CSCI 5451: Introduction to Parallel Computing

Lecture 2: Parallel Architectures

computer science
& engineering

UNIVERSITY OF MINNESOTA
Driven to Discover®

Announcement

- Slack Channel is up
    - Use this link to get access
    - As a reminder, this is where course communications will take place

Lecture Overview

- ❑ Logical Parallel Hardware Models
  - Flynn's Taxonomy
  - Shared vs. Distributed Memory
  - Architecture of Ideal Computer
- ❑ Physical Parallel Hardware models
  - Superscalar Architectures (Early Parallelism)
  - Network Topology (Choices)
    - ✔ Direct vs. Indirect
    - ✔ Connecting Processors to Memory
    - ✔ Connecting Processors to one another
  - Network Topology (Evaluations)

Lecture Overview

- ❑ **Logical Parallel Hardware Models**
  - **Flynn’s Taxonomy**
  - Shared vs. Distributed Memory
  - Architecture of Ideal Computer
- ❑ Physical Parallel Hardware models
  - Superscalar Architectures (Early Parallelism)
  - Network Topology (Choices)
    - ✓ Direct vs. Indirect
    - ✓ Connecting Processors to Memory
    - ✓ Connecting Processors to one another
  - Network Topology (Evaluations)

Recall Von Neumann Execution

- Fetch next instruction from memory into instruction registers
- Fetch operands from memory into data registers
- Execute instruction
- Load result into memory
- Repeat: fetch next instruction…

Recall Von Neumann Execution

- Fetch next instruction from memory into instruction registers
- Fetch operands from memory into data registers
- Execute instruction
- Load result into memory
- Repeat: fetch next instruction…

Fetch One Instruction

Recall Von Neumann Execution

- Fetch next instruction from memory into instruction registers
- $\boxed{\text{Fetch operands from memory into data registers}}$
- Execute instruction
- Load result into memory
- Repeat: fetch next instruction…

Fetch One Operand (Datum)

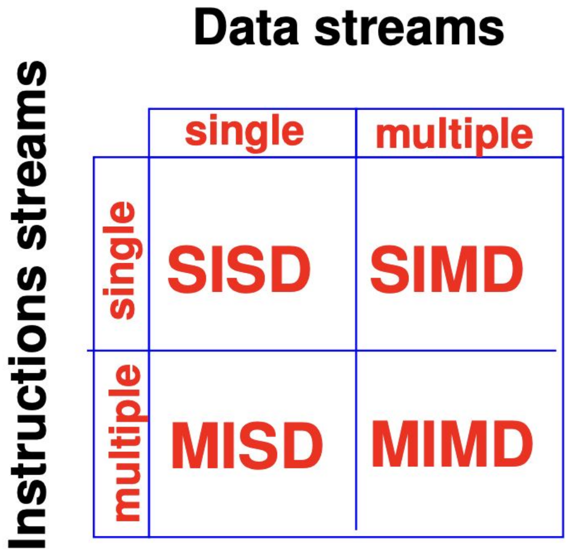
Recall Von Neumann Execution

- Fetch next instruction from memory into instruction registers
- Fetch operands from memory into data registers
- Execute instruction
- Load result into memory
- Repeat: fetch next instruction…

What if we parallelized these steps?

Flynn's taxonomy

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

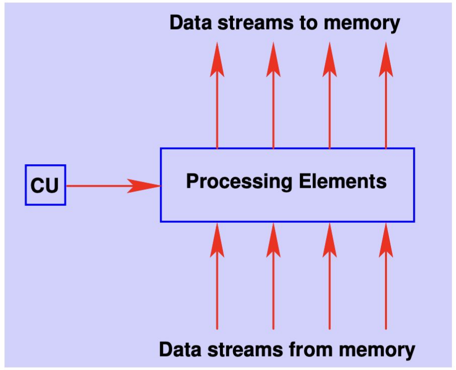


Flynn’s taxonomy

- ❑ Single Instruction Single Data (SISD)

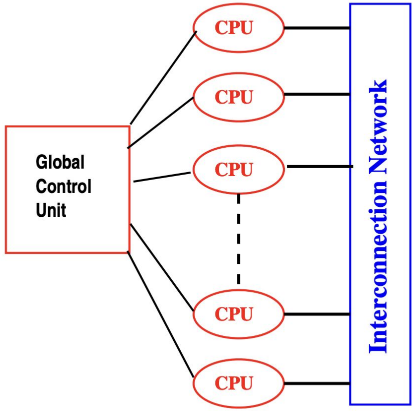
Standard Von Neumann Architecture

Flynn’s taxonomy



- Single Instruction Multiple Data (SIMD)

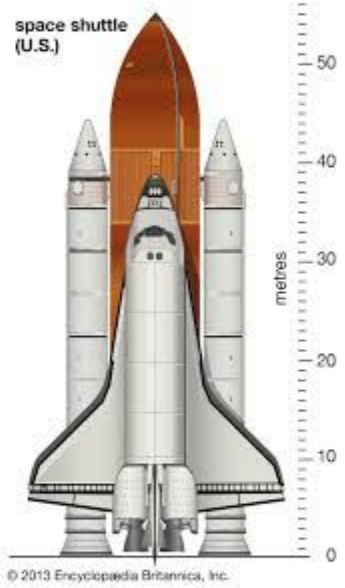
Pipelined Vector Computers, AVX, part of GPU computing



Flynn's taxonomy

- ❑ Multiple Instruction Single Data (MISD)

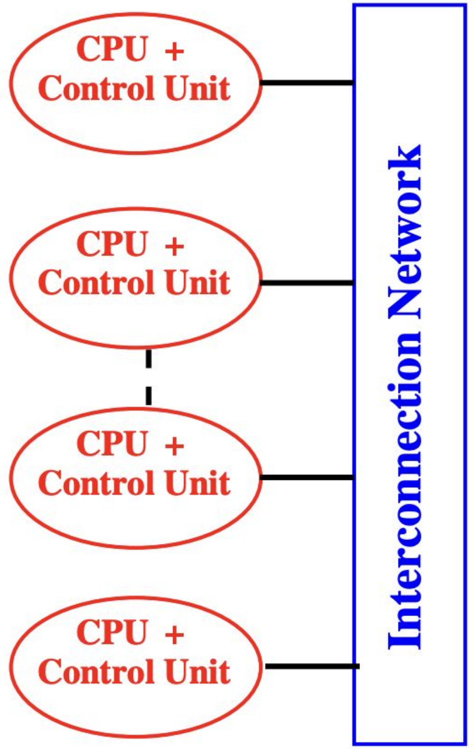
Very rare - used on Space Shuttle for redundancy of checking data



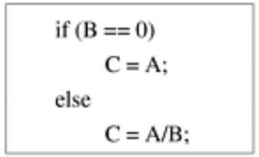
Flynn’s taxonomy

Very common - any time multiple processors/computers must be linked together

- Multiple Instruction Multiple Data (MIMD)



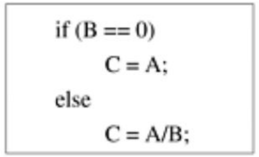
SIMD Example



```
if (B == 0)
        C = A;
else
        C = A/B;
```

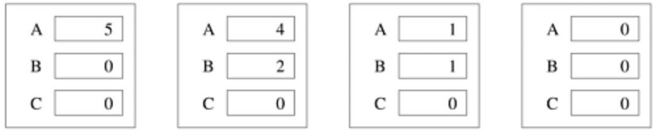
(a)

SIMD Example



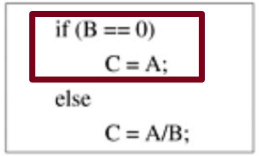
```
if (B == 0)
        C = A;
else
        C = A/B;
```

(a)



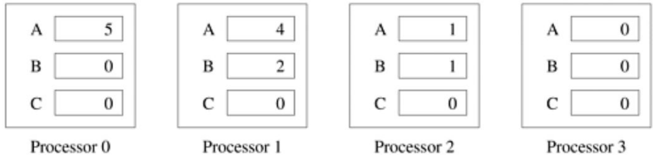
Initial values

SIMD Example

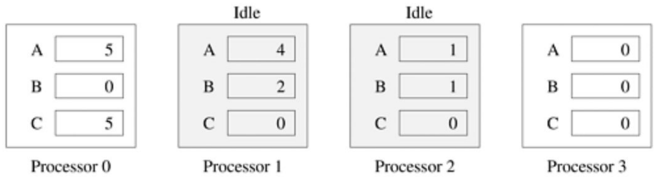


```
if (B == 0)
        C = A;
else
        C = A/B;
```

(a)

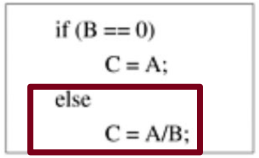


Initial values



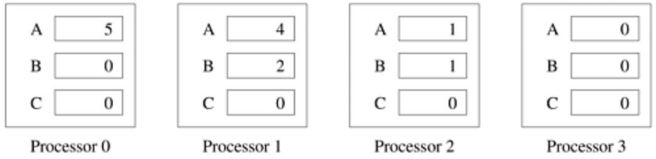
Step 1

SIMD Example

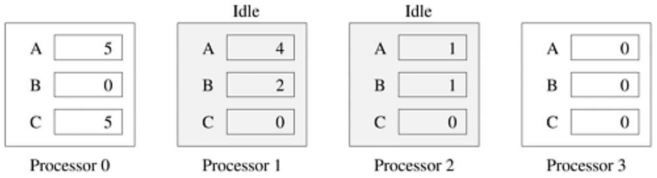


```
if (B == 0)
        C = A;
else
        C = A/B;
```

