Computational Optimization
ISE 407

Lecture 9

Dr. Ted Ralphs

Reading for this Lecture

- Aho, Hopcroft, and Ullman, Chapter 1
- Miller and Boxer, Chapters 1 and 5
- Fountain, Chapter 4
- "Introduction to High Performance Computing", V. Eijkhout, Chapter 2.
- "Introduction to High Performance Computing for Scientists and Engineers," G. Hager and G. Wellein, Chapter 1.

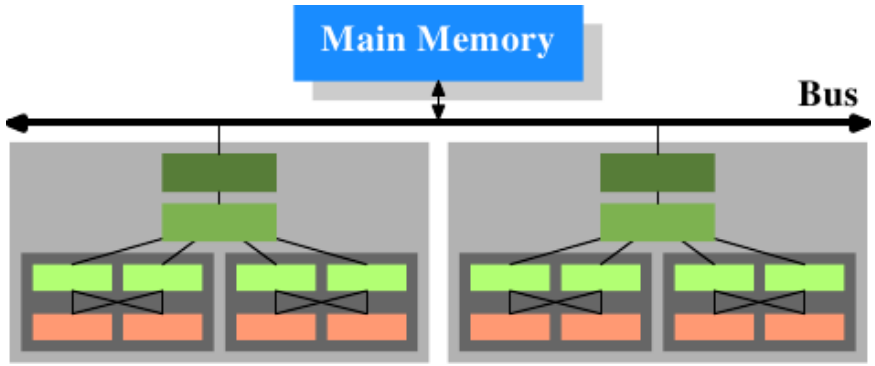
Parallel Algorithms and Parallel Platforms

- A *sequential algorithm* is a procedure for solving a given (optimization) problem that executes one instruction at a time, as in the RAM model.
- A *parallel algorithm* is a scheme for performing an equivalent set of computations that can execute more than one instruction at a time.
- Analyzing parallel algorithm is inherently more difficult, since the assumptions we make about storage and data movement can make a huge difference.
- A *parallel platform* is a combination of the
  - Hardware
  - Software
  - OS
  - Toolchain
  - *Communication Infrastructure*

  which enable a given parallel algorithm to be implemented and executed.
- Measuring practical performance of a parallel algorithm on a particular parallel platform is an alternative that is also challenging.
- It may be difficult to identify what components are affecting performance.

Parallel Architectures

- There is a wide variety of architectures when it comes to parallel computers.
- The simplest parallel computer is a single CUP with multiple cores.
- A single computer (compute node) can have multiple CPUs.
- Multiple compute nodes can be connected by a communication infrastructure that allows them to function as a "cluster".
- We can connect "clusters" into a "grid" with communications happening over the Internet.
- And so on.



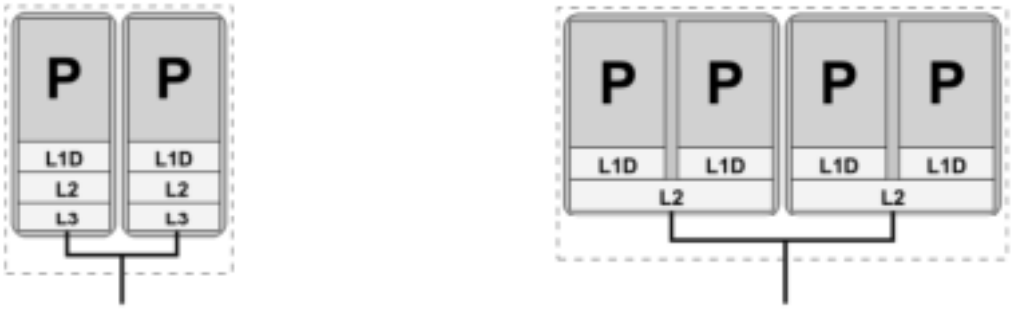
Source: https://lwn.net/Articles/250967

Platform Categories

- High Performance Parallel Computers
  - Massively parallel
  - Distributed
- "Off the shelf" Parallel Computers
  - Small shared memory computers
  - Multi-core computers
  - GPUs
  - Clusters (of multi-core computers)

The Storage Hierarchy

- It is clear that the storage hierarchy can become *very complex*.
- In a multi-core CPU, cache may be shared by groups of cores or each may have its own (and different levels might be different).
- In a multi-CPU computer, there may be multiple memory controllers or a single one.



Source: Hager and Wellein, Figure 1.2

Distributed versus Shared Memory

- When we move to analyzing clusters and grids, it becomes much more important to understand data movement.
- The cost of moving data become mu=ch more pronounced and the importance of optimizing such data movements become more than just "icing on the cake."
- With respect to a single compute core, we can roughly divide available memory into that which directly addressable and that which is not.
- Generally speaking, all the RAM associated with a compute node is addressable by all cores (*shared memory*).
- The memory that is not directly addressable is generally memory attached to other compute nodes (*distributed memory*).
- Accessing shared memory will generally be orders of magnitude faster than accessing distributed memory.

