Introduction to Parallel Processing Architecture

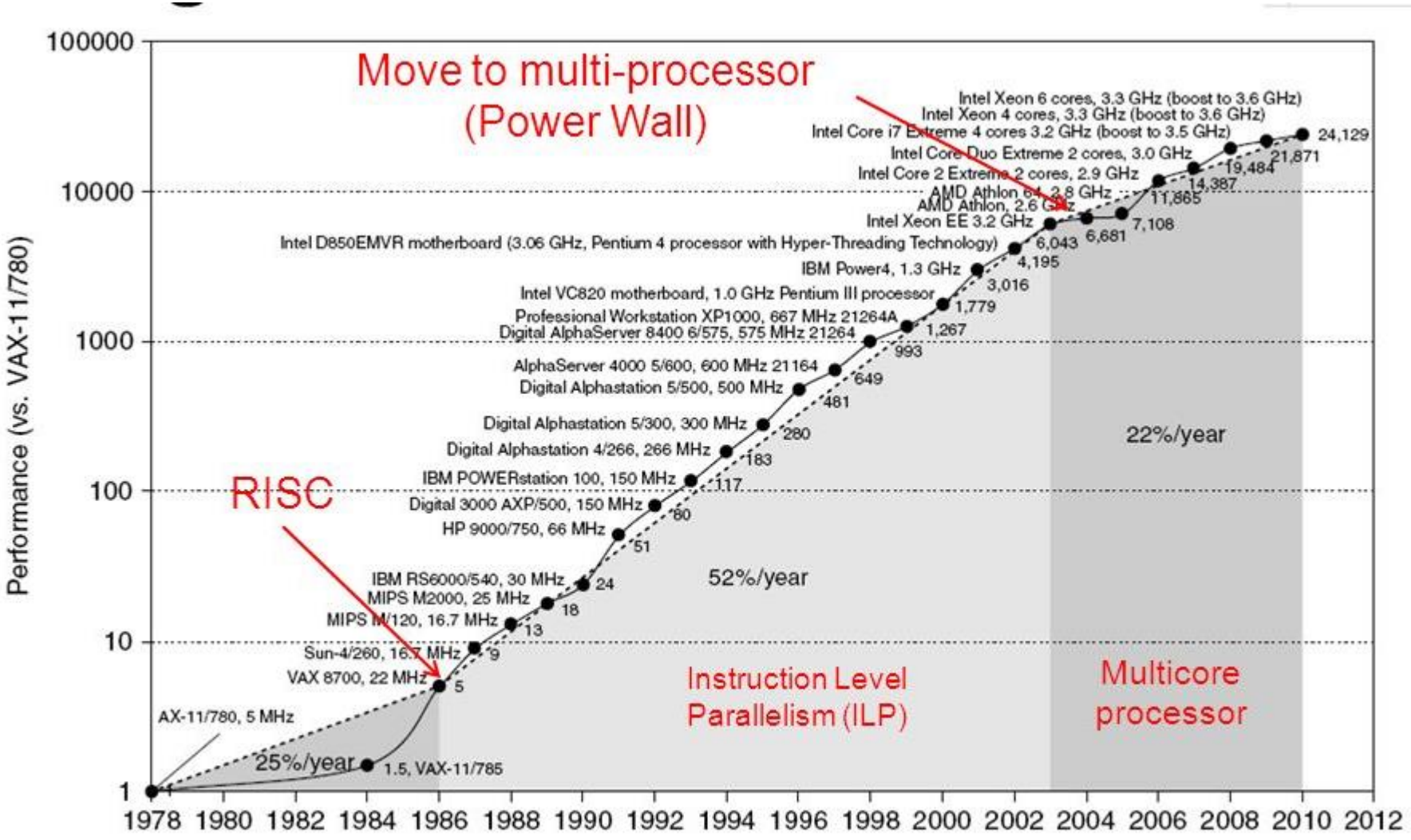
Introduction

- Performance improvements:
- Improvements in semiconductor technology
  - Feature size, clock speed
- Improvements in computer architectures
  - Enabled by HLL compilers, UNIX
  - Lead to RISC architectures
- Together have enabled:
  - Lightweight computers
  - Productivity-based managed/interpreted programming languages
  - SaaS, Virtualization, Cloud
- Applications evolution:
  - Speech, sound, images, video, “augmented/extended reality”, “big data”

Moore's Law

- **Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.**

Single Processor Performance



- So, we need Parallel Computing!