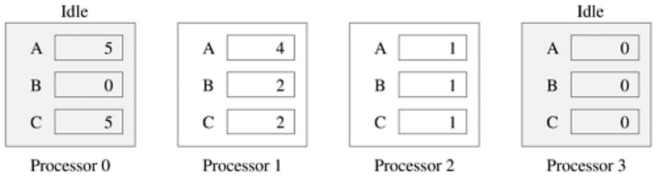
(a)



Initial values



Step 1

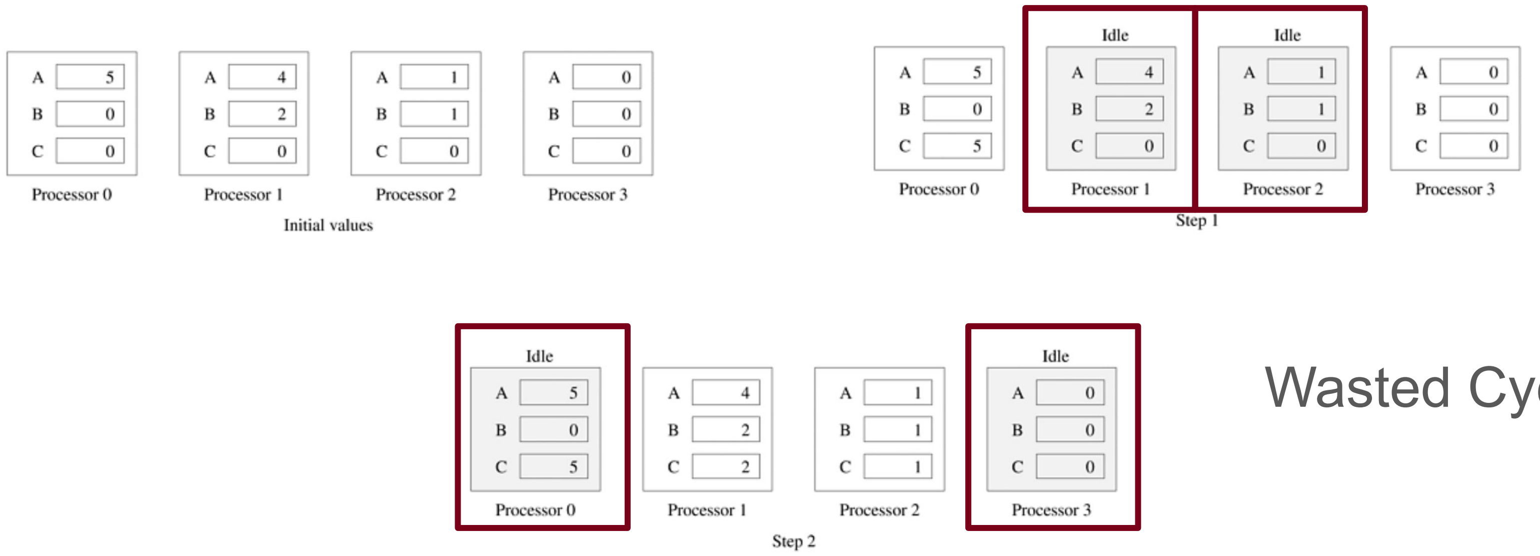


Step 2

SIMD Example

```
if (B == 0)
        C = A;
else
        C = A/B;
```

(a)



Wasted Cycles!

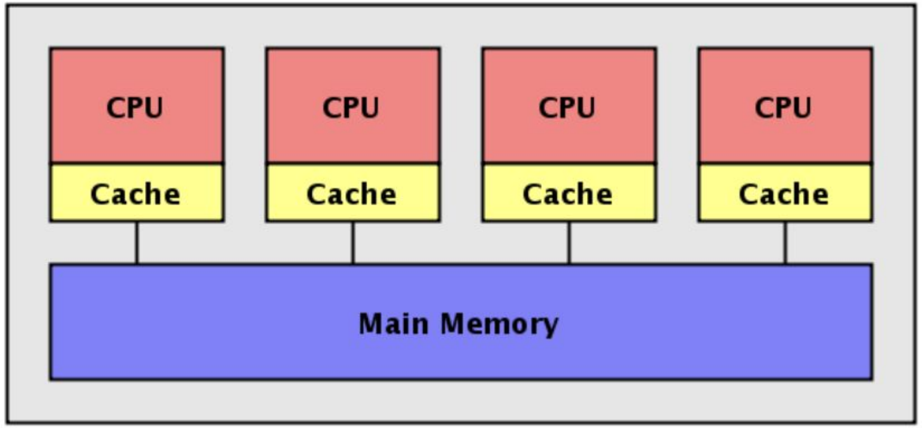
Lecture Overview

- ❑ **Logical Parallel Hardware Models**
  - Flynn's Taxonomy
  - **Shared vs. Distributed Memory**
  - Architecture of Ideal Computer
- ❑ Physical Parallel Hardware models
  - Superscalar Architectures (Early Parallelism)
  - Network Topology (Choices)
    - ✔ Direct vs. Indirect
    - ✔ Connecting Processors to Memory
    - ✔ Connecting Processors to one another
  - Network Topology (Evaluations)

Where are we loading instructions/data from?

Shared memory

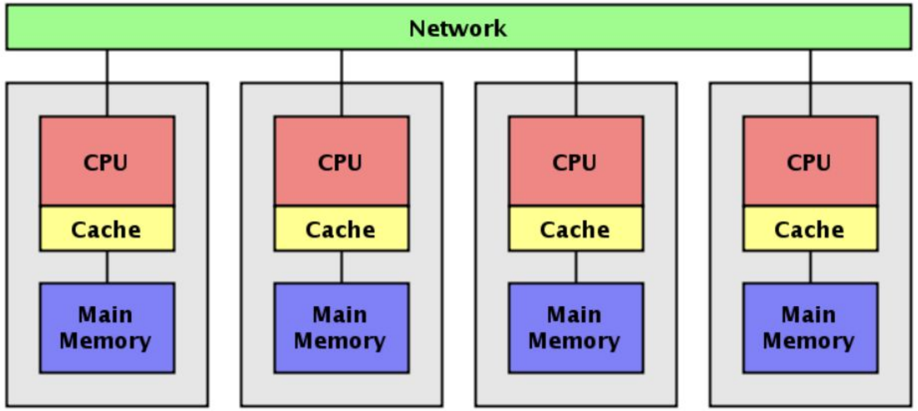
- ❑ Very little local/cache memory
- ❑ Sharing information not explicitly required
- ❑ Convenience → No having to read/write between processes
- ❑ Challenge → Have to be mindful not to overwrite information which other processes are working on



Source: Kaminsky/Parallel Java

Distributed Memory Models

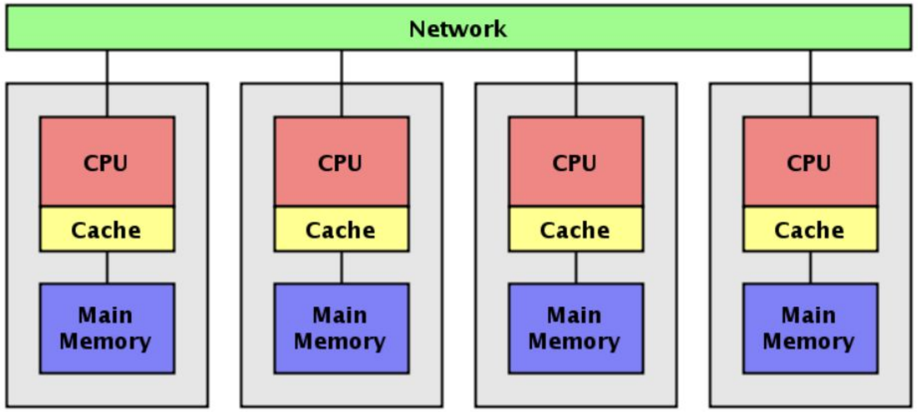
- No shared global address space
- Each processor has its own memory
- Far easier to scale (things are plug & play)
- Puts more responsibility onto the programmer for explicitly sending/receiving information



Source: Kaminsky/Parallel Java

Distributed Memory Models

- No shared global address space
- Each processor has its own memory
- Far easier to scale (things are plug & play)
- Puts more responsibility onto the programmer for explicitly sending/receiving information



Source: Kaminsky/Parallel Java

Lecture Overview

- **Logical Parallel Hardware Models**
  - Flynn’s Taxonomy
  - Shared vs. Distributed Memory
  - **Architecture of Ideal Computer**
- Physical Parallel Hardware models
  - Superscalar Architectures (Early Parallelism)
  - Network Topology (Choices)
    - ✔ Direct vs. Indirect
    - ✔ Connecting Processors to Memory
    - ✔ Connecting Processors to one another
  - Network Topology (Evaluations)

PRAM Model (Idealistic View)

