# ELE 455/555
# Computer System Engineering

## Section 4 – Parallel Processing

## Class 2 – Architectures

# Parallel Processing

## Hardware Multithreading

- HW Multithreading
  - Allow multiple threads to share the same processor
    - Thread = minimal process
    - Must retain the state for each thread
      - PC
      - Registers
      - SP
    - Use virtual memory technique to manage memory spaces
    - Must be fast – possibly switch every clock cycle
      - No OS interaction

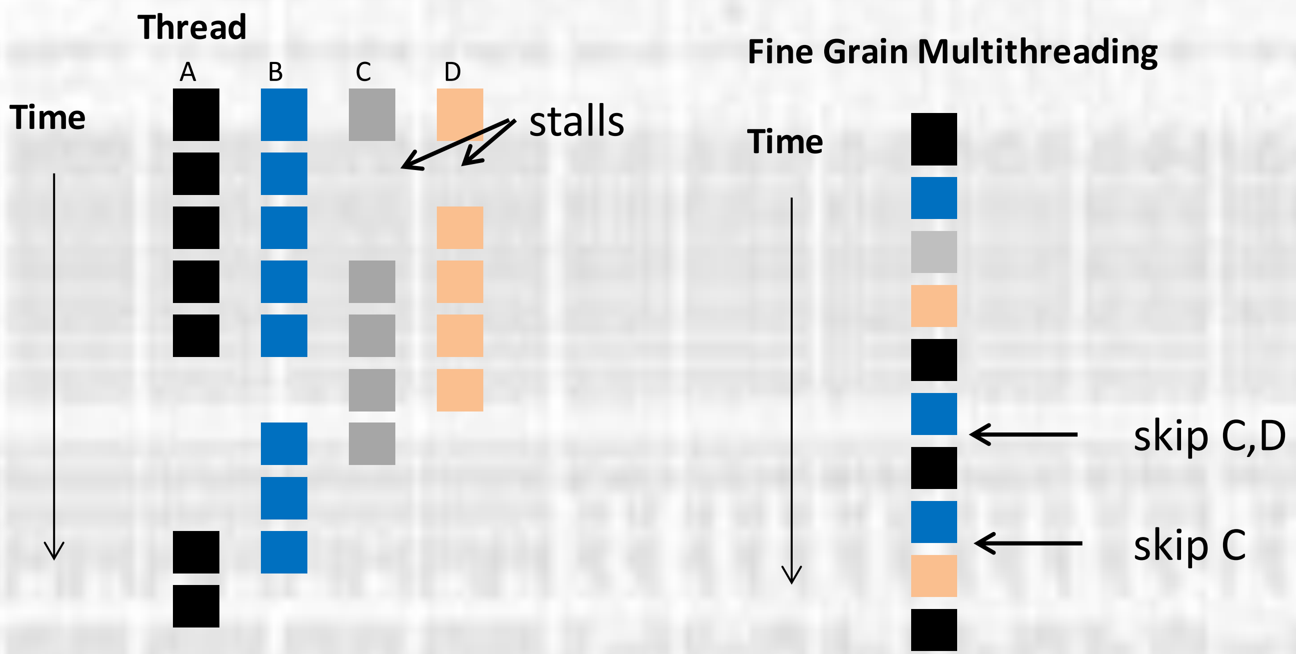
# Parallel Processing

## Hardware Multithreading

- Fine Grained Multithreading
  - Switch threads every clock cycle
  - Interleaved instructions executed in a round-robin fashion
    - Any thread (instruction) that is stalled is skipped
  - Reduces the impacts of long and short stalls
  - Increases the execution time for every thread

