Parallel Computing for Science & Engineering (PCSE 374C/394C)

Computer Architectures: parallel computers

Instructors:
Victor Eijkhout, Cyrus Proctor

Some theory

- ....before we get into the hardware
- Optimally, P processes give $T_P = T_1/P$
- Speedup $S_P = T_1/T_p$, is P at best
- Superlinear speedup not possible in theory, sometimes happens in practice.
- Perfect speedup in "embarrassingly parallel applications"
- Less than optimal: overhead, sequential parts, dependencies

Some more theory

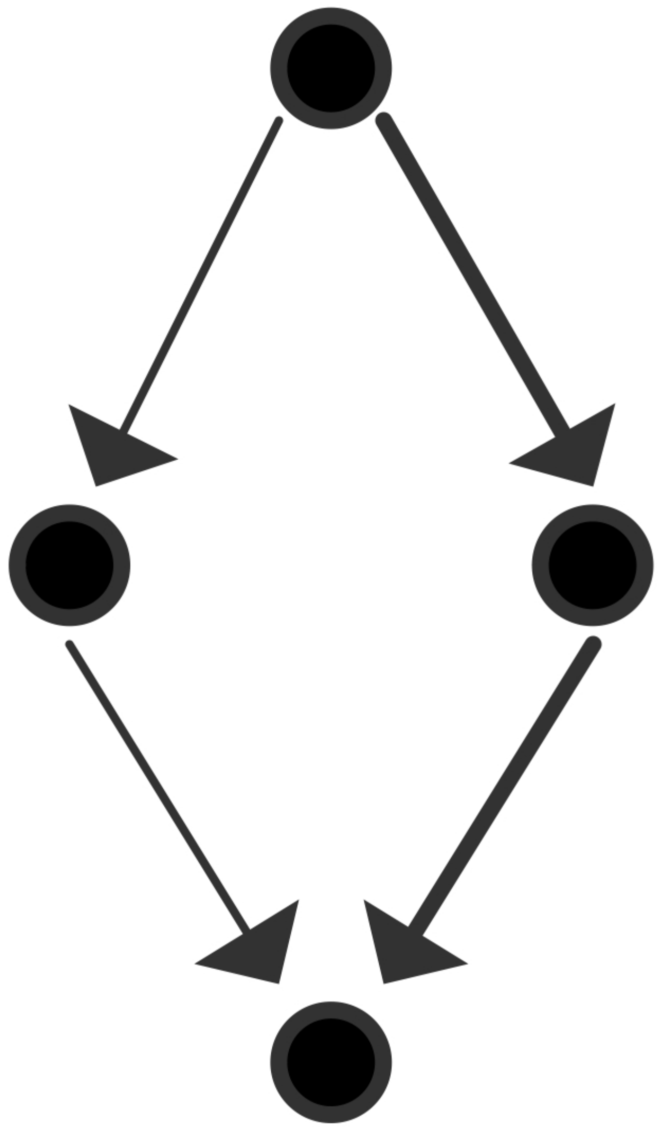
- ….before we get into the hardware
- Optimally, P processes give $T_P=T_1/P$
- Speedup $S_P = T_1/T_p$, is P at best
- Efficiency $E_P = S_p/P$
- Scalability: efficiency bounded below

Amdahl's Law

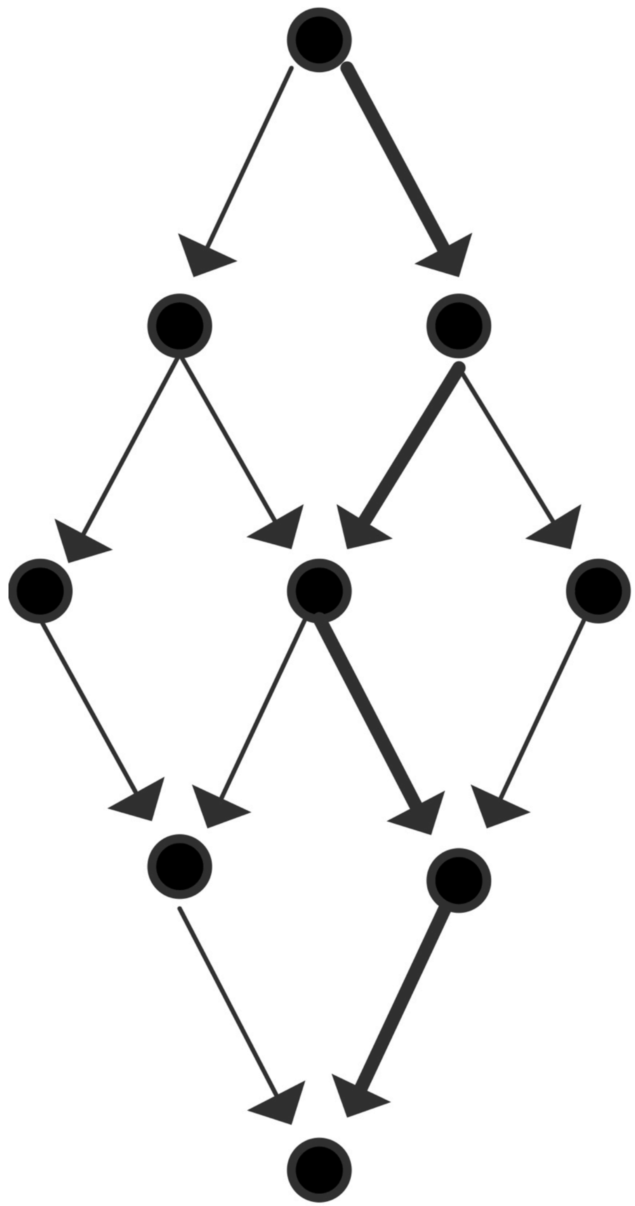
- Some parts of a code are not parallelizable
- => they ultimately become a bottleneck
- For instance, if 5% is sequential, you can not get a speedup over 20, no matter P.
- Formally: $F_p+F_s=1$, $T_p=T_1(F_s+F_p/p)$, so T_p approaches T_1F_s as p increases