- **Current Trends in Architecture**
- Cannot continue to leverage Instruction-Level parallelism (ILP)
- Single processor performance improvement ended in 2003

- New model for performance: **Parallelism**
- These require explicit restructuring of the application

Level of Parallelism

- Bit level parallelism: 1970 to ~1985
  - 4 bits, 8 bit, 16 bit, 32 bit microprocessors
- Instruction level parallelism: ~1985 - today
  - Pipelining
  - Superscalar
  - VLIW
  - Out-of-order execution / Dynamic Instr. Scheduling
- Process level or thread level parallelism
  - Servers are parallel
  - Desktop dual processor PCs
  - Multicore architectures (CPUs, GPUs)

Parallelism

- Classes of parallelism in applications:
- Data-Level Parallelism (DLP)
- Task-Level Parallelism (TLP)

- Methods of exploiting architectural parallelism by hardware:
- Instruction-Level Parallelism (ILP)
- Vector architectures/Graphic Processor Units (GPUs)
- Thread-Level Parallelism
- Request-Level Parallelism

CPU Design

Reduced instruction set computer (RISC) architecture

- RISC-CPUs developed for high-performance computers and now used broadly.
- It increases the arithmetic speed of the CPU by decreasing the number of instructions the CPU.

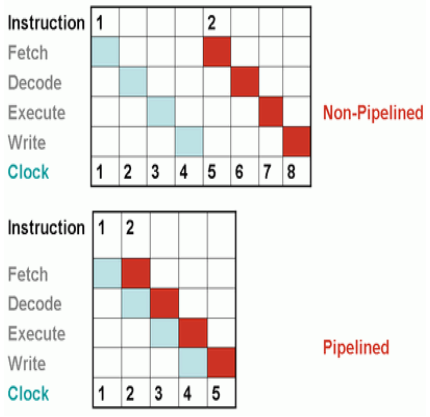
Complex instruction set computer (CISC), architecture

- Pipelining: concurrent preparation of several instructions that are then executed successively.
- High level compilers, to improve performance.

CPU Design

Reduced instruction set computer (RISC) architecture

- **Pipelining**
- An instruction can be executed while the next one is being decoded and the next one is being fetched.
- It overlaps various stages of instruction execution to achieve performance
- The speed of a pipeline is eventually limited by the slowest stage.



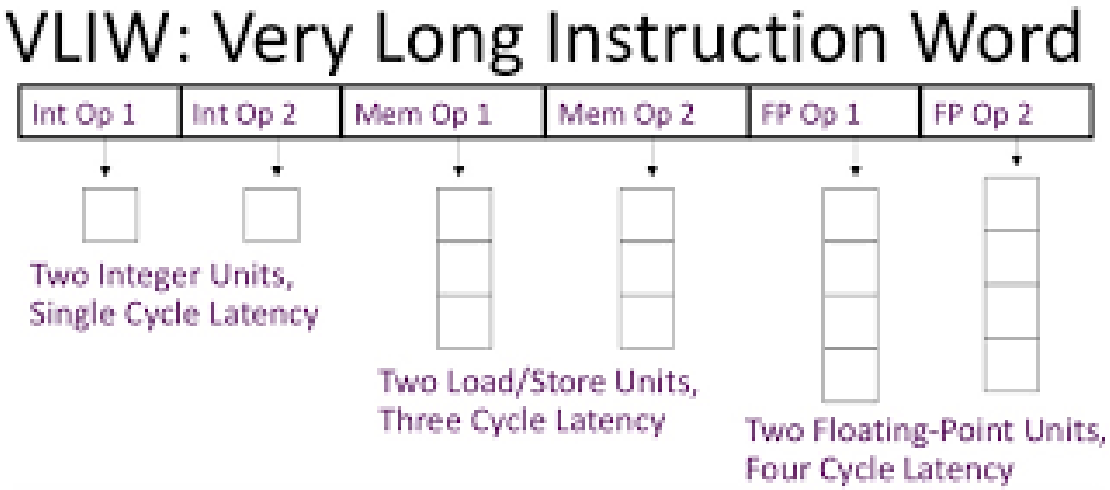
Multiple pipelines.