Parallel Random Access Machine → extends Random Access Machine

"*p* processors and a global memory of unbounded size that is uniformly accessible to all processors"

Grama et. al

PRAM Model (Idealistic View)

- **Exclusive-read, exclusive-write (EREW) PRAM**
- **Concurrent-read, exclusive-write (CREW) PRAM**
- **Exclusive-read, concurrent-write (ERCW) PRAM**
- **Concurrent-read, concurrent-write (CRCW) PRAM**

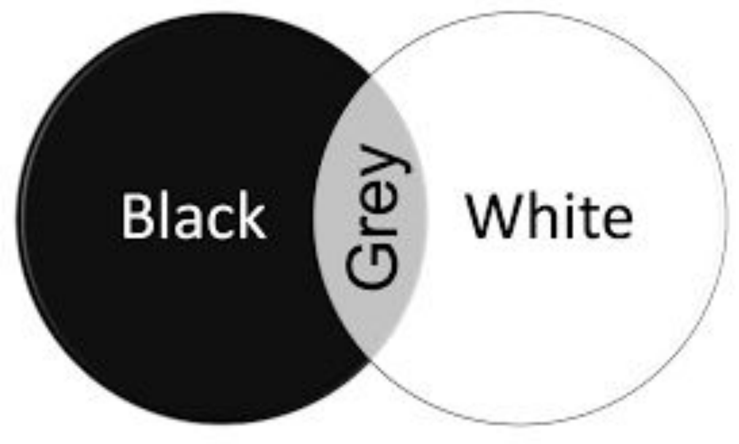
PRAM Model (Idealistic View)

Approaches for resolving concurrent writes

- ❑ Common → Concurrent write is allowed if all the values from all processors attempting to write is the same
- ❑ Arbitrary → Processor who gets to write is chosen at random
- ❑ Priority → All processes are placed into a prioritized list - the processor with the highest priority is chosen for the write

Boundaries of Classification in Complex Systems

- ❑ Like many theoretical models, these models break in contact with reality
- ❑ Their boundaries are less black-white & more gray
- ❑ Use them as intuition for systems, not laws



Boundaries of Classification in Complex Systems

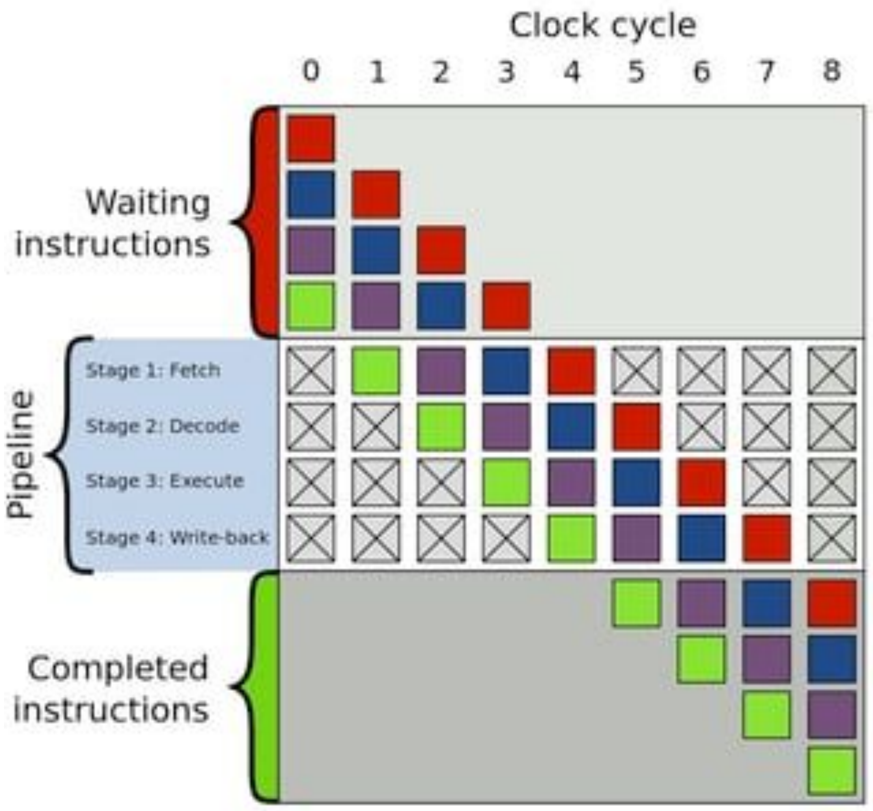
- GPUs
  - Make use of both SIMD + MIMD concepts concurrently
- Caching/Networked Storage
  - Even in shared-address systems, there will be cached memory reserved for only the given process (i.e. they will have *some* states which are not shared)
  - There is almost always some shared space of storage for distributed memory programs (i.e. they will share *some* state)
- PRAM
  - Infinite memory & uniform access are not possible

Lecture Overview

- ❑ Logical Parallel Hardware Models
  - Flynn’s Taxonomy
  - Shared vs. Distributed Memory
  - Architecture of Ideal Computer
- ❑ **Physical Parallel Hardware models**
  - **Superscalar Architectures (Early Parallelism)**
  - Network Topology (Choices)
    - ✔ Direct vs. Indirect
    - ✔ Connecting Processors to Memory
    - ✔ Connecting Processors to one another
  - Network Topology (Evaluations)

What is Pipelining?

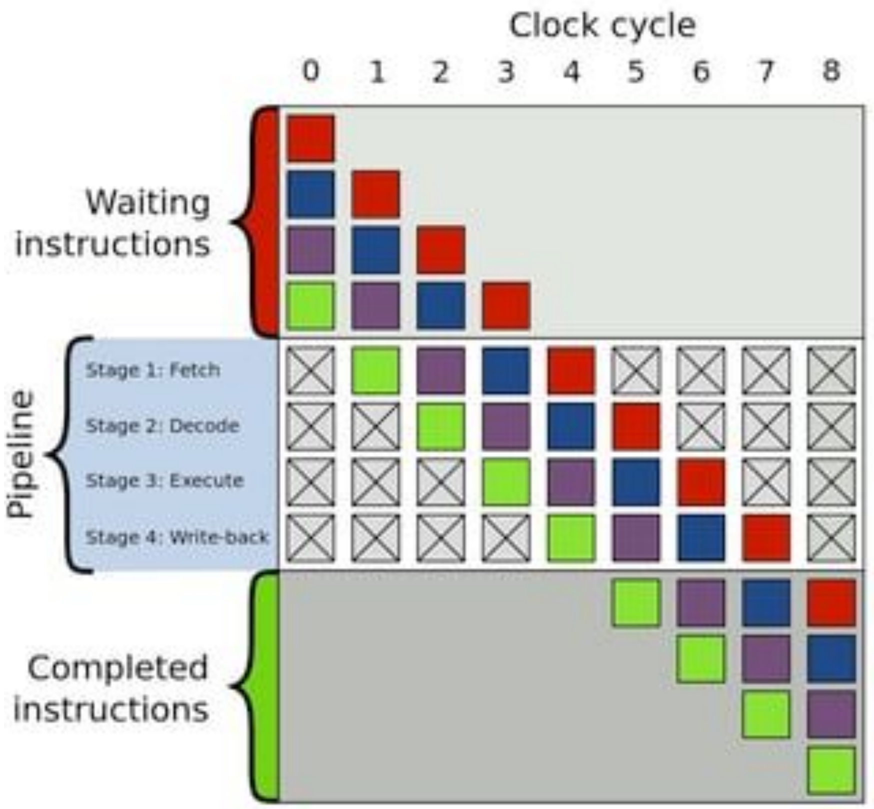
- ❑ Like an assembly line
- ❑ Technique to overlap instruction execution stages,
- ❑ Divides instruction processing into sequential stages
- ❑ Multiple instructions processed simultaneously, each at a different stage of the pipeline.



What is Pipelining?

Modern CPUS have ~5=20+ stages

- ❑ Like an assembly line
- ❑ Technique to overlap instruction execution stages,
- ❑ Divides instruction processing into sequential stages
- ❑ Multiple instructions processed simultaneously, each at a different stage of the pipeline.



More than one instruction at a time?

- Many more transistors == more units for identical work
  - More +, -, *, / (etc.)
  - At each stage in the pipeline, there is usually more than 1 element capable of performing that stage
- Analogizing back to an assembly line → each stage of production can have more than one (1) person doing that task.
- This kind of duplication allows for hardware-levels of *implicit* parallelism

More than one instruction at a time?

