Parallel Architectures

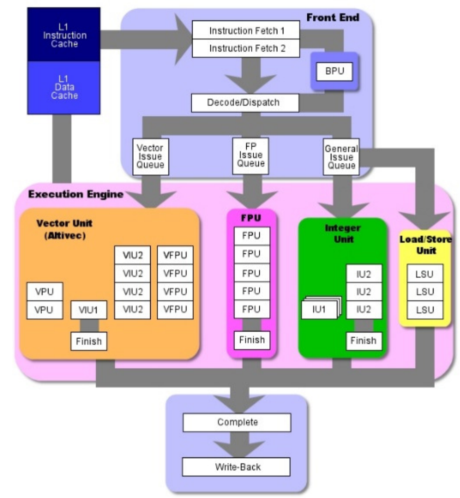
CS121 Parallel Computing

Spring 2018

Parallelism hierarchy

- Parallelism exists at many different levels of a computer system.
  - Within a single core.
  - A multicore processor.
  - A medium scale shared memory parallel computer.
  - A large scale distributed memory system.

Implicit parallelism - Pipelining



- CPU contains multiple functional units, e.g. instruction fetch / decode, load / store, integer / floating point units, etc.
- Implicit parallelism executes straight line code in parallel on multiple FUs.
- Pipelining
  - Break up one instruction and execute pieces in pipeline.
    - Ex 5 stage pipeline potentially offers 5X speedup.
  - Modern processors have 10-20 stages.
  - If code branches, must guess how to fill pipeline.
    - Branch misprediction requires flushing pipeline.
    - Typical code branches every 5 instructions, so requires accurate prediction.

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Implicit parallelism - Superscalar

- Superscalar processors issue multiple instructions each clock.
- Execution must respect data dependencies, where one instruction uses results of another.
  - All three code fragments add 4 numbers, but first code has fewer data dependencies (so more parallelism) than second.
  - Third code requires look-ahead and reordering hardware to detect independence of instructions 1 and 3.

Source: *Introduction to Parallel Computing*, Grama et al., 2003

```
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)

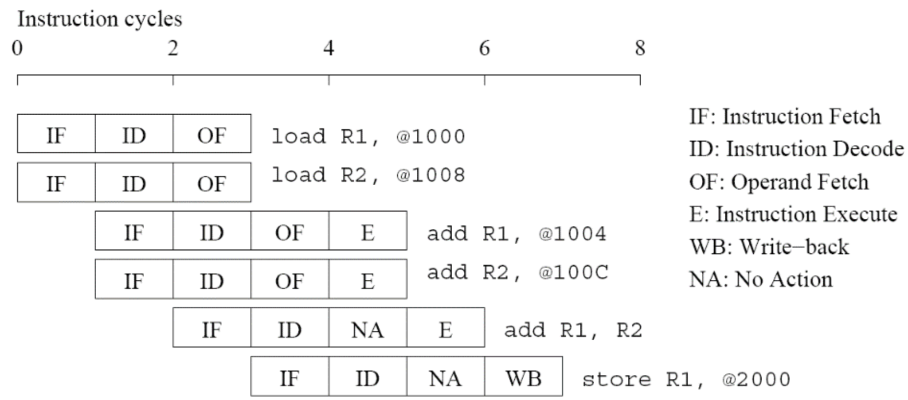
```
1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000
```

(ii)

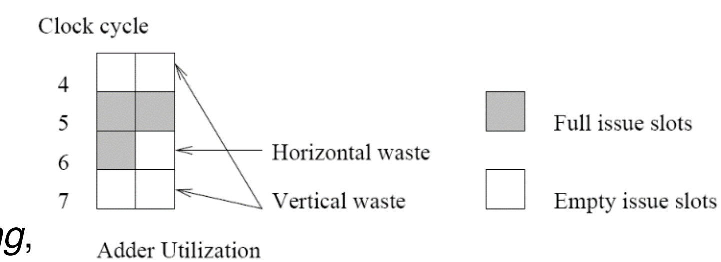
```
1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(iii)

(a) Three different code fragments for adding a list of four numbers.



(b) Execution schedule for code fragment (i) above.



(c) Hardware utilization trace for schedule in (b).

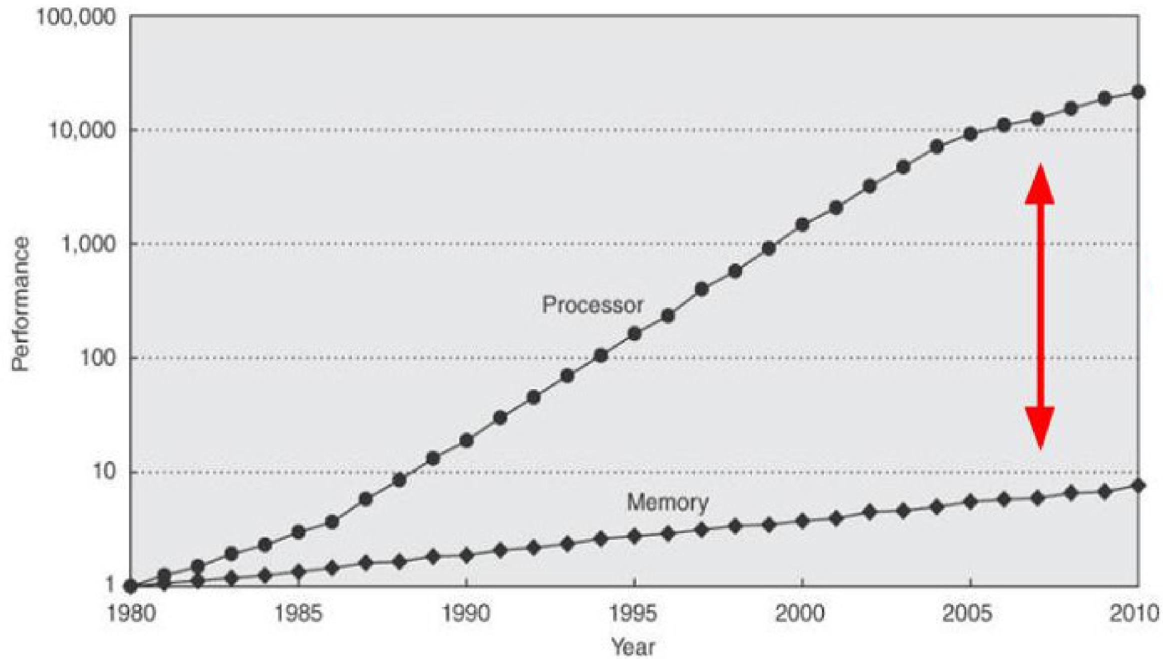
Implicit parallelism - VLIW

- Superscalar
  - Resource dependency is when multiple instructions use same hardware unit.
  - Branch dependency occurs when code can take different paths based on a conditional.
  - Waste occurs when no instruction issued in a clock cycle.
- VLIW (very long instruction word) finds and packs parallelizable instructions at compile time.
  - Superscalar finds parallelism at runtime.
  - Pro Compiler can do more sophisticated search.
  - Con Compiler doesn’t have runtime state, e.g. branch history, cache misses, for efficient scheduling.

Memory performance

- We need data before we can compute. So processor performance often depends on memory performance.
- Key measures are latency and bandwidth.
- Latency is amount of time for processor to access piece of data.
- Bandwidth is amount of data transferred from memory to processor per unit time.
- Ex Consider a highway.
  - Latency is the how fast you can drive on the highway (or alternatively, the length of the highway).
  - Bandwidth is how many lanes the highway has.
- Latency and bandwidth are independent.
  - Ex you can have low latency & low bandwidth, high latency & high bandwidth, etc.
- Latency is more important if frequently transfer small amounts of data.
- Bandwidth matters when transferring large piece of data.