CPU Design

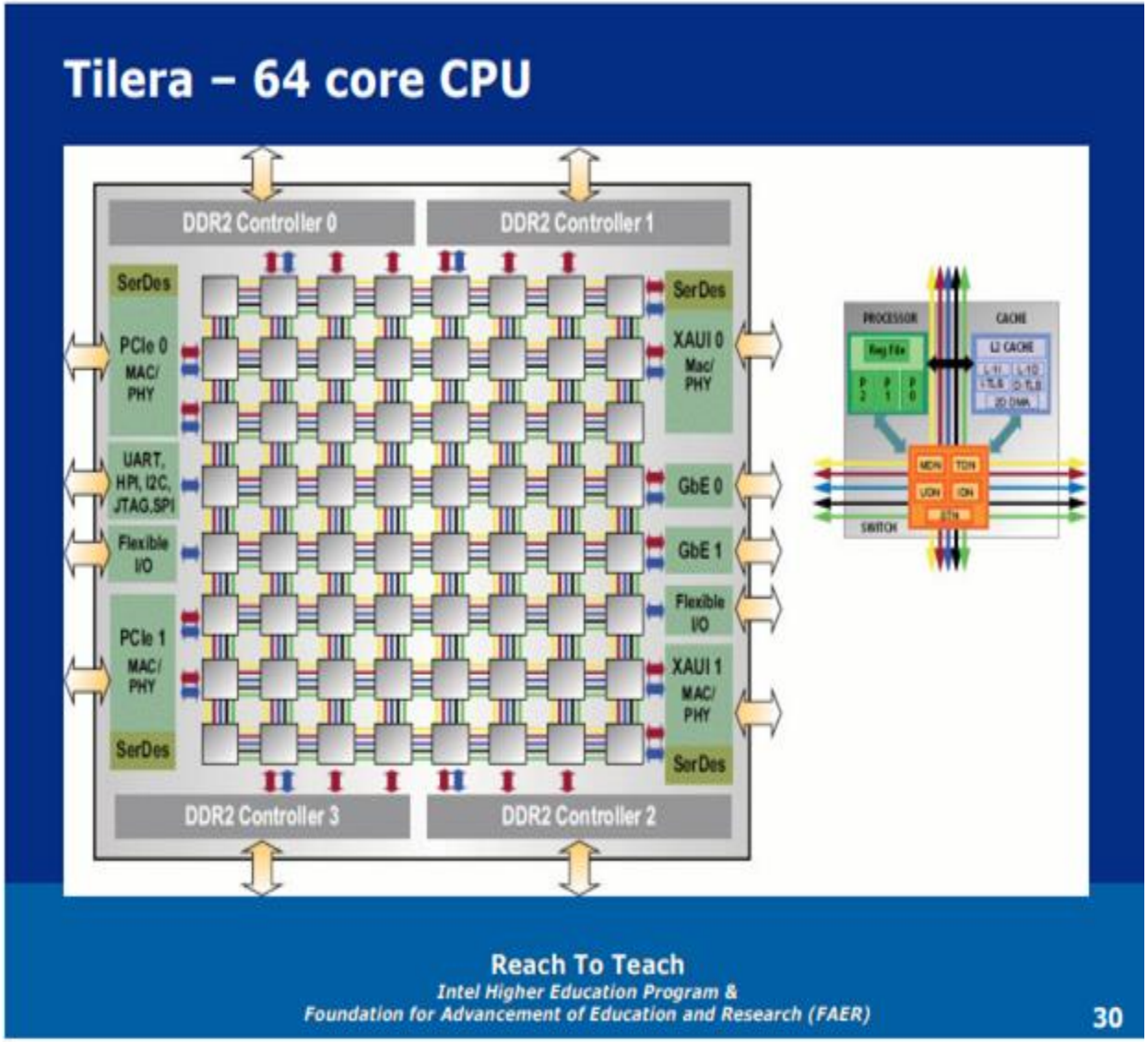
Very Long Instruction Word (VLIW) Processors

- VLIW processors rely on compile time analysis to identify and bundle together instructions that can be executed concurrently.
- These instructions are packed and dispatched together, and thus the name very long instruction word

Intel IA64 processors.



Multicore Processor



- A multi-core processor implements multiprocessing in a single physical package.
- Cores in a multi-core device may be coupled together tightly or loosely.
- For example, cores may or may not share caches, and they may implement message passing or shared memory inter-core communication methods.
- Common network topologies to interconnect cores include: bus, ring, 2-dimentional mesh, and crossbar.
- All cores are identical in *symmetric* multi-core systems and they are not identical in a*symmetric* multi-core systems.
- Just as with single-processor systems, cores in multi-core systems may implement architectures such as superscalar, vector processing, or multithreading.

- Multi-core processors are widely used across many application domains including: general-purpose, embedded, network, digital signal processing, and graphics.
- The amount of performance gained by the use of a multi-core processor is strongly dependent on the software algorithms and implementation.
- Multi-core processing is a growing industry trend as single core processors rapidly reach the physical limits of possible complexity and speed.
- Companies that have produced or are working on multi-core products include AMD, ARM, Broadcom, Intel, and VIA.