Processes and Threads

- Although all memory on a compute node is addressable by all cores, a computer will generally have multiple processes executing simultaneously.
- For security reasons, these processes are assigned separate memory address spaces by the OS and have no direct means of communicating.
- A process can, however, have multiple threads that execute independently but share memory.
- In a multi-core system, different threads from the same process can execute on different cores.

Cache Coherency

- A challenge for shared memory architectures is to maintain "cache coherency."
- Since each core may have its own cache, there may be multiple copies of the same data.
- If a cached copy is over-written, then it becomes "dirty" and other cached copies are invalidated.
- This can lead to inefficiency if different cores are trying to access the same memory locations simultaneously.

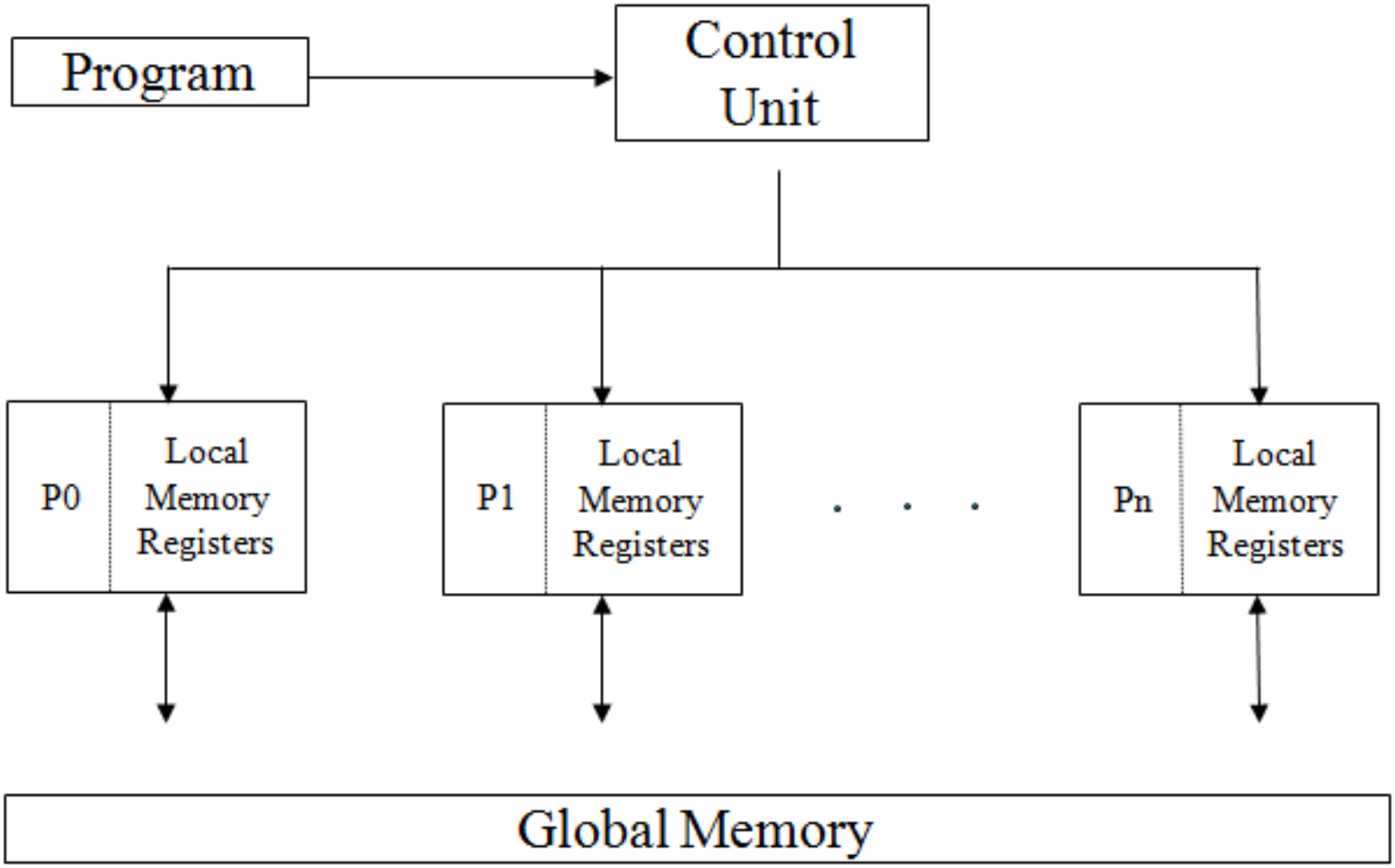
Hyperthreading

- Hyperthreading is a technique for allowing multiple threads to execute efficiently using the same core.
- When one thread is idle due to a cache miss (i.e., waiting for data to be retrieved), other threads can be run.
- In practice, this may create speed-ups similar to what one would observe with multiple cores.

Analysis of Parallel Algorithms

- The analysis of parallel algorithms is more difficult.
- The assumptions of the model make a much bigger difference.
- It is no longer true that all reasonable models are polynomially equivalent.