CPU-memory gap



- Processors today can compute much faster than memory can transfer data.
- Ex 100 GFLOPS, 20 GB/s bandwidth and 100 ns latency to main memory.
  - Performance is bandwidth limited. Can only transfer 5 billion floats / sec, only 5 GFLOPS!

Memory hierarchy

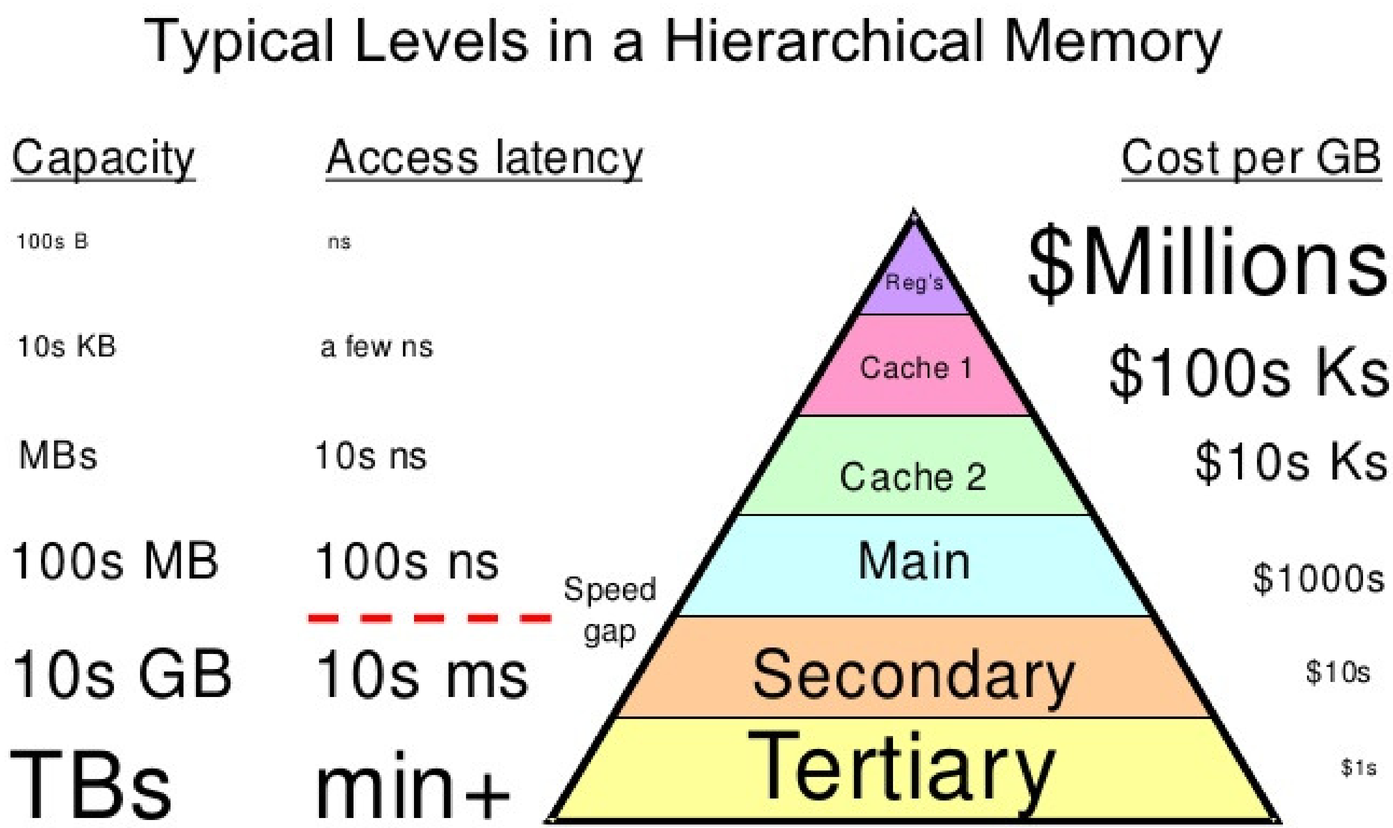
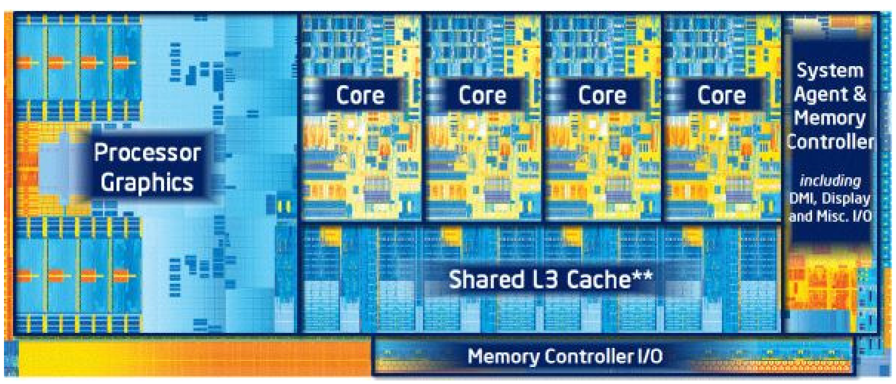
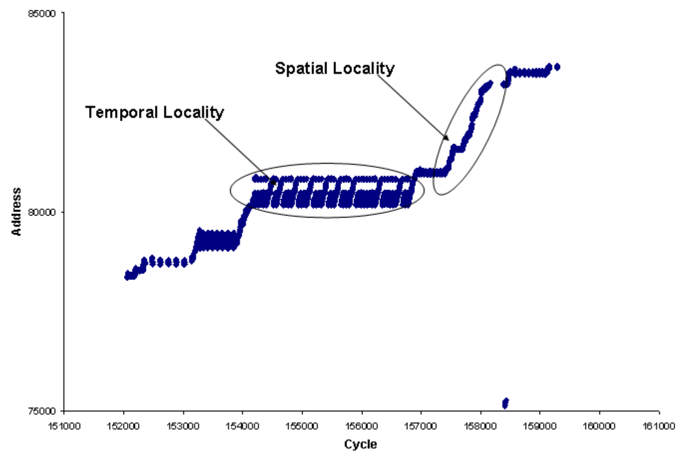


Fig. 17.14 Names and key characteristics of levels in a memory hierarchy.

Caching

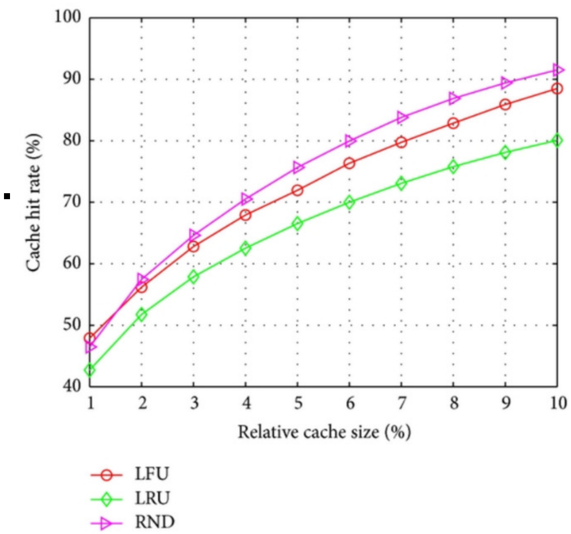
- Cache is small but fast piece of memory built directly on CPU die.
  - Low latency, high bandwidth.
  - Used to decrease (effective) latency. Also helps alleviate bandwidth limitations.
- When accessing data, processor first checks cache, and only goes to memory if data isn't in cache.
- Caches effective due to temporal and spatial locality in (most) code.
- Temporal locality Small set of data accessed repeatedly.
  - Store in cache and access quickly.
- Spatial locality
  - Nearby pieces of data accessed together.
  - Ex Iterating through an array.





