Performance

Parallel Computing – COSC 3P93

Course Outline

- Introduction
- Parallel Hardware
- Parallel Software
- **Performance**
- Parallel Programming Models
- Patterns
- SMP with Threads and OpenMP
- Distributed-Memory Programming with MPI
- Algorithms
- Tools
- Parallel Program Design

Today Class

- Performance
  - Scalable parallel execution
  - Parallel execution models
  - Isoefficiency

Scalable Parallel Computing

- Scalability in parallel architecture
  - Processor numbers
  - Memory architecture
  - Interconnection network
  - Avoid critical architecture bottlenecks
- Scalability in computational problem
  - Problem size
  - Computational algorithms
    - Computation to memory access ratio
    - Computation to communication ratio
- Parallel programming models and tools
- Performance scalability

Why Aren't Parallel Applications Scalable?

- **Sequential performance**
- **Critical Paths**
  - Dependencies between computations spread across processors
- **Bottlenecks**
  - One processor holds things up
- **Algorithmic overhead**
  - Some things just take more effort to do in parallel
- **Communication overhead**
  - Spending increasing proportion of time on communication
- **Load Imbalance**
  - Makes all processor wait for the "slowest" one
  - Dynamic behaviour
- **Speculative loss**
  - Do A and B in parallel, but B is ultimately not needed

Critical Paths

- **Long chain of dependence**
  - Main limitation on performance
  - Resistance to performance improvement
- **Diagnostic**
  - Performance stagnates to a (relatively) fixed value
  - Critical path analysis
- **Solution**
  - Eliminate long chains if possible
  - Shorten chains by removing work from critical path

Bottlenecks

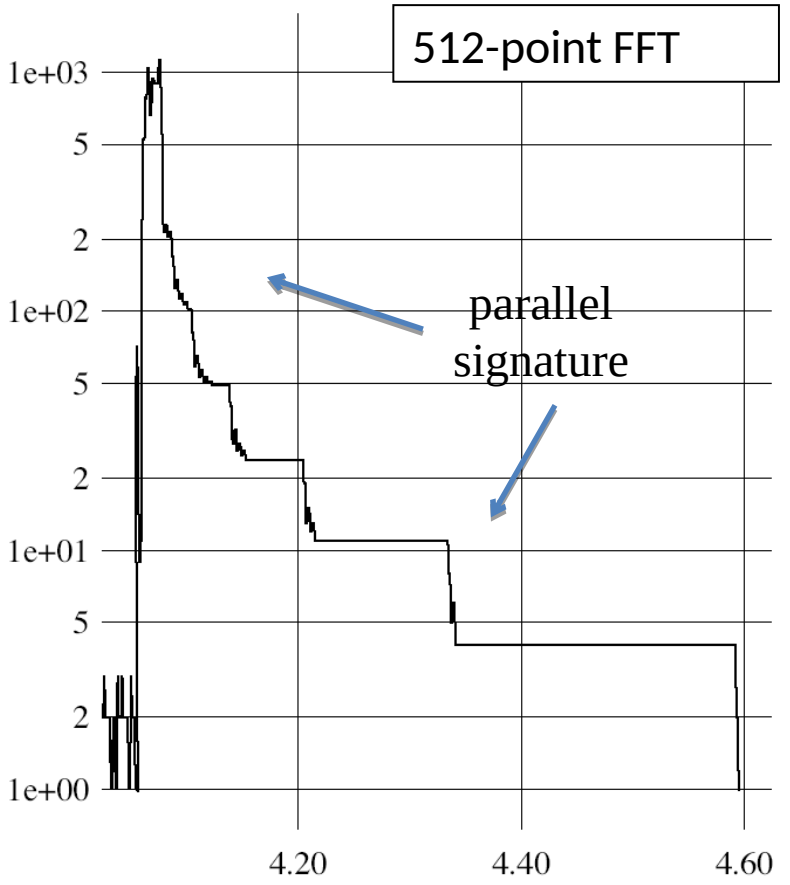
- **How to detect?**
  - One processor A is busy while others wait
  - Data dependency on the result produced by A
- **Typical situations**
  - N-to-1 reduction / computation / 1-to-N broadcast
  - One processor assigning job in response to requests
- **Solution techniques**
  - More efficient communication
  - Hierarchical schemes for master slave
- **Program may not show ill effects for a long time**
- Shows up when scaling

