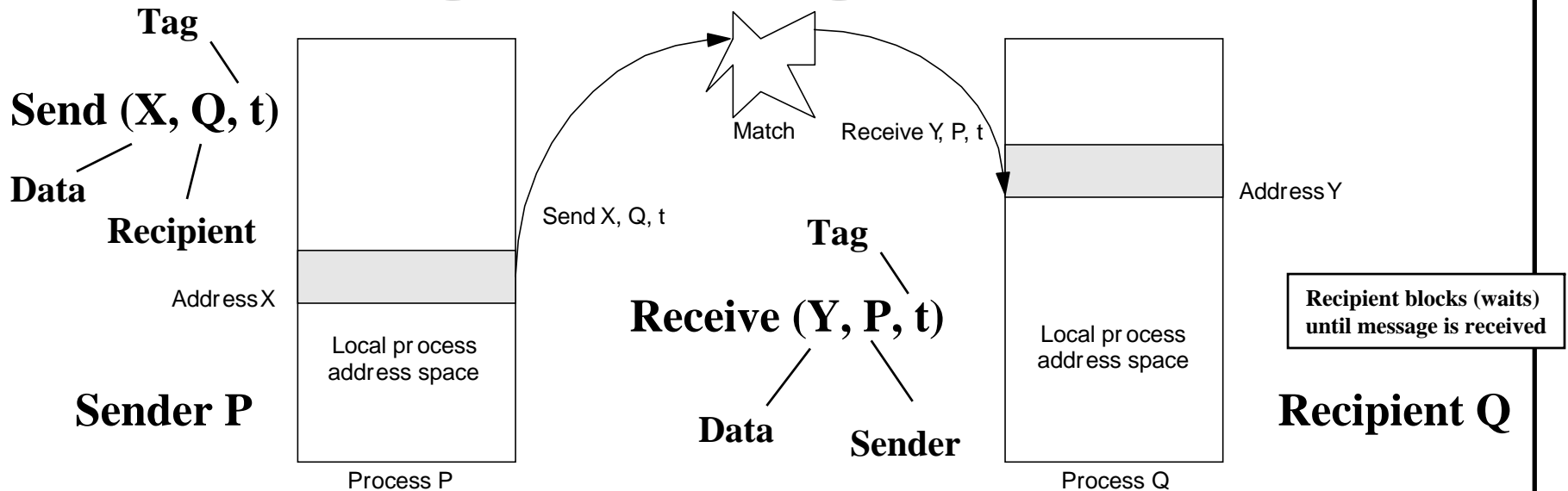
- Include:

- 1 – A number of commercial Massively Parallel Processor systems (MPPs).
- 2 – Computer clusters that utilize commodity of-the-shelf (COTS) components.

Also called Loosely-Coupled Parallel Computers

EECC756 - Shaaban

Message-Passing Abstraction

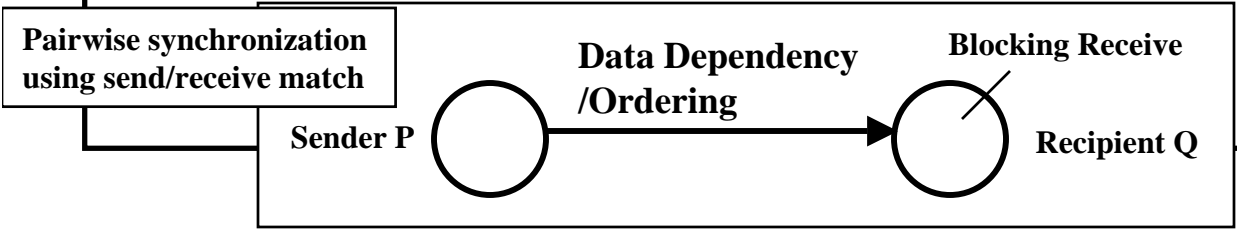


- Send specifies buffer to be transmitted and receiving process.
- Receive specifies sending process and application storage to receive into.
- Memory to memory copy possible, but need to name processes.
- Optional tag on send and matching rule on receive.
- User process names local data and entities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