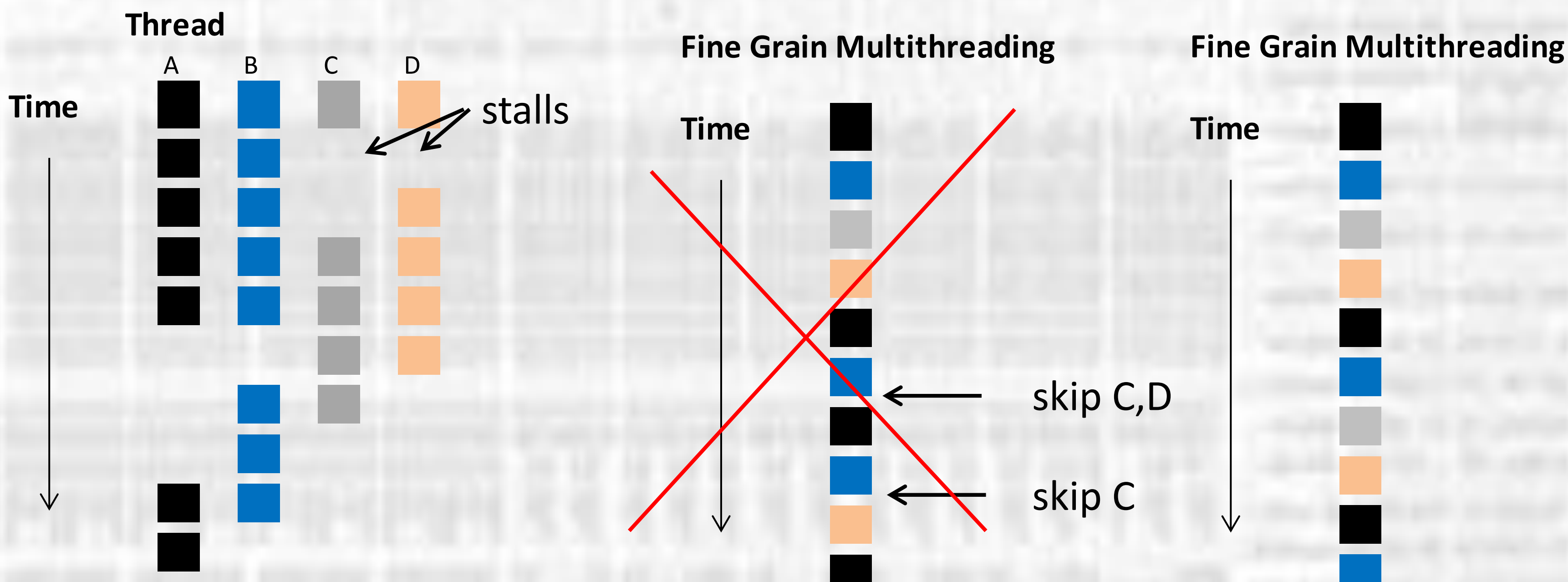
# Parallel Processing
## Hardware Multithreading

- Fine Grained Multithreading



# Parallel Processing
## Hardware Multithreading

- Fine Grained Multithreading



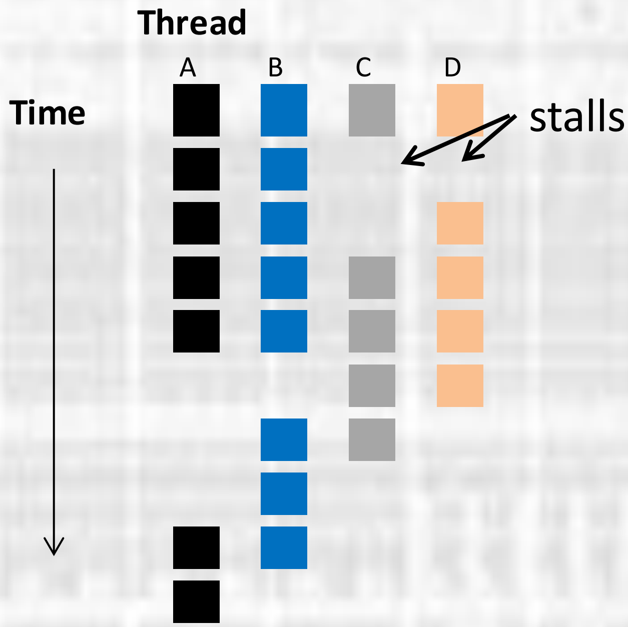
# Parallel Processing

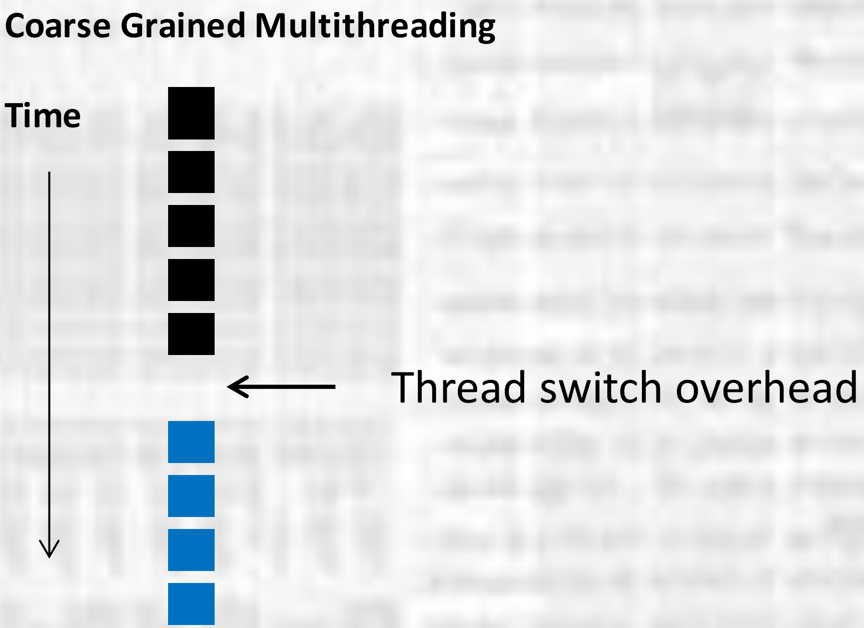
## Hardware Multithreading

- Coarse Grained Multithreading
  - Switch threads on long stalls
  - Threads execute until they reach a long stall
    - e.g. L2 cache miss
    - The next thread is started
  - Reduces the requirement for switching to be super fast
  - Has penalties associated with pipeline filling on each thread switch