Flynn's Taxonomy

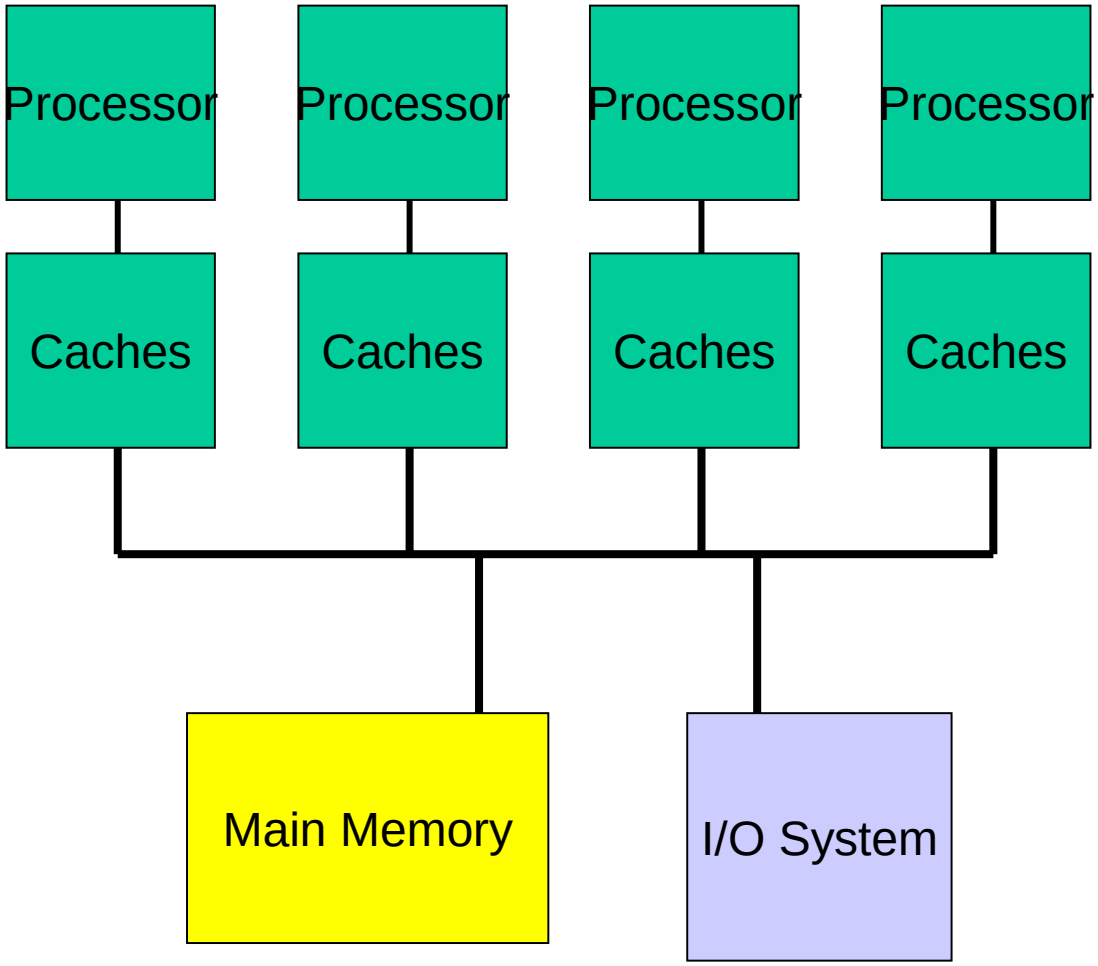
- Single instruction stream, single data stream (SISD)
  - uniprocessor
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
  - Imagine data going through a pipeline of execution engines
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD
  - Most multiprocessors today: easy to construct with off-the-shelf computers, most flexibility

Multiple-instruction, multiple-data (MIMD)

- MIMD machines are broadly categorized based on the way Processing Elements (PE) are coupled to the main memory.

- **Shared-memory MIMD (Types: Centralized and Distributed)**
- Tightly coupled multiprocessor systems
- Connected to a single global memory and they all have access to it.
- Symmetric Multi-Processing.
- Less likely to scale
- **Distributed-memory MIMD**
- Loosely coupled multiprocessor systems
- All PEs have a local memory
- Communication between PEs in this model takes place through the interconnection network.