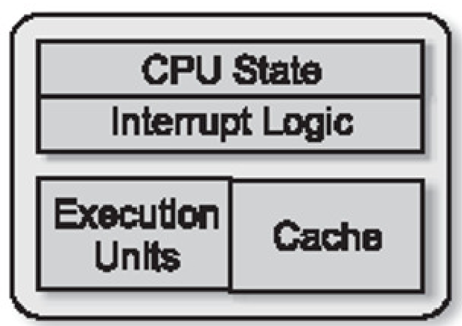
Caching performance

- Cache hit rate is the proportion of data accesses serviced from the cache.
  - Typically a concave function in the cache size.
- Processors have several layers of cache, each layer faster but smaller.
- Algorithms can sometimes be transformed to have greater locality.
- Caches can dramatically improve performance.
- Ex 1 GHz processor with 100 ns latency DRAM.
  - Transfer 10M words / sec $\Rightarrow$ 10 MFLOPS.
  - Suppose cache has 5 ns latency and 80% hit rate.
    - Avg latency per word = 5*0.8 + 100*0.2 = 24 ns.
    - Transfer 42M words / sec $\Rightarrow$ 42 MFLOPS.
  - With 90% hit rate, latency = 5*0.9+100*0.1 = 14.5 $\Rightarrow$ 69 MFLOPS.



Processor architectures

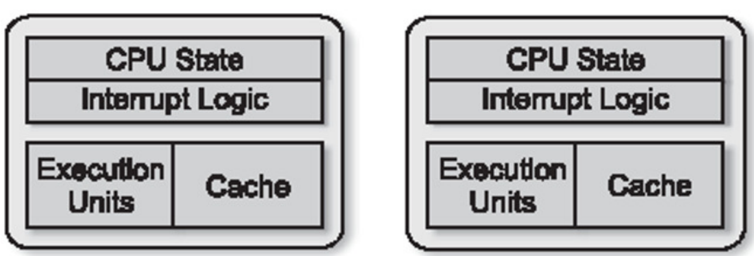
- A core is an independent execution unit that can run one or more threads.
- Core contains memory controller, instruction fetch / decoder, execution units, registers storing CPU state, etc.
- Cores and caches can be combined in various ways to form processors.



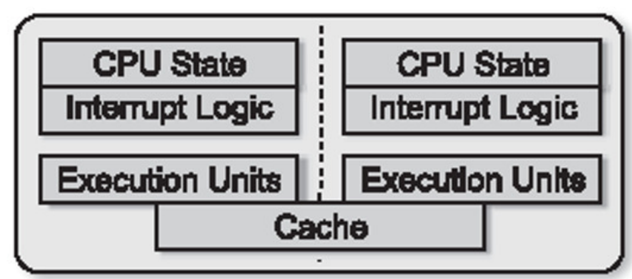
Single core processor

Processor architectures

- A multiprocessor contains several cores communicating through memory.
- A multicore (aka CMP, or chip multiprocessor) has several cores on a single die.
  - Each core is a complete processor, with own execution unit and CPU state.
  - Cores share an L2 / L3 cache.
    - Increases cache utilization, but might cause thrashing.
  - Threads on different cores run simultaneously.
  - Allows fast communication (i.e. cache coherency) between cores.
  - Cheap to manufacture, simpler board design, very popular.



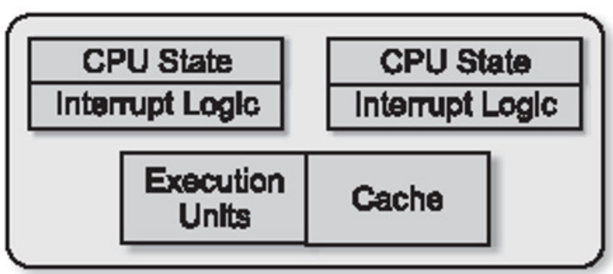
Multiprocessor



Multicore processor

Threads and multithreading

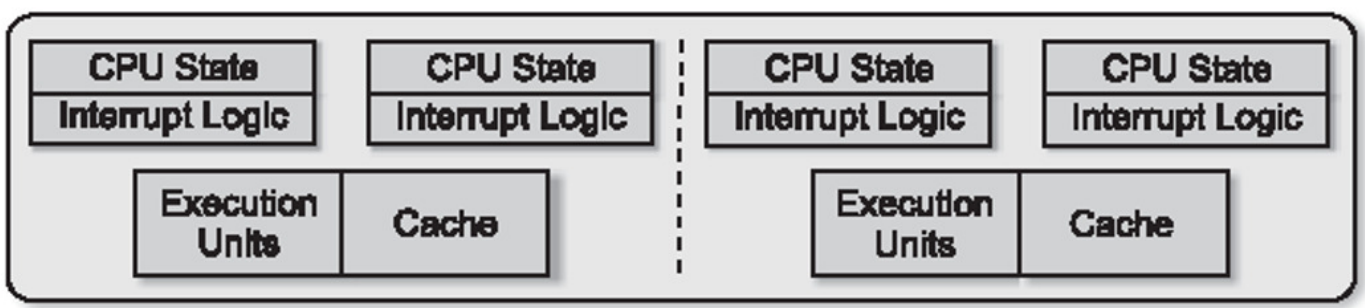
- Thread is an independent instruction stream with its own state, i.e. CPU registers, program counter and stack.
- A program can contain multiple threads running concurrently.
  - Threads can communicate through memory and cooperate on a problem.
- Simultaneous multithreading (SMT)
  - Relatively cheap to maintain the state of a thread.
  - A logical processor can be created using only the thread state.
  - Can fit several logical processors in one core.
    - Execution units and cache still shared, only state hardware duplicated.
  - Cheap way to get (some) extra performance and hide latency.
- Intel supports two threads per core (hyperthreading), Sun UltraSparc supports 8 threads, etc.