# Parallel Processing
## Hardware Multithreading

- Coarse Grained Multithreading



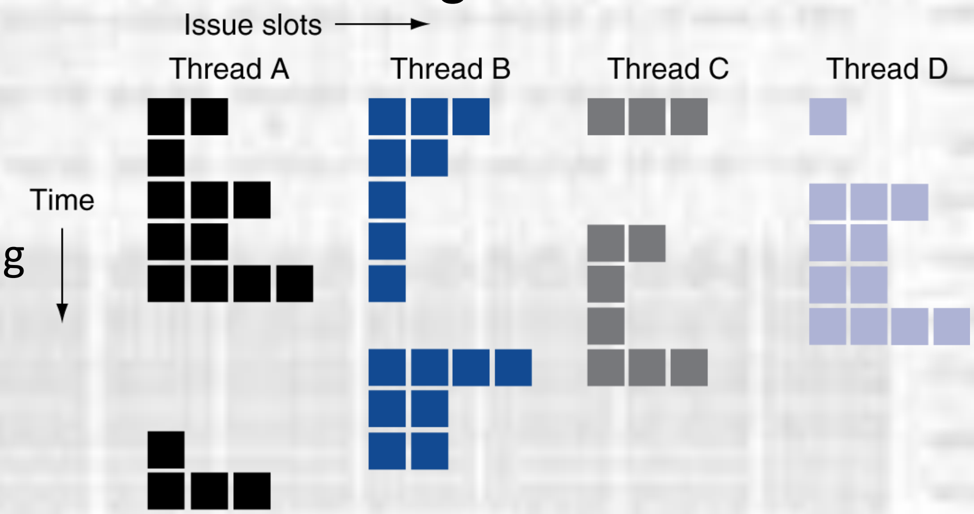
# Parallel Processing
## Hardware Multithreading

- Simultaneous Multithreading (SMT)
  - Superscalar version of multithreading
  - Assuming a multiple issue, dynamically scheduled pipeline
    - The processor has more execution units than some of the threads can effectively use
    - Register renaming and dynamic scheduling can handle conflicts between threads
  - Does not "switch" between threads, but is always running multiple threads and letting the HW figure it out

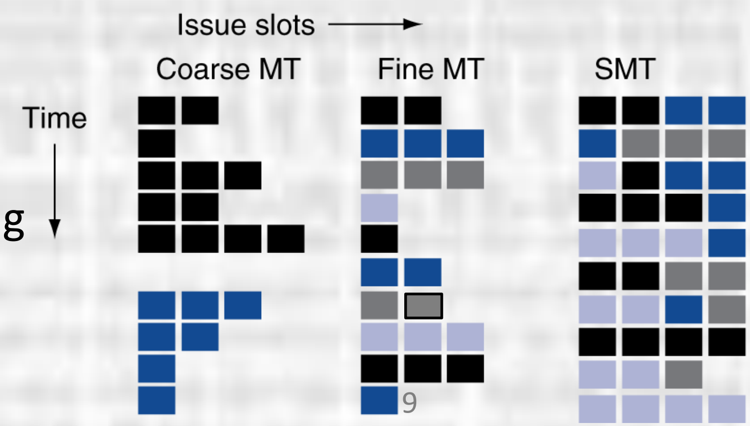
# Parallel Processing
## Hardware Multithreading

- Simultaneous Multithreading

Superscalar (4 slot)
w/o HW Multithreading




Superscalar (4 slot)
with HW Multithreading




SMT
Leverages Instruction level parallelism (multiple slots) and thread level parallelism