The Basic PRAM model



Assumptions of the PRAM model

- This is a synchronous model with shared memory.
- There are a fixed number of cores (bounded).
- All cores execute the same program, but each one can be in a different place.
- At each time step, each core performs one read, one elementary operation, and one write.
- Memory access is performed in constant time.
- Cores are not linked directly.
- Communication issues are not considered.

Concurrent Memory Access

- What if two cores try to read/write to/from the same memory location in the same time step?
- We have to resolve these conflicts.
- Four possible models:
  - CREW $\Leftarrow$ We will use this one (most of the time)
  - CRCW
  - EREW
  - ERCW

Assessment of the PRAM Model(s)

- This model is not as "robust" as the RAM model.
- However, it allows us to do rigorous analysis.
- It is a reasonable model of a small parallel machine.
- It is not "scalable."
- It does not model distributed memory or interconnection networks.
- How do we fix it?

Distributed PRAM Model

- Attempt to model the interconnection network.
- Eliminate global memory.
- Each core can read or write only from its neighbors' registers.
- This will likely increase the complexity of many algorithms, but is more realistic and scalable.

What is an interconnection network?

- A graph (directed or undirected)
- The nodes are the processors
- The edges represent direct connections
- Properties and Terms
  - Degree of the Network
  - Communication Diameter
  - Bisection Width
  - Processor Neighborhood
  - Connectivity Matrix
  - Adjacency Matrix

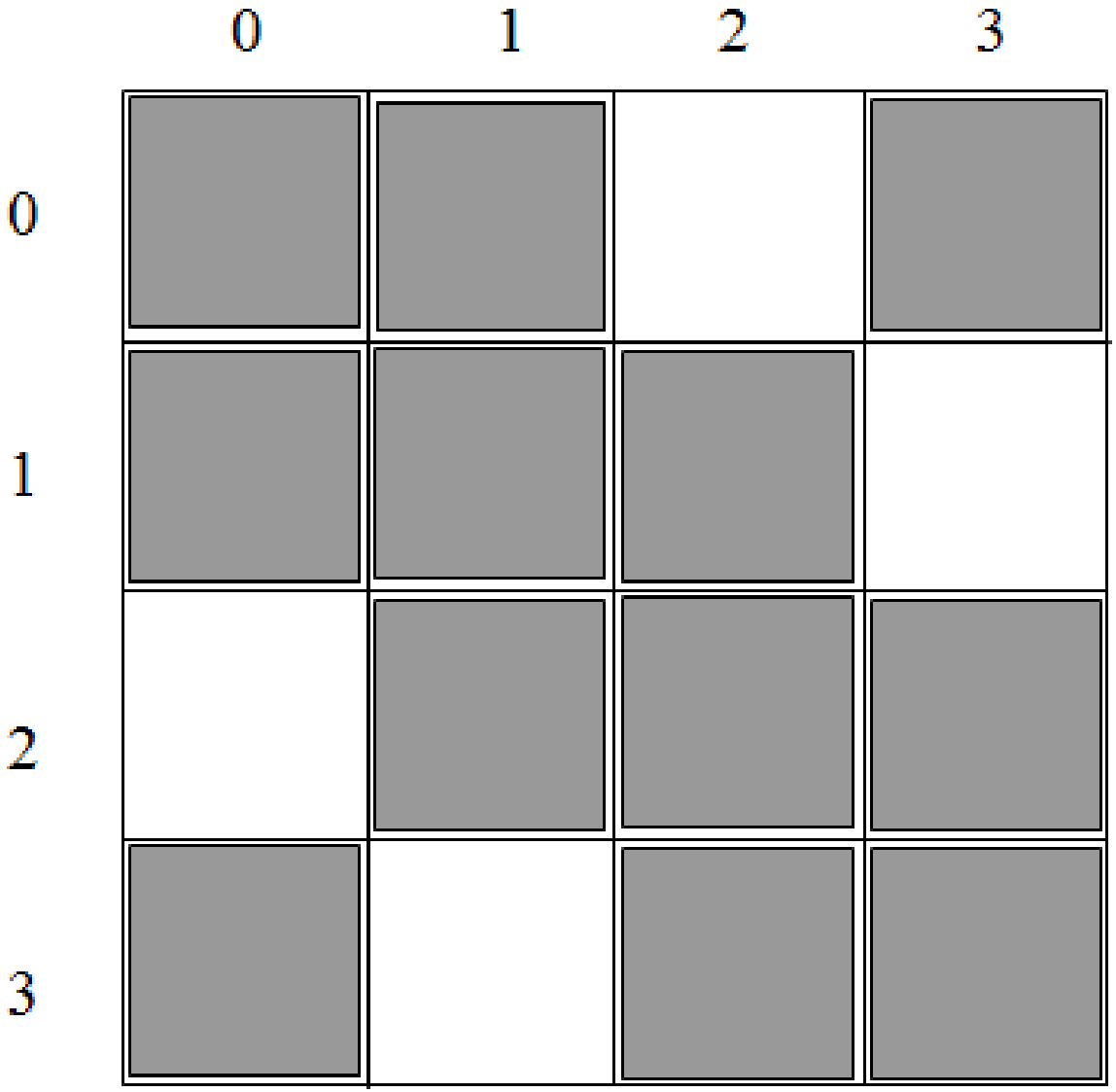
Measures of Goodness

- Communication diameter: The maximum shortest path between two processors.
- Bisection width: The minimum cut such that the two resulting sets of processors have the same order of magnitude.
- Connectivity Matrix
- Adjacency Matrix