Communication is explicit via sends/receives

i.e event ordering, in this case

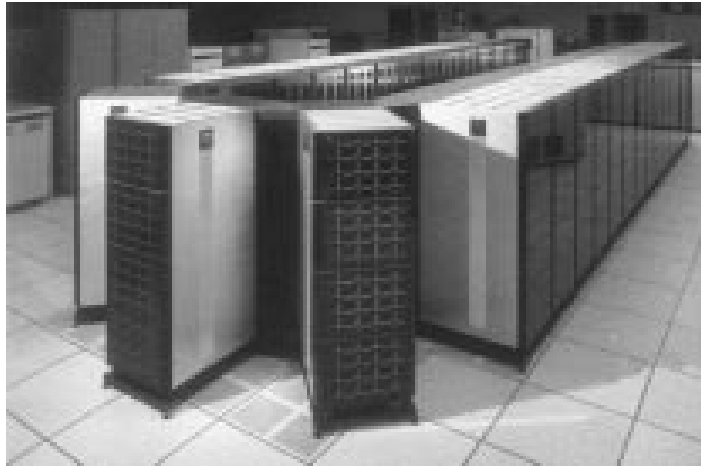
Synchronization is implicit



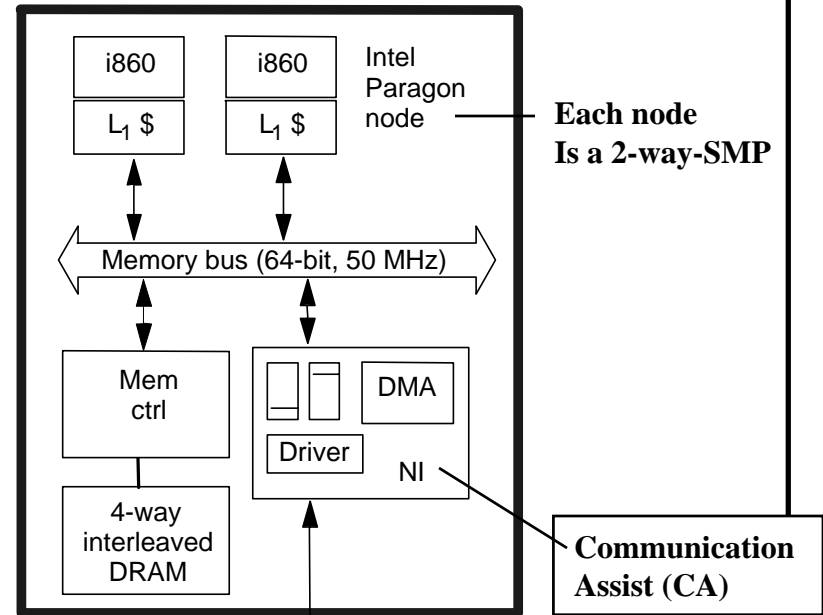
EECC756 - Shaaban

Message-Passing Example: Intel Paragon

Circa 1983

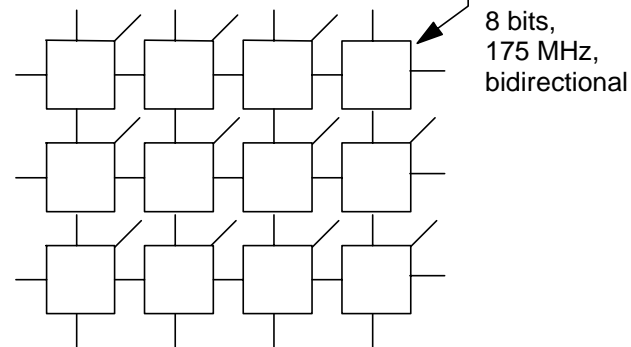


Sandia's Intel Paragon XP/S-based Supercomputer



2D grid network with processing node attached to every switch

2D grid point to point network



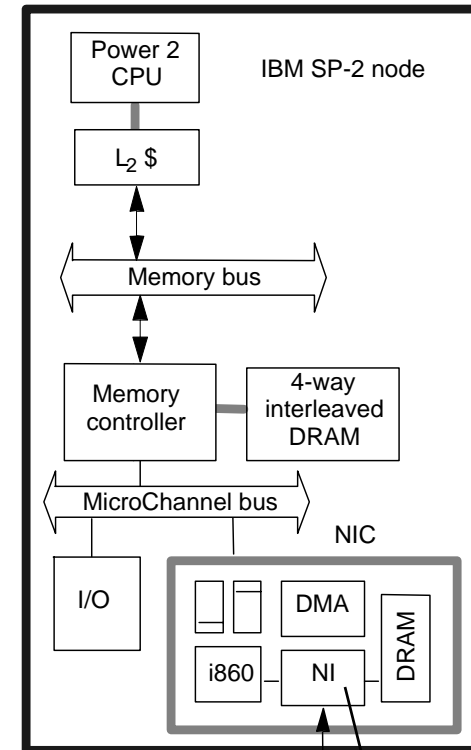
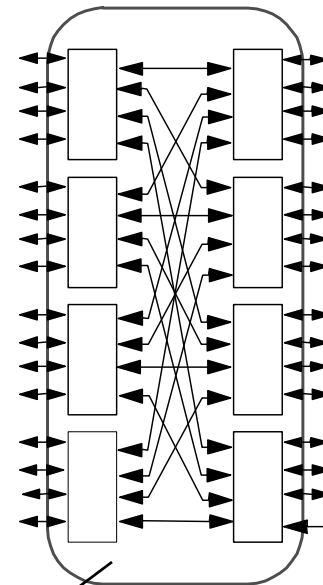
Message-Passing Example: IBM SP-2

MPP

Circa 1994-1998



General interconnection network formed from 8-port switches



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

Multi-stage network

EECC756 - Shaaban

MPP = Massively Parallel Processor System

Message-Passing MPP Example:

