

# CS3210 Cheat Sheet

## Chapter 2

### Primitives

- Locks
- Semaphores - Binary/Mutex and Counting (Wait, Signal)
- Condition Variables (Wait, Signal, Broadcast)
- Monitors (Mutex + CV)
- Barrier
- Starvation, Deadlock

### Producer-consumer with finite buffer

Producer	Consumer
■ event = waitForEvent ()	■ items.wait ()
■ spaces.wait ()	■ mutex.wait ()
■ mutex.wait ()	□ event = buffer.get ()
□ buffer.add ( event )	■ mutex.signal ()
■ mutex.signal ()	■ spaces.signal ()
■ items.signal ()	■ event.process ()

### Lightswitch Definition

```
class Lightswitch :
■ def __init__ ( self ):
□ self.counter = 0
□ self.mutex = Semaphore (1)
■ def lock (self , semaphore ):
□ self.mutex.wait ()
■ self.counter += 1
■ if self.counter == 1:
□ semaphore.wait ()
■ self.mutex.signal ()
■ def unlock (self , semaphore ):
□ self.mutex.wait ()
■ self.counter -= 1
■ if self.counter == 0:
□ semaphore.signal ()
□ self.mutex.signal ()
```

### Readers-Writers Turnstile

Writers	Readers
■ turnstile.wait ()	■ turnstile.wait ()
□ roomEmpty.wait ()	■ turnstile.signal ()
■ # critical section for writers	■ readSwitch.lock ( roomEmpty )
■ turnstile.signal ()	□ # critical section for readers
■ roomEmpty.signal ()	■ readSwitch.unlock ( roomEmpty )

### Readers-Writers with priorities

Writers	Readers
■ writeSwitch.lock (noReaders)	■ noReaders.wait ()
□ noWriters.wait ()	□ readSwitch.lock (noWriters)
□ # critical section for writers	■ noReaders.signal ()
□ noWriters.signal ()	■ # critical section for readers
■ writeSwitch.unlock (noReaders)	■ readSwitch.unlock ( noWriters )

## Chapter 3

### Levels of Parallelism

- Single Processor:
  - Bit Level
  - Instruction Level
  - Thread Level
  - Process Level
- Multiple Processors:
  - Prcessor Level

### Bit Level

Word size (16-bit, 32-bit, 64-bit)

### Instruction Level

1. Pipelining - split instruction execution into multiple stages, then allow multiple instructions to occupy different stages in the same clock cycle; number of pipeline stages == maximum achievable speedup
2. Superscalar - duplicate pipelines, allow multiple instructions to pass through the same stage

### Thread Level Parallelism

- Simultaneous Multi-threading - Hyper-threading; Run multiple (2) threads at the same time

### Process Level Parallelism

- Instead of multiple threads, can use multiple processes to work in parallel.
- Each process needs an independent set of processor context → can be mapped to multiple processor cores

### Flynn's Taxonomy

- Single Instruction Single Data (SISD)
- Single Instruction Multiple Data (SIMD) - SSE, AVX instructions
- Multiple Instruction Single Data (MISD) - Space Shuttle
- Multiple Instruction Multiple Data (MIMD) - multiprocessor

### Memory Organization

- Distributed-Memory Multicomputers:
  - \* Memory in a node is private, use message-passing to exchange data
- Shared-memory Multiprocessors
  - \* Data-exchanges between nodes through shared variables
  - Uniform Memory Access (UMA)
    - \* Latency of accessing main memory is same for all processors
    - \* Main memory is congregated at some other area separate from processors
  - Non-Uniform Memory Access (NUMA)
    - \* Also known as distributed SHARED-MEMORY
    - \* Physically distributed memory of all processing elements combined to form a global shared-memory address space
    - \* Access local memory is faster than remote memory for a processor → non-uniform access time
    - \* Related: Cache Coherent NUMA (ccNUMA) - Each node has cache memory to reduce contention
  - Cache-only Memory Access (COMA)
    - \* Quite similar to NUMA, replace memory with a cache
    - \* Data migrates dynamically and continuously according to cache coherence scheme
- Hybrid (Distributed-Shared Memory)