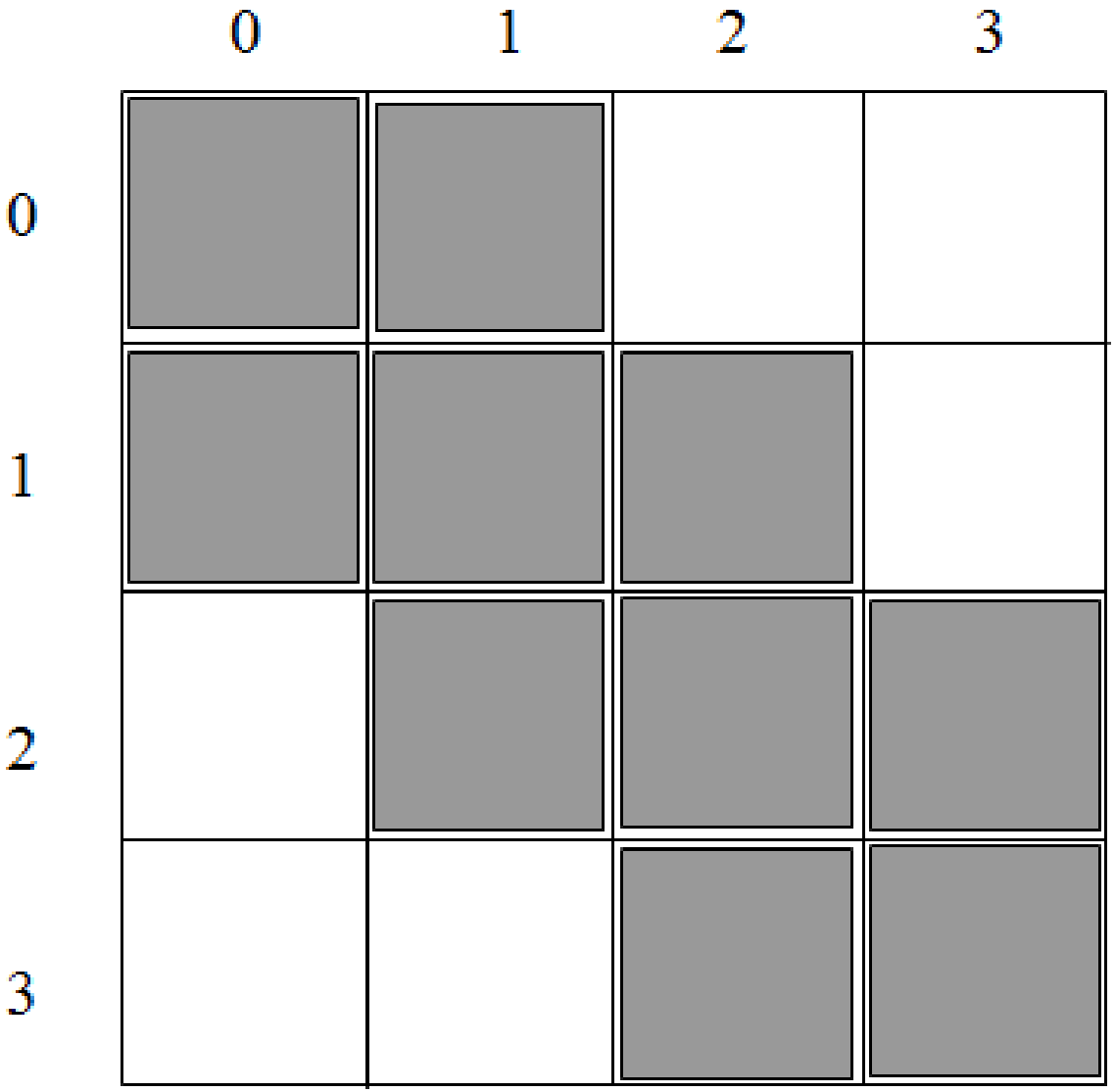
Bottlenecks

- The communication diameter indicates how long it may take to send information from one processor to another.
- Thus, it may be the bottleneck in any algorithm in which the data are initially distributed equally.
- The bisection width is the bottleneck when processors must exchange large amounts of information.
- The bisection width is a lower bound for sorting.