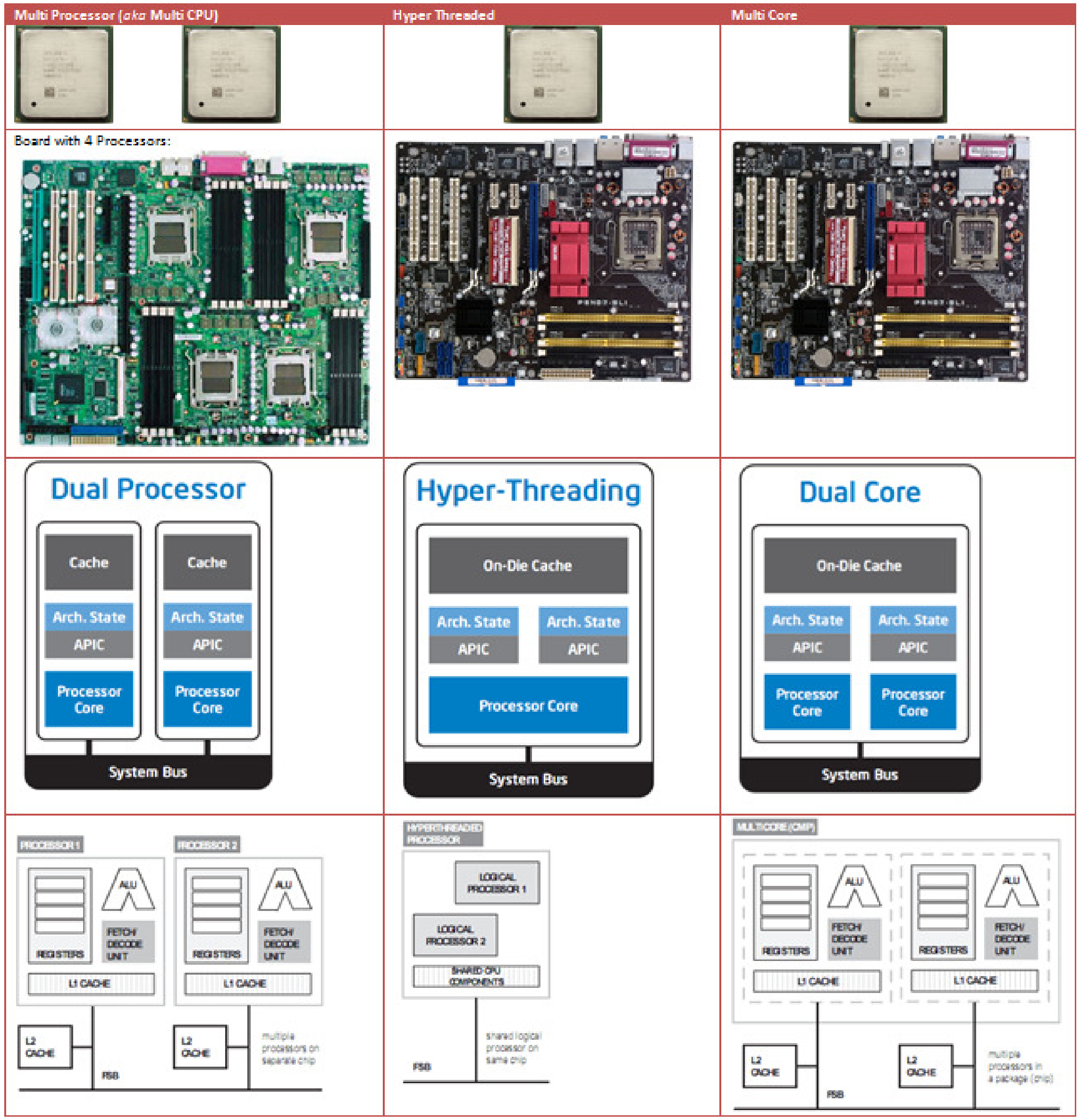
Hyperthread processor

Multithreaded multicore

- Individual cores can use multithreading. Combine several cores in one package.
- Widely used, e.g. Intel Core, Atom, Xeon, Sun UltraSparc, IBM POWER8, etc.
- Hyperthreading improves performance by 15-30%, depending on application and scheduling.
  - Threads may compete for resources such as cache or execution units and actually decrease performance.
  - Processors allow deactivating multithreading.



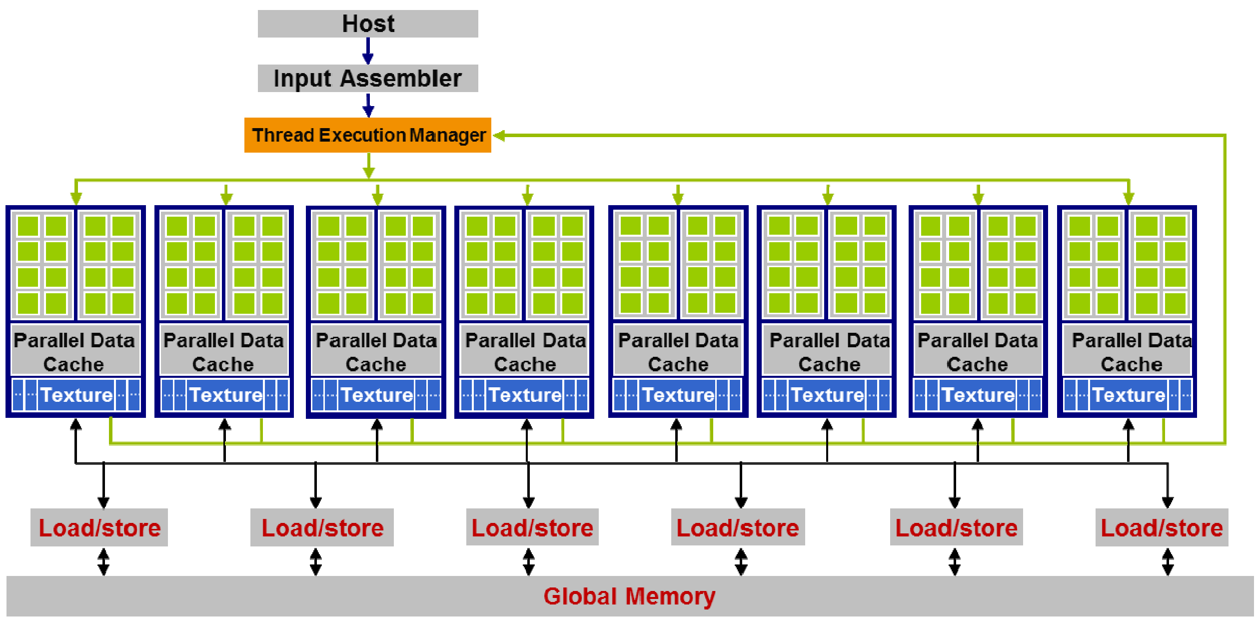
Multicore hyperthreading processor



Source: http://superuser.com/questions/214331/what-is-the-difference-between-multicore-and-multiprocessor

GPGPU

- General purpose graphics processing unit (or just GPU).
  - Using massive parallelism of graphics cards for general purpose computation.
  - Dozens of “streaming multiprocessors”.
    - Each SM contains 32 simple cores and runs 32 threads simultaneously.
    - All cores do the same instruction.
  - Tens of thousands of active threads.
    - Hardware quickly switches between them to hide I/O latency.

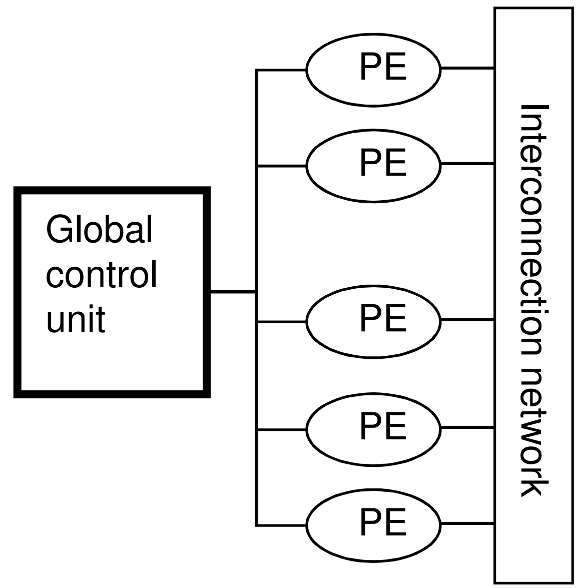


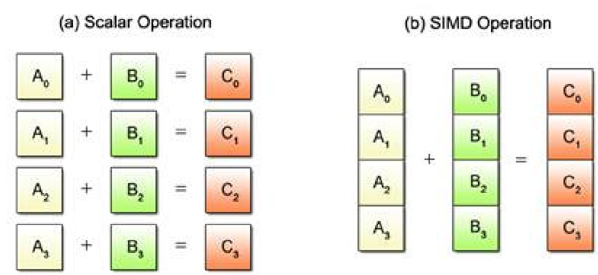
Flynn's taxonomy

- A parallel system consists of multiple communicating, cooperating processors.
  - Processors may be single / multicore, single / multithreaded, GPUs / accelerators, etc.
- Unlike sequential computing, which follows the von Neumann architecture, there is a wide variety of parallel system architectures.
- Flynn (1966) classified parallel systems based on instructions and data used by the processors.
  - **SISD** Single instruction, single data
    - Conventional sequential processor.
  - **SIMD** Single instruction, multiple data
    - Single control unit for multiple processing units.
  - **MISD** Multiple instruction, single data
  - **MIMD** Multiple instruction, multiple data
- Different hardware designs strive to software characteristics.