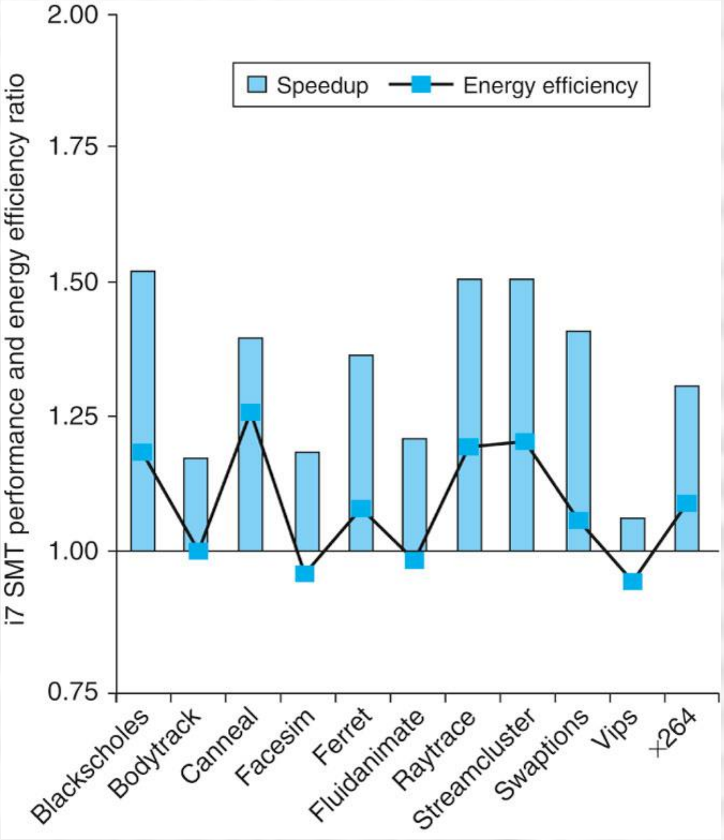
# Parallel Processing
## Hardware Multithreading

- Simultaneous Multithreading

Core I7 single processor with HW support for 2 threads

Ave. speedup = 1.31

Ave. improvement in energy efficiency = 1.07



# Parallel Processing
## Hardware Multithreading

- Simultaneous Multithreading
  - Works well with a single processor
  - What can we do in a multicore environment?
    - How do we share threads?

# Parallel Processing

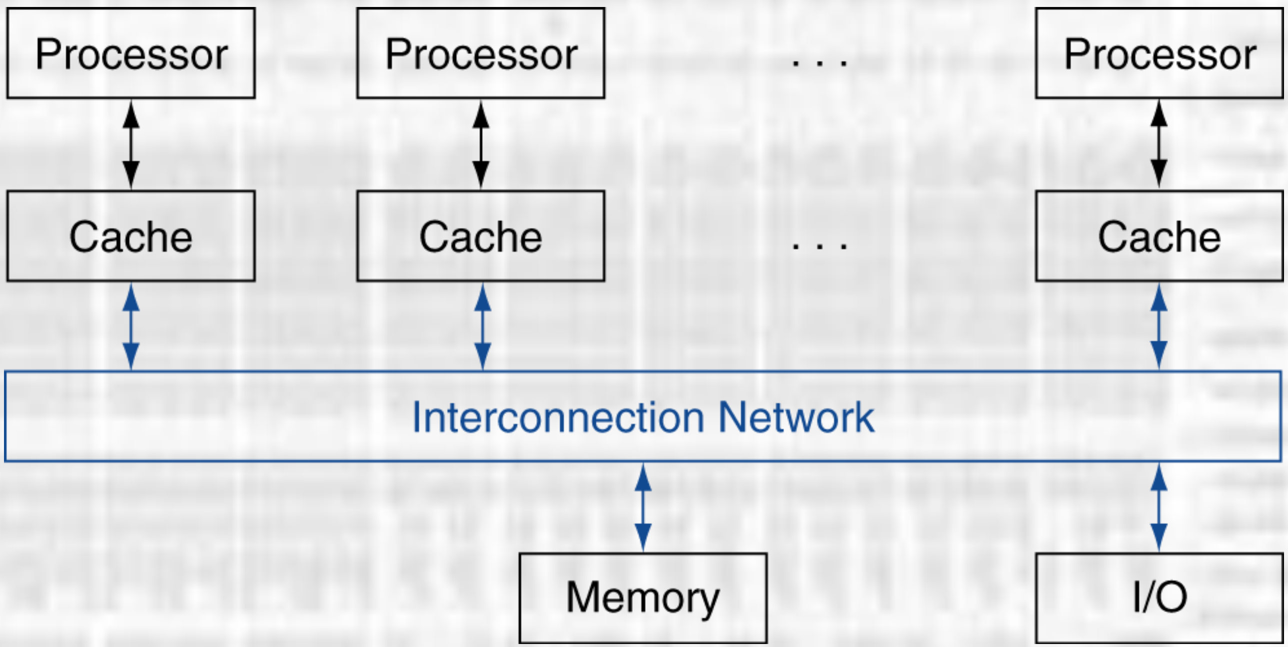
## Memory Sharing

- Two approaches to memory sharing
  - Give each processor its own physical address space
    - Can share data explicitly
    - Requires processors to pass data back and forth
    - Message Passing Multiprocessor
  - Have processors share a common physical address space
    - Data can be shared by reference (memory location)
    - Requires a cache coherency mechanism
    - Shared Memory Multiprocessor (SMP)
- Still use virtual memory to keep processes separate