- Many more transistors == more units for identical work
  - More +, -, *, / (etc.)
  - At each stage in the pipeline, there is usually more than 1 element capable of performing that stage
- Analogizing back to an assembly line → each stage of production can have more than one (1) person doing that task.
- This kind of duplication allows for hardware-levels of *implicit* parallelism

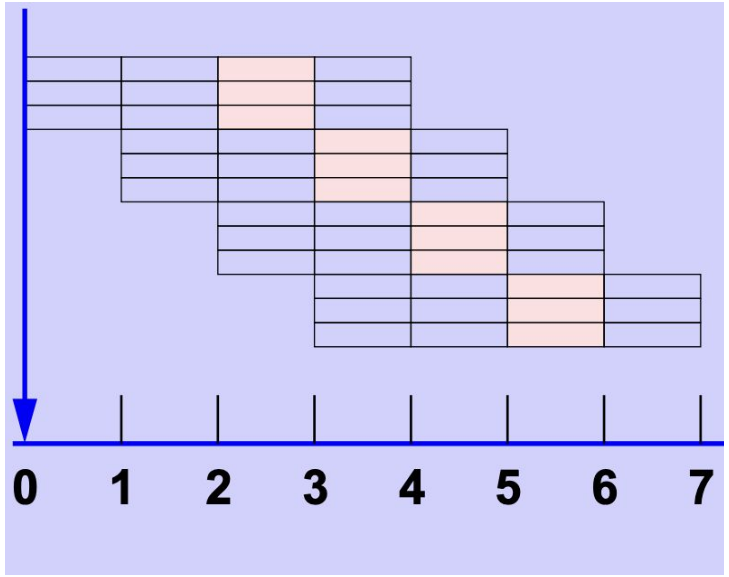
Superscalar Execution

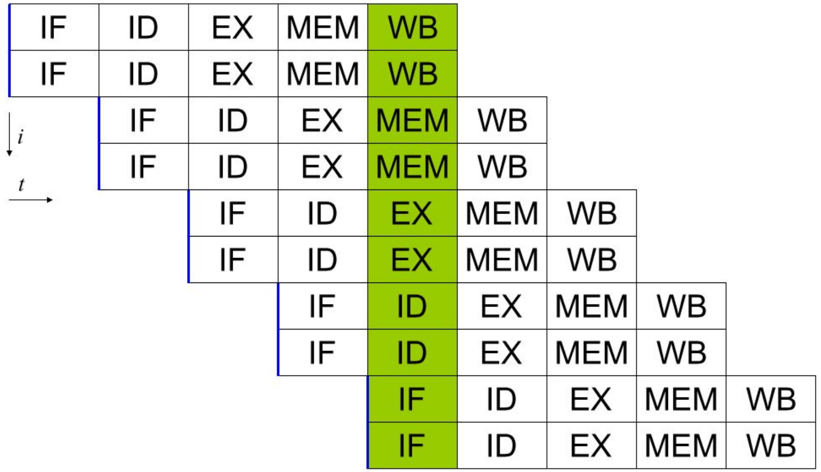
How Does the Hardware Know What to Parallelize?

Compiler Level Dependency Analysis

Hardware Level Dependency Analysis

M-Way Superscalar Execution

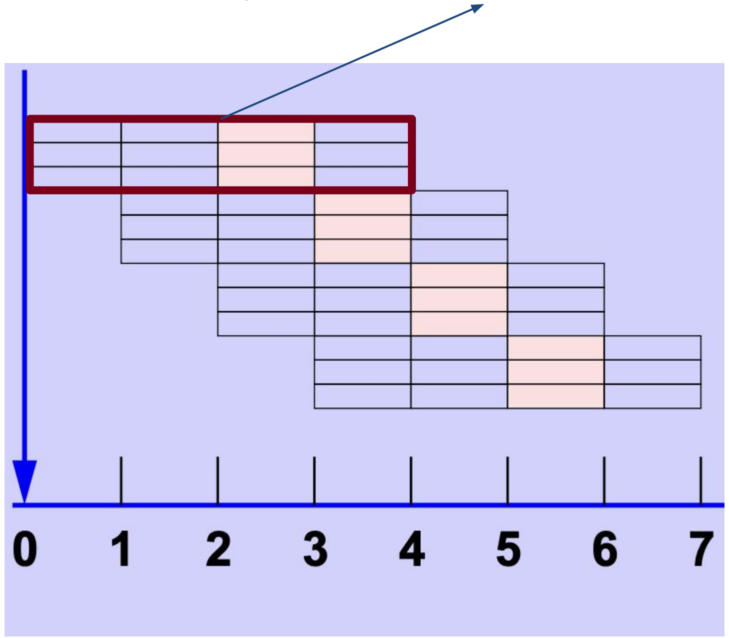


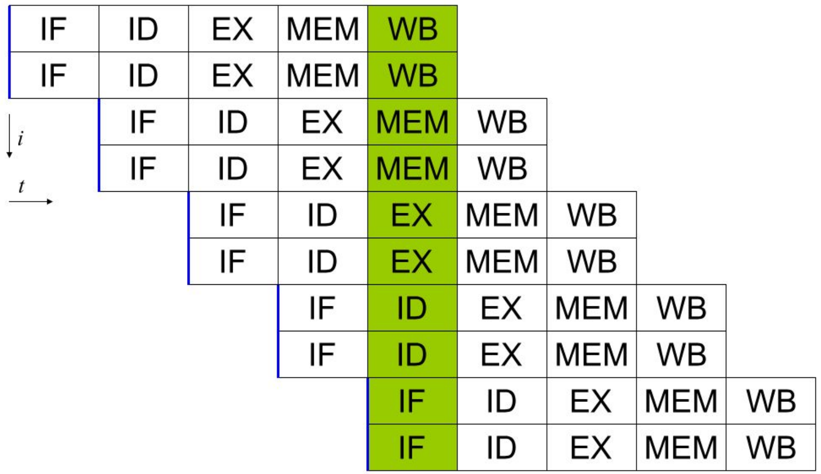


IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

M-Way Superscalar Processor

3-way Superscalar Processor

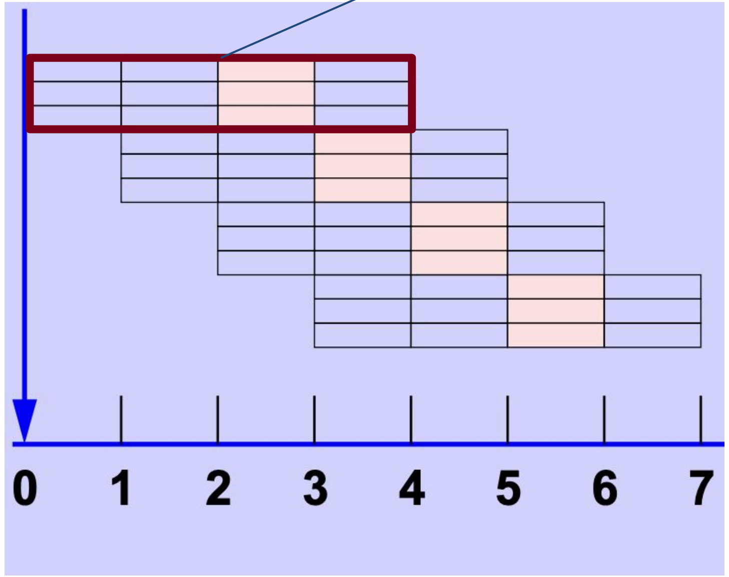




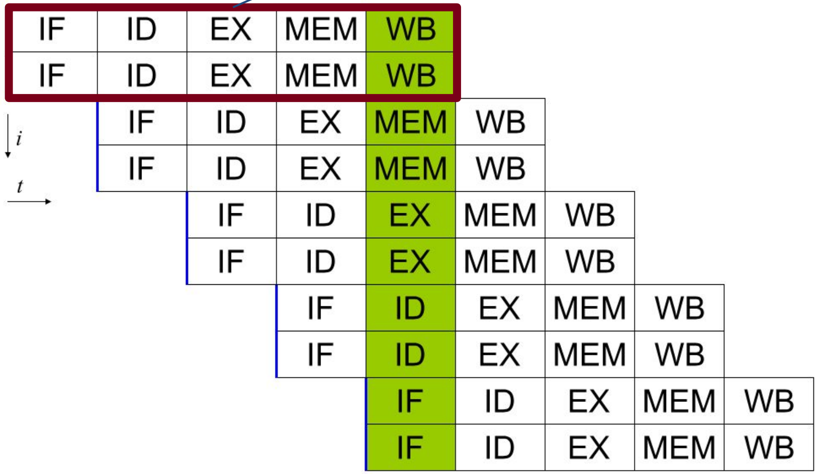
IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

M-Way Superscalar Processor

3-way Superscalar Processor



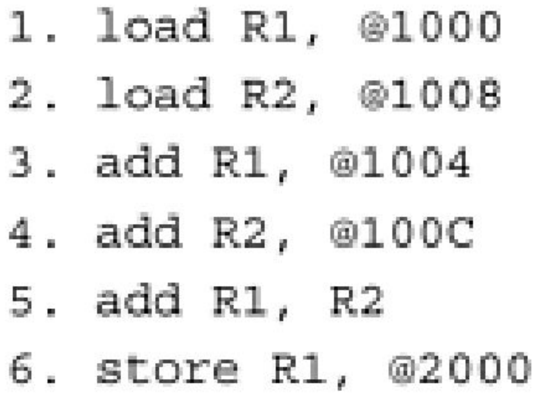
2-way Superscalar Processor



IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

Superscalar Example

- How will the following code stub (which sums a list of 4 numbers) be computed on a 2-way superscalar processor? Which steps will run concurrently?