IBM Blue Gene/L

Circa 2005

(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

2.8 Gflops peak
per processor core

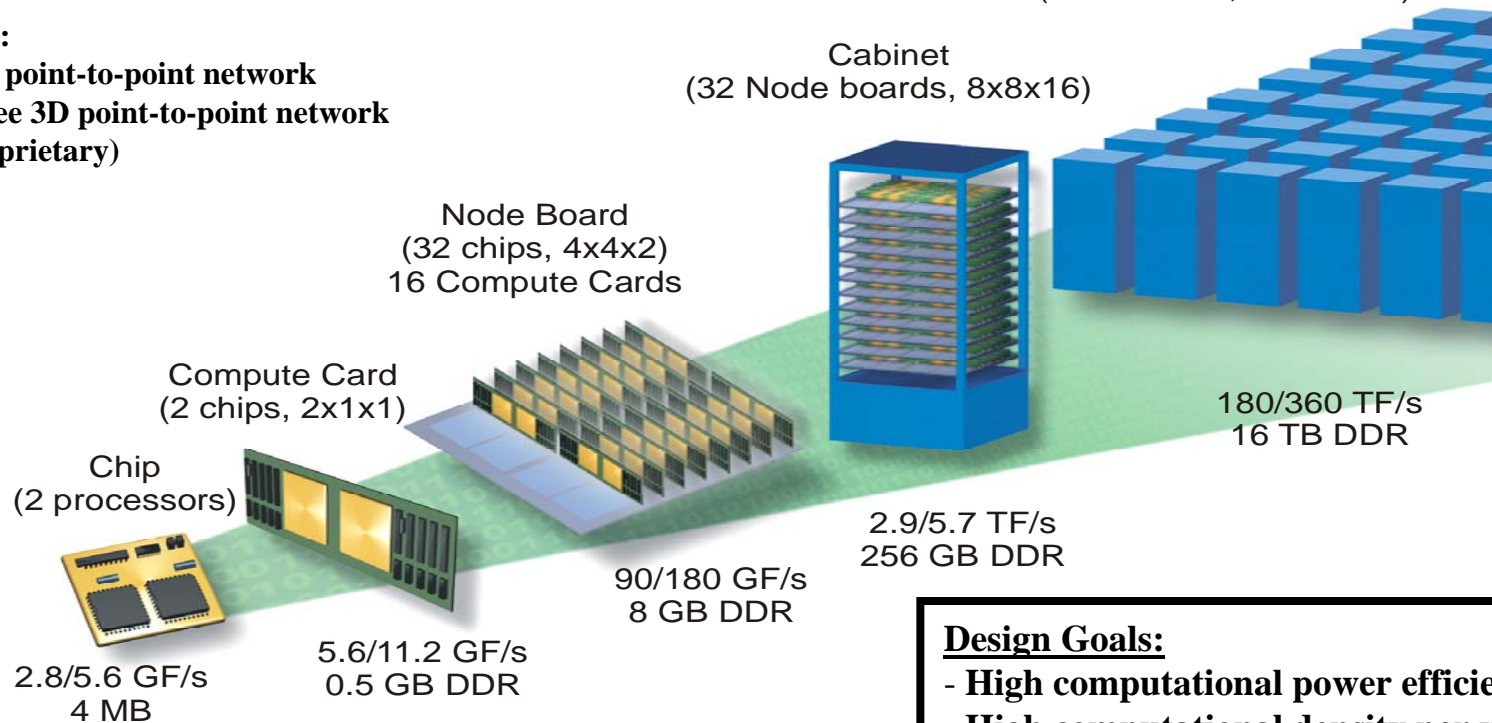
System
(64 cabinets, 64x32x32)

Networks:

3D Torus point-to-point network

Global tree 3D point-to-point network

(both proprietary)



LINPACK Performance:

280,600 GFLOPS = 280.6 TeraFLOPS = 0.2806 Peta FLOP

Top Peak FP Performance:

Now about 367,000 GFLOPS = 367 TeraFLOPS = 0.367 Peta FLOP

Design Goals:

- High computational power efficiency
- High computational density per volume

EECC756 - Shaaban

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 be 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.

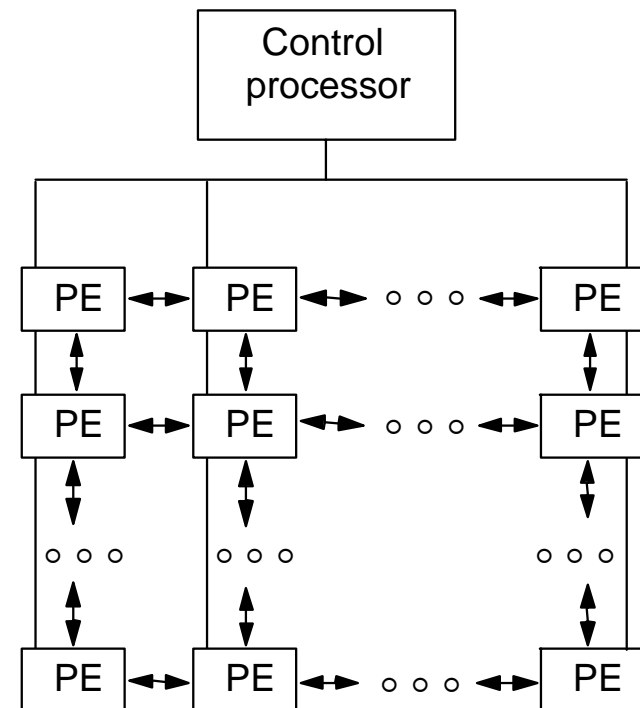
Both MPI & PVM are examples of the explicit parallelism approach to parallel programming

Both MPI and PVM are portable (platform-independent) and allow the user to explicitly specify parallelism

EECC756 - Shaaban

Data Parallel Systems SIMD in Flynn taxonomy

- Programming model (Data Parallel)
 - Similar 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 processors each with little memory
 - Processors don't sequence through instructions
 - Attached to a control processor that issues instructions
 - Specialized and general communication, global synchronization
- Example machines:
 - Thinking Machines CM-1, CM-2 (and CM-5)
 - Maspar MP-1 and MP-2,



All PE are synchronized
(same instruction or operation
in a given cycle)

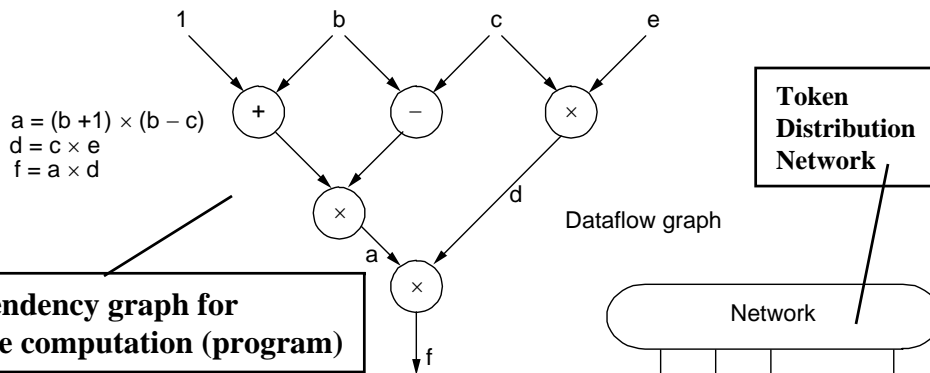
Other Data Parallel Architectures: Vector Machines