Algorithmic Overhead

- Different sequential algorithms to solve the same problem
- All parallel algorithms are sequential
  - When run on 1 processor
- All parallel algorithms introduce addition operations (Why?)
  - Parallel overhead
- Where should the starting point be for a parallel algorithm?
  - Best sequential algorithm might not parallelize at all
  - Or, it doesn't parallelize well (e.g., not scalable)
- What to do?
  - Choose algorithmic variants that minimize overhead
  - Use two level algorithms
- Performance is **the rub**
  - Are you achieving better parallel performance?
  - Must compare with the best sequential algorithm

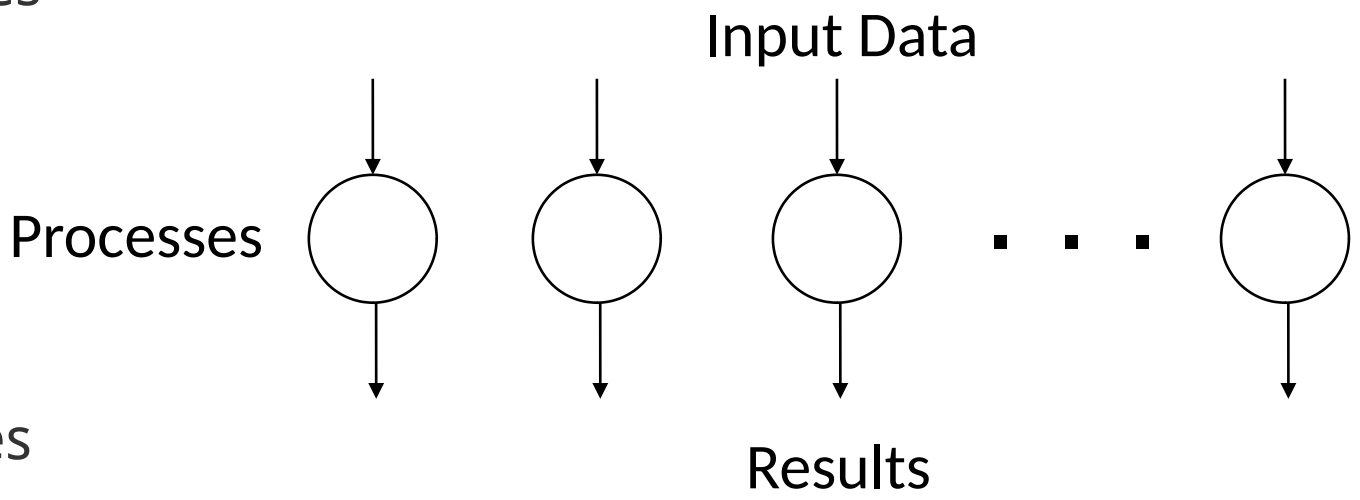
What is the maximum parallelism possible?

- Depends on application, algorithm, program
  - Data dependencies in execution
- Remember MaxPar
  - Analyzes the earliest possible “time” any data can be computed
  - Assumes a simple model for time it takes to execute instruction or go to memory
  - Result is the maximum parallelism available
- Parallelism varies!



Embarrassingly Parallel Computations

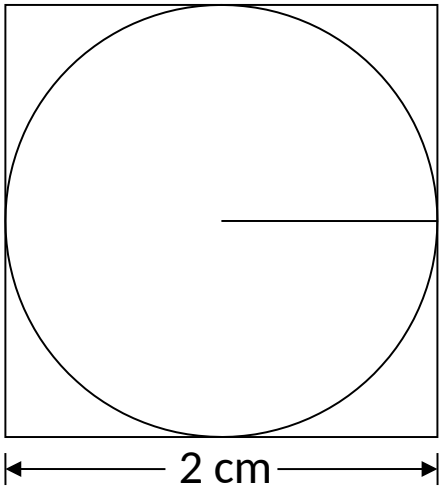
- No or very little communication between processes
- Each process can do its tasks without any interaction with other processes



- Examples
  - Numerical integration
  - Mandelbrot set
  - Monte Carlo methods

Calculating Pi with Monte Carlo

- Consider a circle of unit radius
- Place circle inside a square box with side of 2 cm