### Shared Memory Systems

- Advantages
  - No need to partition code or data
  - No need to physically move data among processors → communication is efficient
- Disadvantages
  - Special synchronization constructions required
  - Lack of scalability due to contention

### Multicore Architecture

- Hierarchical design
- Pipelined design
- Network-based design - Interconnection networks

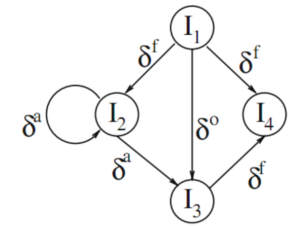
## Chapter 4

What limits parallelism? Dependencies, Overheads in parallelism (context switching), Synchronization

### Instruction Parallelism

- Flow dependency - Read after Write, aka True dependency
- Anti-dependency - Write after Read
- Output dependency - Write after Write

I<sub>1</sub>: R<sub>1</sub> ← A  
I<sub>2</sub>: R<sub>2</sub> ← R<sub>2</sub>+R<sub>1</sub>  
I<sub>3</sub>: R<sub>1</sub> ← R<sub>3</sub>  
I<sub>4</sub>: B ← R<sub>1</sub>



**Instructions:** I<sub>1</sub>, I<sub>2</sub>, I<sub>3</sub>, I<sub>4</sub>  
**Registers:** R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
**Memory addresses:** A, B

$\delta^f$ : (**RAW**) flow dependency  
 $\delta^a$ : (**WAR**) anti-dependency  
 $\delta^o$ : (**WAW**) output dependency

### Loop Parallelism

#### Data Parallelism

- Partition the data used in solving the problem among the processing units; each processing unit carries out similar operations on its part of the data.
- SIMD computers / instructions exploit data parallelism
- (Data Parallelism on MIMD) SPMD (Single Program Multiple Data) - one parallel program executed by all processors in parallel (both shared and distributed address space); example is MPI

#### Task Parallelism

- Partition the tasks in solving the problem among the processing units
- Example: Different components of an SQL statement

#### Task Dependence Graph

- Critical Path Length: Minimum (slowest) completion time
- Degree of concurrency = Total Work / Critical Path Length

### Parallel Programming Patterns

- Fork-Join
- Parbegin-Parend - OpenMP
- SIMD - SSE instruction treats xmm registers (128 bit) as 4 32-bit floating point values
- SPMD - MPI
- Master-slave
- Client-Server (MPMD model)
- Pipelining
- Task (Work) Pools
- Producer-Consumer

## Chapter 5

### CPU Time (No memory miss)

$$\text{Time}_{\text{user}}(A) = N_{\text{cycle}}(A) \times \text{Time}_{\text{cycle}} \quad (1)$$

$$N_{\text{cycle}}(A) = \sum_{i=1}^n n_i(A) \times \text{CPI}_i \quad (2)$$

$$\text{Time}_{\text{user}}(A) = N_{\text{instructions}}(A) \times \text{CPI}(A) \times \text{Time}_{\text{cycle}} \quad (3)$$

### CPU Time (With memory miss)

#### Memory Access Time

$$\text{Time}_{\text{user}}(A) = (N_{\text{cycle}}(A) + N_{\text{mm\_cycle}}(A)) \times \text{Time}_{\text{cycle}} \quad (4)$$

#### Consider a one-level cache:

$$N_{\text{mm\_cycle}}(A) = N_{\text{read\_cycle}}(A) + N_{\text{write\_cycle}}(A) \quad (5)$$

$$N_{\text{read\_cycle}}(A) = N_{\text{read\_op}}(A) \times R_{\text{read\_miss}}(A) \times N_{\text{miss\_cycles}}(A) \quad (6)$$

$$N_{\text{write\_cycle}}(A) = N_{\text{write\_op}}(A) \times R_{\text{write\_miss}}(A) \times N_{\text{miss\_cycles}}(A) \quad (7)$$

