Parallel and Distributed Computing

CS3006

Lecture 3

Flynn's Taxonomy

29th January 2024

Slides provided by: Dr. Rana Asif Rehman

Additions for Spring 2024 by: Dr. Abdul Qadeer

Agenda

- A Quick Review
- Flynn’s Taxonomy
  - SISD
  - MISD
  - SIMD
  - MIMD
- Physical Organization of Parallel Platforms
  - PRAM
- Routing techniques and Costs

Quick Review to the Previous Lecture

- **Amdahl's Law of Parallel Speedup**
  - Maximum speedup limited by serial portion: **Serial bottleneck**
  - **But this law says nothing about the parallelism overheads**
- **Parallelism Overhead**
  - Parallel portion is usually not perfectly parallel
    - Synchronization overhead (e.g., updates to shared data)
    - Load imbalance overhead (imperfect parallelization)
    - Resource sharing overhead (contention among P processors)
- **Karp-Flatt Metric e: guidance about parallelism overheads**
  - Find e experimentally with fixed data size, but changing p (p >1)
  - If e remains constant => Overhead not a primary reason
  - If e increases => Overhead primary reason

Multi-processor vs Multi-Computer

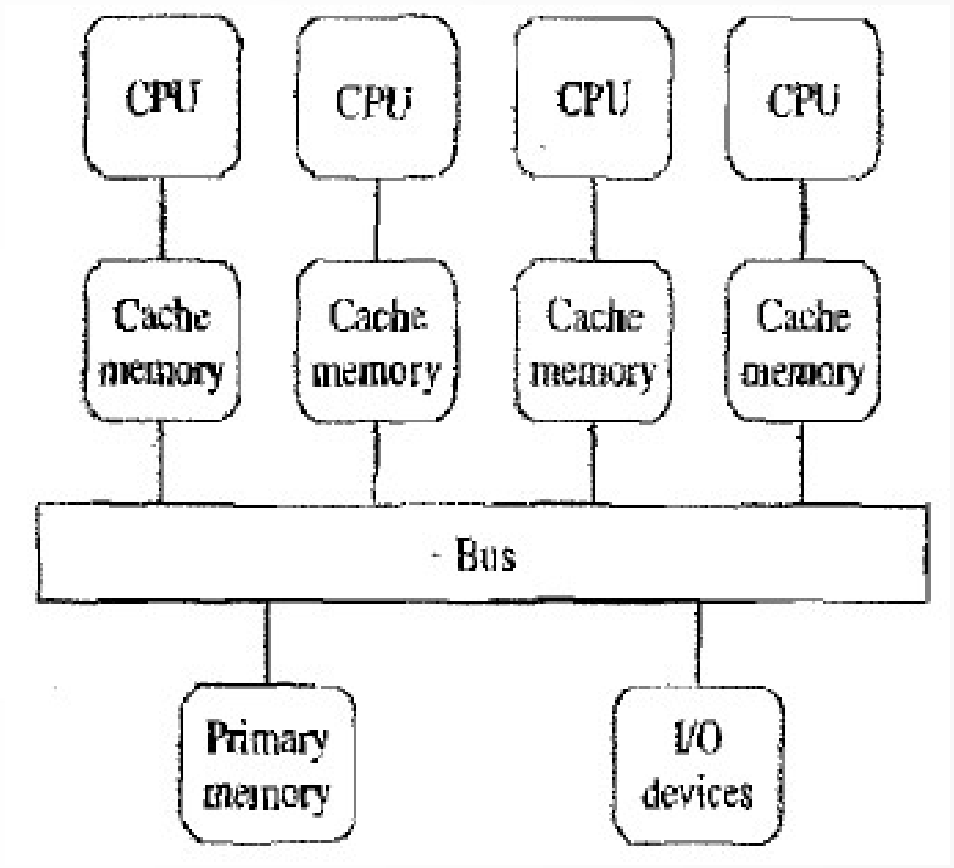
Multi-Processor

- Multiple-CPUs with a shared memory
- The same address on two different CPUs refers to the same memory location.
- **Generally two categories:-**
  1. Centralized Multi-processors
  2. Distributed Multi-processor

Multi-Processor

i. **Centralized Multi-processor**

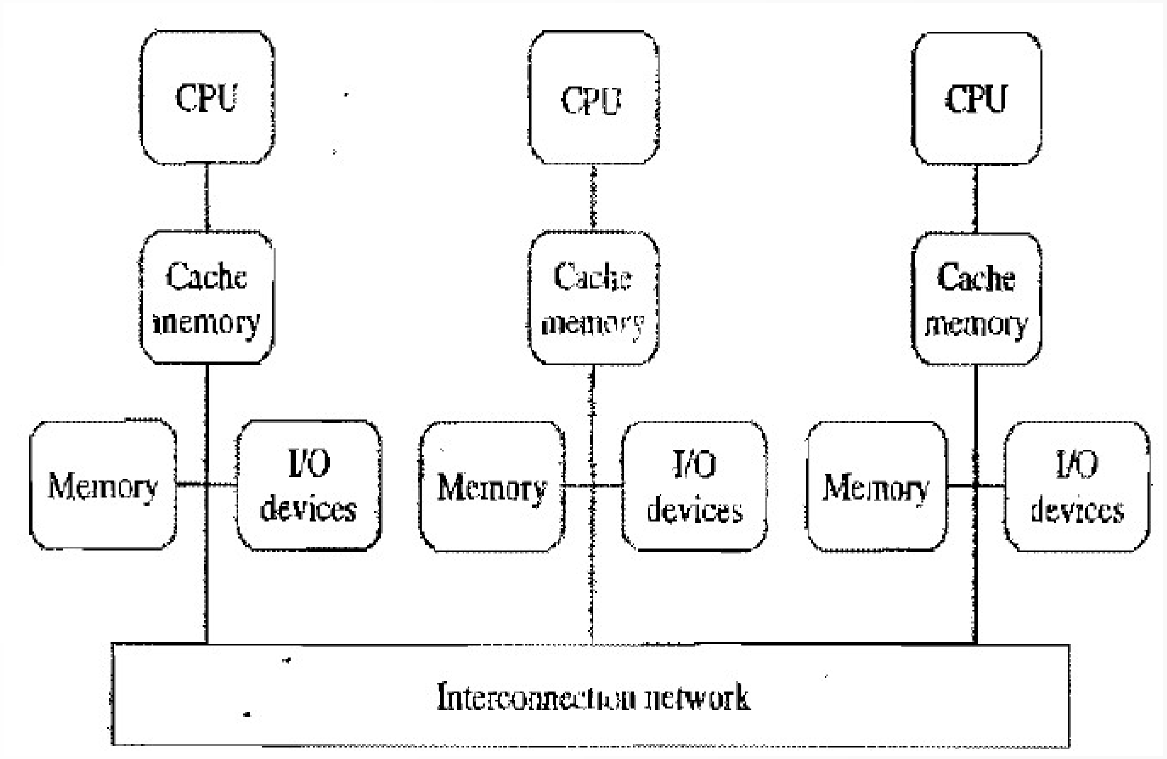
- Additional CPUs are attached to the system bus, and all the processors share the same primary memory
- All the memory is at one place and has the **same access time from every processor**
- Also known to as **UMA**(Uniform Memory Access) multi-processor or **SMP** (symmetrical Multi-processor)



Multi-Processor

ii. **Distributed Multi-processor**

- Distributed collection of memories forms one logical address space
- Again, the same address on different processors refers to the same memory location.
- Also known as non-uniform memory access (**NUMA**) architecture
- Because, **memory access time varies significantly, depending on the physical location** of the referenced address



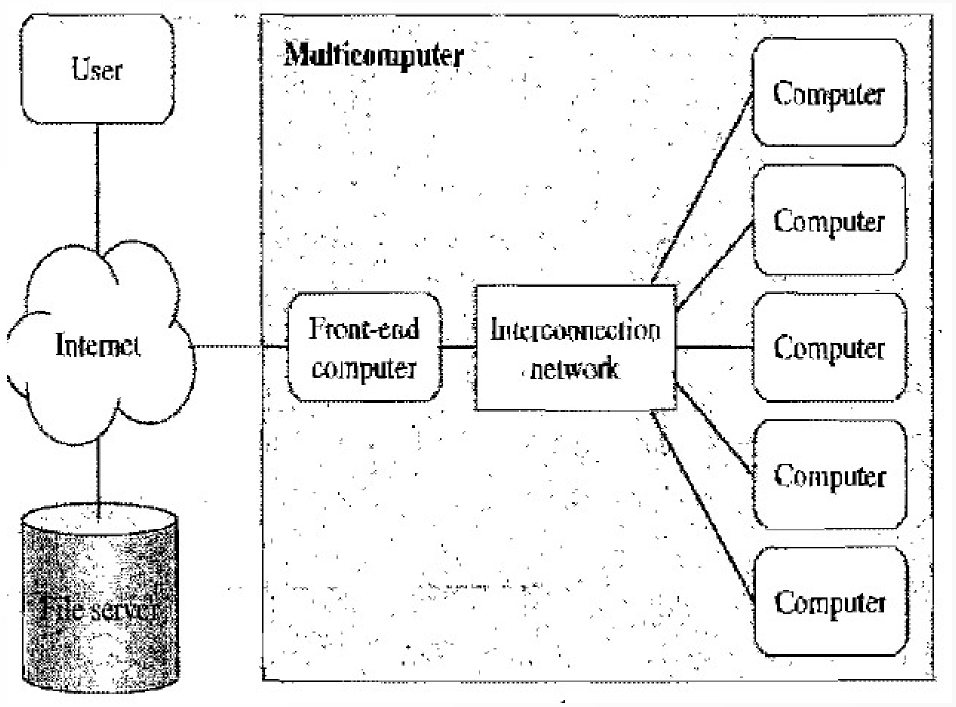
Multi-Computer

- Distributed-memory, multi-CPU computer
- Unlike **NUMA** architecture, a multicomputer has **disjoint local address spaces**
- Each processor has direct access to their local memory only.
- The same address on different processors refers to two different physical memory locations.
- Processors interact with each other through **passing messages**
- **Example:**
  - **When we do Remote Procedure Call, we need to serialize all the function parameters before calling**