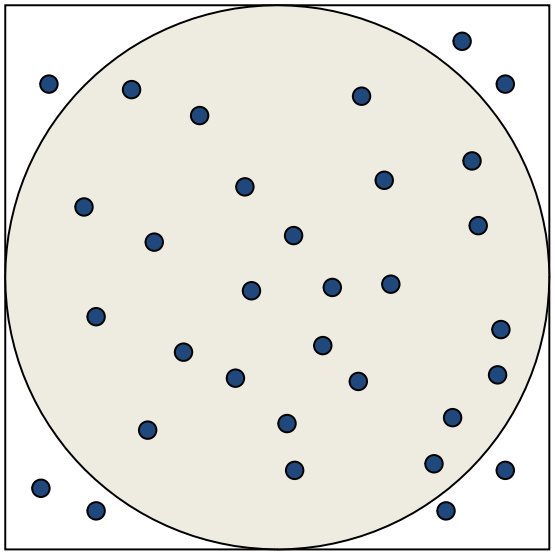


- The ratio of the circle area to the square area is

$$\frac{\pi * 1 * 1}{2 * 2} = \frac{\pi}{4}$$

Monte Carlo Calculation of Pi

- Randomly choose a number of points in the square
- For each point p
  - Determine if p is inside the circle
- The ratio of points in the circle to points in the square will give an approximation of Pi/4



Performance Metrics Review

- T1 is the execution time on a single processor
- Tp is the execution time on a **p** processor system
- S(p) (Sp) is the speedup

$$S(p) = \frac{T_1}{T_p}$$

- E(p) (Ep) is the efficiency

$$Efficiency = \frac{S_p}{p}$$

- Cost(p) (Cp) is the cost

$$Cost = p \times T_p$$

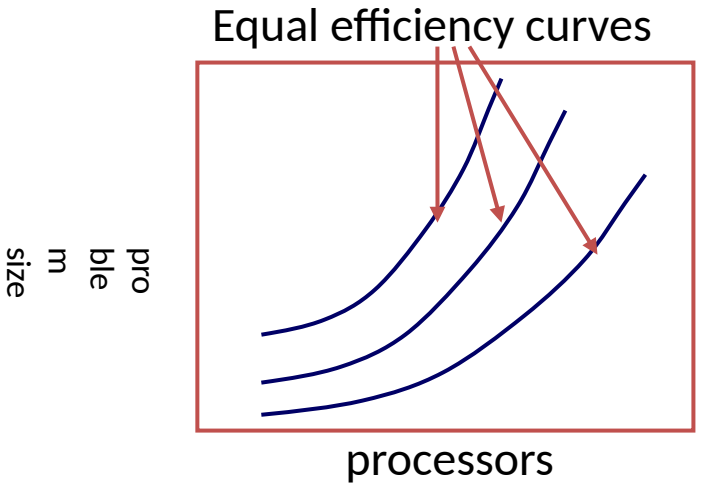
- Parallel algorithm is cost-optimal
  - Parallel time = sequential time ($C_p = T_1$, $E_p = 100\%$)
-

Analytical / Theoretical Techniques

- Involves simple algebraic formulas and ratios
  - Typical variables
    - Data size *(N)*, number of processors *(P)*, machine constants
  - To model performance of individual operations, components, algorithms in terms of the above
    - Be careful to characterize variations across processors
    - Model them with max operators
  - **Constants are important in practice**
    - Use asymptotic analysis carefully
- Scalability analysis
  - Isoefficiency (Kumar)

Isoefficiency

- Goal is to quantify scalability
- How much increase in problem size is needed to retain the same efficiency on a larger machine?
- Efficiency
  - $T_1 / (p * T_p)$
  - T_p = computation + communication + idle
- Isoefficiency
  - Equation for equal-efficiency curves
  - If no solution
    - Problem is not scalable in the sense defined by isoefficiency



Scalability of Adding n Numbers

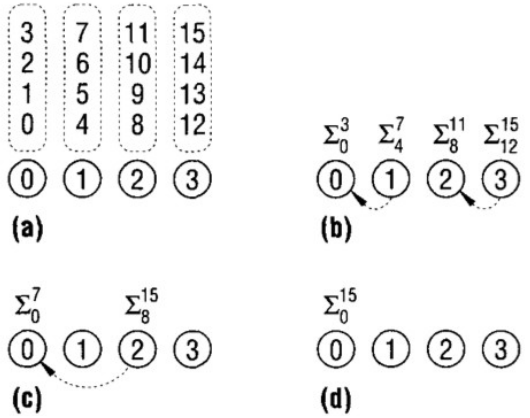
- Scalability of a parallel system is a measure of its capacity to increase speedup with more processors
- Adding **n** numbers on **p** processors with strip partition