SIMD

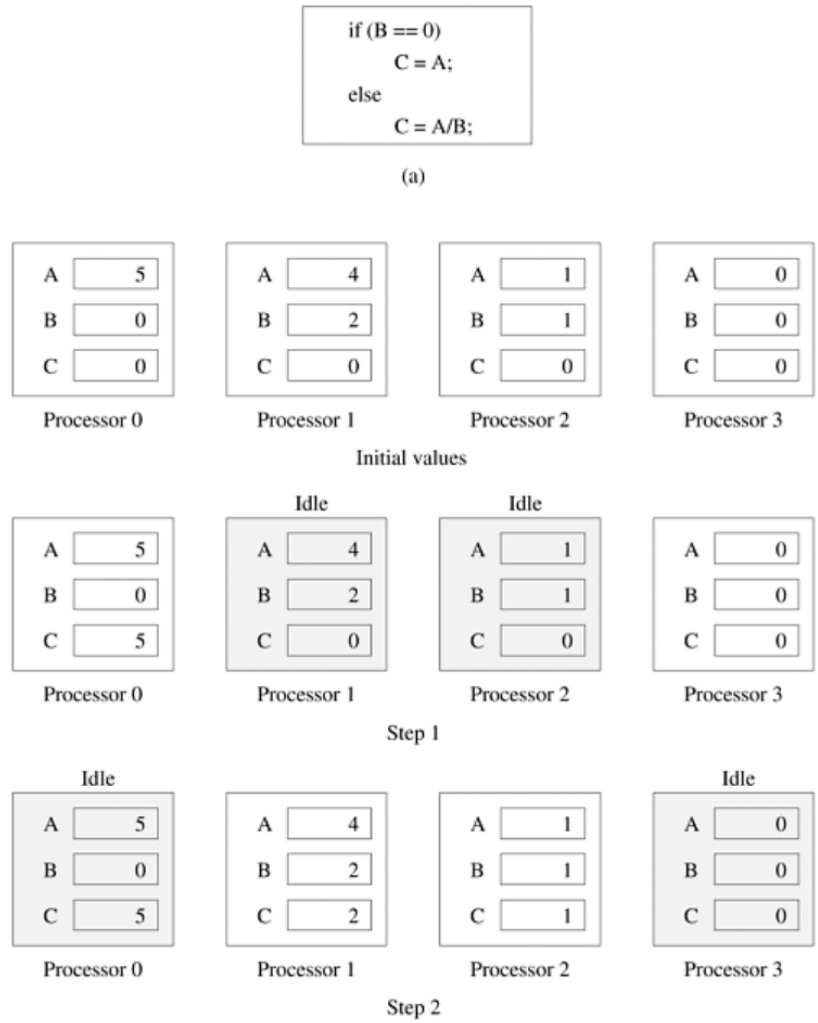
- Control unit fetches single instruction and broadcasts it to processing units.
- PEs each applies instruction to its own data item, in synchrony.
- Ex Adding two arrays.
  - All PEs perform addition, on different coordinates of the arrays.
- Very popular today.
  - Effective for data parallel programs, e.g. graphics and video, dense linear algebra, machine learning.
  - Cheap to implement, don't need to duplicate hardware for fetch / decode, branch prediction, OOO, etc.
  - Used in GPUs, Intel AVX, Xeon Phi, IBM Cell SPU.
  - Early generations supercomputers were SIMD with very wide lanes (1000s bits).
  - Modern SIMD execute 4-32 instructions.





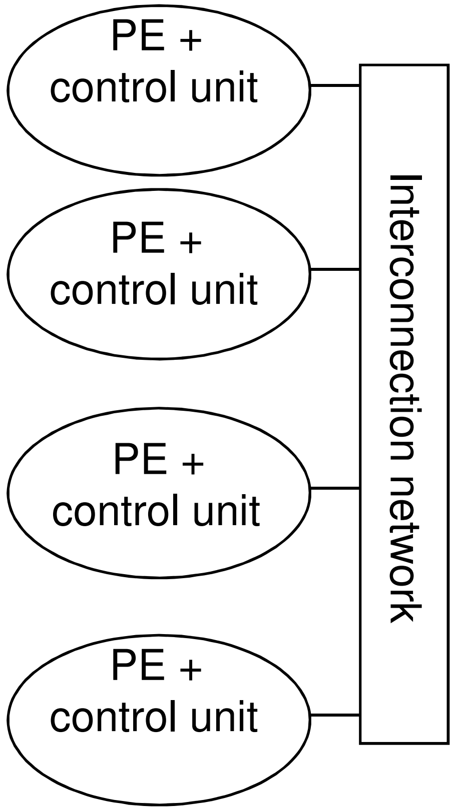
Instruction divergence

- SIMD processors can perform different instructions, but need an activity mask to deactivate processors in certain steps.
  - Takes k steps to do k different instructions. In each step all processors doing same instruction execute synchronously.
- SIMD works poorly for heavily branching code, where execution is data dependent.
- Also doesn't work well when threads not balanced, e.g. graph algorithms, sparse linear algebra.



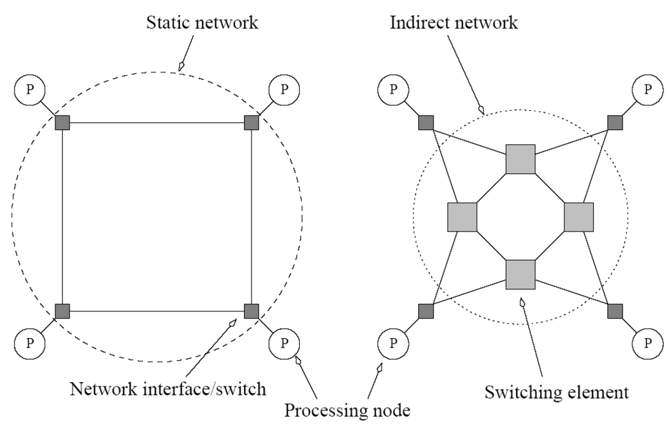
MIMD

- General purpose multiprocessor system.
  - Each processor has its own instruction and data.
  - Processors communicate through an interconnection network.
- A broad category covering most parallel computers.
- Most commodity processors are a combination of MIMD with SIMD capability.
  - Ex Intel Xeon. Different cores are MIMD. But each core implements SIMD AVX.
- Ex Most supercomputers contain commodity processors plus SIMD coprocessors.