# Parallel Processing

## Memory Sharing

- Shared Memory Multiprocessor



# Parallel Processing
## Memory Sharing

- Shared Memory Multiprocessor
  - Uniform Memory Access (UMA)
    - All processors have equal delays to access memory
  - Non-uniform Memory Access (NUMA)
    - Not all processors have equal delays to access memory
    - Locality of physical memory
    - Modularity (processor clusters and memory controllers)

# Parallel Processing
## Memory Sharing

- Shared Memory Multiprocessor
  - Require synchronization
    - Prevent processors from using invalid data
      - Currently being worked on
    - "lock" the data while in use
      - A processor will lock the data while using it and unlock it when done
      - No processor can access data that has been locked by another processor

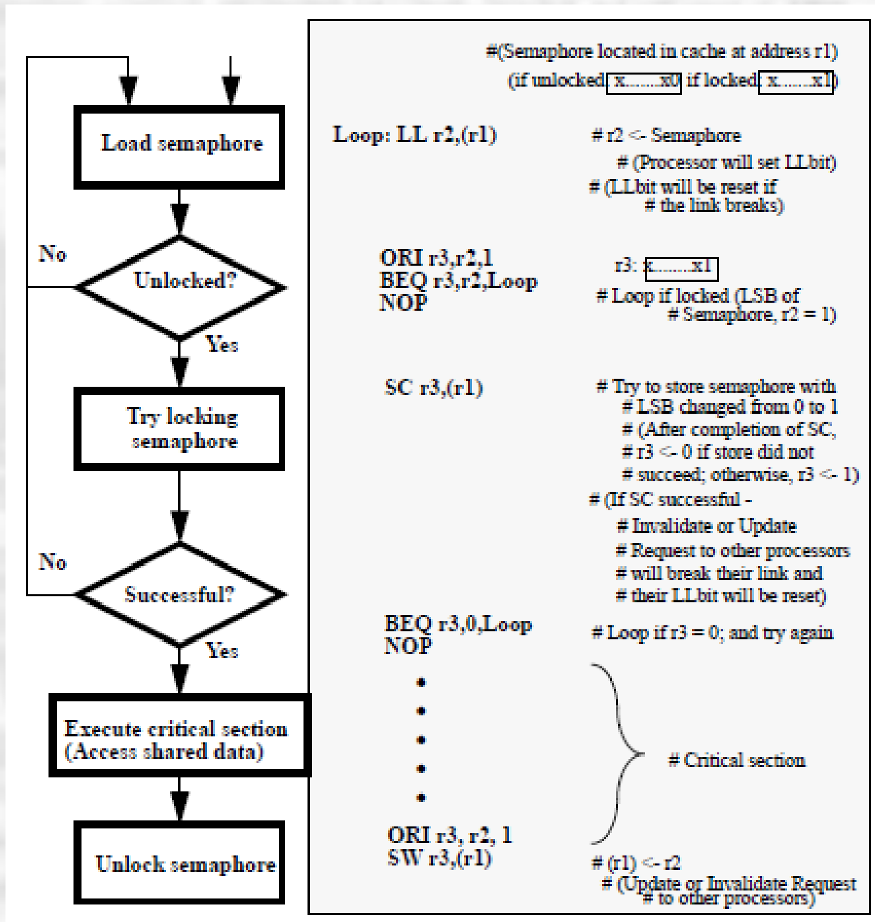
# Parallel Processing

## Memory Sharing

- Shared Memory Multiprocessor
  - Simple Lock
    - Special location in memory attached to the shared data
    - If set – some processor has the data and no one else can access it
    - If clear – free to use
  - Special instructions required to read/write to shared data
    - LL – Load linked, sets a special system bit (LLbit)
    - SC – Store Conditional, only does the store if the LLbit is set
      - Required since other threads may execute between the LL and SC

# Parallel Processing
## Memory Sharing

- Shared Memory Multiprocessor
  - Simple Lock



# Parallel Processing

## Programming

- Multiprocessor Programming
  - Sum 100,000 numbers on 100 processor UMA
    - Each processor has ID: $0 \le Pn \le 99$
    - Partition 1000 numbers per processor
    - Initial summation on each processor