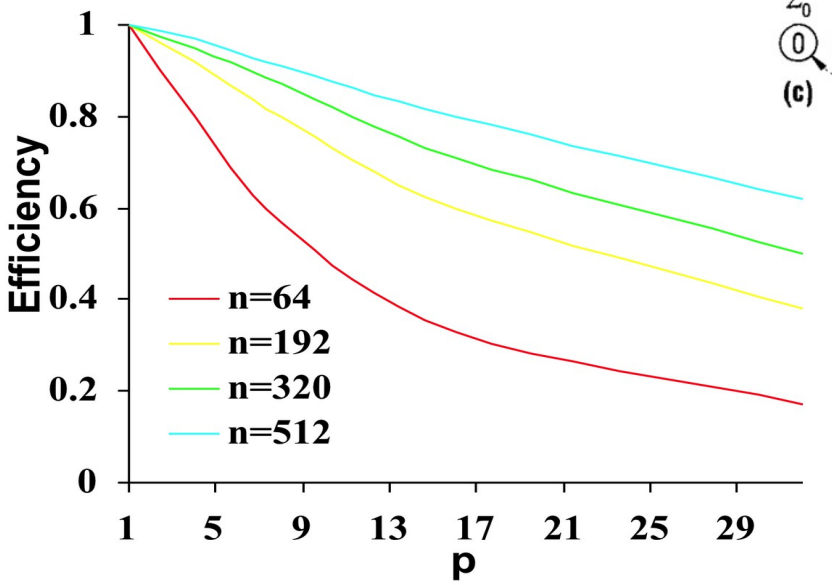
$$T_{par} = \frac{n}{p} - 1 + 2\log p$$

$$Speedup = \frac{n - 1}{\frac{n}{p} - 1 + 2\log p}$$

$$\approx \frac{n}{\frac{n}{p} + 2\log p}$$

$$Efficiency = \frac{S}{p} = \frac{n}{n + 2p\log p}$$





Problem Size and Overhead

- Informally, problem size is expressed as a parameter of the input size
- A consistent definition of the size of the problem is the total number of basic operations (T_{seq})
  - Also refer to problem size as work ($W = T_{seq}$)
- Overhead of a parallel system is defined as the part of the cost not in the best serial algorithm
- Denoted by T_O, it is a function of W and p

$$T_O(W,p) = pT_{par} - W \qquad (pT_{par} \text{ includes overhead})$$

$$T_O(W,p) + W = pT_{par}$$

Isoefficiency Function

- With a fixed efficiency, W is as a function of **p**

$$T_{par} = \frac{W + T_O(W, p)}{p} \qquad W = T_{seq}$$

$$Speedup = \frac{W}{T_{par}} = \frac{Wp}{W + T_O(W, p)}$$

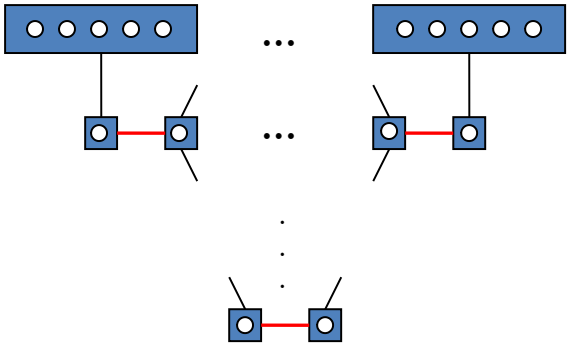
$$Efficiency = \frac{S}{p} = \frac{W}{W + T_O(W, p)} = \frac{1}{1 + \frac{T_O(W, p)}{W}}$$

$$E = \frac{1}{1 + \frac{T_O(W, p)}{W}} \rightarrow \frac{T_O(W, p)}{W} = \frac{1 - E}{E}$$

$$W = \frac{E}{1 - E} T_O(W, p) = K T_O(W, p) \quad \text{Isoefficiency Function}$$

Isoefficiency Function of Adding n Numbers

- Overhead function:
  - $T_O(W,p) = pT_{par} - W = 2p\log(p)$
- Isoefficiency function
  - $W=K*2p\log(p)$
- If p doubles
  - W needs also to be doubled to roughly maintain the same efficiency
- Isoefficiency functions can be more difficult to express for more complex algorithms



More Complex Isoefficiency Functions

- A typical overhead function T_o can have several distinct terms of different orders of magnitude with respect to both **p** and **W**
- We can balance **W** against each term of T_o and compute the respective isoefficiency functions for individual terms
  - Keep only the term that requires the highest grow rate with respect to p
  - This is the asymptotic isoefficiency function