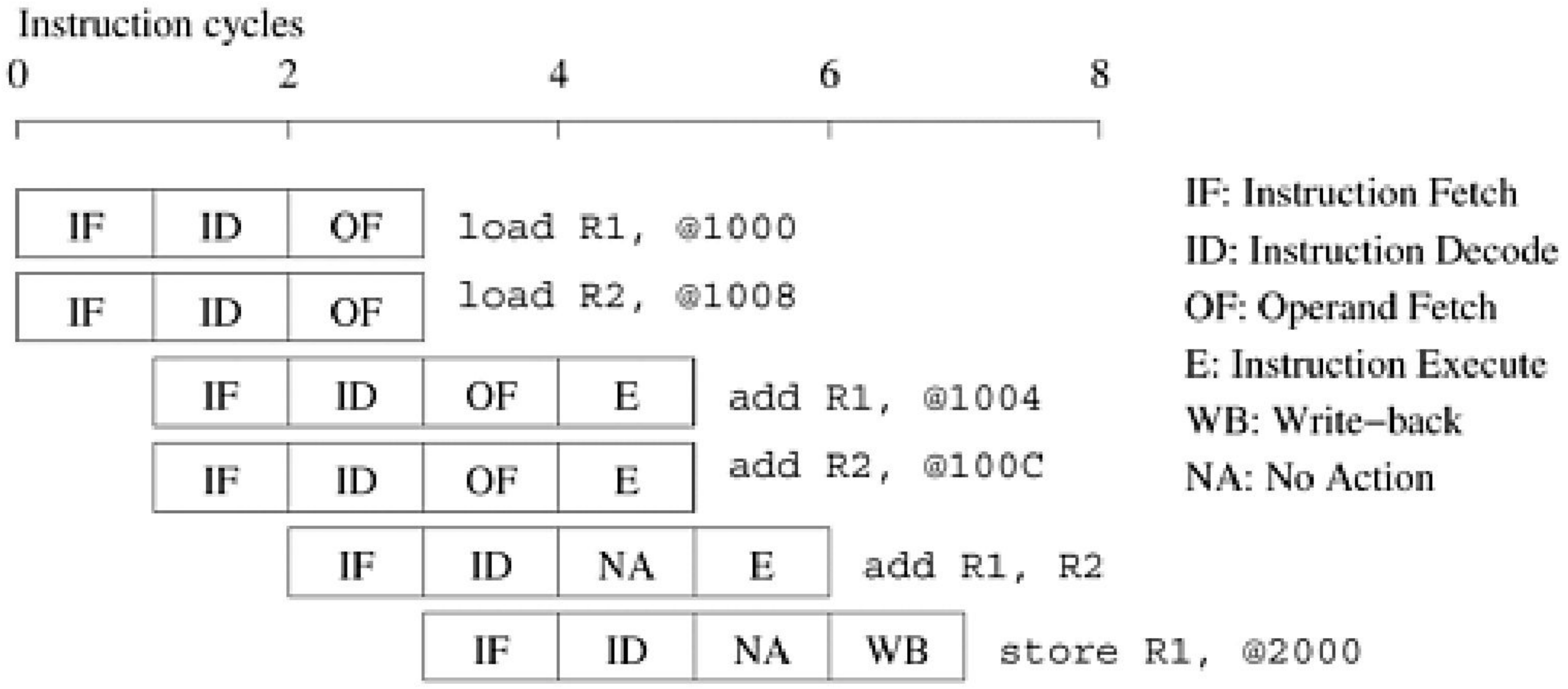


```
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(i)

Superscalar Example

- ANS: (1, 2) , (3, 4)



Why couldn't hardware & compilers just solve parallelism?

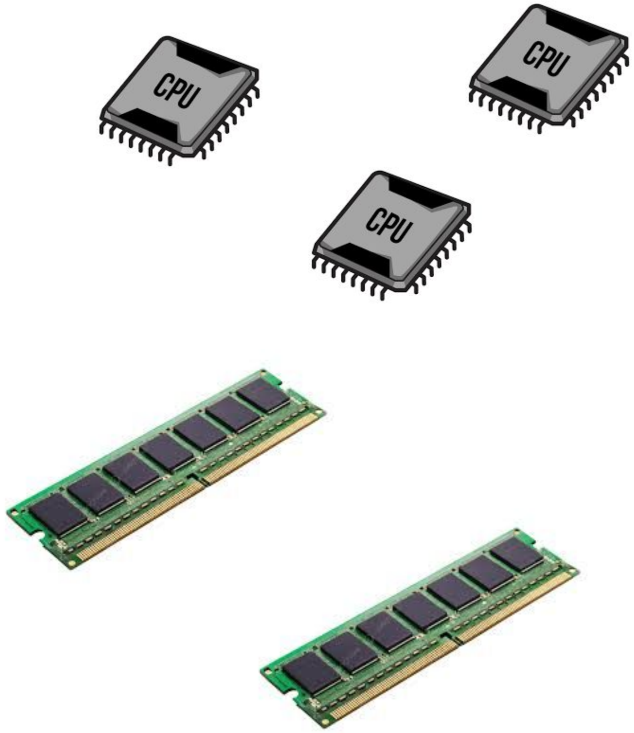
- ❑ Instruction-lookahead (seeing how dependencies will exactly work out) is quite difficult
- ❑ Parallelizing via hardware logic alone is quite complicated
- ❑ Compilers help, but only so much
- ❑ The increasing complexity of the hardware and the general nature of computation means that relying on these superscalar architectures alone is not feasible (most mainstream m-way superscalar processors in use today have m ~4-6)

Lecture Overview

- Logical Parallel Hardware Models
  - Flynn's Taxonomy
  - Shared vs. Distributed Memory
  - Architecture of Ideal Computer
- **Physical Parallel Hardware models**
  - Superscalar Architectures (Early Parallelism)
  - **Network Topology (Choices)**
    - ✔ **Direct vs. Indirect**
    - ✔ **Connecting Processors to Memory**
    - ✔ **Connecting Processors to one another**
  - Network Topology (Evaluations)

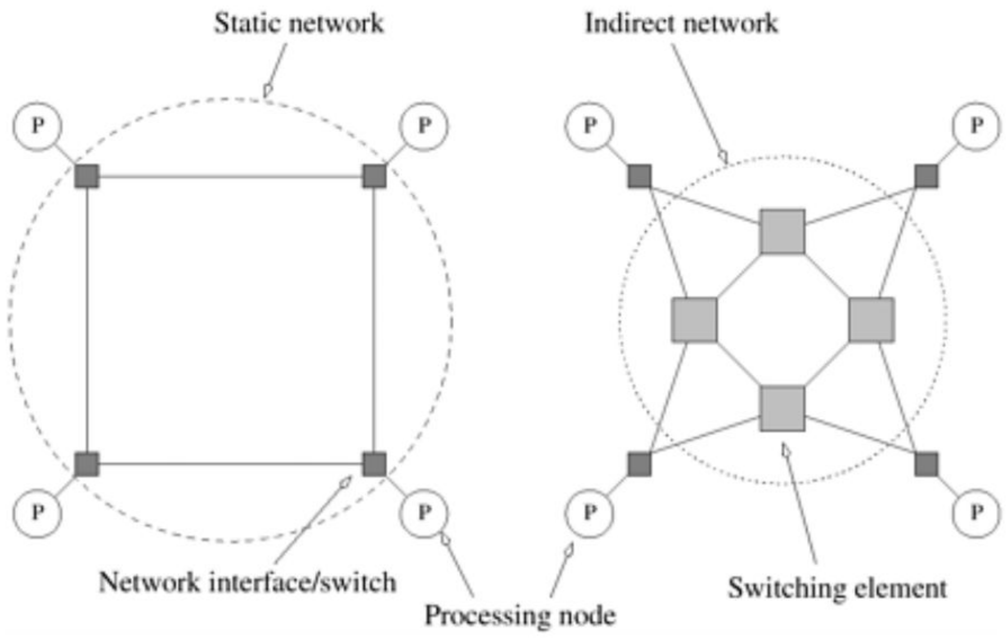
Network Topology Choices

- How are we going to connect a large set of computers to a large set of memory banks? How can we reduce communication times and make our programs as fast as possible?