Centralized Shared-Memory



- **Advantages**

- Global address space provides a user-friendly programming perspective to memory

- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

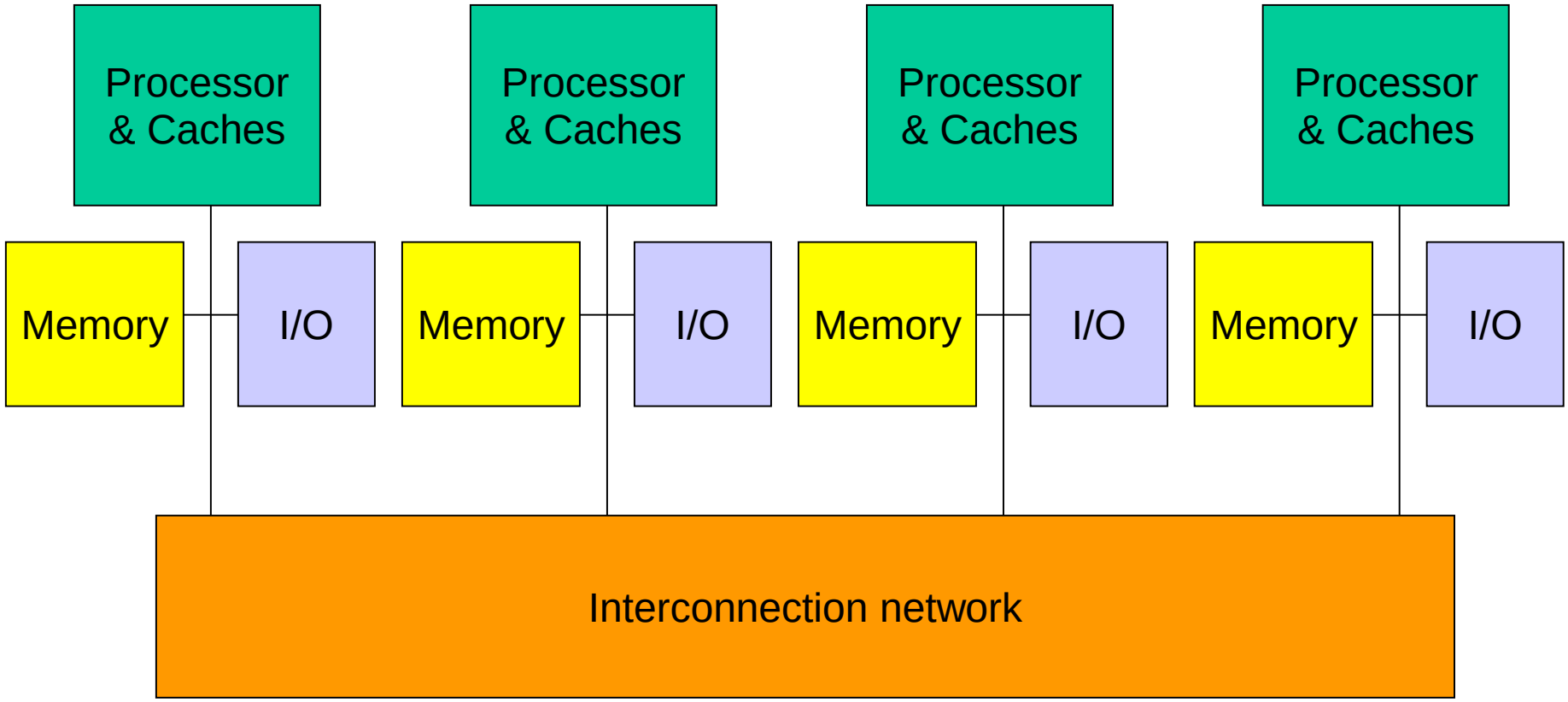
- **Disadvantages**

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.

- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory Multiprocessors



Distributed Memory Multiprocessors

- Require a communication network to connect inter-processor memory.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor
- No concept of global address space across all processors.
- Changes to local memory have no effect on the memory of other processors.

- **Advantages**

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

- **Disadvantages**

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

Interconnection Network

- Both shared memory and message passing architectures can use
  - **static interconnection network**
    - can not be modified without a physical re-designing of a system
    - static networks usually in message passing computers
  - **dynamic interconnection network**
    - dynamic networks usually in shared memory computers
    - enable changing (reconfiguring) of the connection structure in a system.