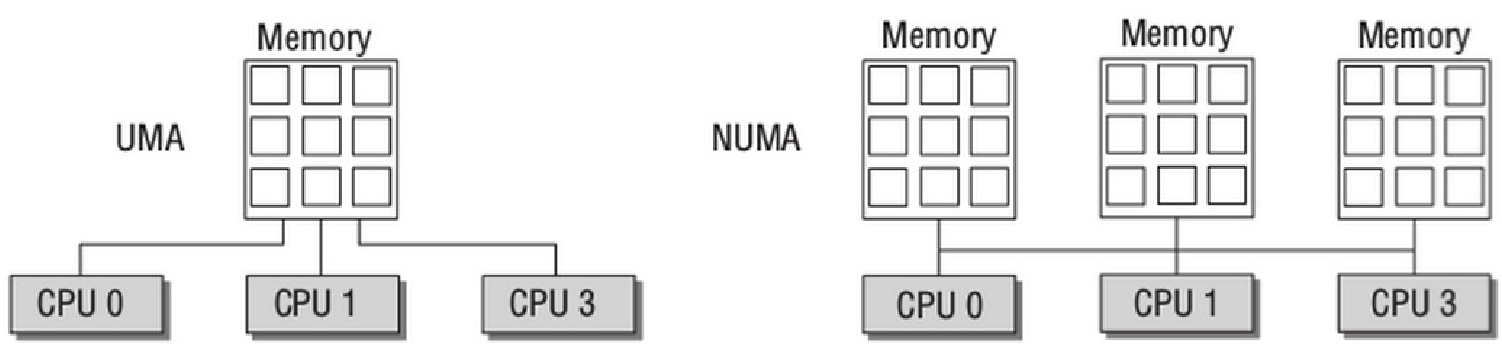
Interconnection networks

- Network allows processors to share data and cooperate.
- Built using links and switches.
  - Links are fixed connection between two processors.
  - Switch can set of processors on input ports with processors on output ports.
- Static or direct networks built from links.
- Dynamic or indirect networks built from switches.
  - Switches route data between processors.
  - Can also buffer, multicast, etc.
  - Wire complexity of switches quadratic in degree, i.e. number of processors on input / output ports.



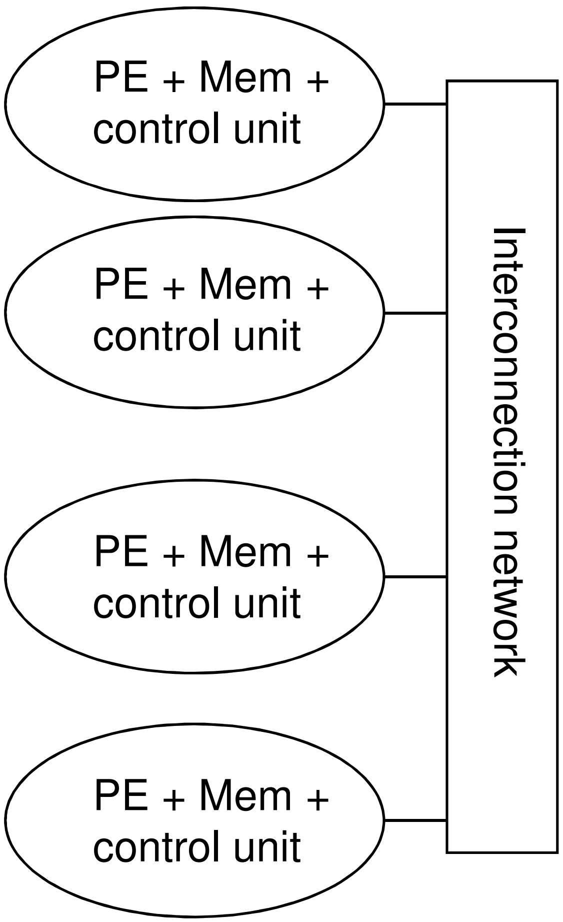
Shared memory architecture

- A single memory address space for all processors.
  - Any address can be accessed by any processor.
  - Easier to program and reason about.
- While memory is logically one block, physically there may be multiple memory banks connected on a network.
  - OS takes care of locating the data.
- Limited scalability (100s of processors) due to bandwidth requirement on interconnect.
- Can attach caches to processors to avoid some accesses to main memory.
  - Needs cache coherence, i.e. changes to data in one processor's cache needs to be reflected in other processor using same data.
  - Coherence traffic also limits scalability.
- Memory access times can be uniform (UMA) or non-uniform (NUMA).
  - In UMA access times for all memory banks roughly equal.
  - In NUMA, accessing physically local memory faster than accessing remote memory.



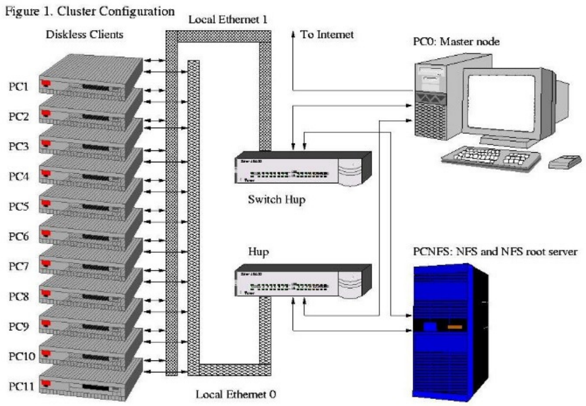
Distributed memory architecture

- Each processor can only directly address its own memory.
- To access remote data, processor sends message over interconnect network to data's owner.
  - Also called message passing architecture.
- Programmer keeps track of where data is located.
  - Harder to program than shared memory, but scales better.
- Large scale parallel computers are all distributed memory, because overhead of providing one logical memory is too high.
- Can also combine distributed and shared memory.
  - Ex Supercomputer is overall a distributed memory system connecting shared memory nodes.



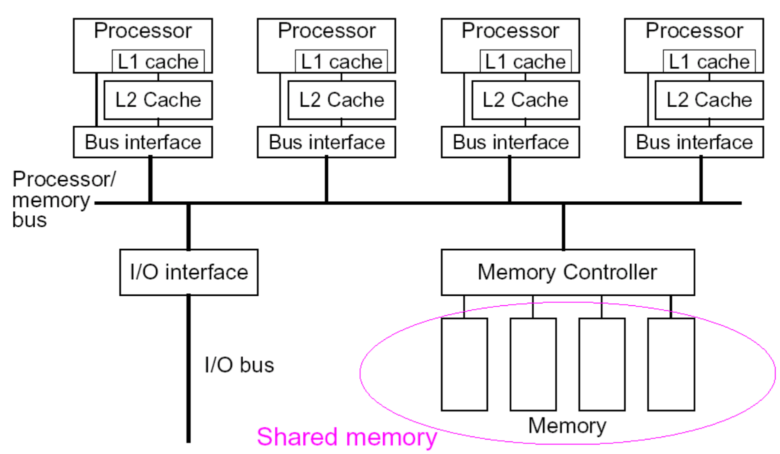
Network of workstations (NOW)

- Networked computers as a multicomputer platform
  - Popularized in 1990's as high performance workstations and networking became commoditized (cheap).
- Advantages
  - Relatively high performance and low cost.
  - The latest processors can easily be incorporated into the system as they become available.
  - Existing software can be used or modified.
- Ex Beowulf, SETI@Home, Folding@Home



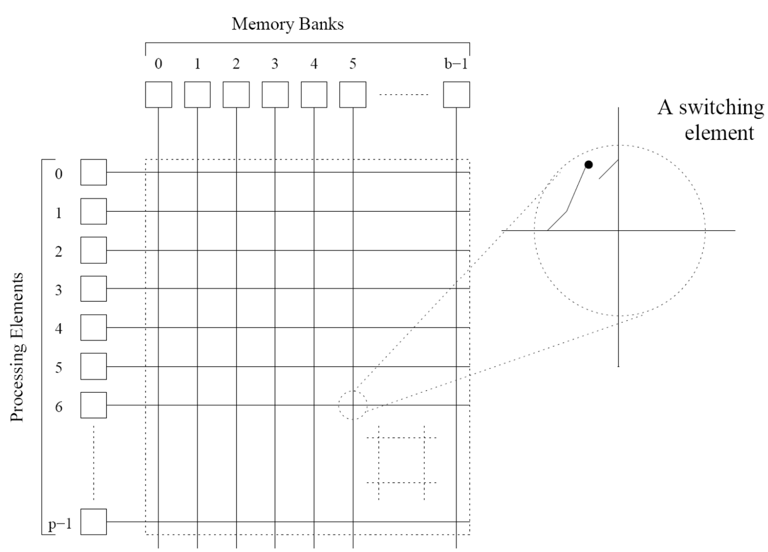
Bus architecture

- All processors communicate with memory through common bus.
  - To communicate, a processor needs exclusive access to the bus.
  - Processors need media access control protocol to control simultaneously.
- Bus has limited bandwidth, becomes communication bottleneck.
- Caches help avoid bus traffic for many memory operations.
- But still limited to small scale systems, (~50 processors).
  - Usually used for shared memory systems.