Multi-Computer

Asymmetric Multi-Computers

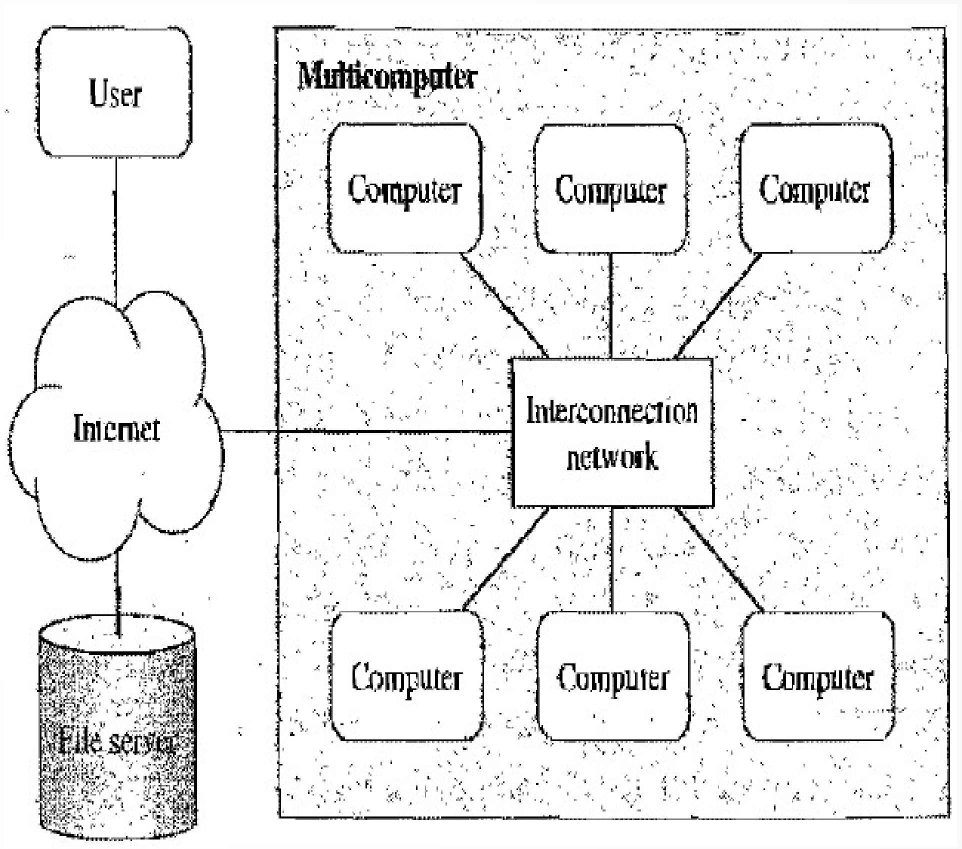
- A front-end computer that interacts with users and I/O devices
- The back-end processors are dedicatedly used for "number crunching"
- Front-end computer executes a full, multiprogrammed OS and provides all functions needed for program development
- The backends are reserved for executing parallel programs
- Examples: Modern data centers with a load balancer,
- ARM's big.Little core model
- PS3's cores



Multi-Computer

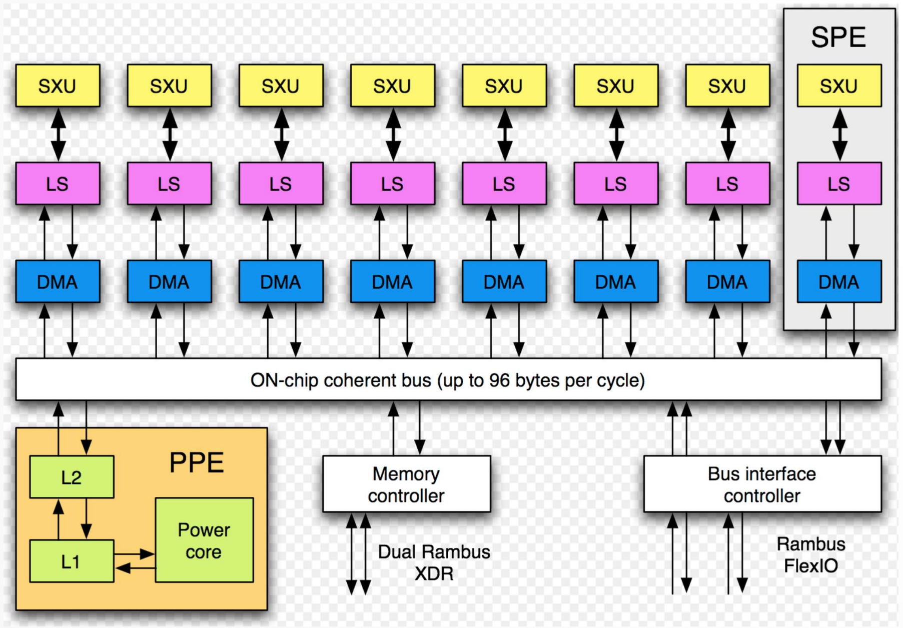
Symmetric Multi-Computers

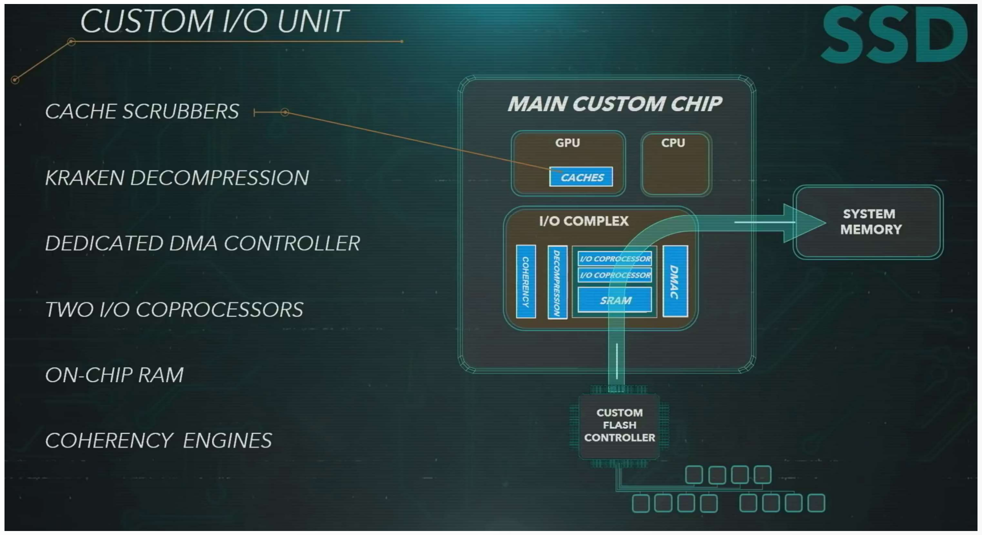
- Every computer executes same OS
- Users may log into any of the computers
- This enables multiple users to concurrently login, edit and compile their programs.
- All the nodes can participate in execution of a parallel program
- Example: Supercomputers for scientific computing



PS3 cell architecture vs PS5 x86 arch

- PS3 was based on IBM's power PC ISA based cell processor
- PS5 is based on X86 SOC by AMD (the usual shared memory system)





Network of Workstations vs Cluster

Cluster	Network of workstations
Usually a co-located collection of low-cost computers and switches, dedicated to running parallel jobs. All computer run the same version of operating system.	A dispersed collection of computers. Individual workstations may have different Operating systems and executable programs
Some of the computers may not have interfaces for the users to login	User have the power to login and power off their workstations
Commodity cluster uses high speed networks for communication such as fast Ethernet@100Mbps, gigabit Ethernet@1000 Mbps and Myrinet@1920 Mbps.	Ethernet speed for this network is usually slower. Typical in range of 10 Mbps

- Modern datacenters usually are divided into clusters
- Network speeds are ~ 40Gbps, 80Gbps etc.

Classes of Parallelism & Parallel Architectures

- There are basically two kinds of parallelism in applications:
    - **Data-level parallelism (DLP)** arises because there are many data items that can be operated on at the same time.
    - **Task-level parallelism (TLP)** arises because tasks of work are created that can operate independently and largely in parallel.
        - We called it **functional parallelism** in last lecture

Classes of Parallelism & Parallel Architectures

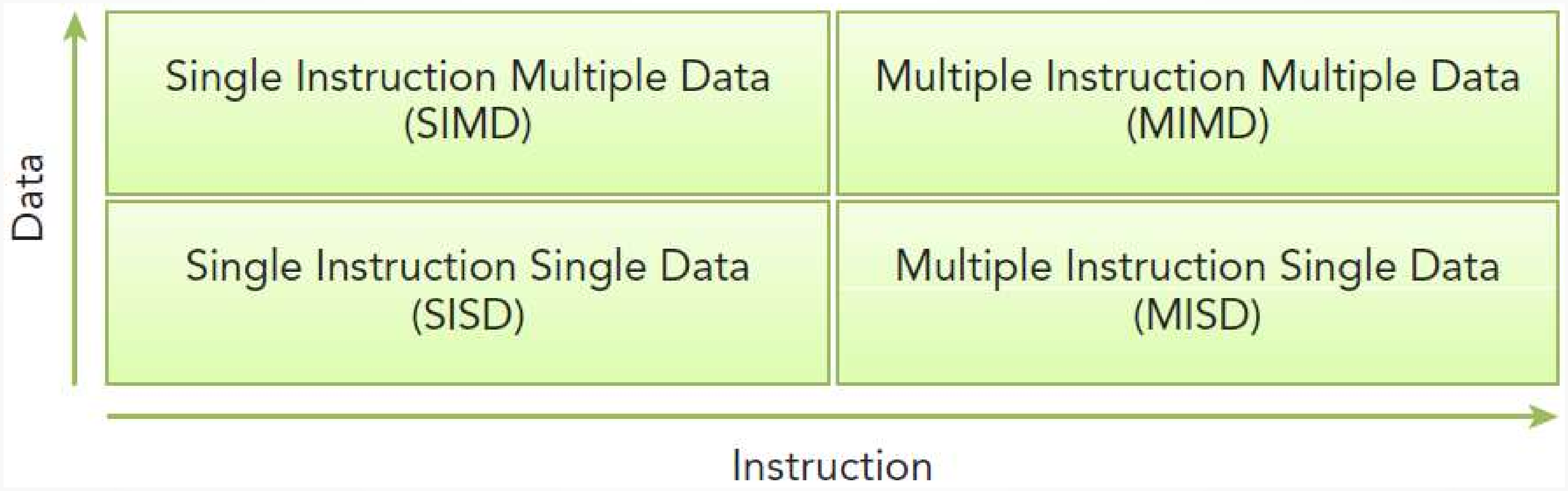
- Computer hardware in turn can exploit two kinds of application parallelism in four major ways:
  - **Instruction-level parallelism** exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution.
  - **Vector** architectures, graphic processor units (**GPU**s), and **multimedia instruction sets** exploit data-level parallelism by applying a single instruction to a collection of data in parallel.