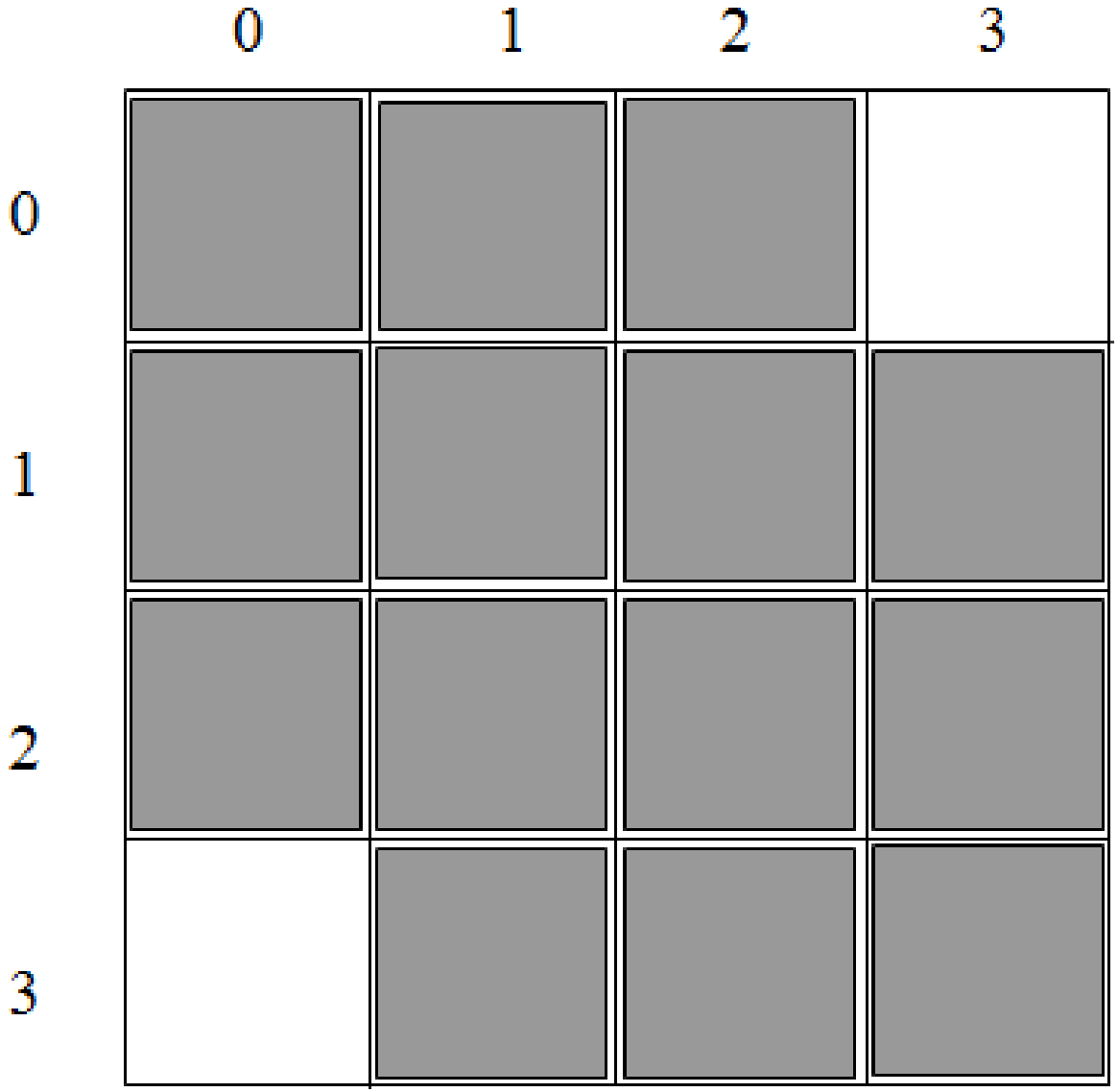
Connectivity Matrix: Example 1



Connectivity Matrix: Example 2



2-step Connectivity Matrix



N-step Connectivity Matrices

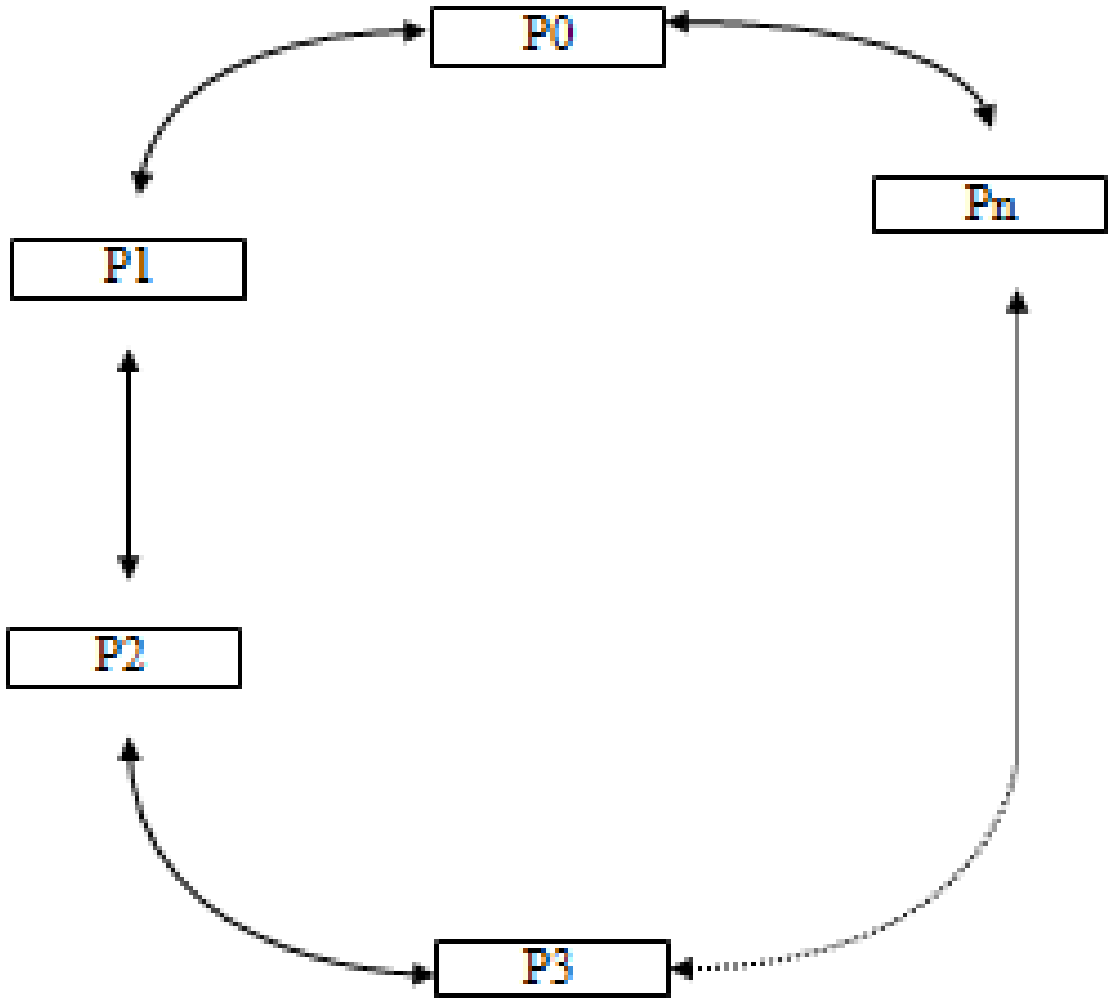
- Indicates the processor pairs that can reach each other in N steps