Isoefficiency

- Consider a parallel system with an overhead function

$$T_o = p^{3/2} + p^{3/4} W^{3/4}$$

- Using only the first term

$$W = K p^{3/2}$$

- Using only the second term

$$W = K p^{3/4} W^{3/4}$$

$$W^{1/4} = K p^{3/4}$$

$$W = K^4 p^3$$

- K^4p^3 is the overall asymptotic isoefficiency function

Parallel Computation (Machine) Models

- **PRAM** (parallel RAM)
  - Basic parallel machine
- **BSP** (Bulk Synchronous Parallel)
  - Isolates regions of computation from communication
- **LogP**
  - Used for studying distributed memory systems
  - Focuses on the interconnection network
- **Roofline**
  - Based in analyzing “feeds” and “speeds”

PRAM Complexity Measures

- For each individual processor
  - **Time**: number of instructions executed
  - **Space**: number of memory cells accessed
- PRAM machine
  - **Time**: time taken by the longest running processor
  - **Hardware**: maximum number of active processors
- Technical issues
  - How processors are activated
  - How shared memory is accessed

PRAM is a Theoretical (Unfeasible) Model

- Interconnection network between processors and memory would require a very large amount of area
- The message-routing on the interconnection network would require time proportional to network size
- Algorithm's designers can forget the communication problems and focus their attention on the parallel computation only
- There exist algorithms simulating any PRAM algorithm on bounded degree networks
- Design general algorithms for the PRAM model and simulate them on a feasible network

Classification of PRAM Models

- EREW (Exclusive Read Exclusive Write)
  - No concurrent read/writes to the same memory location
- CREW (Concurrent Read Exclusive Write)
  - Multiple processors may read from the same global memory location in the same instruction step
- ERCW (Exclusive Read Concurrent Write)
  - Concurrent writes allowed
- CRCW (Concurrent Read Concurrent Write)
  - Concurrent reads and writes allowed
- CRCW > (ERCW,CREW) > EREW

Complexity of PRAM Algorithms

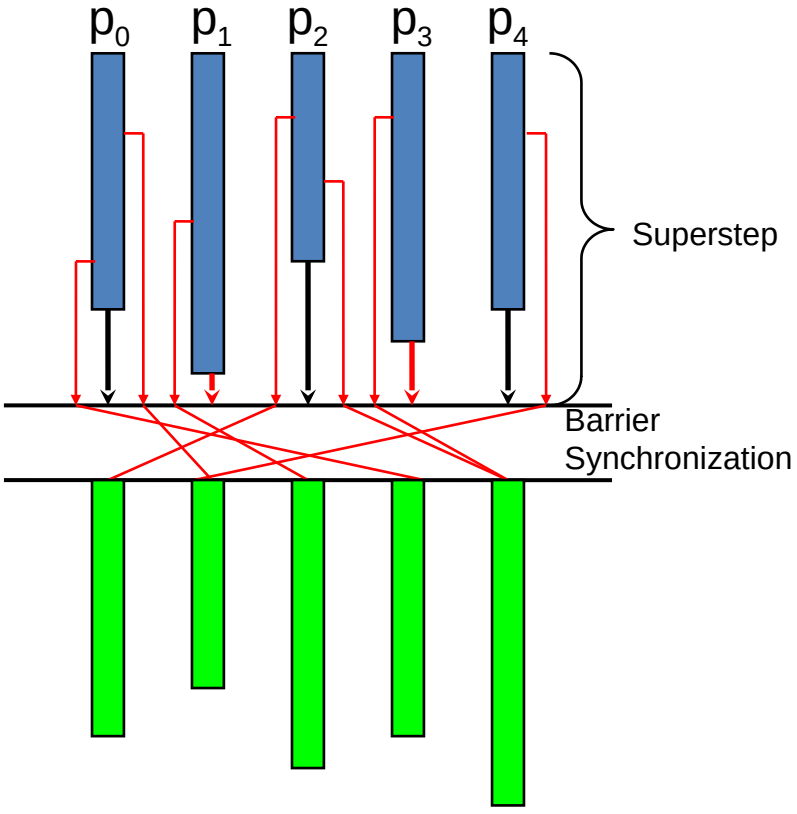
Problem / Model	EREW	CRCW
Search	O(log n)	O(1)
List Ranking	O(log n)	O(log n)
Prefix	O(log n)	O(log n)
Tree Ranking	O(log n)	O(log n)
Finding Minimum	O(log n)	O(1)