#### Refinement with Memory Access Time

$$\text{Time}_{\text{user}}(A) = (N_{\text{instructions}}(A) \times \text{CPI}(A) + N_{\text{rw\_op}}(A) \times R_{\text{rw\_miss}}(A) \times N_{\text{rw\_cycles}}(A)) \times \text{Time}_{\text{cycle}} \quad (8)$$

Average Memory Access Time

Average read access time = Time for read hit + Time for read miss

T\_read\_access(A) = T\_read\_hit(A) + R\_read\_miss(A) x T\_read\_miss(A)

Two-level Cache example:

T\_read\_access(A) = T\_read\_hit^L1(A) + R\_read\_miss^L1(A) x T\_read\_miss^L1(A)

T\_read\_miss^L1(A) = T\_read\_hit^L2(A) + R\_read\_miss^L2(A) x T\_read\_miss^L2(A)

Global Miss Rate:

R\_read\_miss^L1(A) x R\_read\_miss^L2(A)

MIPS, MFLOPS

MIPS(A) = (N\_instr(A) / Time\_user(A) x 10^6) = (clock\_frequency / CPI(A) x 10^6)

MFLOPS(A) = (N\_fl\_ops(A) / Time\_user(A) x 10^6)

Parallel Execution Time

T\_p(n) - time for p processors to work on problem of size n
C\_p(n) = p x T\_p(n)

- C\_p(n) - cost of a parallel program with input size n executed on p processors
- Parallel program is cost optimal if it executes the same total number of operations as the fastest sequential program

S\_p(n) = (T\_best\_seq(n) / T\_p(n))

- S\_p(n) is the speedup of the parallel program on p processors
- Theoretically S\_p(n) <= p always holds
- In practice S\_p(n) > p can occur due to better cache locality, early termination

E\_p(n) = (T\_\*(n) / C\_p(n)) = (S\_p(n) / p) = (T\_\*(n) / (p x T\_p(n)))

- Use T\_\*(n) as a shorthand for T\_best\_seq(n)
- Efficiency measures the actual degree of speedup performance achieved compared to the maximum
- In an ideal speedup S\_p(n) = p -> E\_p(n) = 1

Parallel Laws

Amdahl's Law

- Speedup of parallel execution is limited by the fraction of the algorithm that cannot be parallelized, f
- f(0 <= f <= 1) - the sequential fraction
- "Fixed-workload" performance

S\_p(n) = (T\_\*(n) / (f x T\_\*(n) + ((1-f)/p) x T\_\*(n))) = (1 / (f + ((1-f)/p))) <= 1/f

S\_p(n) = (p / (1 + (p-1)f))

Gustafson's Law

- In many computing problems, f is not a constant
- Depends on problem size n: f is a function of n, f(n)
- An effective parallel algorithm is:

lim (n -> infinity) f(n) = 0

- Thus speedup:

lim (n -> infinity) S\_p(n) = (p / (1 + (p-1)f(n))) = p

- In such cases, we can have

S\_p(n) <= p

S\_p(n) = ((tau\_f + tau\_v(n, 1)) / (tau\_f + tau\_v(n, p)))

Assume parallel program is perfectly parallelizable (without overheads):

tau\_v(n, 1) = T^\*(n) - tau\_f and tau\_v(n, p) = ((T^\*(n) - tau\_f) / p)

S\_p(n) = ((tau\_f + T^\*(n) - tau\_f) / (tau\_f + ((T^\*(n) - tau\_f) / p))) = ((tau\_f / (T^\*(n) - tau\_f) + 1) / ((tau\_f / (T^\*(n) - tau\_f) + 1/p)))

If T \* (n) increase strongly monotonically with n, then

lim (n -> infinity) S\_p(n) = p

Chapter 6

Memory Consistency Models

Relaxed Consistency

- Only if instructions operate on different memory locations
- Write-to-Read Program Order
  - Total Store Ordering (TSO)
  - Processor Consistency (PC)
- Write-to-Write Program Order
  - Partial Store Ordering (PSO)