Classes of Parallelism & Parallel Architectures

- Computer hardware in turn can exploit two kinds of application parallelism in four major ways:
  - **Thread-level parallelism** exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction between parallel threads.
  - **Request-level parallelism** exploits parallelism among largely decoupled tasks specified by the programmer or the operating system.

Flynn’s Taxonomy

- Widely used architectural classification scheme
  - Flynn studied the parallel computing efforts in the 1960s, he found a simple classification whose abbreviations we still use today.
- Classifies architectures into four types
- The classification is based on how data and instructions flow through the cores.


Single Instruction Multiple Data (SIMD)
Multiple Instruction Multiple Data (MIMD)
Single Instruction Single Data (SISD)
Multiple Instruction Single Data (MISD)
Data
Instruction

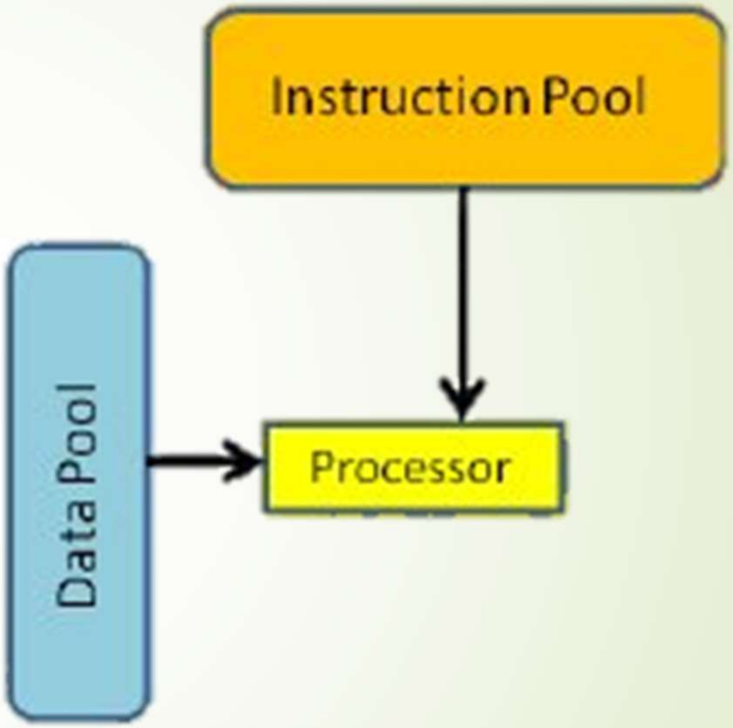
Flynn’s Taxonomy

- This taxonomy is a coarse model:
    - many parallel processors are **hybrids** of the SISD, SIMD, and MIMD classes.
    - Nonetheless, it is useful to put a framework on the **design space** for the computers we will see in this course

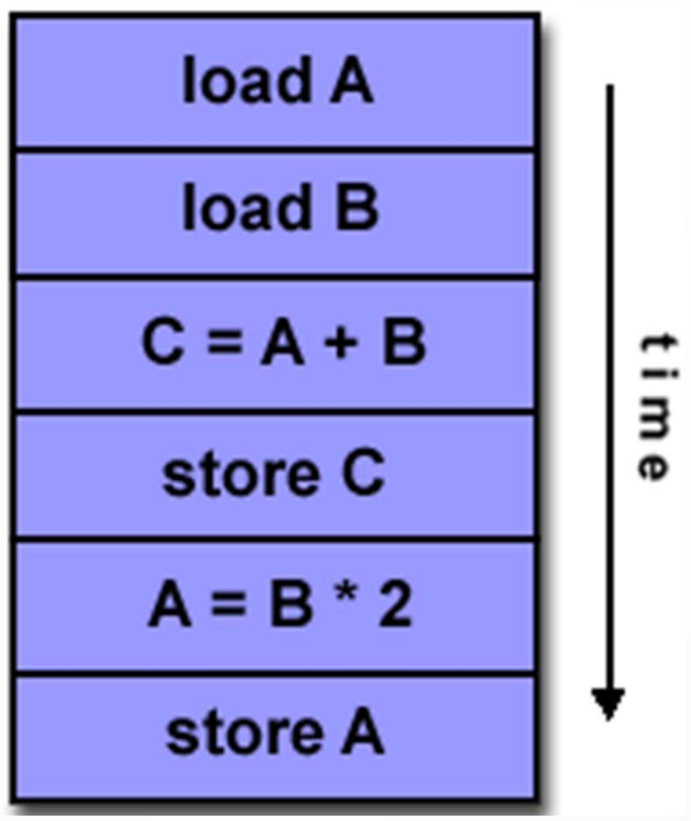
Flynn's Taxonomy

SISD (Single Instruction Single Data)

- Refers to traditional computer: a serial architecture
- This architecture includes single core computers
- Single instruction stream is in execution at a given time
- Similarly, only one data stream is active at any time



Example of SISD:

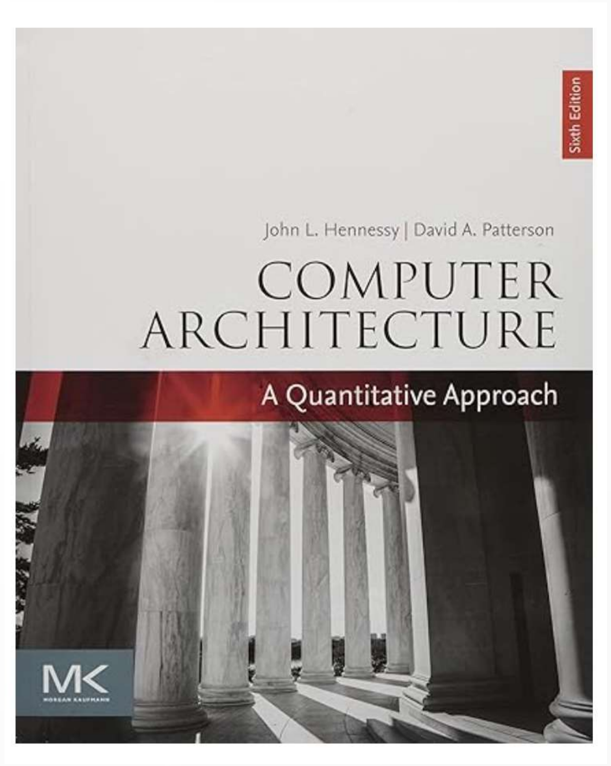


SISD: Some Explanations

- The programmer thinks of it as the standard sequential computer, but it **can exploit ILP**.
- ILP techniques used can be:
  - superscalar (for example via pipelining where multiple instructions are in progress)
  - speculative execution (for example fetching instruction streams using predictions
  - Many **challenges** to get pipelining and predictive executions right
    - Hazards (read-after-write, write-after-write, write-after-read)
    - Security issues (meltdown, specter)
      - Side-channel attacks
      - Meltdown mostly Intel specific

Some references for further reading

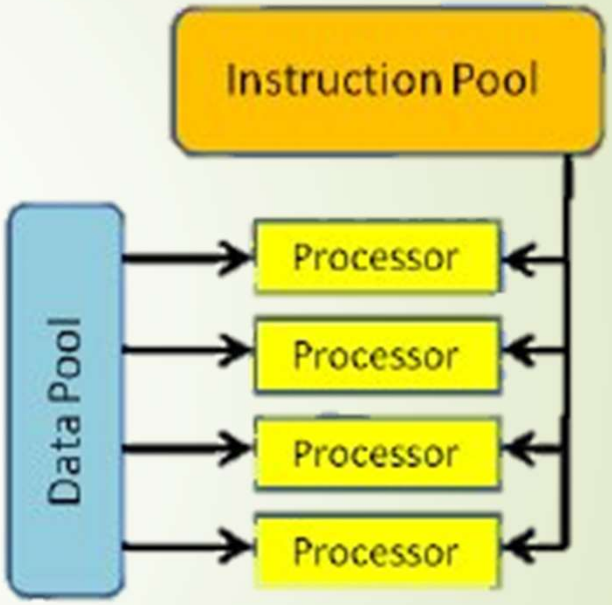
- There are tons of details about SISD implementations and ILP
- Refer to chapter 3 of:
  - Computer Architecture: A quantitative approach (6th edition)