Definition of parallelism

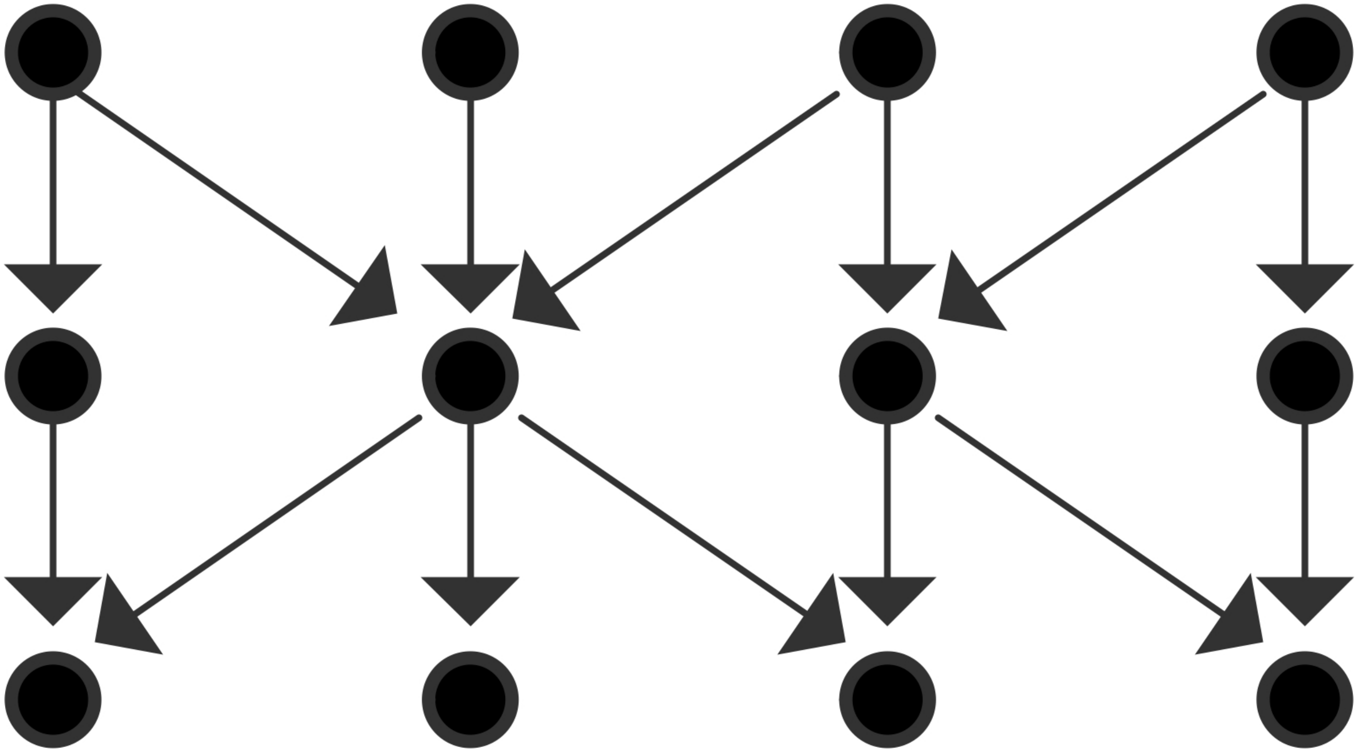
- T_1 : time on a single processor
- T_∞ : time on p processors
- T_∞ : time with unlimited processors
- P_∞ : value of p for which T_∞ is attained
- Brent's theorem:
  If there are W operations, and the critical path has length S, p processors can achieve time $S + \text{floor}(W/p)$



- T_1 : Sequential time?
- T_∞ : What is the best you can do, and with how many processors?



- T_1 : Sequential time?
- Maximal parallelism?
- T_∞ : What is the best you can do, and with how many processors?



- Maximal parallelism is 4.
- Can you find a solution with p=3 that has T_3=4 and therefore E=1?

Brent's theorem

- If there are W operations, and the critical path has length S, p processors can achieve time $S + \text{floor}(W/p)$

Scaling

- Increasing the number of processors for a given problem makes sense up to a point: p>n/2 in the addition example has no use
- Strong scaling: problem constant, number of processors increasing
- More realistic: scaling up problem and processors simultaneously, for instance to keep data per processor constant: Weak scaling
- Weak scaling not always possible: problem size depends on measurements or other external factors.

Theoretical characterization of architectures

Classification #1: instruction streams

Parallel Computers Architectures

- **Parallel computing** means using multiple processors, possibly comprising multiple computers
- Flynn's (1966) taxonomy is a first way to classify parallel computers into one of four types:
  - (SISD) Single instruction, single data
    - Your desktop (unless you have a newer multiprocessor one)
  - (SIMD) Single instruction, multiple data:
    - Thinking machines CM-2
    - Cray 1, and other vector machines (there's some controversy here)
    - Parts of modern GPUs
  - (MISD) Multiple instruction, single data
    - basically doesn't exist
  - (MIMD) Multiple instruction, multiple data
    - Nearly all of today's parallel machines
  - (SPMD) Single program, multiple data: MIMD, but identical executables.