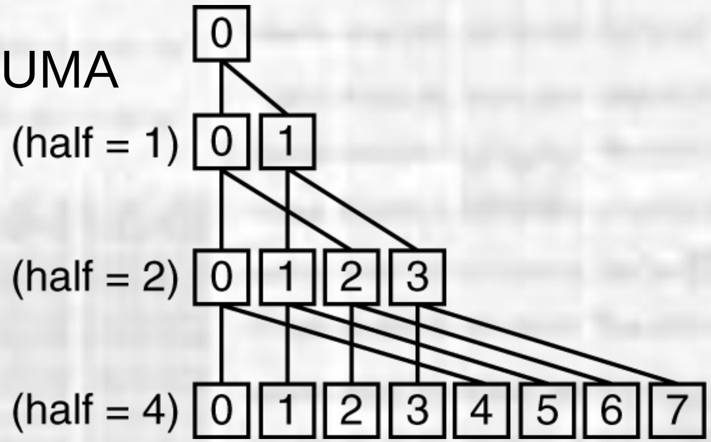
```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
sum[Pn] = sum[Pn] + A[i];
```

    - Now need to add these partial sums
    - Reduction: divide and conquer
    - Half the processors add pairs, then quarter, …
    - Need to synchronize between reduction steps

# Parallel Processing

## Programming

- Multiprocessor Programming
  - Sum 100,000 numbers on 100 processor UMA




```
half = 100;
do
    synch();
    if (half%2 != 0 && Pn == 0)
        sum[0] = sum[0] + sum[half-1];
            /* Conditional sum needed when half is odd;
                Processor0 gets missing element */
        half = half/2; /* dividing line on who sums */
        if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
while (half > 1);
```

# Parallel Processing
## Programming

- Parallel Programming Interface

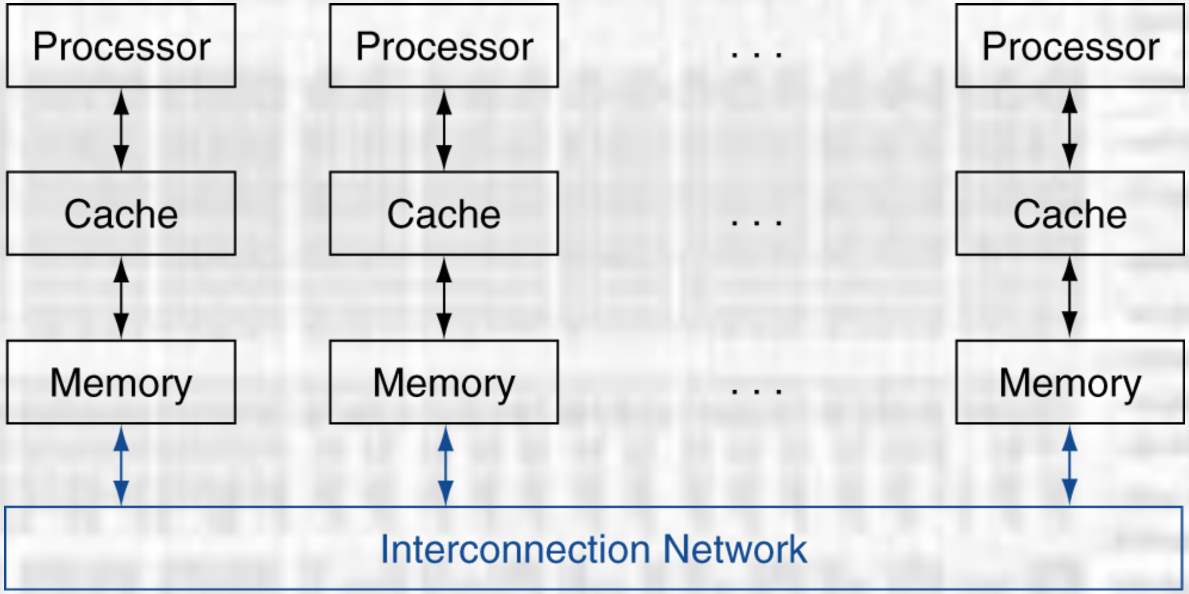
  - OpenMP – API for parallel programming
    - Pragma – compiler directives

```
# define P 100              /* define 100 processors */
#pragma  omp  parallel  num_threads(P)      /* use 100 threads */

#pragma  omp  parallel for
for (Pn = 0; Pn < P; Pn += 1)
    for(i = 1000*Pn; i < 1000*(Pn + 1); i += 1)
        sum[Pn] += A[i];

#pragma  omp  parallel  for  reduction(+ : FinalSum)
for (i=0; i<P; i += 1)
    FinalSum += sum[i];        /* compiler figures out how to minimize effort */
```

# Parallel Processing

## Memory Sharing

- Message Passing Multiprocessor



# Parallel Processing

## Memory Sharing

- Message Passing Multiprocessor
  - Processors send and receive messages back and forth
    - send message routine
    - receive message routine
    - ack
  - Interconnection network
    - Local area network (LAN)
    - Custom networks
  - Most effective when applications have little need to share memory