- Computed using Boolean matrix multiplication
- The corresponding adjacency matrix indicates how many disjoint paths connect each pair.

	0	1	2	3
0	1	1	1	
1	1	1	1	1
2	1	1	1	1
3		1	1	1

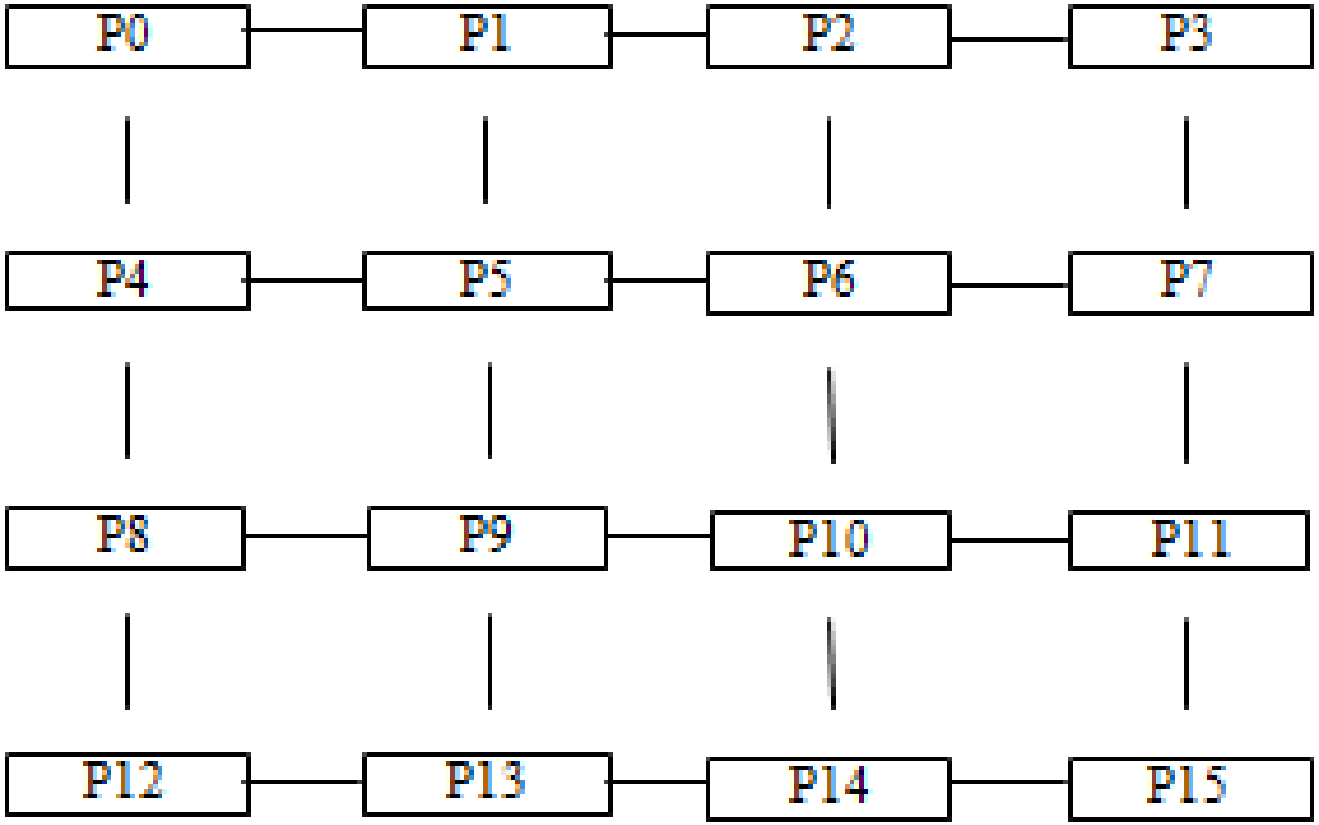
	0	1	2	3
0	1	1	2	1
1	1	1	1	2
2	2	1	1	1
3	1	2	1	1