SIMD

- Based on regularity of computation: all processors often doing the same operation: *data parallel*
- Big advantage: processor do not need separate ALU
- ==> lots of small processors packed together
- Ex: Goodyear MPP: 64k processors in 1983
- Use masks to let processors differentiate

SIMD then and now

- There used to be computers that were entirely SIMD (usually attached processor to a front end)
- SIMD these days:
  - SSE instructions in regular CPUs
  - GPUs are SIMD units (sort of)

Classification #2: memory model

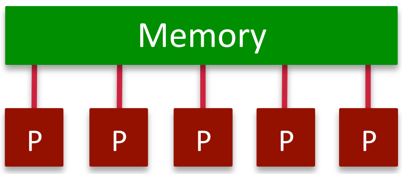
Parallel Computer Architectures

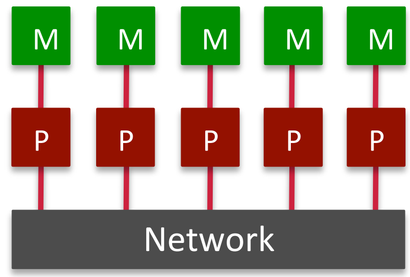
- Top500 List now dominated by MPPs and Clusters
- The MIMD model "won".
- SIMD exists only on smaller scale
- A much more useful way to classification is by memory model
  - *shared* memory
  - *distributed* memory

Two memory models

- Shared memory: all processors share the same address space
  - OpenMP: directives-based programming
  - PGAS languages (UPC, Titanium, X10)
- Distributed memory: every processor has its own address space
  - MPI: Message Passing Interface

Shared and distributed memory

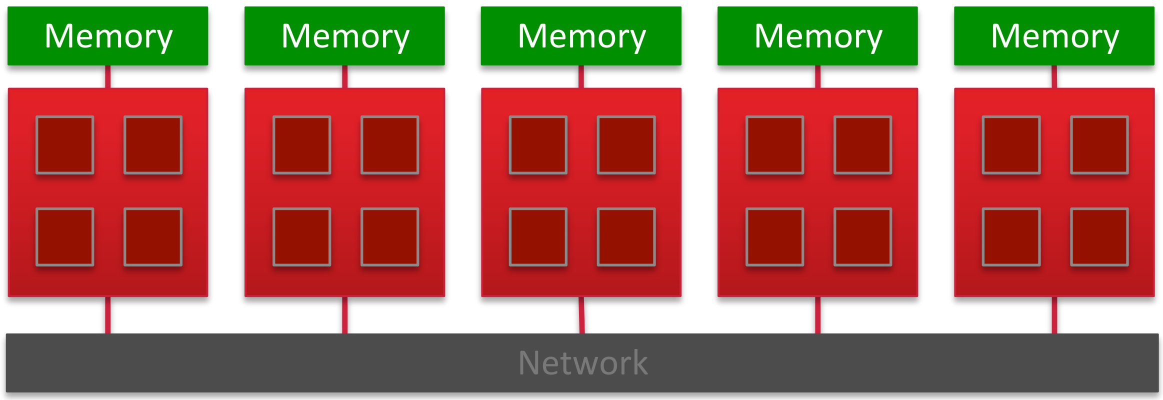




- All processors have access to a pool of shared memory
- Access times vary from CPU to CPU in NUMA systems
- Example: SGI Altix (SMP), multicore processors

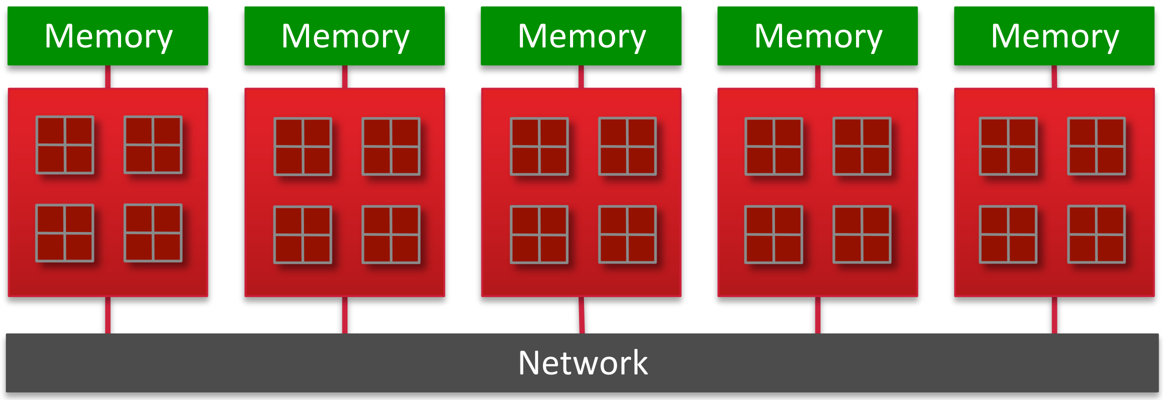
- Memory is local to each processor
- Data exchange by message passing over a network
- Example: Clusters with single-socket blades

Hybrid systems



- A limited number, N, of processors have access to a common pool of shared memory
- To use more than N processors requires data exchange over a network
- Example: Cluster with multi-socket blades