# Parallel Processing

## Memory Sharing

- Message Passing Multiprocessor
  - Clusters
    - Collection of computers configured for message passing
      - Communicate via I/O
      - Connected through standard network hardware (switches)
      - Each with its own copy of the code
    - High dependability
      - Easy to replace
      - Easy to expand
      - One failure does not impact other nodes

# Parallel Processing

## Memory Sharing

- Message Passing Multiprocessor
  - Clusters
    - Familiar examples
      - Google search, Amazon, Facebook, Twitter
      - Multiple data centers with 10s of 1000s of servers

# Parallel Processing
## Warehouse Scale Computing

- Warehouse Scale Computers (WSC)
  - Clusters taken to the extreme
  - Provide support for many users and applications
    - Software as a Service (SaaS)
  - Additional infrastructure requirements
    - Power
    - Cooling
    - I/O bandwidth

# Parallel Processing
## Warehouse Scale Computing

- Warehouse Scale Computers (WSC)
  - WSC vs. Servers
  - WSC relies on many users and many applications
    - Very little coordination needed
    - Very few messages between users / applications
  - Scale leads to operational cost concerns
    - 30% of cost may be for infrastructure
  - Economy of Scale
    - Thousands of identical servers leads to volume discounts
  - Result: Cloud Computing

# Parallel Processing

## Grid Computing

- Grid Computing
  - Separate computers interconnected by long-haul networks
    - E.g., Internet connections
    - Work units farmed out, results sent back
  - Can make use of idle time on PCs
    - E.g., SETI@home, World Community Grid

# Parallel Processing

## Network Topologies

- Networking Topologies

  - Multi core processors require on-chip networks
  - Clusters require networks

  - Network cost factors
    - # of switches
    - # of links / switch
    - Width of a link (# of bits)
    - Length of link

  - Network performance factors
    - Unloaded / Loaded network latency (send/receive messages)
    - Throughput (# messages possible)
    - Network contention

# Parallel Processing

## Network Topologies

- Networking Topology Metrics
  - Total Network Bandwidth

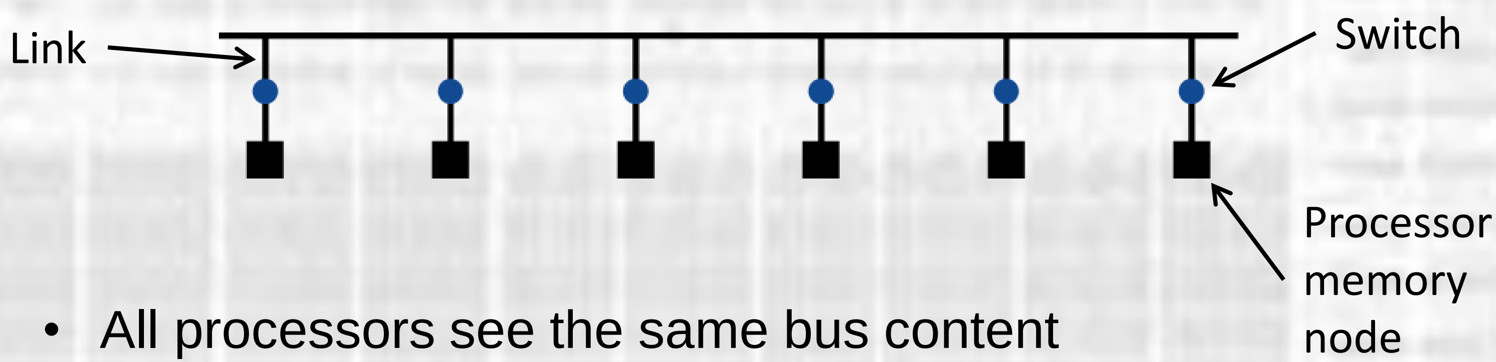
    *Bandwidth / link   x   total # of links*

    - Measure of ideal peak performance
  - Bisection Bandwidth
    - Cut network in half and measure the bandwidth between the two halves
    - Close to the worst case performance
    - For asymmetric topologies – choose the worst case bisection