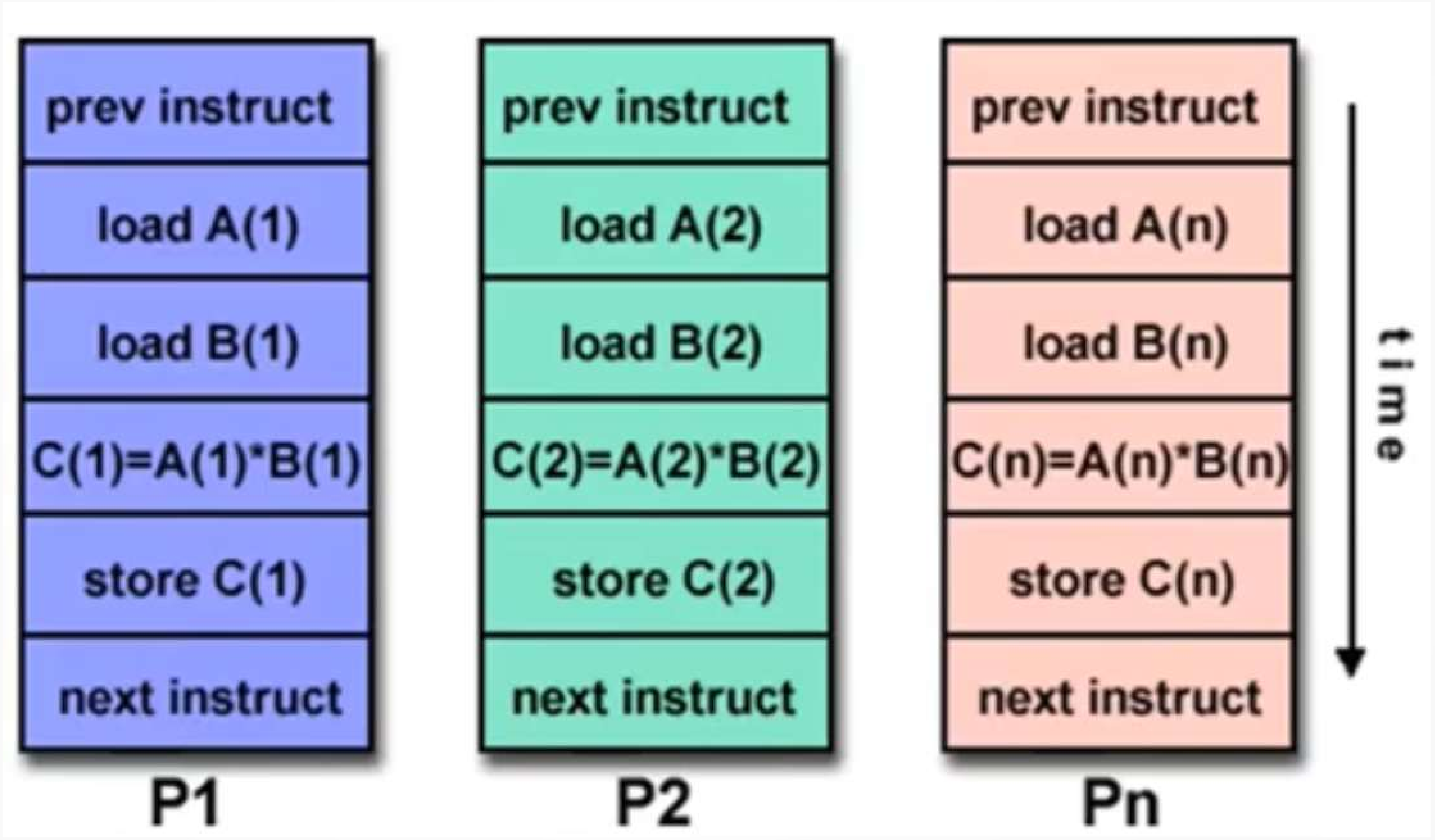
Flynn’s Taxonomy

SIMD (Single Instruction Multiple Data)

- Refers to parallel architecture with multiple cores
- All the cores execute the same instruction stream at any time but, data stream is different for the each.
- Well-suited for the scientific operations requiring large matrix-vector operations
- Vector computers (Cray vector processing machine) and Intel co-processing unit ‘MMX’ fall under this category.
- Used with array operations, image processing and graphics

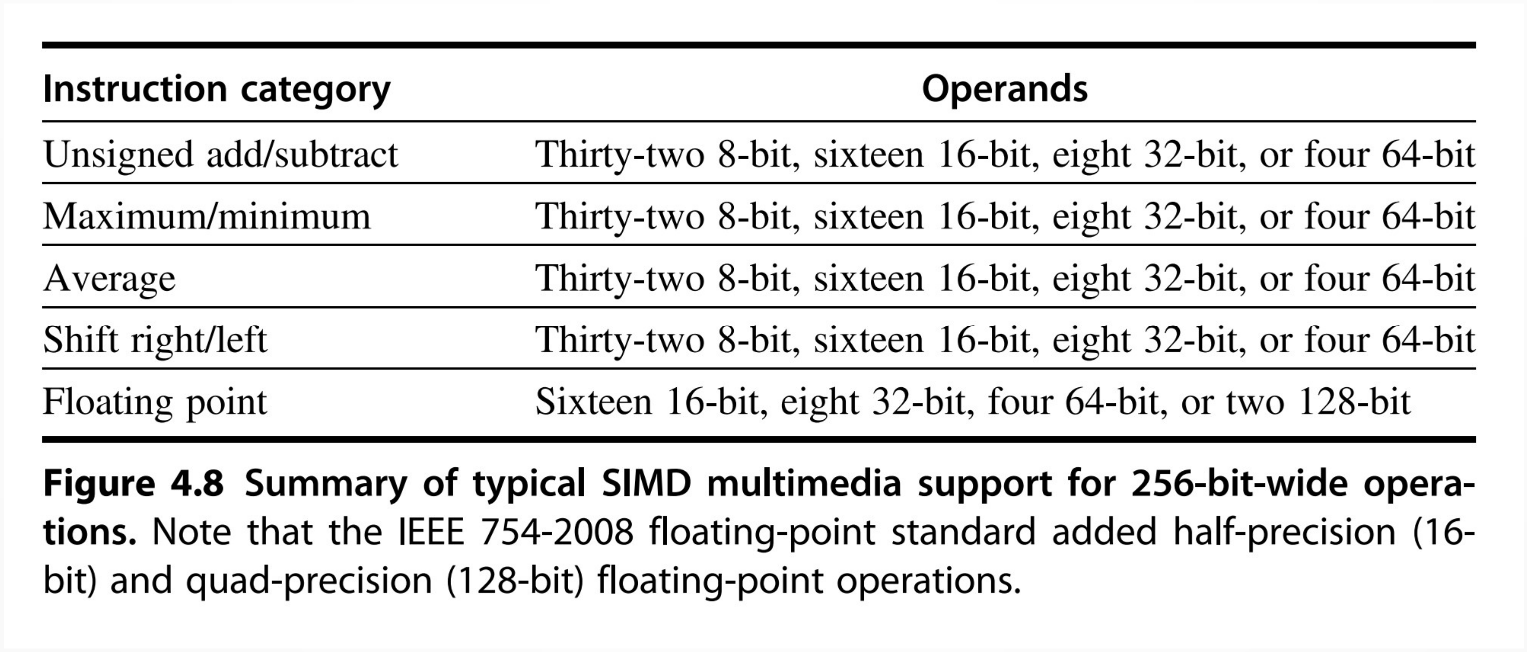


Example of SIMD:



SIMD: Some Explanations

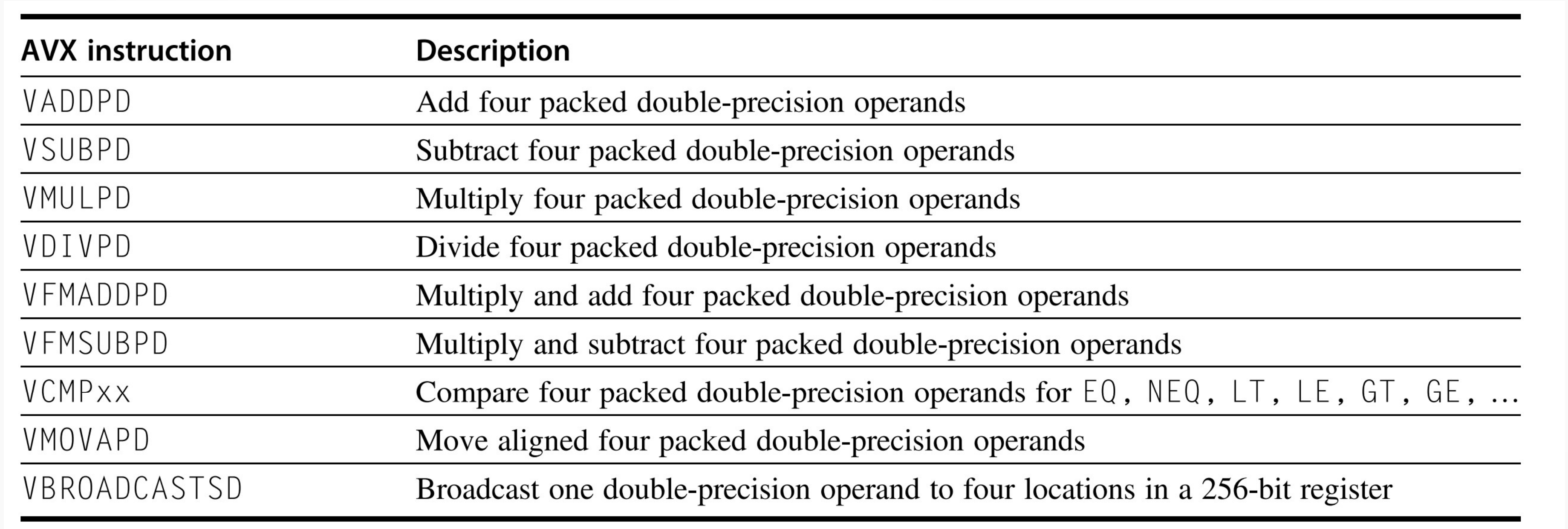
- SIMD computer exploit data-level parallelism
- Examples:
  - Vector architectures
  - Multimedia extensions to standard instructions
  - GPUs



Instruction category	Operands
Unsigned add/subtract	Thirty-two 8-bit, sixteen 16-bit, eight 32-bit, or four 64-bit
Maximum/minimum	Thirty-two 8-bit, sixteen 16-bit, eight 32-bit, or four 64-bit
Average	Thirty-two 8-bit, sixteen 16-bit, eight 32-bit, or four 64-bit
Shift right/left	Thirty-two 8-bit, sixteen 16-bit, eight 32-bit, or four 64-bit
Floating point	Sixteen 16-bit, eight 32-bit, four 64-bit, or two 128-bit

Figure 4.8 Summary of typical SIMD multimedia support for 256-bit-wide operations. Note that the IEEE 754-2008 floating-point standard added half-precision (16-bit) and quad-precision (128-bit) floating-point operations.