TSO

- Can reorder W -> R
- All processors see updates in the same order

PC

- Can reorder W -> R
- Different processors can see updates in different orders
- Note: Ordering should still be consistent for updates coming from the same processor
- P1 executes X -> Y, if P2 saw Y, then P2 must have seen X
- But if P1 executes X and P2 executes Y, if P3 sees X first, it is possible for P4 to see Y first instead

PSO

- Can reorder W -> R
- Can reorder W -> W
- Similar to TSO, processors see updates in same order

Interconnection Networks

Direct Interconnect

- Diameter - maximum distance between any pair of nodes. Small diameter ensures small distances for message transmission.
- Node Degree - number of direct neighbours of node. Small node degree reduces the node hardware overhead.
- Graph Degree - maximum degree of a node in network G.
- Bisection width - minimum number of edges that must be removed to divide network into two equal halves. (Bottlenecks) capacity of network to transmit messages simultaneously.
- Bisection bandwidth - total bandwidth available between the two bisected portion of the network.
- Node connectivity - minimum number of nodes that must fail to disconnect the network. Determines the robustness of the network.
- Edge connectivity - minimum number of edges that must fail to disconnect the network. Determine number of independent paths between any pair of nodes.

network G with n nodes	degree	diameter	edge-connectivity ec(G)	bisection bandwidth B(G)
complete graph	n - 1	1	n - 1	(n/2)^2
linear array	2	n - 1	1	1
ring	2	floor(n/2)	2	2
d-dimensional mesh (n = r^d)	2d	d(floor(n^(1/d)) - 1)	d	n^(d-1/d)
d-dimensional torus (n = r^d)	2d	d(floor(n^(1/d))	2d	2n^(d-1/d)
k-dimensional hyper-cube (n = 2^k)	log n	log n	log n	n/2
k-dimensional CCC-network (n = k2^k for k >= 3)	3	2k - 1 + floor(k/2)	3	n/(2k)
complete binary tree (n = 2^k - 1)	3	2 log((n+1)/2)	1	1
k-ary d-cube (n = k^d)	2d	d(floor(k/2)	2d	2k^(d-1)

Indirect Interconnect

- Bus Network
- Crossbar Network - n x m switches
- Multistage Switching Network
  - Omega network
    - n x n Omega network has log n stages
    - n/2 switches per stage
    - Switch position: (alpha, i)
    - alpha: position of switch within a stage
    - i: stage number
    - Edge between (alpha, i) and (beta, i + 1) where
    - beta = alpha by a cyclic left bit shift
    - beta = alpha by a cyclic left bit shift + inversion of LSBit
  - Butterfly network
    - Should be same number of switches and stages as Omega
    - Node (alpha, i) connects to:
      - (alpha, i + 1), straight edge
      - (alpha', i), alpha and alpha' differ in the (i + 1)th bit from the left, i.e. cross edge
  - Baseline network

Routing

Classification

- Based on path length
  - Minimal or Non-minimal routing: whether shortest path is always chosen
- Based on adaptivity
  - Deterministic: Always same path for same pair of (source, destination) node
  - Adaptive: May take into account network status and adapt accordingly, e.g. avoid congested path, avoid dead nodes, etc.

XY Routing for 2D Mesh

- (X\_src, Y\_src) -> (X\_dst, Y\_dst)
- Move in X direction until X\_src == X\_dst
- Move in Y direction until Y\_src == Y\_dst

E-Cube Routing for Hypercube

- (alpha\_{n-1}, alpha\_{n-2}, ..., alpha\_1, alpha\_0) -> (beta\_{n-1}, beta\_{n-2}, ..., beta\_1, beta\_0)
- Start from MSB to LSB (or LSB to MSB)
- Find first different bit
- Go to the neighboring node with the bit corrected
- At most n hops

XOR-Tag Routing for Omega Network

- Let T = Source Id xor Destination Id
  - At stage-k:
  - Go straight if bit k of T is 0
  - Crossover if bit k of T is 1
- Copyright © 2018 Edmund Mok