- We have a series of tools to choose from (not necessarily mutually exclusive) which have tradeoffs.
- More interconnects == $$$$$



Should memory/processors connect directly to one another or via switches?

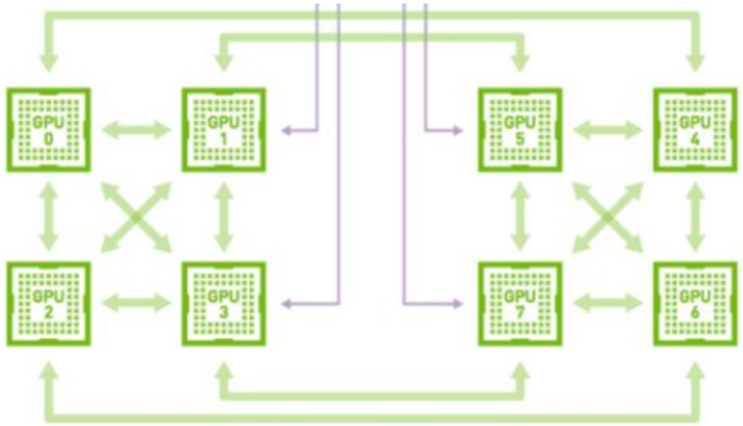
Static (Direct) vs. Dynamic (Indirect) Networks



Direct vs Indirect (NVLink/NVSwitch)

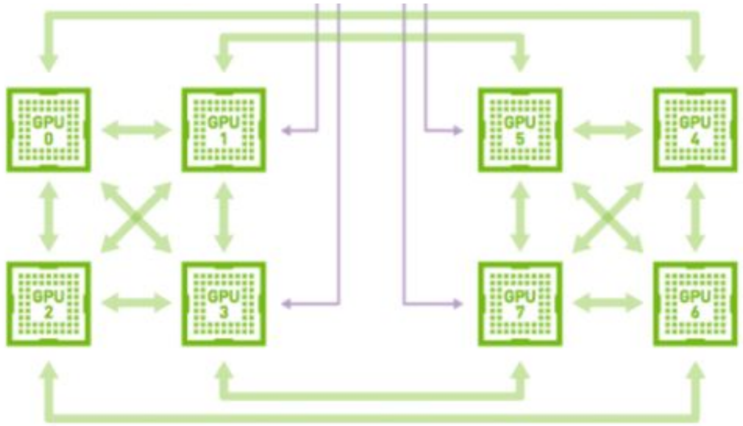
Direct vs Indirect (NVLink/NVSwitch)

Direct → NVLink

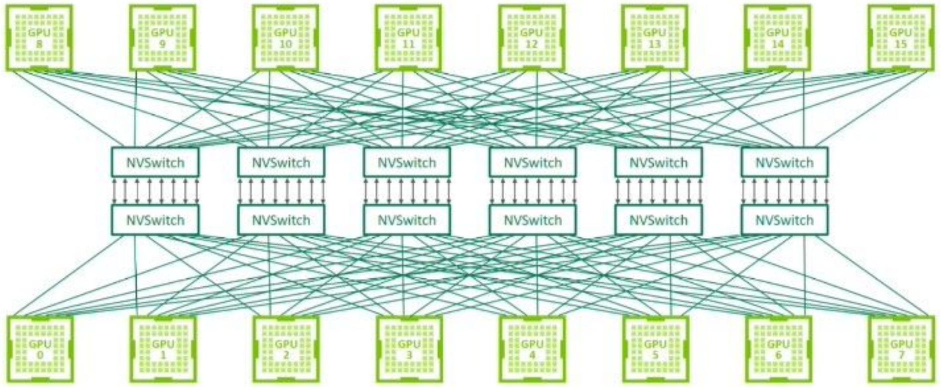


Direct vs Indirect (NVLink/NVSwitch)

Direct → NVLink



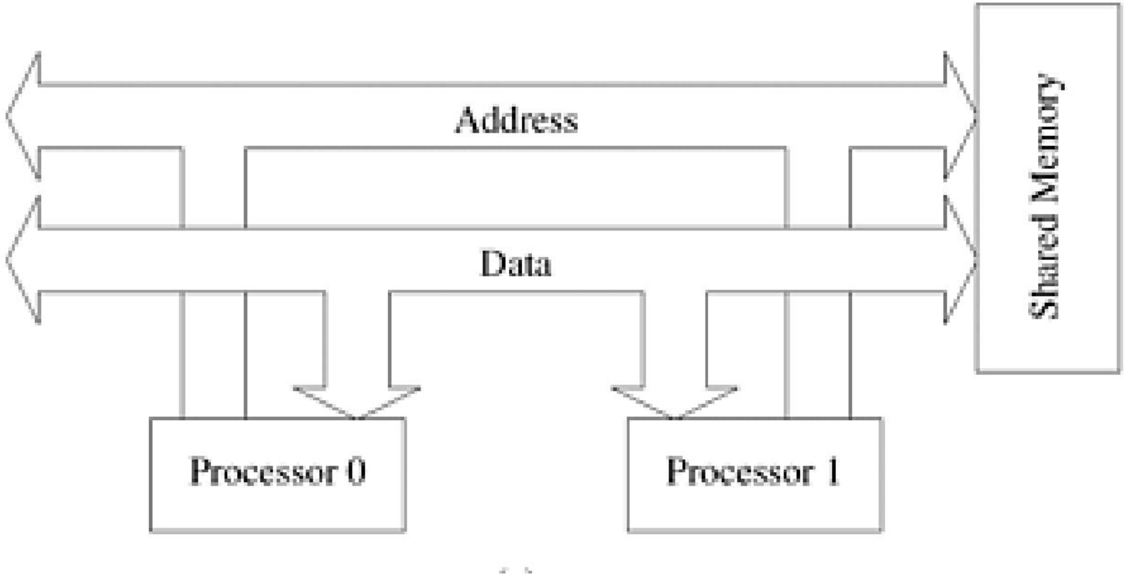
Indirect → NVSwitch



How to connect Processors to Memory?

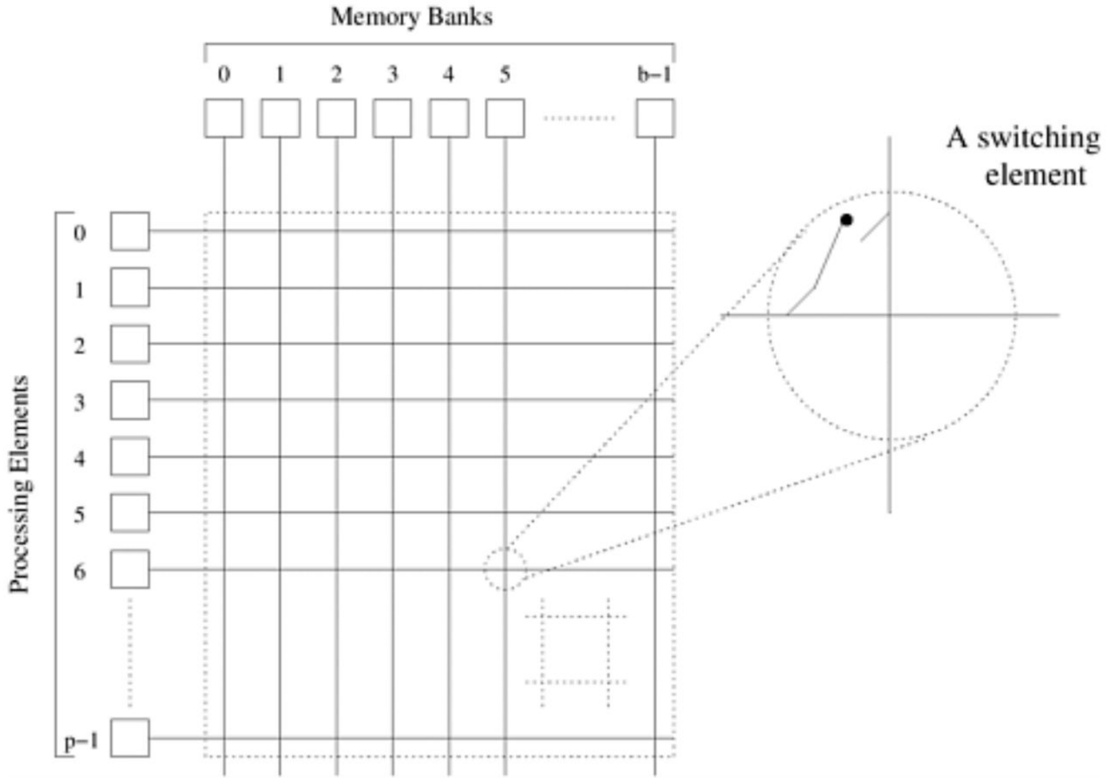
Memory Connection - Bus

- ❑ Connects Processors to a shared memory bank via a direct connection
- ❑ Cheap, but unscalable with more processors, memory banks



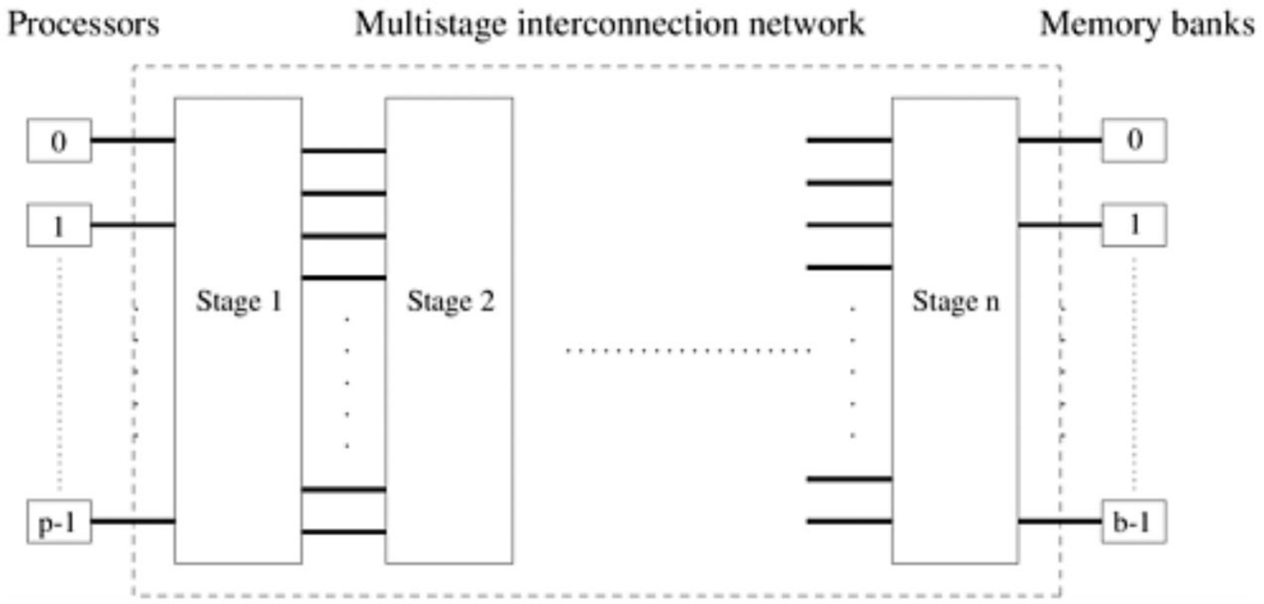
Memory Connection - Crossbar

- Uses switching nodes at each interconnect between PEs and Banks
- Non-Blocking
- Very Expensive
- More scalable in performance



Memory Connection - Multistage

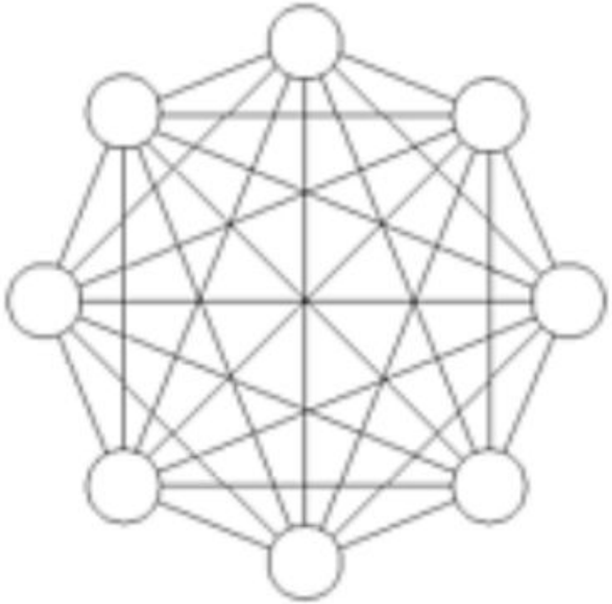
- ❑ Shuffles connections at each stage
- ❑ Worse than Bus on price, better than Crossbar
- ❑ Worse than Crossbar on scalable performance, better than Bus
- ❑ Uses log(p) stages

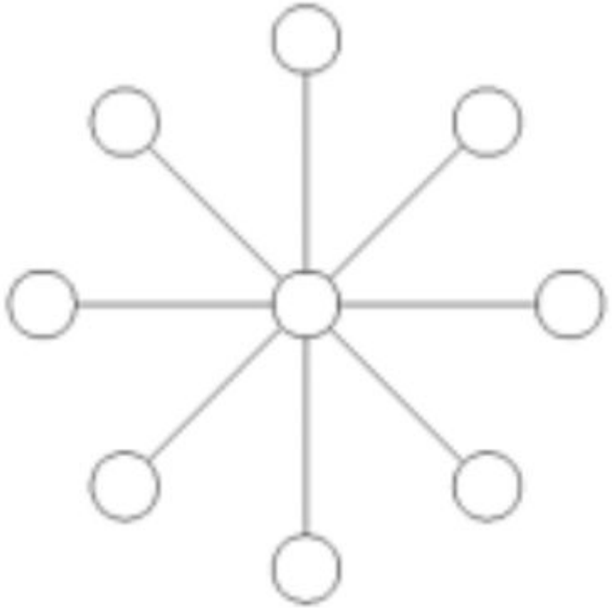