SIMD: Some Explanations

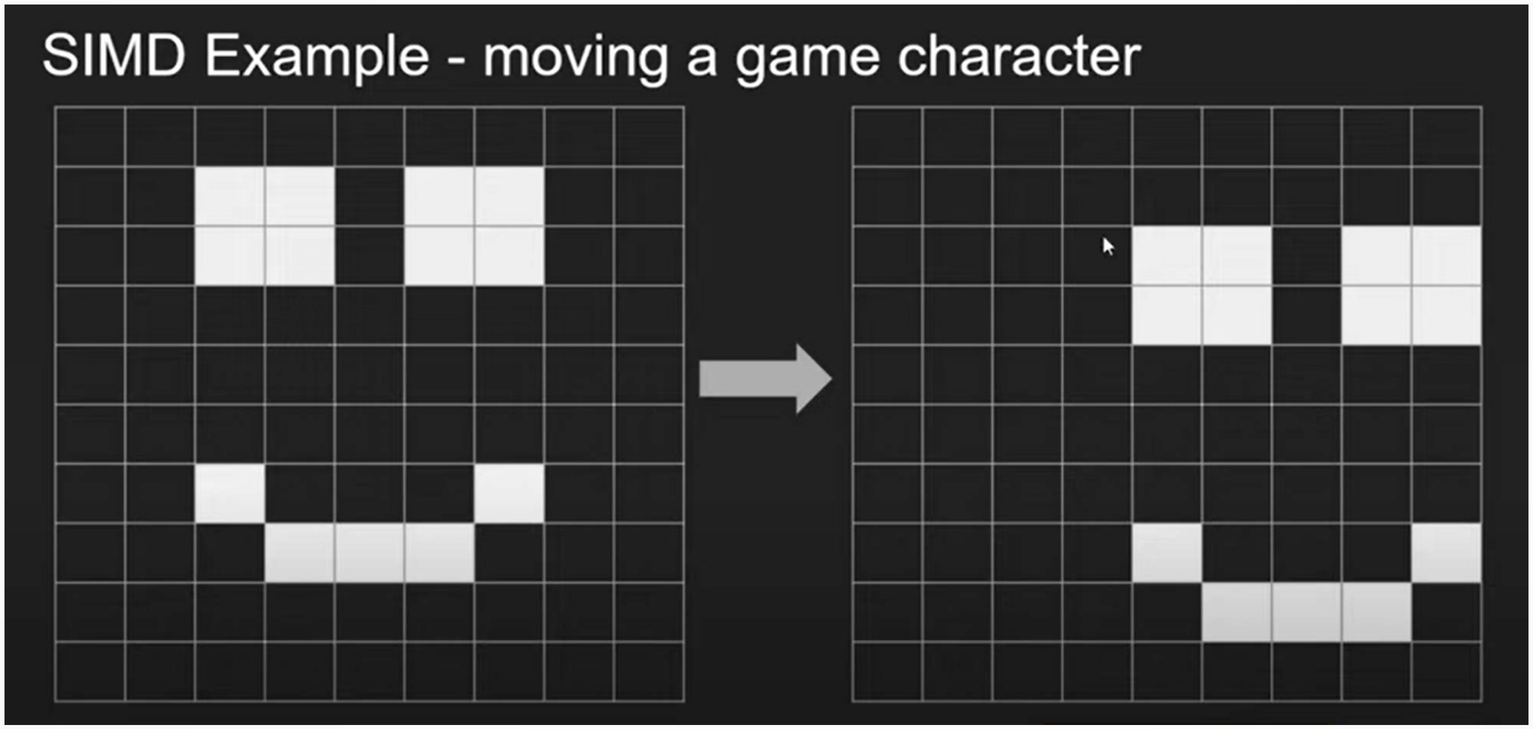


AVX instruction	Description
`VADDPD`	Add four packed double-precision operands
`VSUBPD`	Subtract four packed double-precision operands
`VMULPD`	Multiply four packed double-precision operands
`VDIVPD`	Divide four packed double-precision operands
`VFMADDPD`	Multiply and add four packed double-precision operands
`VFMSUBPD`	Multiply and subtract four packed double-precision operands
`VCMPxx`	Compare four packed double-precision operands for `EQ`, `NEQ`, `LT`, `LE`, `GT`, `GE`, …
`VMOVAPD`	Move aligned four packed double-precision operands
`VBROADCASTSD`	Broadcast one double-precision operand to four locations in a 256-bit register

Figure 4.9 AVX instructions for x86 architecture useful in double-precision floating-point programs. Packed-double for 256-bit AVX means four 64-bit operands executed in SIMD mode. As the width increases with AVX, it is increasingly important to add data permutation instructions that allow combinations of narrow operands from different parts of the wide registers. AVX includes instructions that shuffle 32-bit, 64-bit, or 128-bit operands within a 256-bit register. For example, BROADCAST replicates a 64-bit operand four times in an AVX register. AVX also includes a large variety of fused multiply-add/subtract instructions; we show just two here.

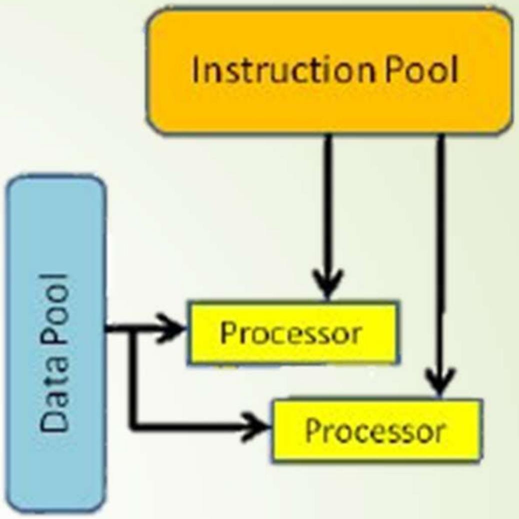
SIMD: Some Explanations

- GPUs: Example pixels around



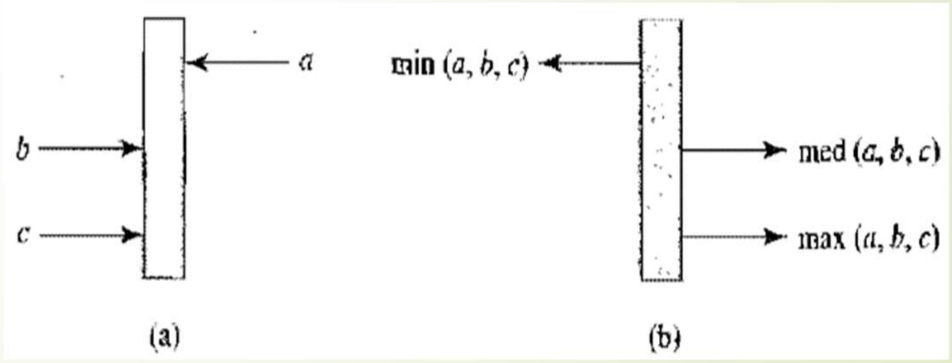
- Credit: Screen grab from: https://www.youtube.com/watch?v=KVOc6369-Lo

Flynn's Taxonomy

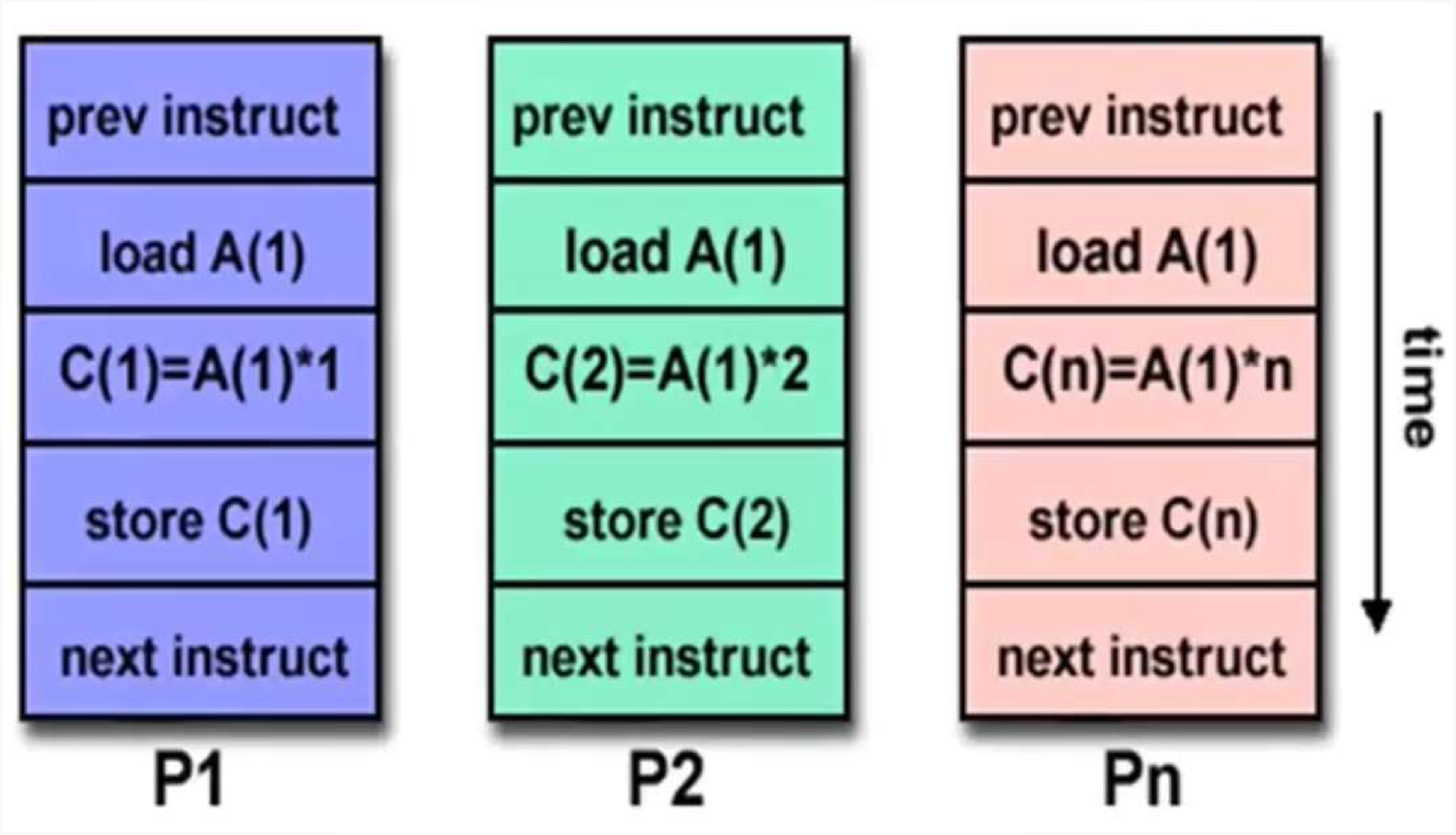


MISD (Multiple Instructions Single Data)

- Multiple instruction stream and single data stream
  - A pipeline of multiple independently executing functional units
  - Each operating on a single stream of data and forwarding results from one to the next
- Rarely used in practice
- E.g., Systolic arrays : network of primitive processing elements that pump data.
- But one can make an argument that systolic arrays are not MISD because data keep on changing as it passes through processing elements.



Example of MISD:



MISD: Some Explanations

- Systolic arrays
  - An array of processing units that in lockstep input data from upstream neighbors, compute partial results, and pass some inputs and results to downstream neighbors

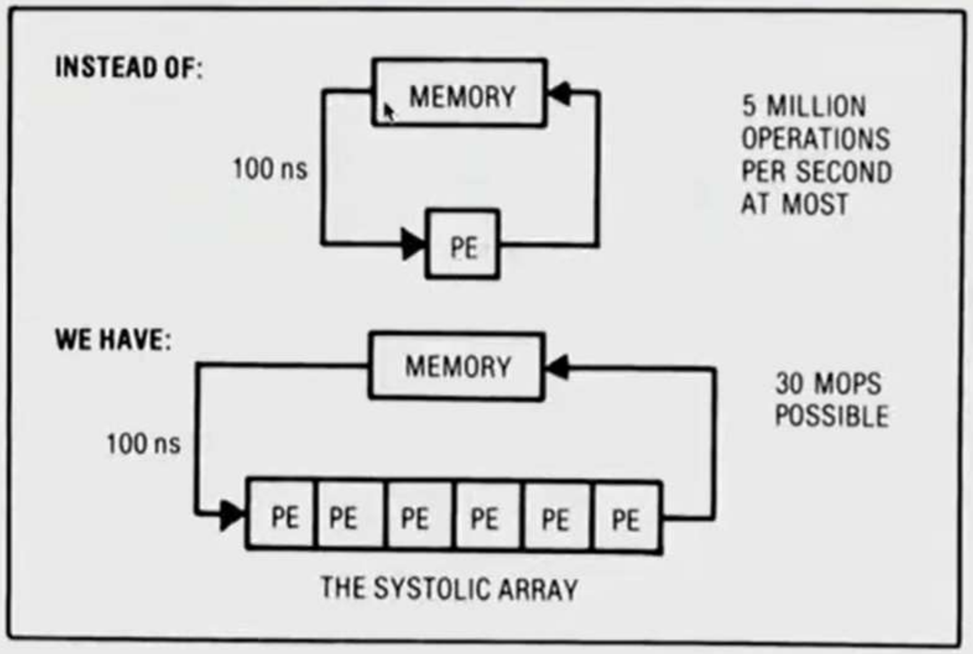
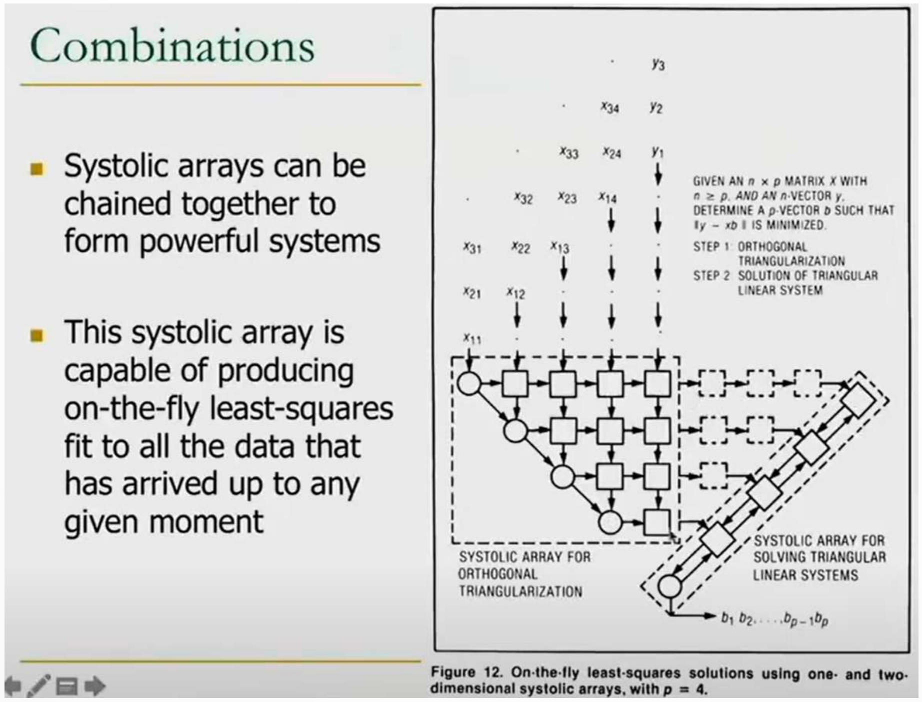


Figure 1. Basic principle of a systolic system.

MISD: Some Explanations

- Systolic arrays
  - Not generalizable.
  - Domain specific computing
  - Example: In Google's TPU systolic arrays for matrix multiply
  - See for more details: https://youtu.be/8zbh4gWGa7I?si=sSJrxUytOXdRLyyr



Combinations

- Systolic arrays can be chained together to form powerful systems
- This systolic array is capable of producing on-the-fly least-squares fit to all the data that has arrived up to any given moment

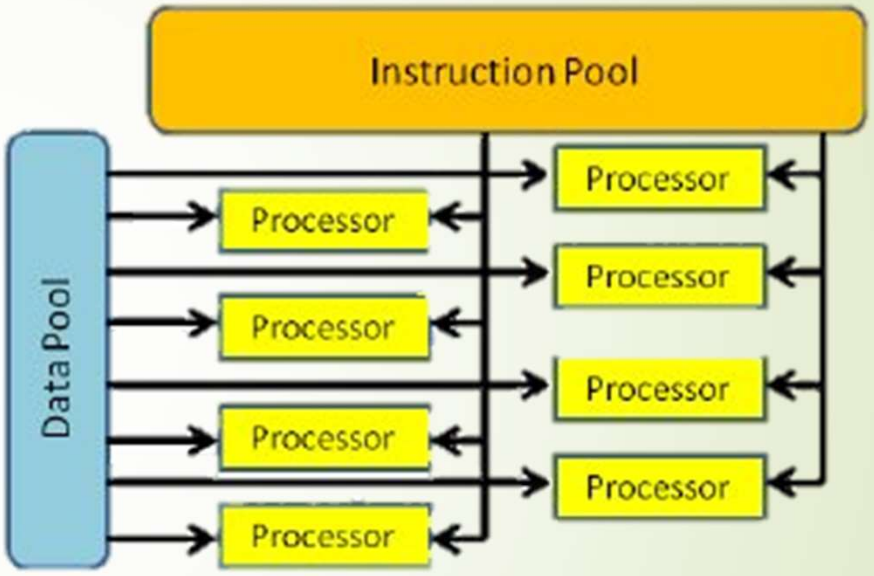
y_3 y_2 y_1
x_{34} x_{33} x_{24} x_{32} x_{23} x_{14} x_{31} x_{22} x_{13} x_{21} x_{12} x_{11}
GIVEN AN $n \times p$ MATRIX X WITH $n \geq p$, AND AN n-VECTOR y, DETERMINE A p-VECTOR b SUCH THAT $\|y - xb\|$ IS MINIMIZED.
STEP 1: ORTHOGONAL TRIANGULARIZATION
STEP 2: SOLUTION OF TRIANGULAR LINEAR SYSTEM
SYSTOLIC ARRAY FOR ORTHOGONAL TRIANGULARIZATION
SYSTOLIC ARRAY FOR SOLVING TRIANGULAR LINEAR SYSTEMS
$b_1\ b_2 \ldots b_{p-1} b_p$

Figure 12. On-the-fly least-squares solutions using one- and two-dimensional systolic arrays, with $p = 4$.

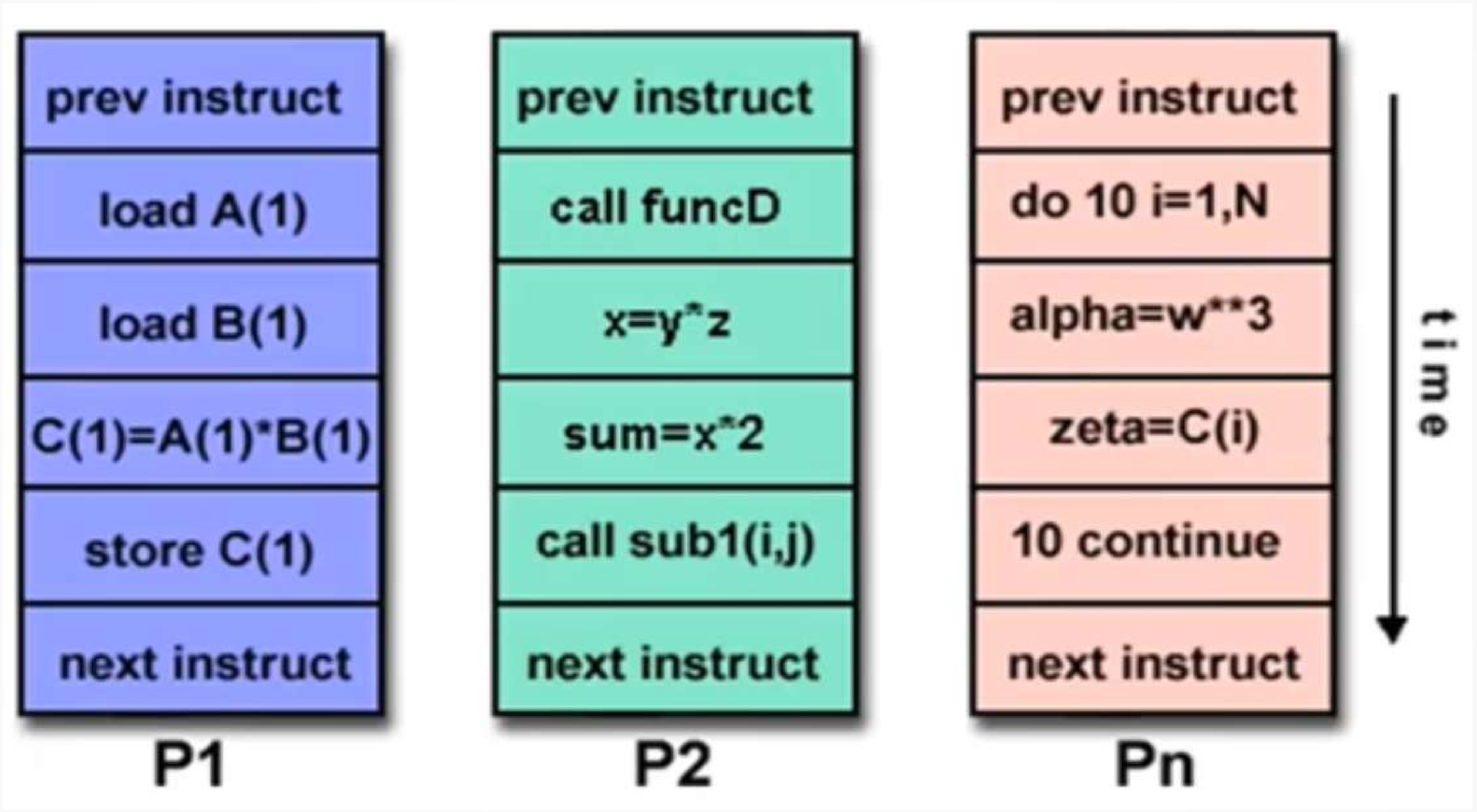
Flynn’s Taxonomy

MIMD (Multiple Instructions Multiple Data)

- Multiple instruction streams and multiple data streams
- Different CPUs can simultaneously execute different instruction streams manipulating different data
- Most of the modern parallel architectures fall under this category e.g., **Multiprocessor** and **multicomputer** architectures
- Many MIMD architectures include SIMD executions by default.



Example of MIMD:

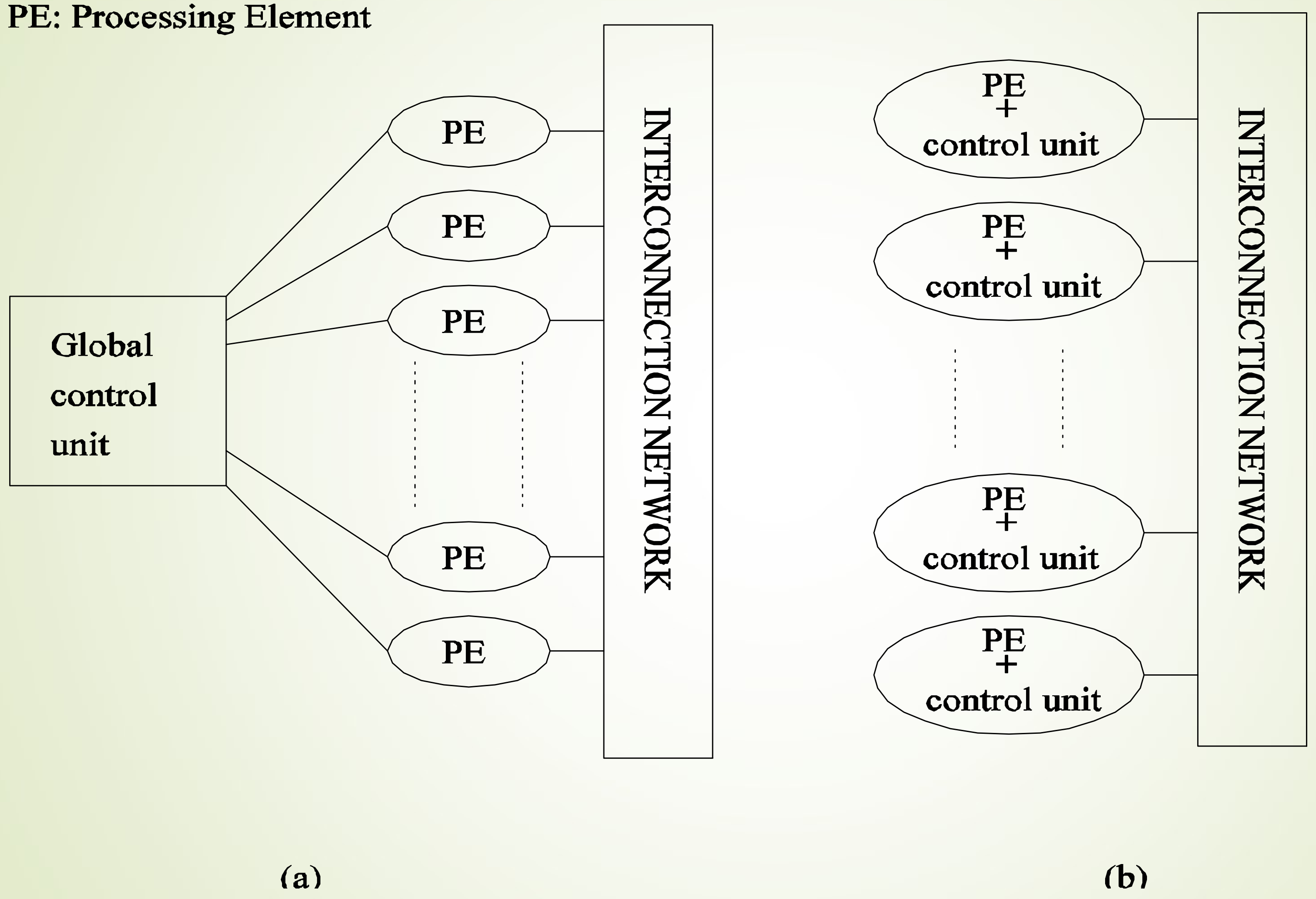


MIMD: Some Explanations

- Each processor fetches its own instructions and operates on its own data
- it targets task-level parallelism
- Tradeoff: In general, MIMD is more flexible than SIMD and thus more generally applicable, but it is inherently more expensive than SIMD.
- **Tightly coupled MIMD architectures**, which exploit thread-level parallelism because multiple cooperating threads operate in parallel.
- **Loosely coupled MIMD architectures**—specifically, clusters and warehouse-scale computers—that exploit request-level parallelism, where many independent tasks can proceed in parallel naturally with little need for communication or synchronization.

Flynn’s Taxonomy

PE: Processing Element



A typical SIMD architecture (a) and a typical MIMD architecture (b).

SIMD-MIMD Comparison

- SIMD computers require less hardware than MIMD computers (single control unit).
- However, since SIMD processors are specially designed, they tend to be expensive and have long design cycles.
- Not all applications are naturally suited to SIMD processors.
- In contrast, platforms supporting the **SPMD (Same Program Multiple Data)** paradigm can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time.
  - The Term SPMD is close variant of MIMD