Interconnection Network

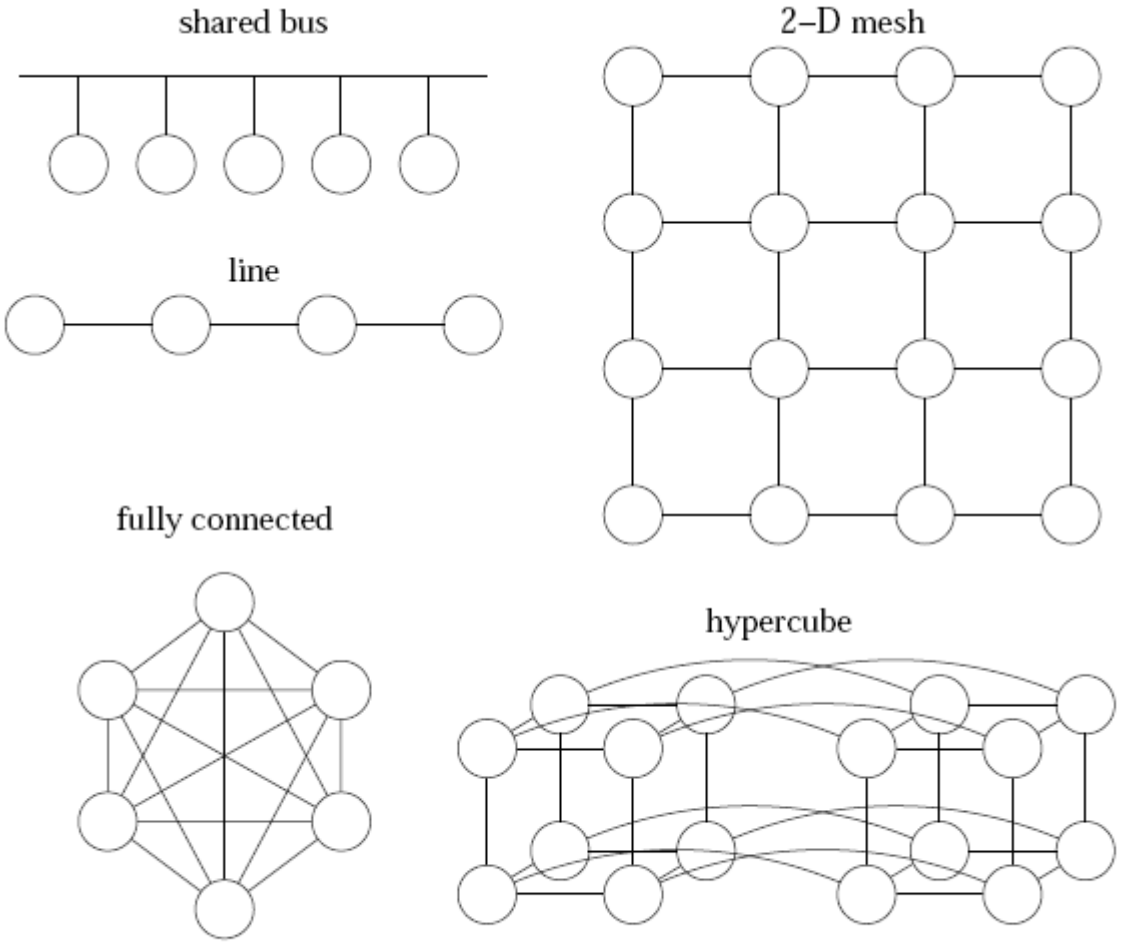
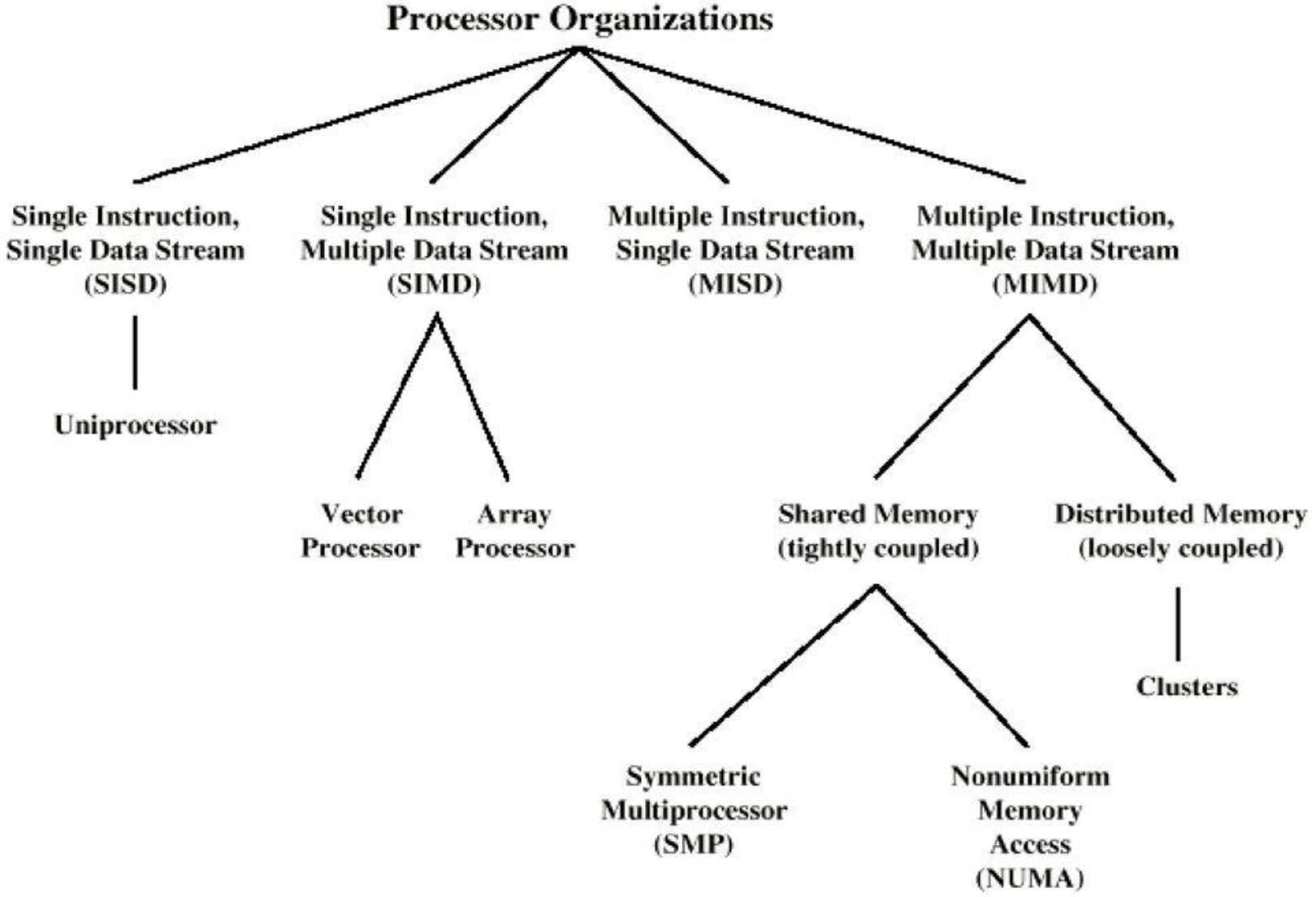