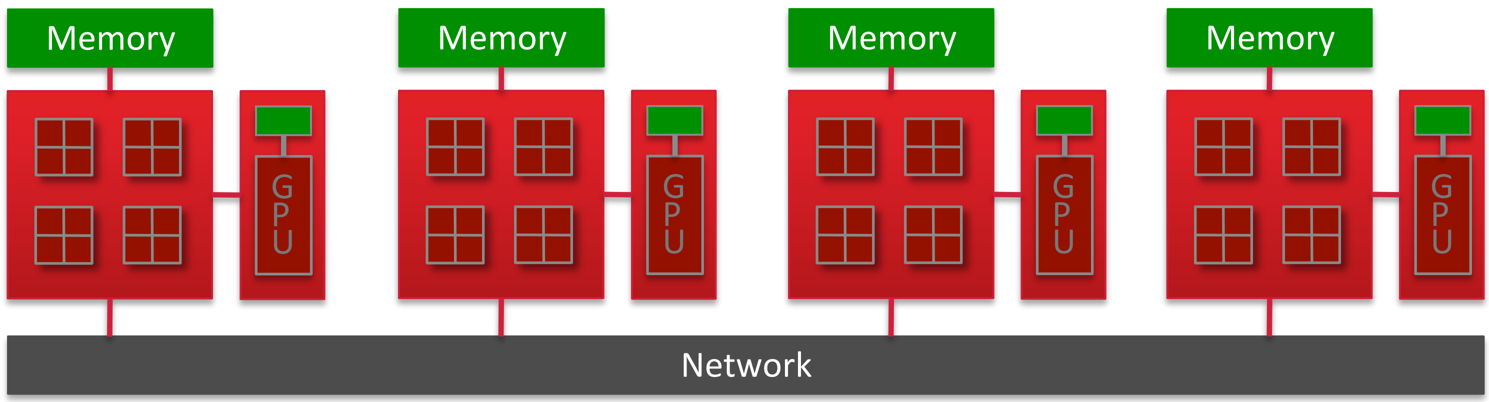
Multi-core systems



- Extension of hybrid model

- Communication details increasingly complex
  - Cache access
  - Main memory access
  - Quick Path / Hyper Transport socket connections
  - Node to node connection via network

Co-processor Systems

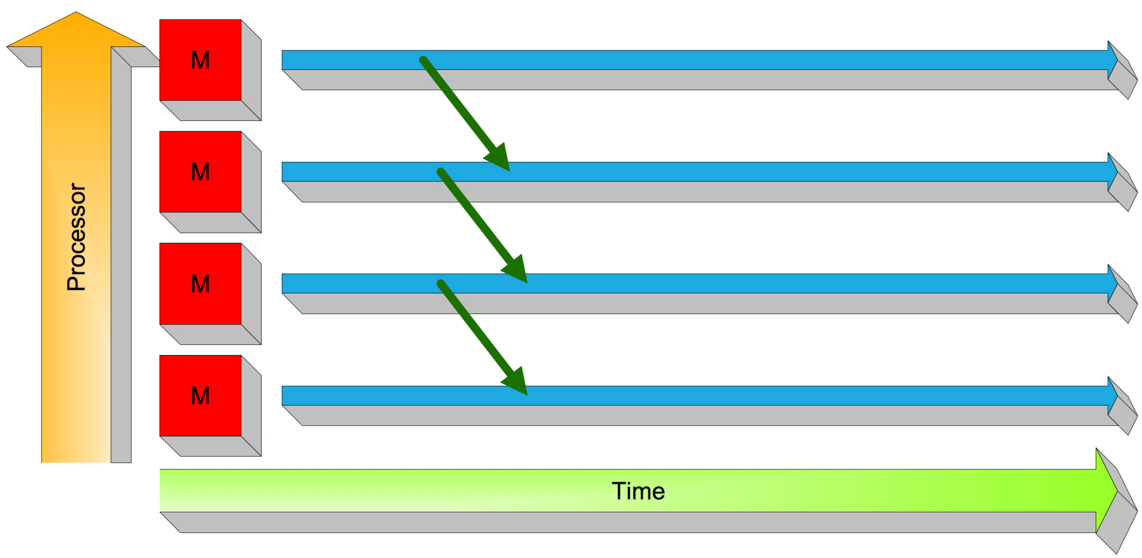


- Calculations made in both CPUs and co-processors (GPU, MIC)
- Programmability is tricky: two different processor types
- Requires specific libraries and compilers (GPU: CUDA, OpenCL, MIC: OpenMP)

Classification #3: process dynamism

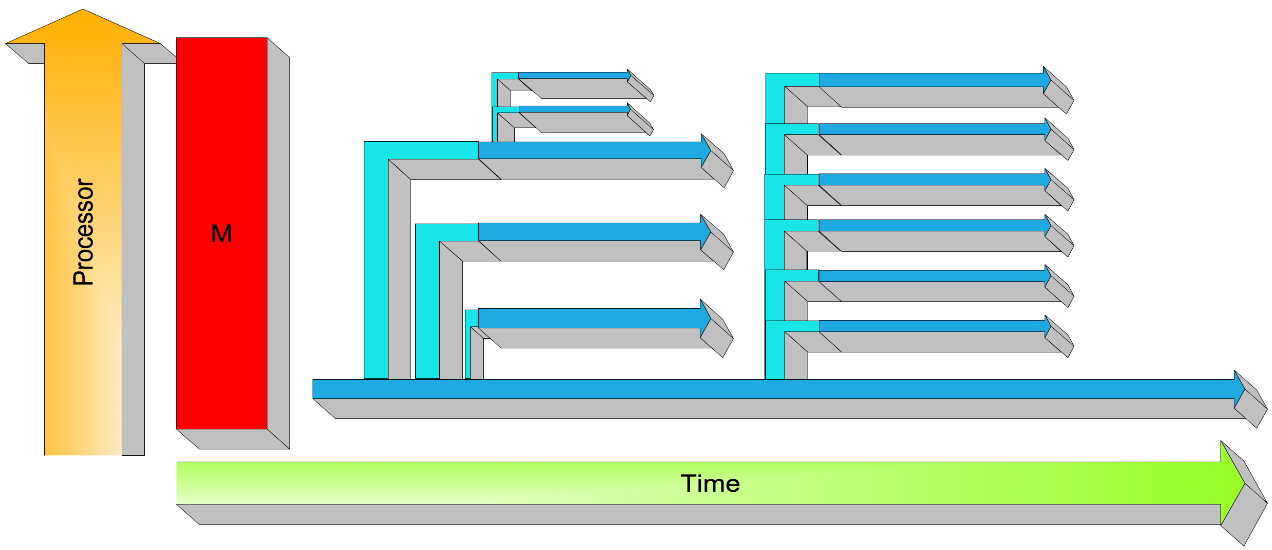
"Process-based" parallelism

- MIMD & SPMD: one process per processor/ core, lives for the life of the run
- Great for distributed memory: task creation and migration is hard.



“Task-based” parallelism

- Threading models: tasks can be created at will, placed on whatever processor/core is free
- Great on shared memory



Dynamic thread creation

- Old: pthreads
- Newer: Cilk+ (Intel), OpenMP (open standard)

```
int sum=0;
void adder(){sum = sum+1;}

int main() {
  int i;
  pthread_t threads[NTHREADS];
  for (i=0; i<NTHREADS; i++)
   pthread_create
     (threads+i,NULL,&adder,NULL);
  for (i=0; i<NTHREADS; i++)
   pthread_join(threads[i],NULL);
```

```
cilk int fib(int n){
  if (n<2) return 1;
  else {
    int rst=0;
    rst += spawn fib(n-1);
    rst += spawn fib(n-2);
    sync;
  return rst;
 }
```

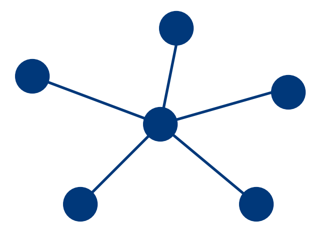
Classification #4: interconnects

Topology of interconnects

- What is the actual ‘shape’ of the interconnect? Are the nodes connect by a 2D mesh? A ring? Something more elaborate?
- => some graph theory

Completely Connected and Star Networks

- Completely Connected : Each processor has direct communication link to every other processor (compare ranger node)

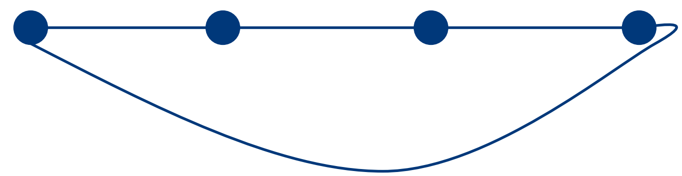


- Star Connected Network : The middle processor is the central processor; every other processor is connected to it.

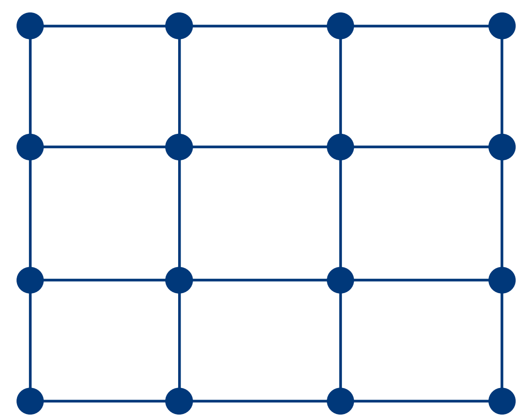
Arrays and Rings

- Linear Array :

- Ring :

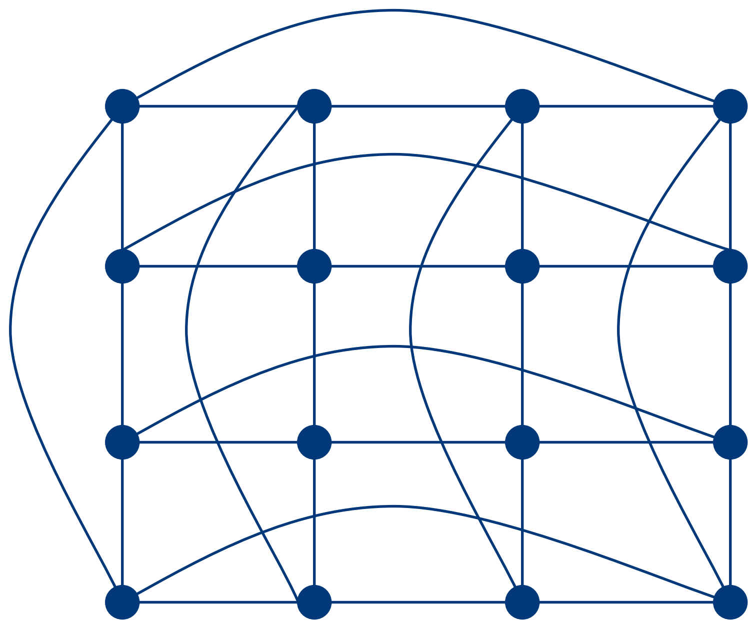


- Mesh Network (e.g. 2D-array)



Torus

2-d Torus (2-d version of the ring)

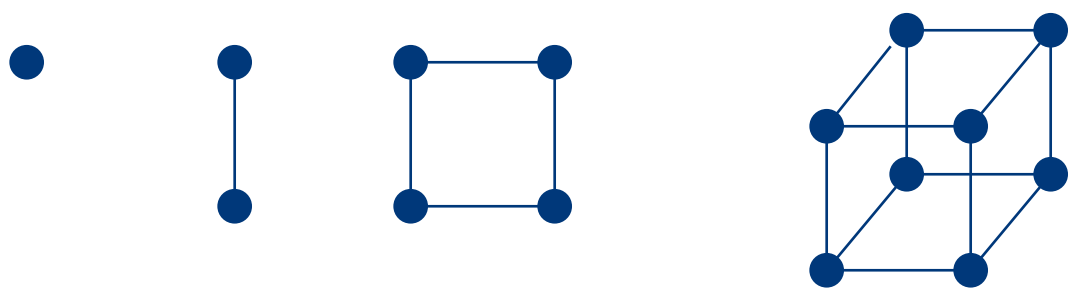


Hypercubes

- Hypercube Network : A multidimensional mesh of processors with exactly two processors in each dimension. A d dimensional processor consists of

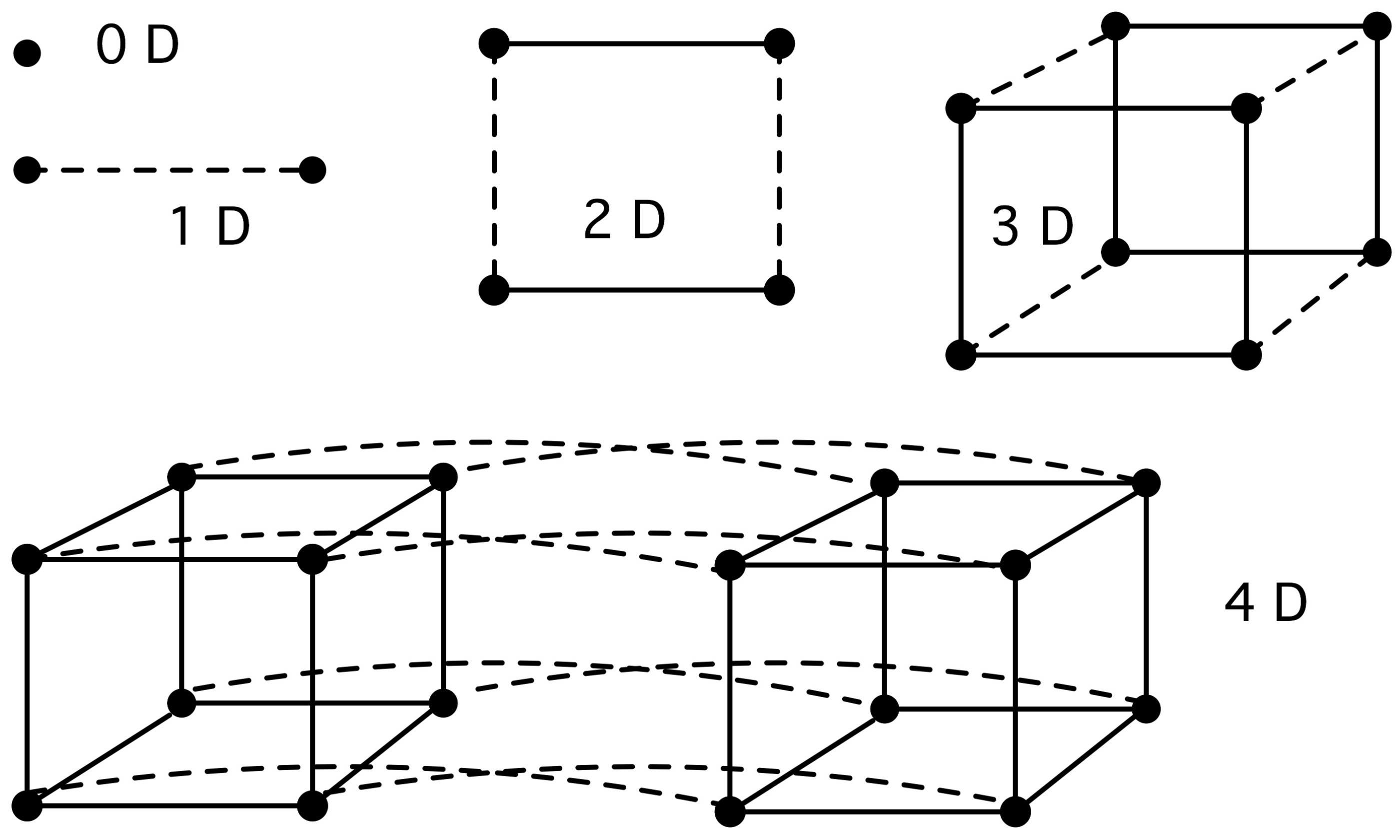
$$p = 2^d \text{ processors}$$

- Shown below are 0, 1, 2, and 3D hypercubes



0-D 1-D 2-D 3-D hypercubes

Inductive definition



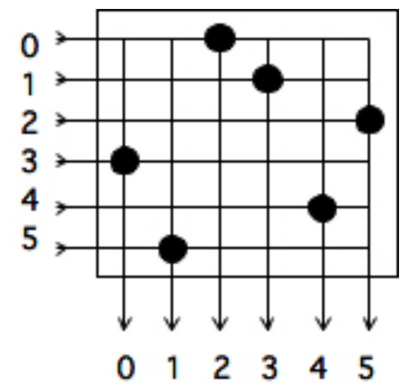
Pros and cons of hypercubes

- Pro: processors are close together: never more than log(P)
- Lots of bandwidth
- Little chance of contention
- Con: the number of wires out of a processor depends on P: complicated design
- Values of P other than 2^p not possible.

Busses/Hubs and Crossbars

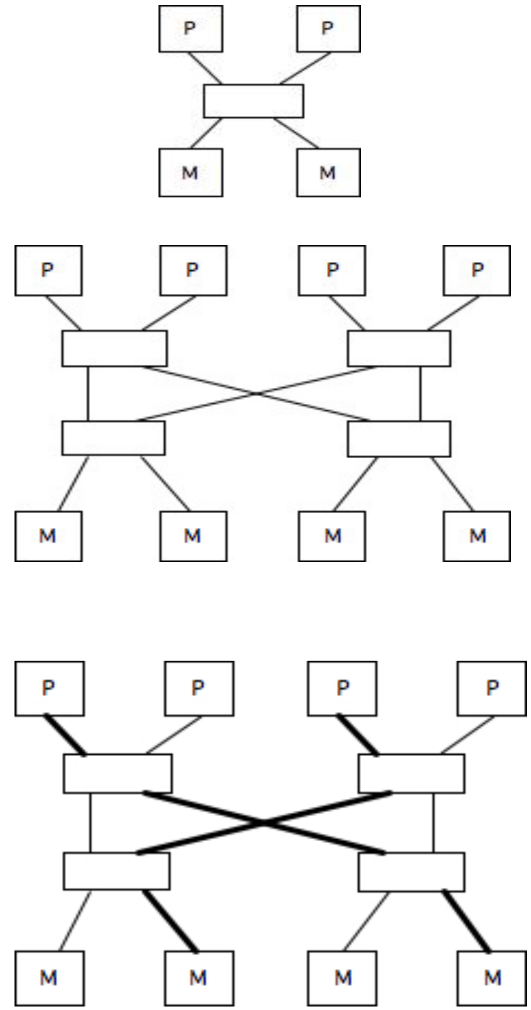
Hub/Bus: Every processor shares the communication links

Crossbar Switches: Every processor connects to the switch which routes communications to their destinations



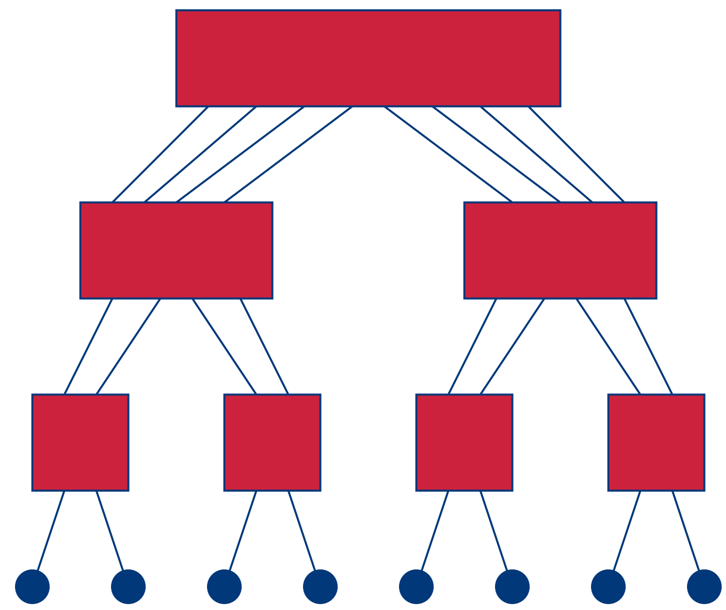
Butterfly exchange network

- Built out of simple switching elements
- Multi-stage; #stages grows with #procs
- Multiple non-colliding paths possible
- Uniform memory access



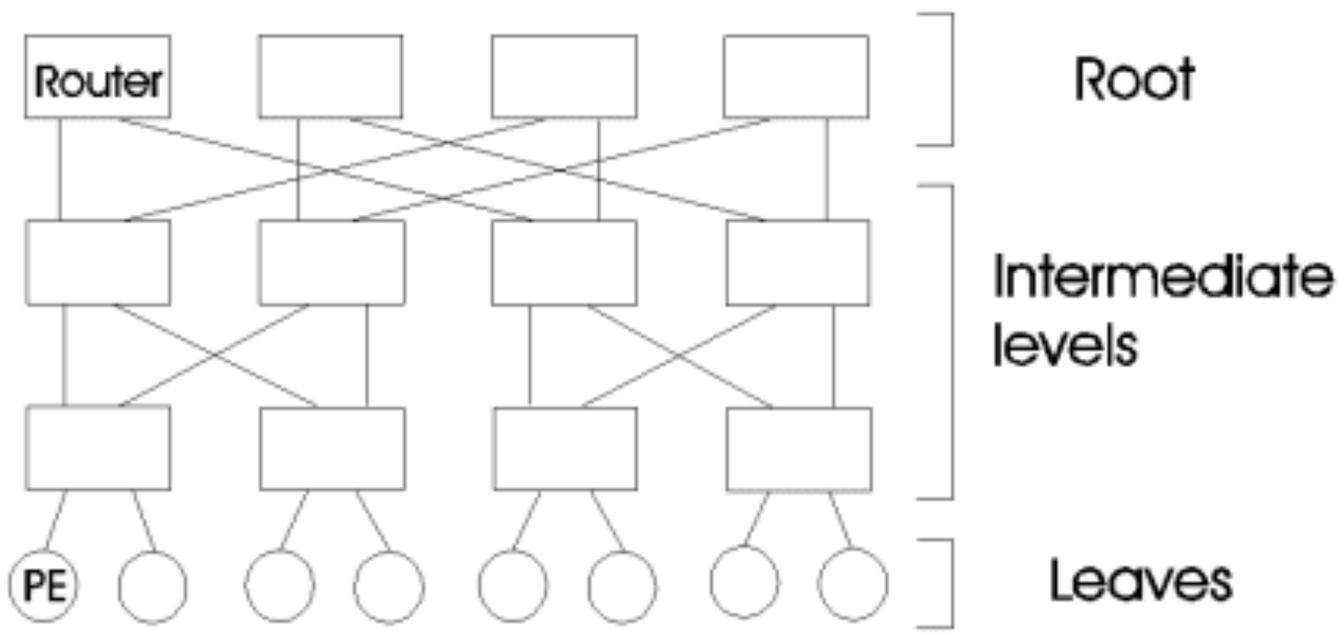
Fat Trees

- Multiple switches
- Each level has the same number of links in as out
- Increasing number of links at each level
- Gives full bandwidth between the links
- Added latency the higher you go



Fat Trees

- in practice emulated by switching network



Interconnect graph theory

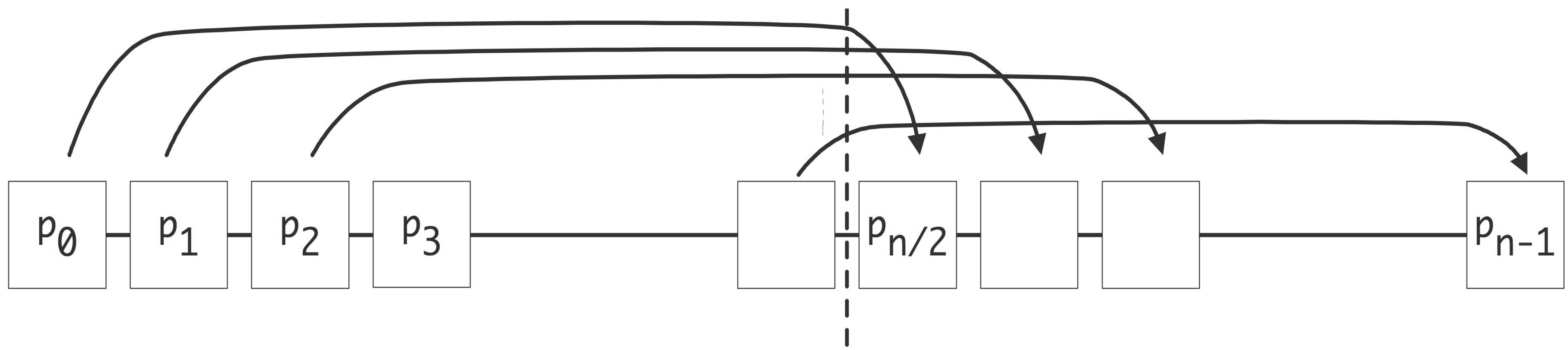
- Degree
  - How many links to other processors does each node have?
  - More is better, but also expensive and hard to engineer
- Diameter
  - maximum distance between any two processors in the network.
  - The distance between two processors is defined as the shortest path, in terms of links, between them.
  - completely connected network is 1, for star network is 2, for ring is p/2 (for p even processors)
- Connectivity
  - measure of the multiplicity of paths between any two processors (# arcs that must be removed to break the connection).
  - high connectivity is desired since it lowers contention for communication resources.
  - 1 for linear array, 1 for star, 2 for ring, 2 for mesh, 4 for torus
  - technically 1 for traditional fat trees, but there is redundancy in the switch infrastructure

Practical issues in interconnects

- Latency : How long does it take to start sending a "message"? Units are generally microseconds or milliseconds.
- Bandwidth : What data rate can be sustained once the message is started? Units are Mbytes/sec or Gbytes/sec.
  - Both point-to-point and aggregate bandwidth are of interest
- Multiple wires: multiple latencies, same bandwidth
- Sometimes shortcuts possible: `wormhole routing'

Measures of bandwidth

- Aggregate bandwidth: total data rate if every processor sending: total capacity of the wires. This can be very high and quite unrealistic.
- Imagine linear array with processor *i* sending to *P/2+i*: `Contention'
- Bisection bandwidth: bandwidth across the minimum number of wires that would split the machine in two.


p_0
p_1
p_2
p_3
$p_{n/2}$
p_{n-1}

Interconnects

- Bisection width
  - Minimum # of communication links that have to be removed to partition the network into two equal halves. Bisection width is
  - 2 for ring, sq. root(p) for mesh with p (even) processors, p/2 for hypercube, (p*p)/4 for completely connected (p even).
- Channel width
  - of physical wires in each communication link
- Channel rate
  - peak rate at which a single physical wire link can deliver bits
- Channel BW
  - peak rate at which data can be communicated between the ends of a communication link
  - = (channel width) * (channel rate)
- Bisection BW
  - minimum volume of communication found between any 2 halves of the network with equal # of procs
  - = (bisection width) * (channel BW)

Summary

- Why so much parallel talk?
  - Every computer is a parallel computer now
  - Good serial computing skills a central to good parallel computing
  - Cluster and MPP nodes are appear largely like desktops and laptops
    - Processing units: CPUs, FPUs, GPUs
    - Memory hierarchies: Registers, Caches, Main memory
    - Internal Interconnect: Buses and Switch-based networks
  - Clusters and MPPs built via fancy connections.

Title

- Test text, filler to see how font and color work with the background image.

```
Code should be written in a
Courier font, in black
```

- Test text, filler to see how font and color work with the background image.
- Test text, filler to see how font and color work with the background image.