We stopped here in lecture 3

Physical Organization of Parallel Platforms

Problem: We want to analyze parallel algorithms

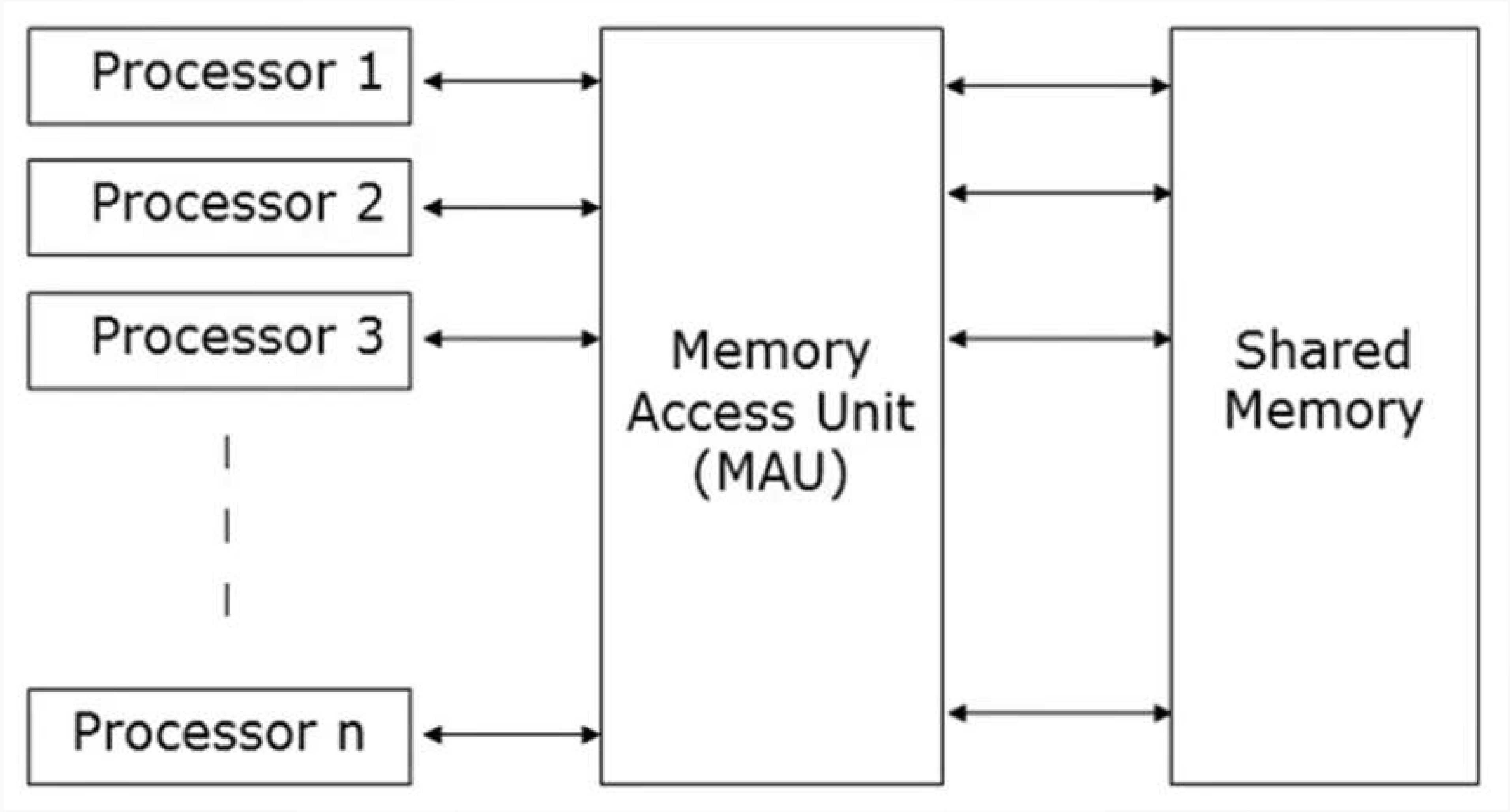
- Just like we used to analyze sequential algorithms, we need to analyze parallel algorithms
- We need some framework for such analyses
- Why can't we use sequential model without change in the parallel context?
- Can we extend the sequential model?
    - Yes (see next few slides)

Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- An extension to ideal sequential model: random access machine (RAM)
  - Another model is cache-oblivious model
- PRAMs consist of ***p*** processors
- A global memory
  - Unbounded size
  - Uniformly accessible to all processors with same address space
- Processors share a common clock but may execute different instructions in each cycle.
- Based on simultaneous memory access mechanisms, PRAM can further be classified.

Graphical representation of PRAM:



Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- PRAMs can be divided into four subclasses.

1. Exclusive-read, exclusive-write (EREW) PRAM
   - No two processors can perform read/write operations concurrently
   - Weakest PRAM model, provides minimum memory access concurrency
2. Concurrent-read, exclusive-write (CREW) PRAM
   - All processors can read concurrently but can't write at same time
   - Multiple write accesses to a memory location are serialized
3. Exclusive-read, concurrent-write (ERCW) PRAM
   - No two processors can perform read operations concurrently, but can write
4. Concurrent-read, concurrent-write (CRCW) PRAM
   - Most powerful PRAM model

Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- Concurrent reads do not create any semantic inconsistencies
- But, What about concurrent write?
- Need of an arbitration(mediation) mechanism to resolve concurrent write access. For example:
    - Locks
    - Atomic operations
    - Transactional memory
    - Etc.

Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- Mostly used arbitration protocols: -
  - **Common:** write only if all values that the processors are attempting to write are identical
  - **Arbitrary:** write the data from a randomly selected processor and ignore the rest.
  - **Priority:** follow a predetermined priority order. Processor with highest priority succeeds and the rest fail.
  - **Sum:** Write the sum of the data items in all the write requests. The sum-based write conflict resolution model can be extended for any of the associative operators, that is defined for data being written .

Architecture of an Ideal Parallel Computer

Physical Complexity of an Ideal Parallel Computer

- Processors and memories are connected via switches.
  - Switch is a device that opens or closes access to certain data bank or word

- Since these switches must operate in $O(1)$ time at the level of words, for a system of p processors and m words, the switch complexity is $O(mp)$.
- Clearly, for meaningful values of p and m, a true PRAM is not realizable.

Communication Costs in Parallel Machines

Communication Costs in Parallel Machines

- Along with idling (doing nothing) and contention (conflict e.g., resource allocation), **communication** is a major overhead in parallel programs.
- The communication cost is usually dependent on a number of features including the following:
  - Programming model for communication
  - Network topology
  - Data handling and routing
  - Associated network protocols
- Usually, distributed systems suffer from major communication overheads.

Message Passing Costs in Parallel Computers

- The total time to transfer a message over a network comprises of the following:
  - ***Startup time* (t_s):** Time spent at sending and receiving nodes (preparing the message[adding headers, trailers, and parity information] , executing the routing algorithm, establishing interface between node and router, etc.).
  - ***Per-hop time* (t_h):** This time is a function of number of hops (steps) and includes factors such as switch latencies, network delays, etc.
    - Also known as **node latency**.
  - ***Per-word transfer time* (t_w):** This time includes all overheads that are determined by the length of the message. This includes bandwidth of links, and buffering overheads, etc.

Message Passing Costs in Parallel Computers

Store-and-Forward Routing

- A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.
- The total communication cost for a message of size ***m*** words to traverse ***l*** communication links is

$$t_{comm} = t_s + (mt_w + t_h)l.$$

- In most platforms, ***t**$_h$* is small and the above expression can be approximated by

$$t_{comm} = t_s + mlt_w.$$

Message Passing Costs in Parallel Computers

Packet Routing

- Store-and-forward makes poor use of communication resources.
- Packet routing breaks messages into packets and pipelines them through the network.
- Since packets may take different paths, each packet must carry routing information, error checking, sequencing, and other related header information.
- The total communication time for packet routing is approximated by: $t_{comm} = t_s + t_h l + t_w m.$
- Here factor t_w also accounts for overheads in packet headers.

Message Passing Costs in Parallel Computers

Cut-Through Routing

- Takes the concept of packet routing to an extreme by further dividing messages into basic units called **flits** or flow control digits.
- Since flits are typically small, the header information must be minimized.
- This is done by forcing all flits to take the same path, in sequence.
- A tracer message first programs all intermediate routers. All flits then take the same route.
- Error checks are performed on the entire message, as opposed to flits.
- No sequence numbers are needed.

Message Passing Costs in Parallel Computers

Cut-Through Routing

- The total communication time for cut-through routing is approximated by:

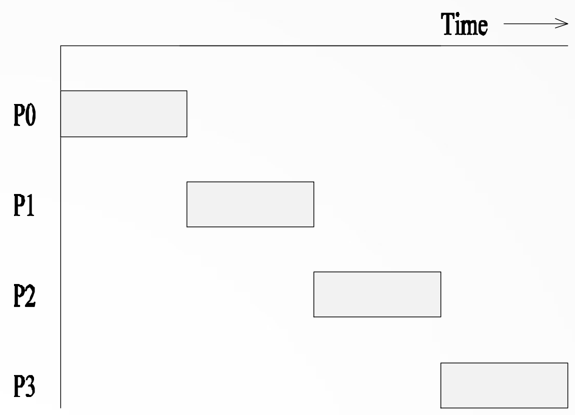
$$t_{comm} = t_s + t_h l + t_w m.$$

- This is identical to packet routing, however, t_w is typically much smaller.

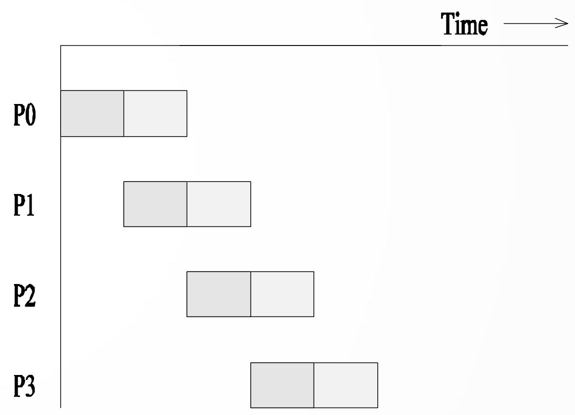
Message Passing Costs in Parallel Computers

(a) through a store-and-forward communication network;

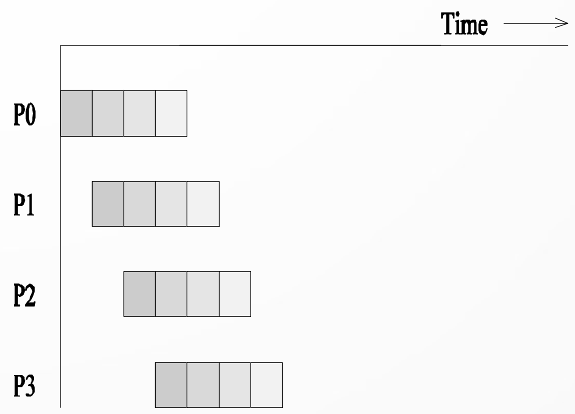
b) and (c) extending the concept to cut-through routing.



(a) A single message sent over a store-and-forward network



(b) The same message broken into two parts and sent over the network.



(c) The same message broken into four parts and sent over the network.

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- The cost of communicating a message between two nodes *l* hops away using cut-through routing is given by

$$t_{comm} = t_s + l t_h + t_w m.$$

- In this expression, t_h is typically smaller than t_s and t_w. For this reason, the second term in the RHS does not show, particularly, when *m* is large.

- For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- It is important to note that the original expression for communication time is valid for only uncongested networks.
- Different communication patterns congest different networks to varying extents.
- It is important to understand and account for this in the communication time accordingly.

Questions

References

1. Flynn, M., “Some Computer Organizations and Their Effectiveness,” IEEE Transactions on Computers, Vol. C-21, No. 9, September 1972.
2. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City, CA: Benjamin/Cummings.
3. Quinn, M. J. Parallel Programming in C with MPI and OpenMP,(2003).