Crossbar architecture

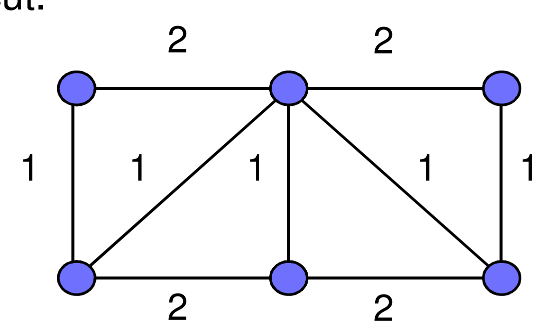
- Switched (dynamic) network for higher end shared memory systems.
- Allows all processors to communicate with all memory modules simultaneously.
  - Nonblocking, i.e. one processor's communication won't prevent another's.
- Higher bandwidth and more scalable than bus.
  - But more complex and expensive to implement.
- Limited to a few hundred processors.



Source: Introduction to Parallel Computing, Grama et al.

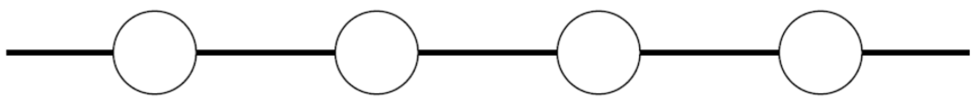
Multihop networks

- Bus and crossbar both one stage networks, i.e. two processors can directly communicate over a link.
- One stage networks have limited bandwidth and scalability, and sometimes high cost.
- Can improve performance using multistage or multihop networks, where messages traverse several links.
- Multihop networks can be characterized by
  - Diameter
    - Distance between farthest pair of processors.
    - Gives worst case latency.
  - Bisection width
    - Minimum number of links to cut to partition the network into two (almost) equal halves.
    - Indicates potential communication bottlenecks.
    - Bisection bandwidth is sum of bandwidths of links cut.
  - Cost
    - Number of links in network.
    - Bisection width.



1D topologies

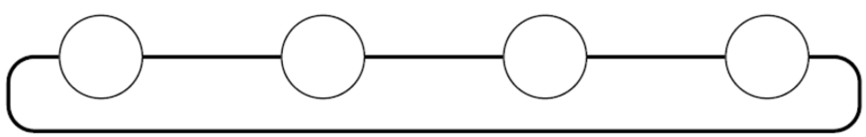
Linear array



Diameter	p - 1
Bisection	1
Cost	p-1

High diameter, low fault tolerance, low cost.

Ring

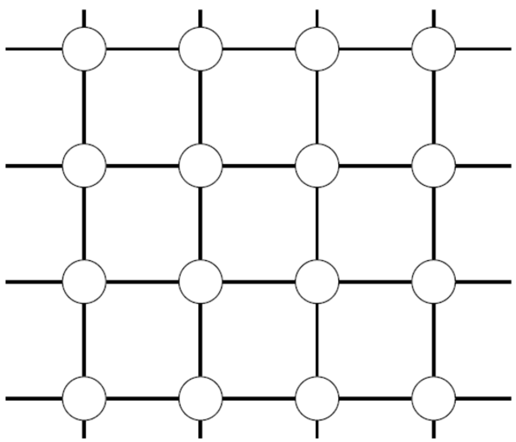


Diameter	p / 2
Bisection	2
Cost	p

Slightly improved diameter and fault tolerance, low cost.
All nodes symmetric.

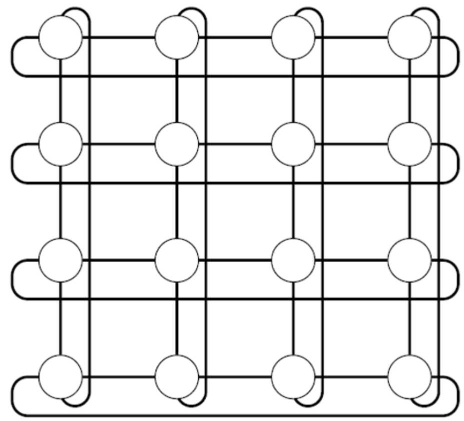
Meshes and tori

2D mesh



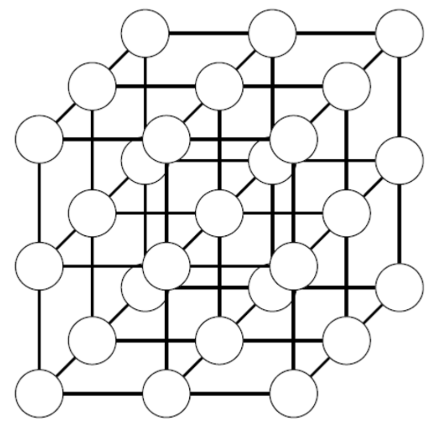
Diameter $2\sqrt{p} - 2$
Bisection $\sqrt{p}$
Cost $2p-2\sqrt{p}$