PE = Processing Element

EECC756 - Shaaban

Dataflow Architectures

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

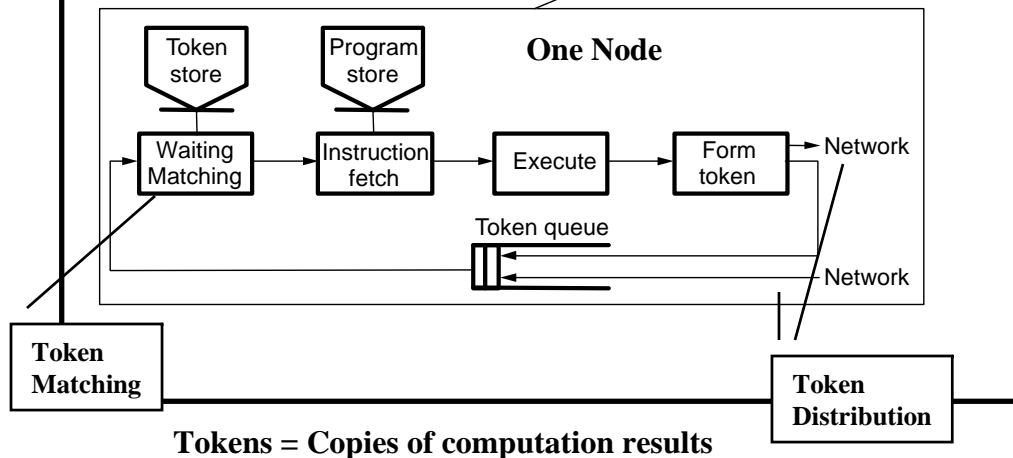


Research Dataflow machine prototypes include:

- The MIT Tagged Architecture
- The Manchester Dataflow Machine

The Tomasulo approach of dynamic instruction execution utilizes dataflow driven execution engine:

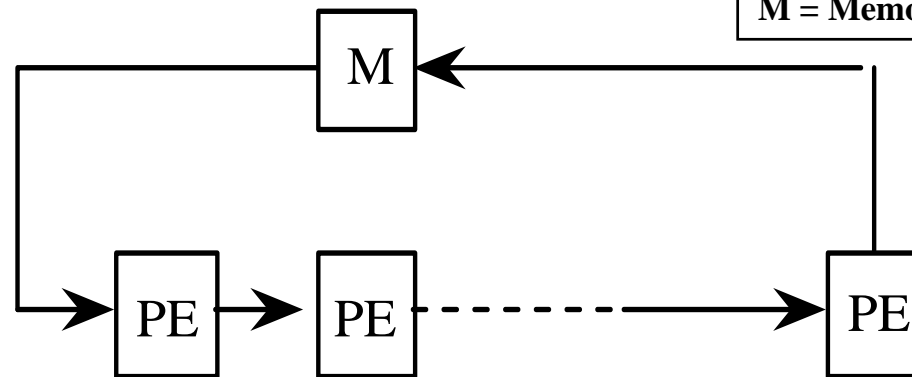
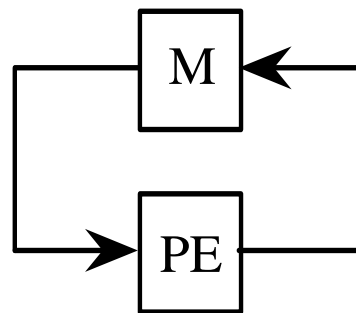
- The data dependency graph for a small window of instructions is constructed dynamically when instructions are issued in order of the program.
- The execution of an issued instruction is triggered by the availability of its operands (data it needs) over the CDB.



Example of Flynn's Taxonomy's MISD (Multiple Instruction Streams Single Data Stream):