Linear Array

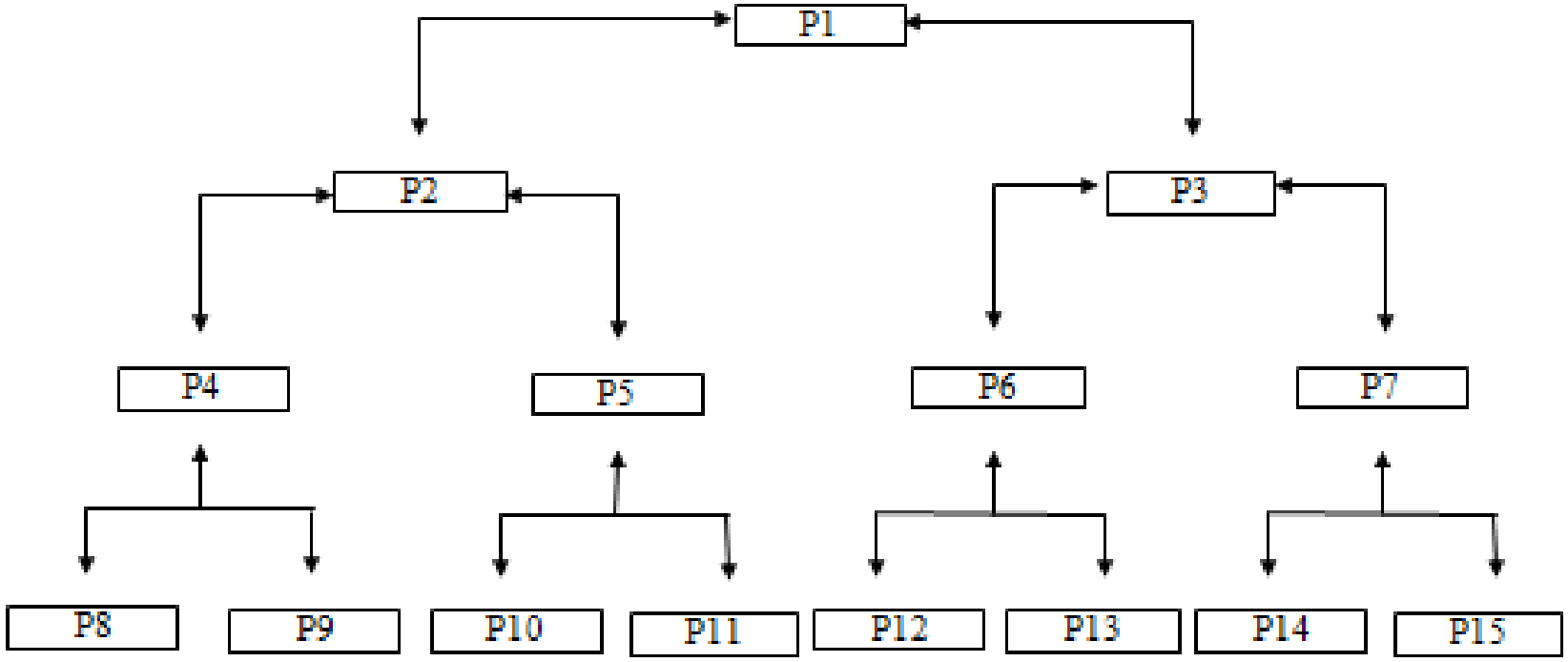
Ring



Mesh



Tree



Other Schemes

- Pyramid: A 4-ary tree where each level is connected as a mesh
- Hypercube: Two processors are connected if and only if their ID #'s differ in exactly one bit.
  - Low communications diameter
  - High bisection width
  - Doesn't have constant degree
- Perfect Shuffle: Processor i is connected one-way to processor $2i \mod N - 1$.
- Others: Star, De Bruijn, Delta, Omega, Butterfly