Figure 5.6: Five important interconnect network topologies.

Taxonomy of Parallel Processor Architectures



- Parallel architectures can be classified in several ways

- Flynn's taxonomy
- Control mechanism
- Address-space organization
- Interconnection network

– Granularity of processors

Granularity of Processors

- Granularity is the ratio of computation time to communication time.
  - Computation time – Time required to perform computation of a task
  - Communication time is the time required to exchange data between processors
- Fine-grained, medium-grained and coarse-grained parallelism

- **Fine-grained parallelism**
  - a program is broken down to a large number of small tasks.
  - Tasks are assigned individually to many processors.
  - The work is evenly distributed among the processors.
- Increases the communication and synchronization overhead

Fine-grained parallelism is best exploited in architectures which support fast communication

Shared memory architecture are most suitable for fine-grained parallelism.

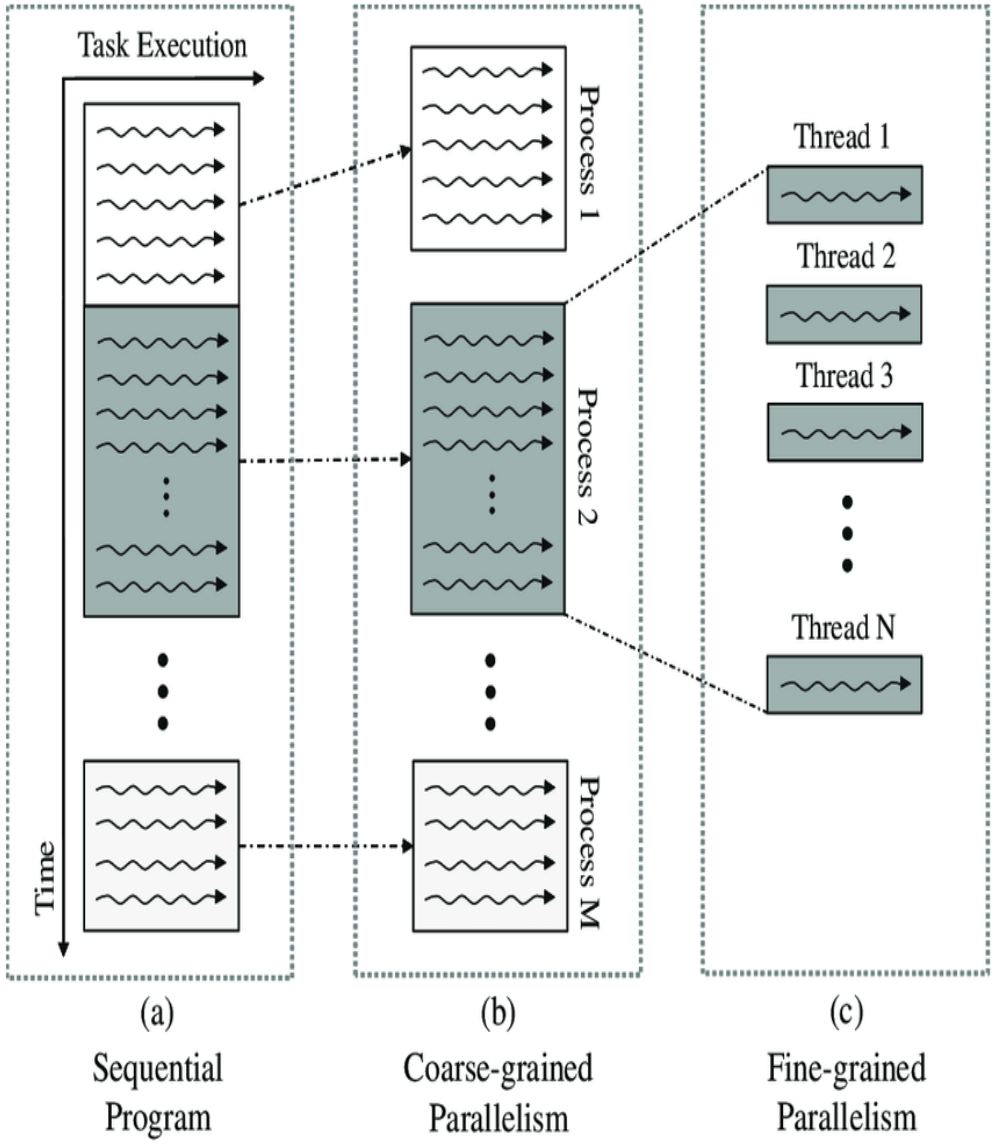
- **Coarse-grained parallelism**

- a program is split into large tasks. a large amount of computation takes place in processors.

- Load imbalance

- Certain tasks process the bulk of the data while others might be idle.

- Low communication and synchronization overhead.


Task Execution
Time
Process 1
Process 2
Process M
Thread 1
Thread 2
Thread 3
Thread N
(a)
Sequential Program
(b)
Coarse-grained Parallelism
(c)
Fine-grained Parallelism

- **Medium-grained parallelism**

- Medium-grained parallelism is used relatively to fine-grained and coarse-grained parallelism.
- Task size and communication time greater than fine-grained parallelism and lower than coarse-grained parallelism
- General-purpose parallel computers fall in this category

- Example :
  - Consider a 10*10 image that needs to be processed, given that, processing of the 100 pixels is independent of each other. Assume there are
- Fine-grained parallelism:
  - 100 processors that are responsible for processing the 10*10 image.
  - 100 processors can process the 10*10 image in 1 clock cycle
- Each processor is working on 1 pixel of the image
- Medium-grained parallelism:
  - 25 processors processing the 10*10 image.
- The processing of the image will now take 4 clock cycles.
- Coarse-grained parallelism
  - we reduce the processors to 2, processing will take 50 clock cycles

- Relationship between levels of parallelism, grain size and degree of parallelism

Levels	**Grain Size**	**Parallelism**
Instruction level	Fine	Highest
Loop level	Fine	Moderate
Sub-routine level	Medium	Moderate
Program level	Coarse	Least

- **Impact of granularity on performance**

- Fine grains or small tasks results in more parallelism.

- Synchronization overhead, scheduling strategies

- Coarse grained tasks have less communication overhead but they often cause load imbalance.

Finding the best grain size, depends on a number of factors and varies

greatly from problem-to-problem.

References

- https://en.wikipedia.org/wiki/Granularity_(parallel_computing)

- Hwang, Kai (1992). Advanced Computer Architecture: Parallelism,Scalability,Programmability (1st ed.). McGraw-Hill Higher Education. ISBN 978-0070316225.