# Parallel Processing

## Network Topologies

- Bus



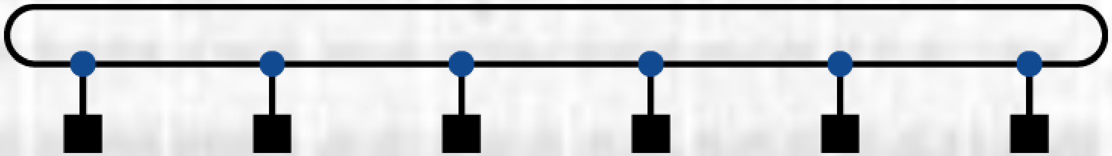

- All processors see the same bus content

# Parallel Processing

## Network Topologies

- Ring

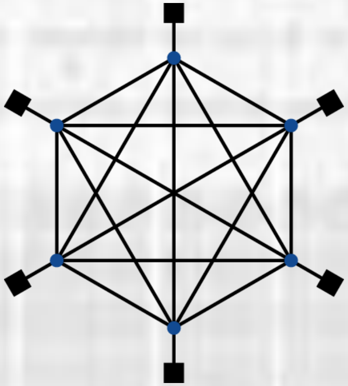


- Some messages may need to “hop” along intermediate nodes
- Multiple transfers can be active at any given time

# Parallel Processing

## Network Topologies

- Fully Connected Network



Total Network Bandwidth = P(P-1)/2

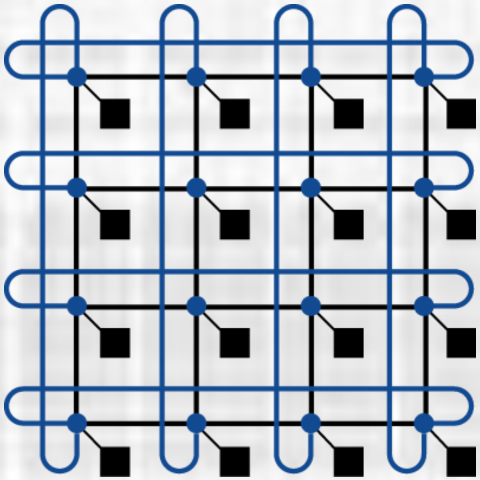
Bisection Bandwidth = $(P/2)^2$

  - All processors are connected together
  - Expensive

# Parallel Processing

## Network Topologies

- 2D Mesh

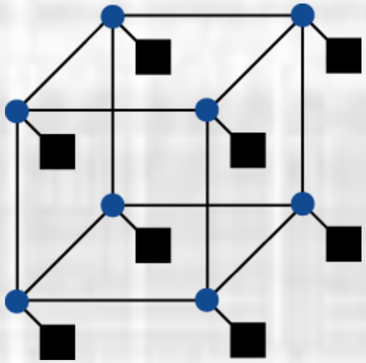


  - Processors connected in a 2-D array

# Parallel Processing
## Network Topologies

- N-Cube



  - N dimensions with processors connected in each dimension

# Parallel Processing

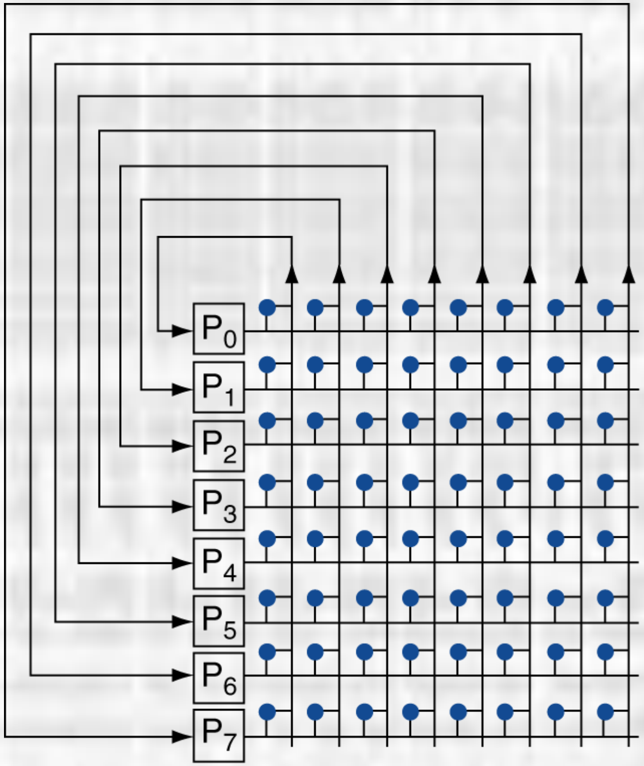
## Network Topologies

- Multistage Networks
  - Not all nodes have processors
  - Use switches to pass along the network

# Parallel Processing

## Network Topologies

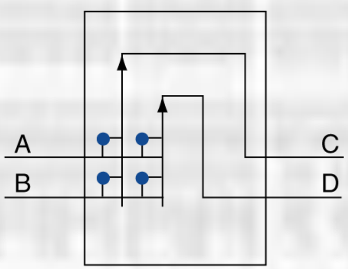
- Multistage Networks
  - Crossbar
    - Unidirectional links



# Parallel Processing

## Network Topologies

- Multistage Networks
  - Omega
    - Unidirectional links




c. Omega network switch box