Asymptotic Analysis of Parallel Algorithm

- In the course of a parallel architecture, small details make a difference.
- Example: broadcasting a unit of data
  - $\Theta(1)$ under the shared-memory CREW model
  - $\Theta(n)$ under the shared-memory EREW model
  - $\Theta(\sqrt{n})$ under the distributed-memory CREW model on a mesh
  - $\Theta(\log n)$ under the distributed-memory tree model
- Note: These models are architecture dependent
- This is the biggest difference between sequential and parallel complexity analysis

Cost of a Parallel Algorithm

- In the case of a RAM algorithm, the measure of effectiveness was the time (number of steps).
- In the PRAM case, we may consider both time and "cost."
  - Running time is the number of steps required in "real time."
  - Parallel cost is the product of running time and number of cores.
- An "optimal" parallel algorithm is one for which the parallel cost function is of the same order as the sequential running time function.
- The difference between the sequential running time and the parallel cost is known as *parallel overhead*.
- It consists of time steps during which a core is idle or doing something not required in the parallel algorithm (e.g., moving data).
- For algorithms that are not optimal, the running time decreases with additional cores, but the cost increases.

Speedup and Parallel Efficiency

- Speedup and parallel efficiency are concepts related to parallel cost.
- Speedup is the ratio of the parallel running time to the sequential running time.
- Efficiency is the speedup divided by the number of cores.
- Optimal algorithms are those whose speedup is equal to the number of cores or with a parallel efficiency of 1.
- Essentially, these are algorithms that balance communication and idle time with time for computation.

Semigroup operations

- Definition: A binary associative operation

$$(x \otimes y) \otimes z = x \otimes (y \otimes z)$$

- Typical semigroup operations.
  - maximum
  - minimum
  - sum
  - product
  - OR
- Can be used to compare parallel architectures.

Semigroup operations example

- RAM Algorithm: Can't do better than sequential search, which is $\Theta(n)$.
- Shared-memory PRAM Algorithm
  Assumptions: n cores, CREW
  Input: An array $x = [x_1, x_2, ..., x_{2n}]$ (2 data elements per core initially)
  Output: The smallest entry of X

```
1  for (i = 0; i < log2(n); i++){
2     parallel for (j = 0; j < 2**(log2(n)-i-1); j++){
3        read x[2j-1] and x[2j];
4           write min(x2j-1, x2j);
5     }
6  }
```

Semigroup operations example (cont'd)

- The parallel cost of this implementation is $\Theta(n \lg n)$, so this is not cost optimal.
- Can we achieve cost optimality?
  - Starting with one data element per core, we can't expect a running time better than $\Theta(\lg n)$.
  - The problem is that we are not fully utilizing all the cores.
  - Including the idle time, there is an overall increase of the number of total steps required of $\lg n$.
  - How do we improve this situation?

Scaling Up

- The problem is that there simply isn't enough data to utilize all the processing power.
- If we had N cores and $n > N$ data elements, what would change?
  - Start with n/N data elements per core.
  - First apply the sequential algorithm to the n/N elements stored on each core.
  - Then combine the results using the original parallel algorithm.
- What should N be, as a function of n?
  - The running time is $\Theta(n/N + \lg N)$.
  - The cost is $\Theta(n + N \lg N)$.
- What should N be to achieve cost optimality?

Scaling Up

- The problem is that there simply isn't enough data to utilize all the processing power.
- If we had N cores and $n > N$ data elements, what would change?
  - Start with n/N data elements per core.
  - First apply the sequential algorithm to the n/N elements stored on each core.
  - Then combine the results using the original parallel algorithm.
- What should N be, as a function of n?
  - The running time is $\Theta(n/N + \lg N)$.
  - The cost is $\Theta(n + N \lg N)$.
- What should N be to achieve cost optimality?
  - We want $N \lg N \approx n$.
  - Taking $N = n/\lg n$ is an approximate solution.

The General Principle

- The previous analysis illustrates a general principle.
- When adding more cores, there is a limit based on the size of the input beyond which we cannot effectively utilize the additional cores.
- We must scale up the input size along with the number of cores in order to achieve "scalability."
- We will examine this phenomena in more detail in a future lecture.

Other Benchmark Problems

- Broadcast:
  - Send value from one processor to all others
  - Limited by diameter
- Sorting:
  - Sort a list of values
  - Limited by bisection bandwidth
- Semigroup
  - Combine values using a binary associative operator
  - Requires bandwidth and diameter to be balanced