BSP Overview

- **Bulk Synchronous Parallelism**
- A parallel programming model
- Invented by Leslie Valiant at Harvard
- Enables performance prediction
- SPMD (Single Program Multiple Data) style
- Supports both direct memory access and message passing semantics
- BSPlib is a BSP library implemented at Oxford

BSP Supersteps

- A BSP computation consists of a sequence of supersteps
- In each superstep
  - Processes execute computations using locally available data
  - And issue communication requests
- Processes synchronized at the end of the superstep
  - At which all communications issued have been completed

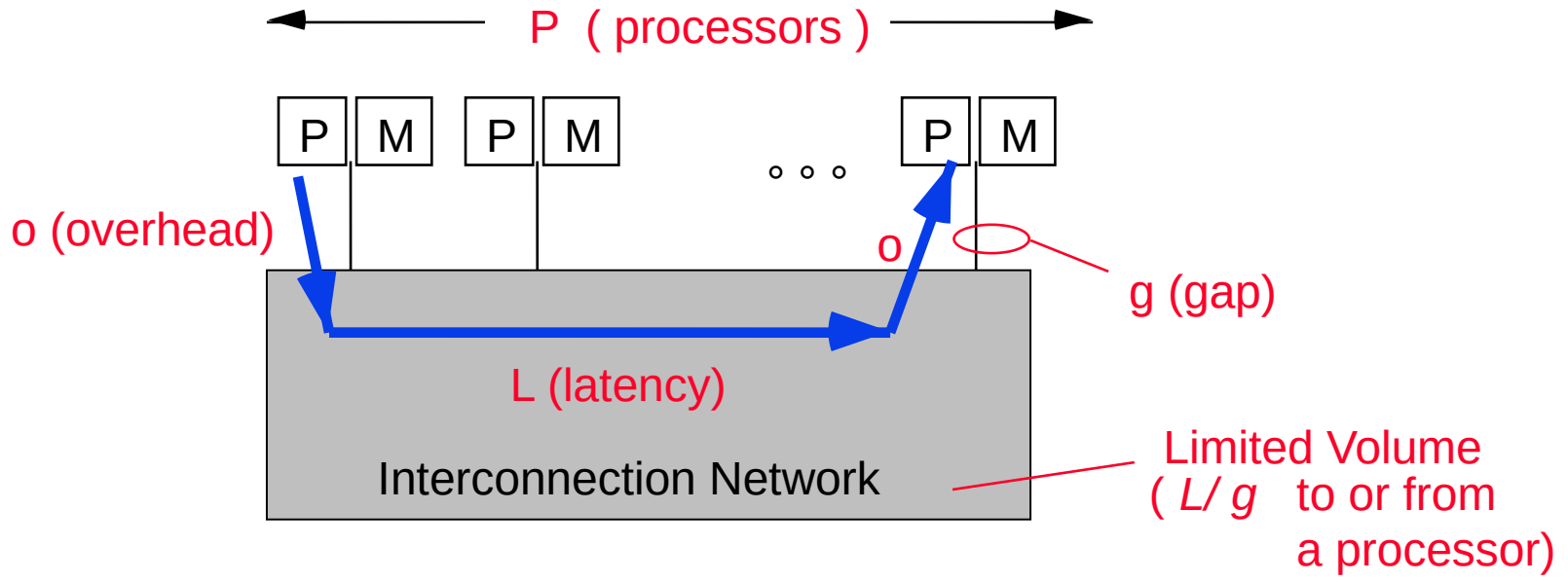


BSP Performance Model Parameters

- p = number of processors
- l = barrier latency, cost of achieving barrier synchronization
- g = communication cost per word
- s = processor speed
- l, g, and s are measured in FLOPS
- Any processor sends and receives at most h messages in a single superstep (called h-relation communication)
- Time for a superstep = max number of local operations performed by any one processor + g*h + l

LogP

- **L**atency in sending a (small) message between modules
- **o**verhead felt by the processor on sending or receiving message
- **g**ap between successive sends or receives (1/BW)
- **P**rocessors



LogP ”Philosophy"

- Strategy
  - Mapping of N words onto P processors
  - Computation within a processor
    - Its cost and balance
  - Communication between processors
    - Its cost and balance
- Characterize processor and network performance
- Do not think about what happens in the network
- This should be enough

Class Recap

- Performance
  - Scalable parallel execution
  - Isoefficiency
  - Parallel execution models

Next Class

- Parallel Programming Models