Lecture concluded here (remaining elements in this set of slides covered in next lecture)

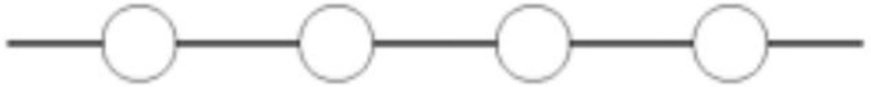
How to connect processors to one another?

Processor Connections

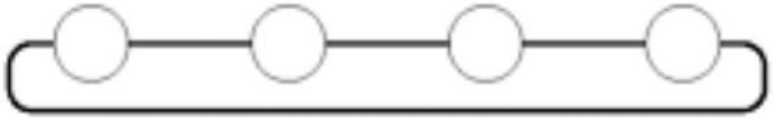

Fully Connected


Star

Processor Connections

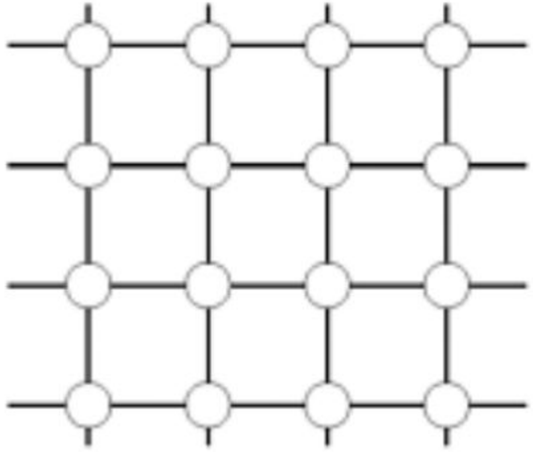


Linear Array
w/o Wraparound

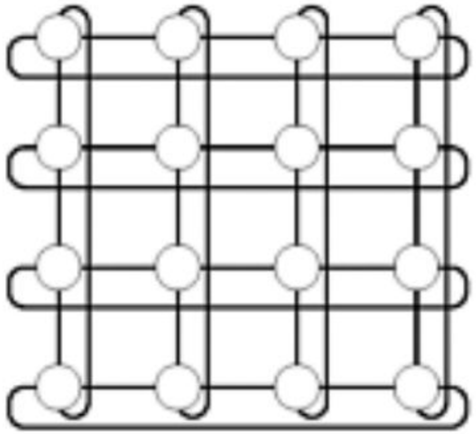


Linear Array
w/Wraparound

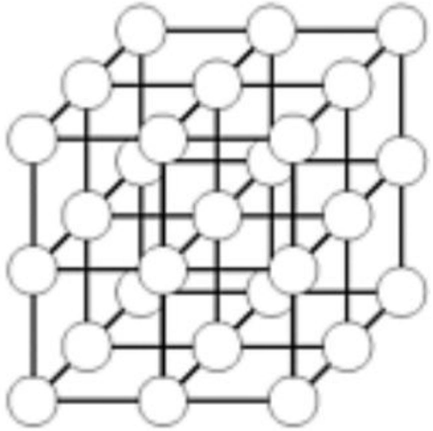
Processor Connections



2-d Mesh
w/o Wraparound



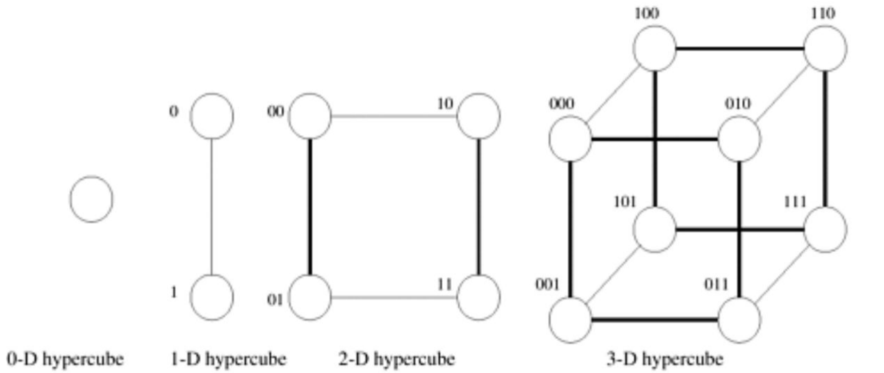
2-d Mesh
w/Wraparound
(2-d torus)

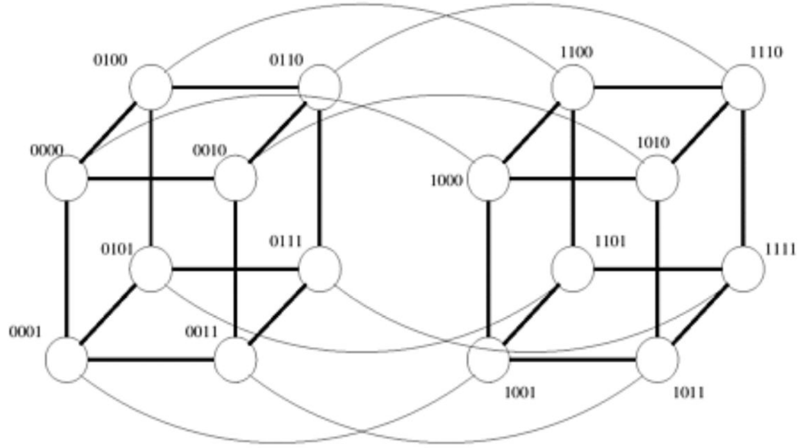


3-d Mesh
w/o Wraparound

Processor Connections

Hypercube Construction





Lecture Overview

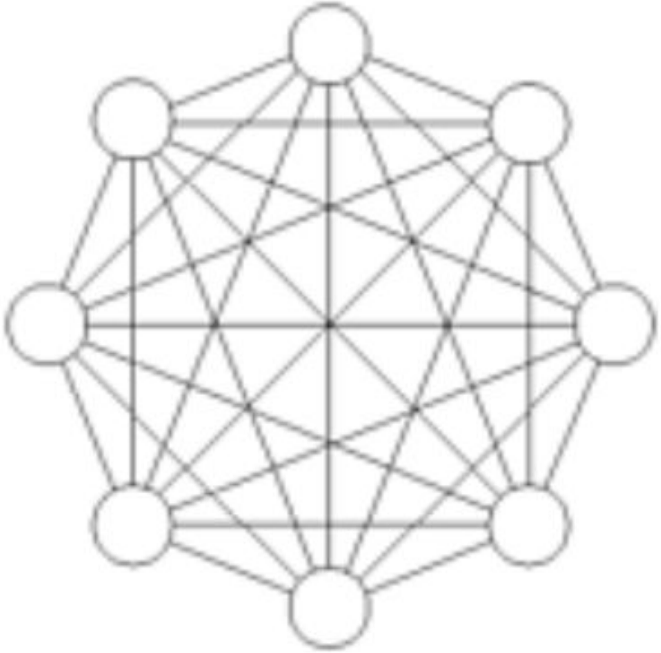
- ❑ Logical Parallel Hardware Models
  - Flynn’s Taxonomy
  - Shared vs. Distributed Memory
  - Architecture of Ideal Computer
- ❑ **Physical Parallel Hardware models**
  - Superscalar Architectures (Early Parallelism)
  - Network Topology (Choices)
    - ✔ Direct vs. Indirect
    - ✔ Connecting Processors to Memory
    - ✔ Connecting Processors to one another
  - **Network Topology (Evaluations)**

Network Evaluation

- ❑ Diameter → The maximum distance between any two nodes in the network, where the distance between any two nodes is defined as the shortest path between them