2D torus

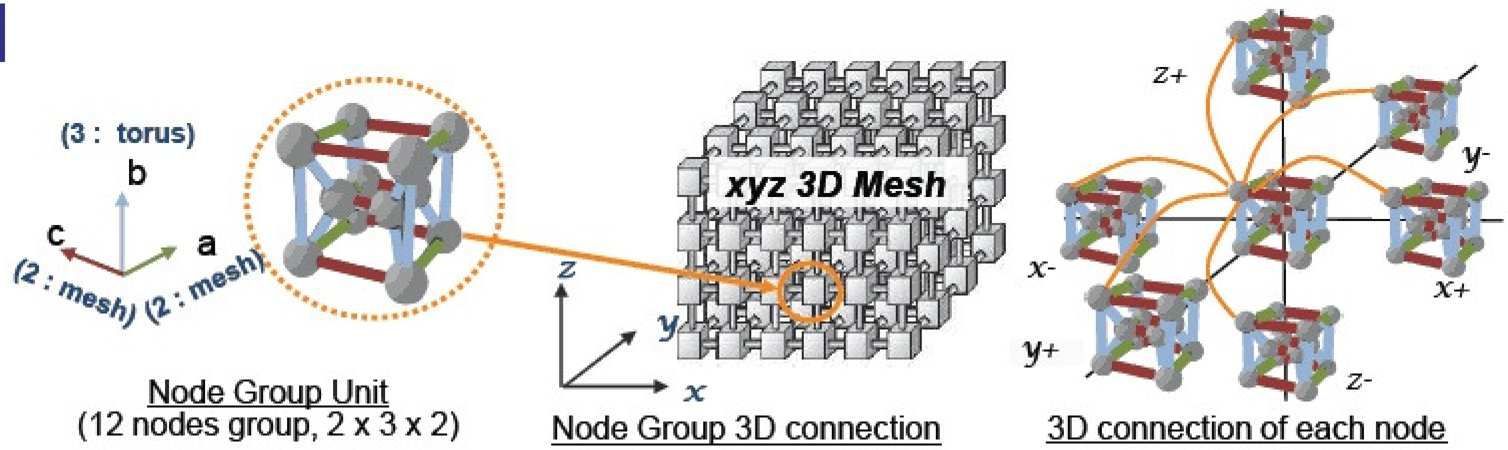


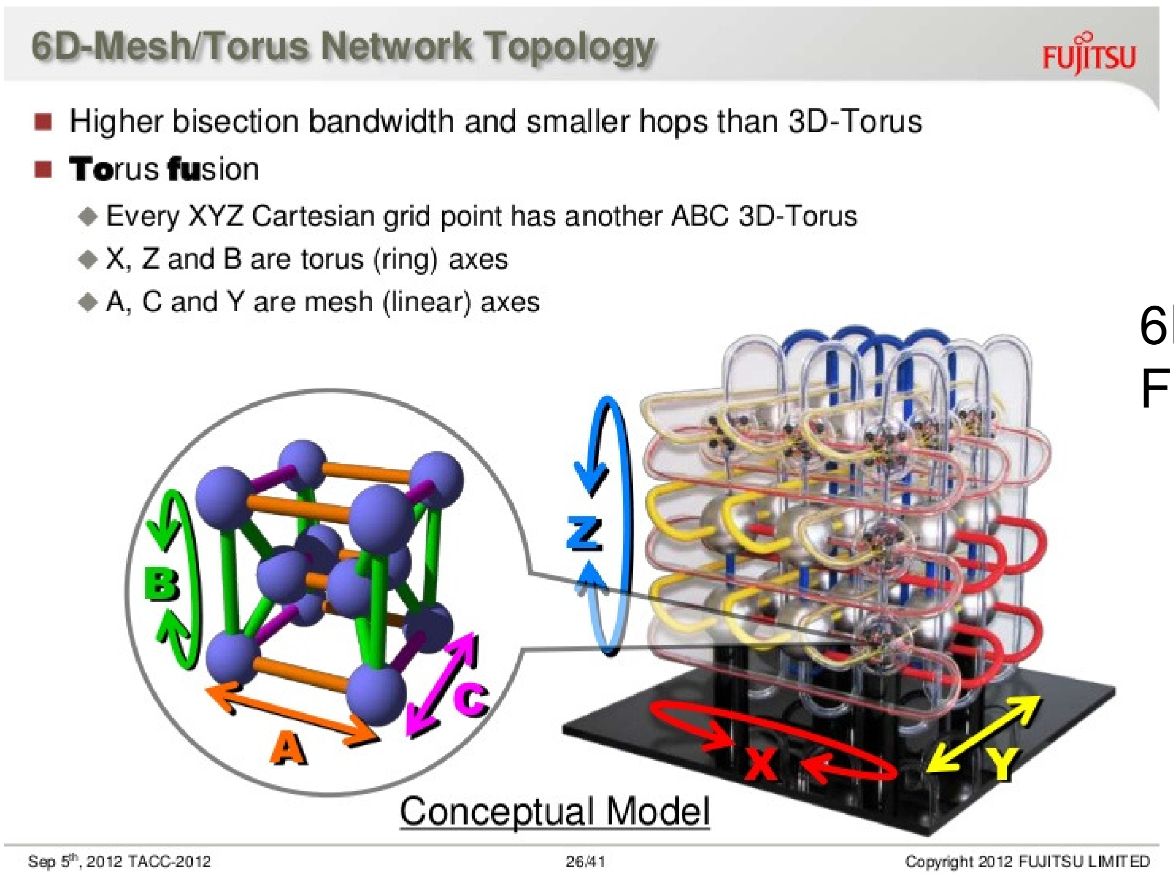
Diameter $\sqrt{p} - 1$
Bisection $2\sqrt{p}$
Cost $2p$

3D mesh



Diameter $3\sqrt[3]{p} - 3$
Bisection $(\sqrt[3]{p})^2$
Cost $3p-3(\sqrt[3]{p})^2$

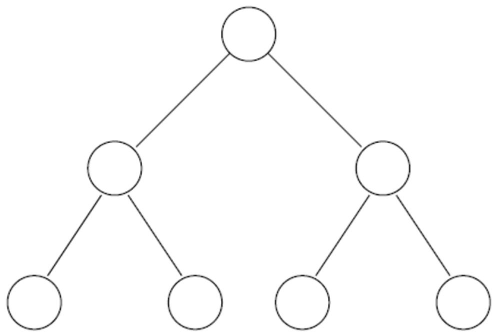




6D “tofu” network on Fujitsu K computer.

Trees

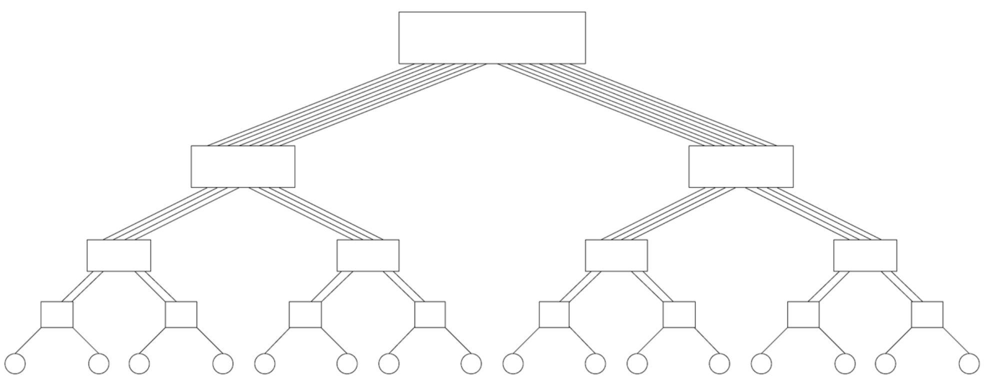
Tree



Diameter	2 log (p+1)
Bisection	1
Cost	p

Excellent diameter and cost, low fault tolerance.
Root is communication bottleneck and single point of failure.

Fat tree



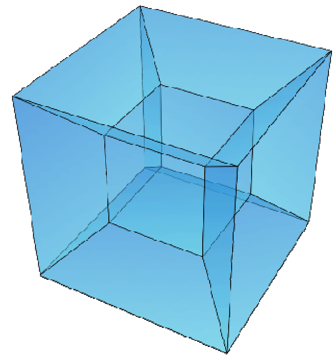
Diameter	2 (log p)
Bisection	p / 2
Cost	p log p / 2

Each processor has twice as many links to its parent as to each child.
When routing, messages traverse random links.
Used in e.g. Tianhe-2.

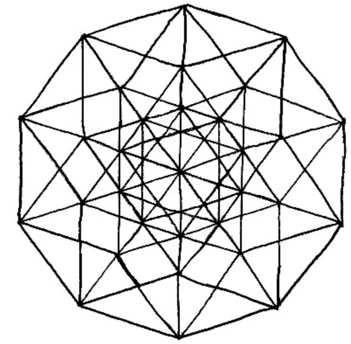
Hypercubes

Diameter	log p
Bisection	p / 2
Cost	p log p / 2

Excellent diameter and fault tolerance.
CM-1 used a 20-dim hypercube!



4D hypercube



5D hypercube