Systolic Architectures

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



PE = Processing Element
M = Memory

- Different from linear 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

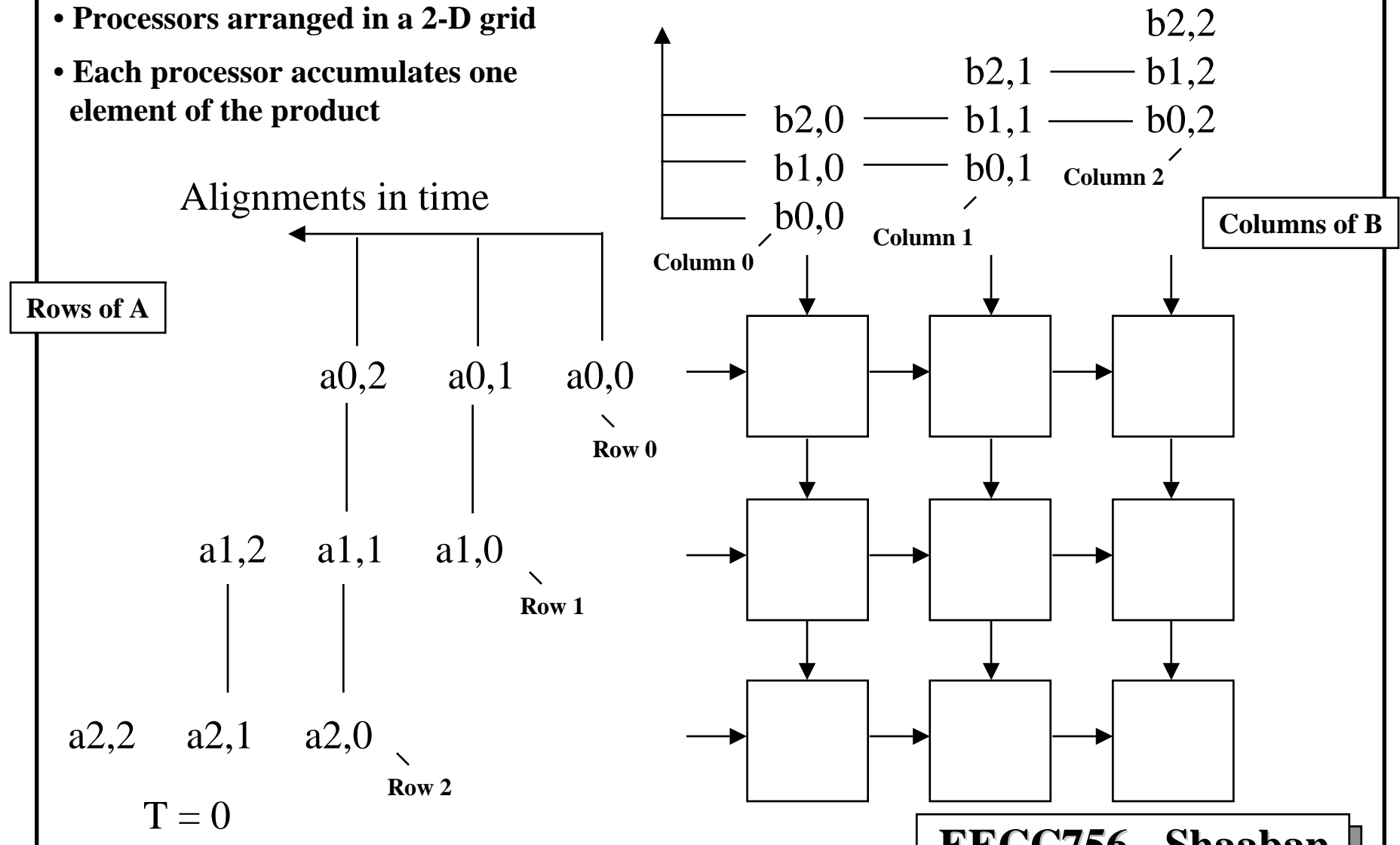
EECC756 - Shaaban

Systolic Array Example:

$$C = A \times B$$

3x3 Systolic Array Matrix Multiplication

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



EECC756 - Shaaban

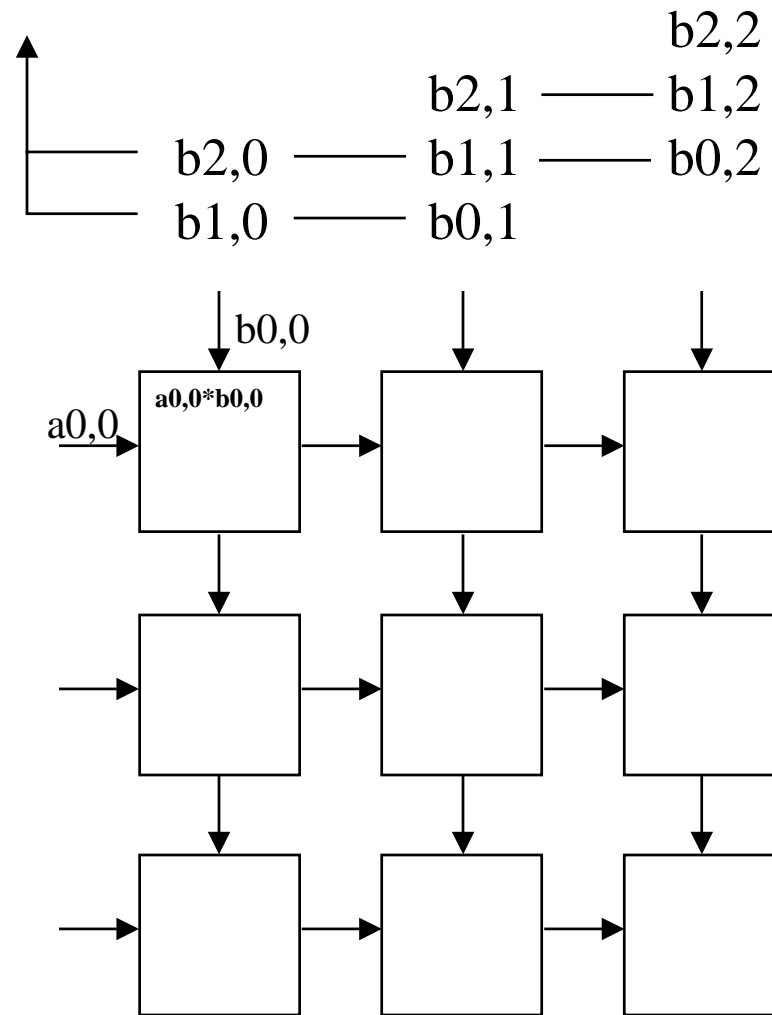
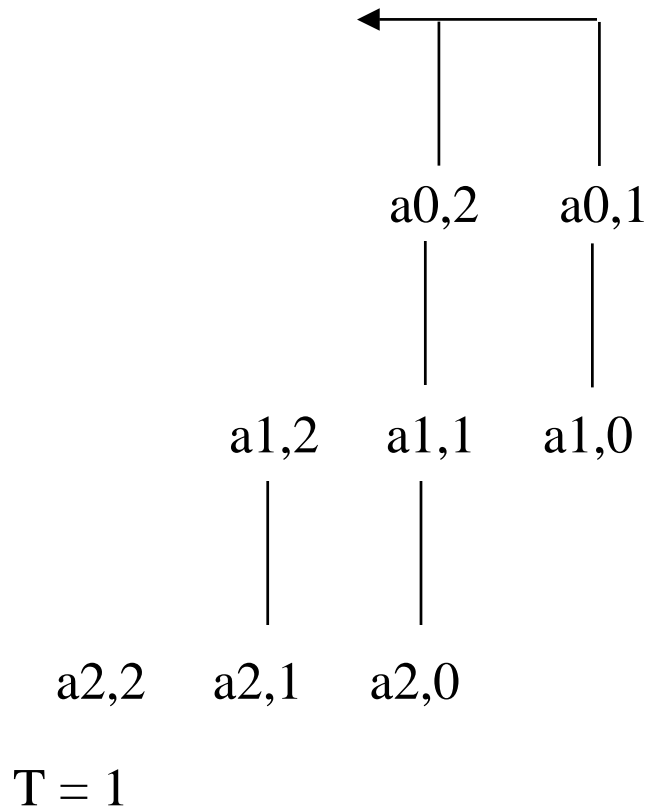
Example source: <http://www.cs.hmc.edu/courses/2001/spring/cs156/>

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

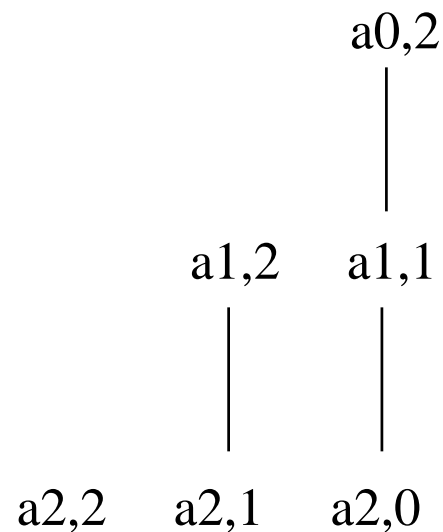


Systolic Array Example:

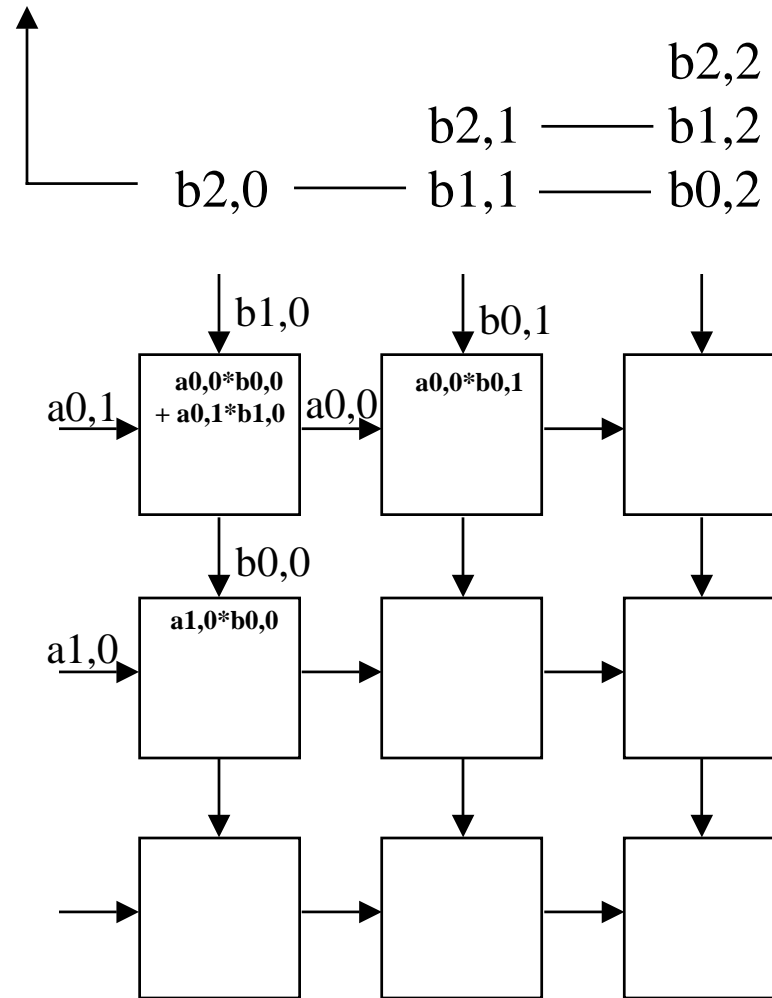
3x3 Systolic Array Matrix Multiplication

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

Alignments in time



$T = 2$



EECC756 - Shaaban

Example source: <http://www.cs.hmc.edu/courses/2001/spring/cs156/>

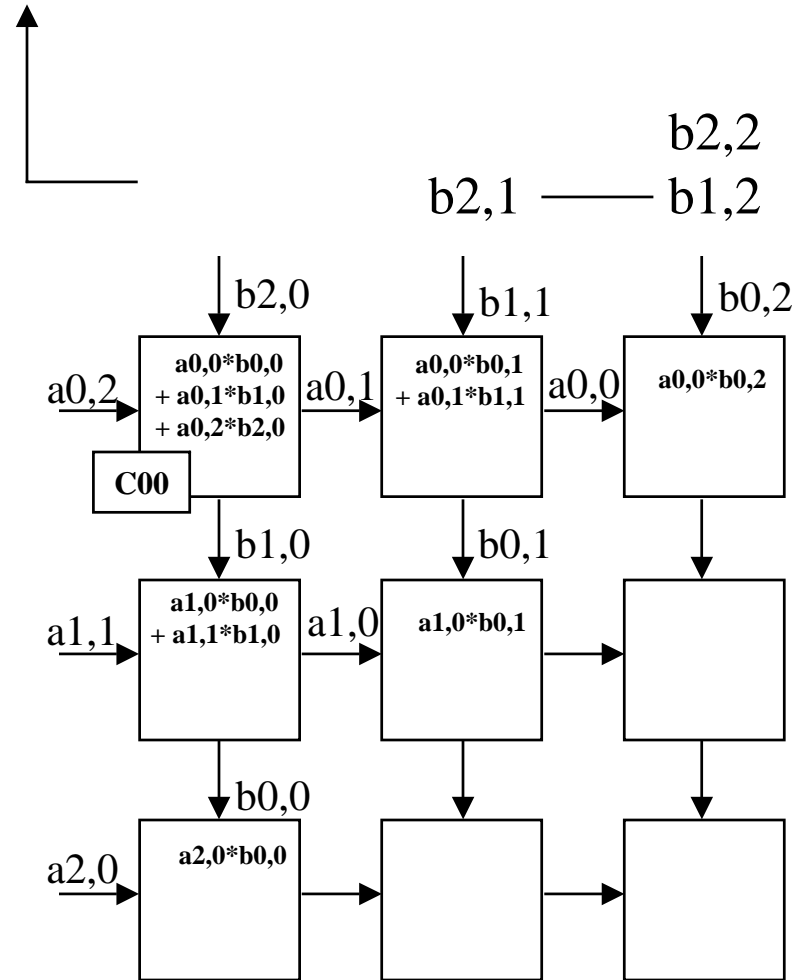
#56 lec # 1 Spring 2010 3-9-2010

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



$T = 3$

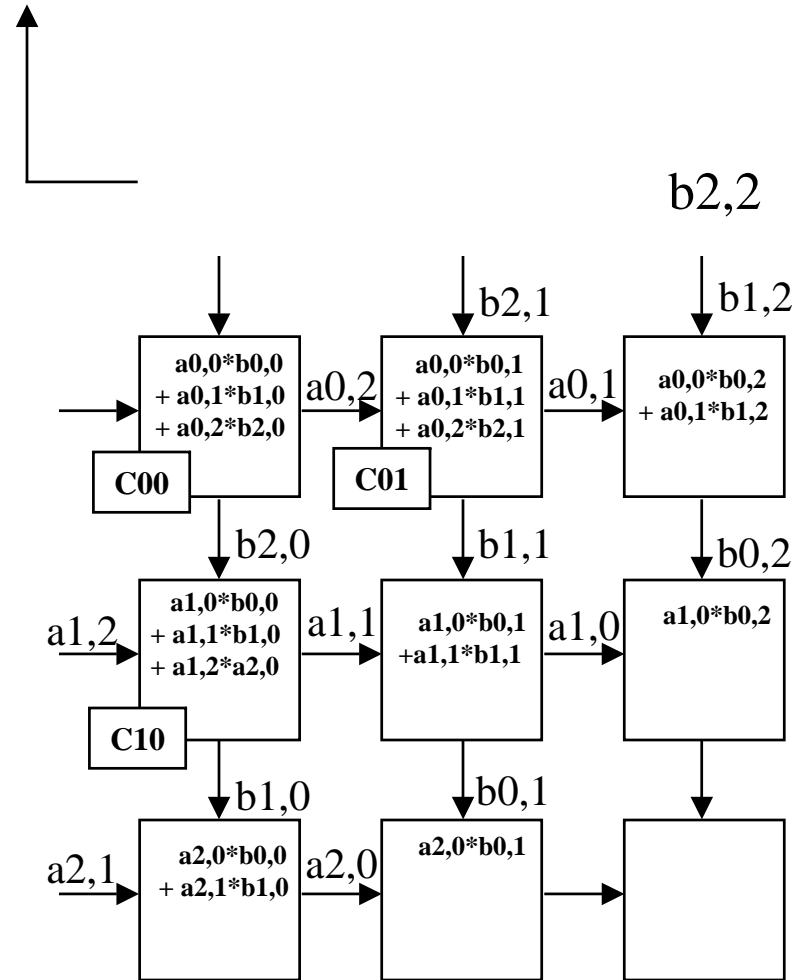
EECC756 - Shaaban

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



$T = 4$

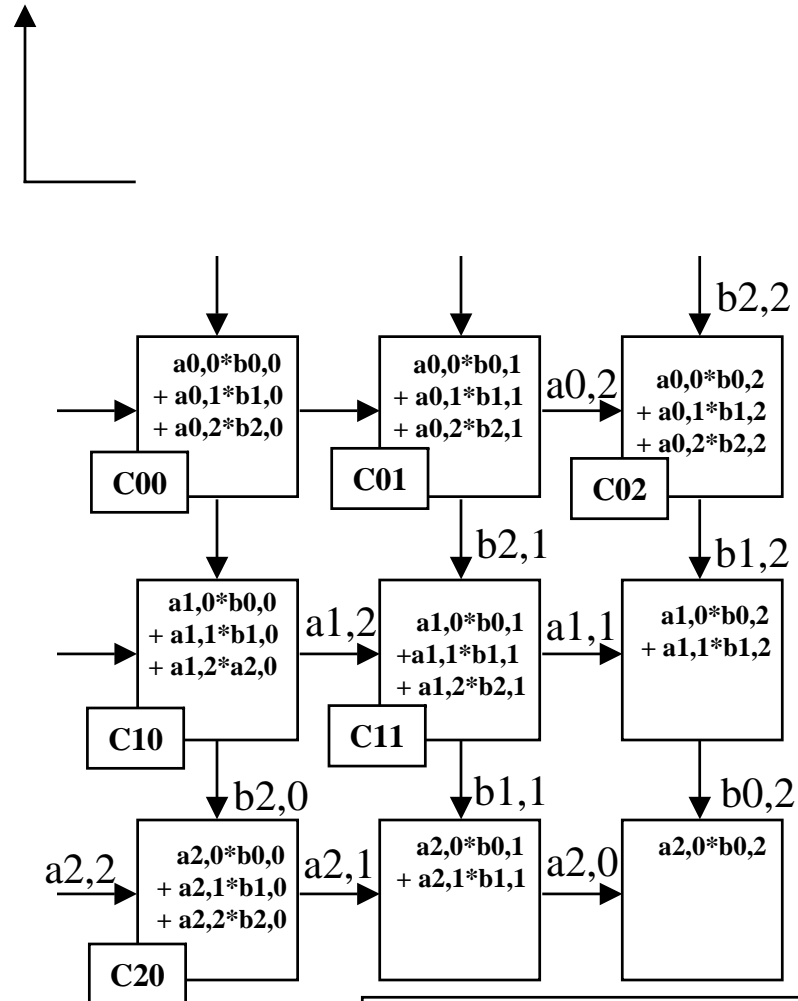
EECC756 - Shaaban

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

EECC756 - Shaaban

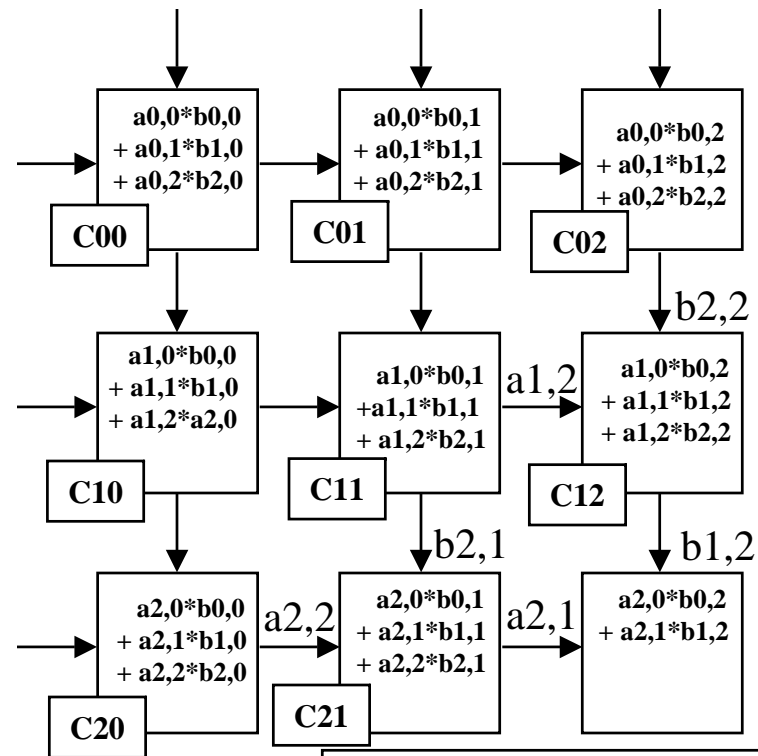
Example source: <http://www.cs.hmc.edu/courses/2001/spring/cs156/>

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



$T = 6$

EECC756 - Shaaban

Example source: <http://www.cs.hmc.edu/courses/2001/spring/cs156/>

#60 lec # 1 Spring 2010 3-9-2010

Systolic Array Example:

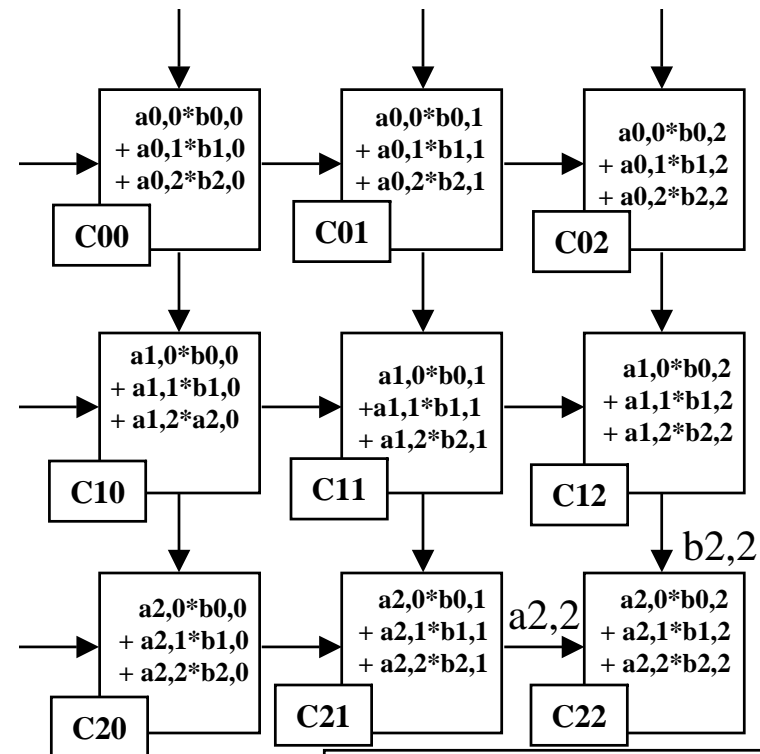
3x3 Systolic Array Matrix Multiplication

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

Alignments in time

Done

$T = 7$



EECC756 - Shaaban