- ❑ Connectivity → Minimum number of links which must be cut to split the network in two
- ❑ Bisection width → Minimum number of links which must be cut to ensure that the network is partitioned into two equal halves.
- ❑ Cost (no of links) → How many links are required for this topology?

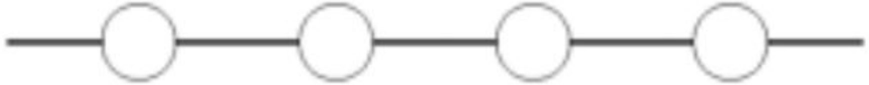
Example : Fully Connected

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p - 1$	$p(p - 1)/2$



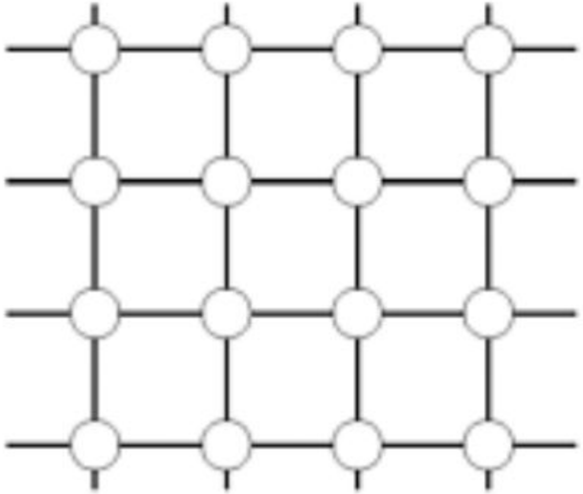
Example Linear Array w/o Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Linear array	$p-1$	1	1	$p-1$



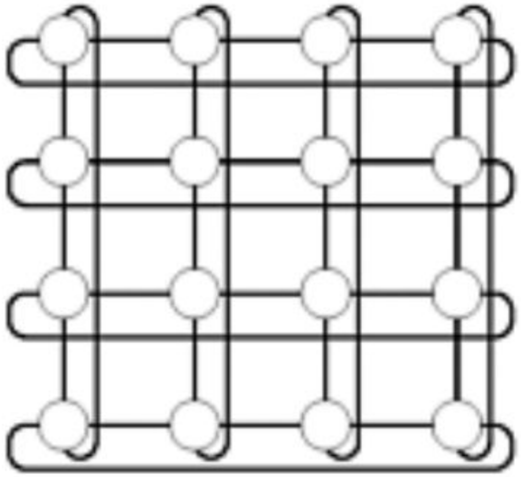
Example: 2-d Mesh w/o Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
2-D mesh, no wraparound	$2(\sqrt{p} - 1)$	$\sqrt{p}$	2	$2(p - \sqrt{p})$



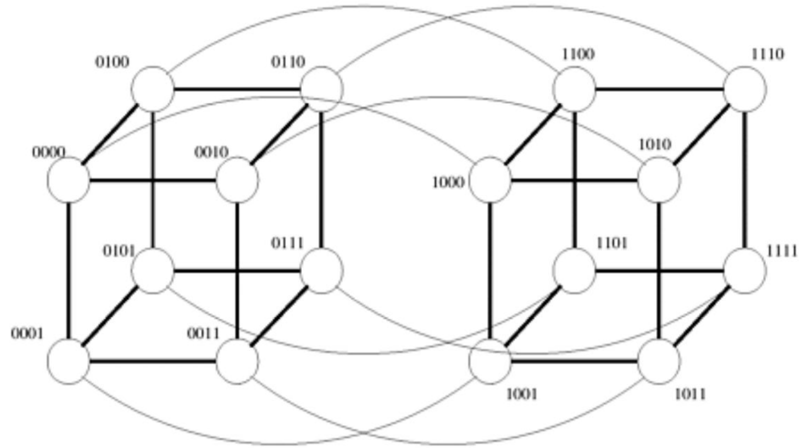
Example: 2-d Mesh w/Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$



Example: Hypercube

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Hypercube	$\log p$	$p/2$	$\log p$	$(p \log p)/2$



Full Table (For Future Reference)

- Feel free to explore further on your own

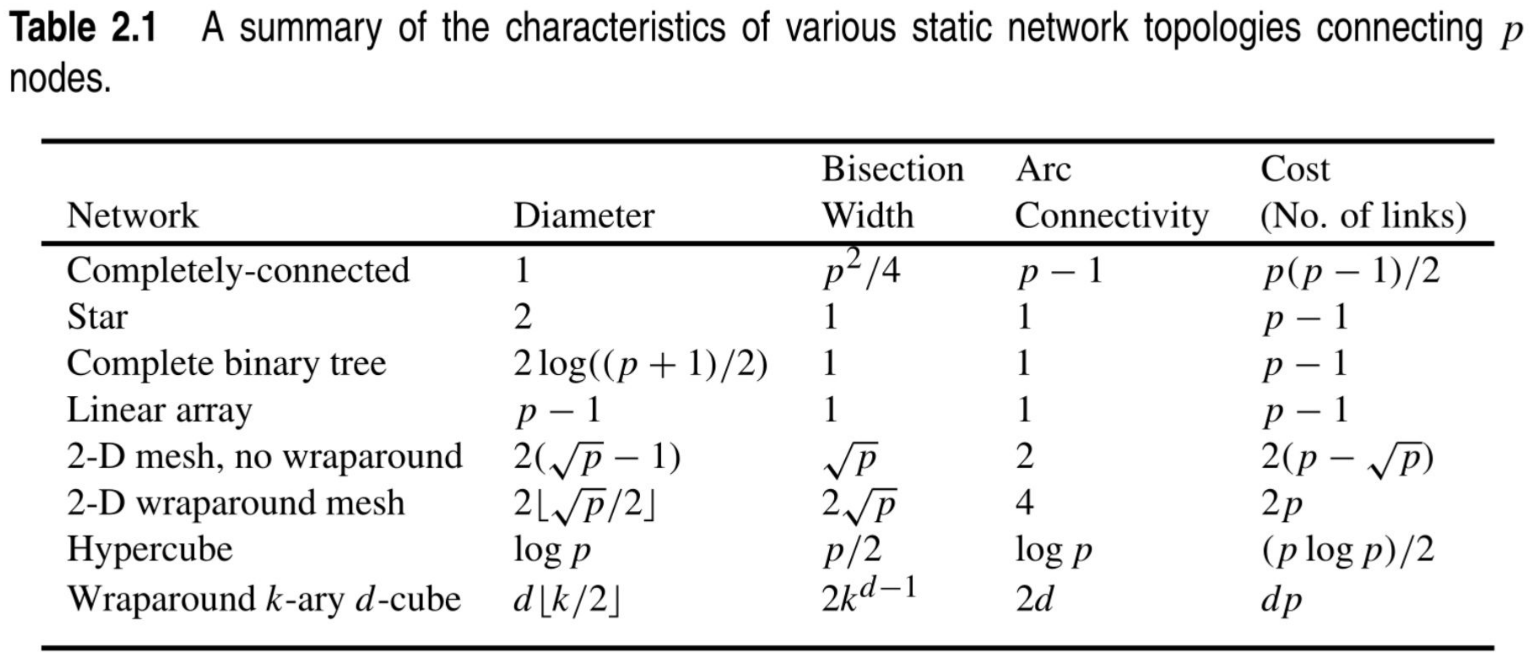


Table 2.1 A summary of the characteristics of various static network topologies connecting p nodes.

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp