CS3210 Cheat Sheet

Chapter 2

Primatives

- 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

```
event = waitForEvent ()
spaces.wait ()
mutex.wait ()
    buffer.add ( event )
mutex.signal ()
items.signal ()
```

Consumer

```
items.wait ()
mutex.wait ()
    event = buffer.get ()
mutex.signal ()
spaces.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

```
turnstile.wait ()
    roomEmpty.wait ()
        # critical section for writers
turnstile.signal ()
roomEmpty.signal ()
```

Readers

```
turnstile.wait ()
turnstile.signal ()
readSwitch.lock ( roomEmpty )
    # critical section for readers
readSwitch.unlock ( roomEmpty )
```

Readers-Writers with priorities

Writers

```
writeSwitch.lock (noReaders)
    noWriters.wait ()
    # critical section for writers
    noWriters.signal()
writeSwitch.unlock (noReaders)
```

Readers

```
noReaders.wait ()
    readSwitch.lock (noWriters)
noReaders.signal ()
# critical section for readers
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 fater 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

- Hierachical design
- Pipelined design
- Network-based design - Interconnection networks

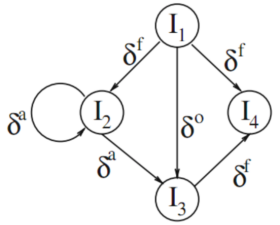
Chapter 4

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_1: $R_1 \leftarrow A$
I_2: $R_2 \leftarrow R_2 + R_1$
I_3: $R_1 \leftarrow R_3$
I_4: $B \leftarrow R_1$



Instructions: I_1, I_2, I_3, I_4
Registers: R_1, R_2, R_3
Memory addresses: A, B

δ^f: (**RAW**) flow dependency
δ^a: (**WAR**) anti-dependency
δ^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 operaitons 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

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

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

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

Average Memory Access Time

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

$$T_{\text{read\_access}}(A) = T_{\text{read\_hit}}(A) + R_{\text{read\_miss}}(A) \times T_{\text{read\_miss}}(A) \quad (9)$$

Two-level Cache example:

$$T_{\text{read\_access}}(A) = T^{L1}_{\text{read\_hit}}(A) + R^{L1}_{\text{read\_miss}}(A) \times T^{L1}_{\text{read\_miss}}(A) \quad (10)$$

$$T^{L1}_{\text{read\_miss}}(A) = T^{L2}_{\text{read\_hit}}(A) + R^{L2}_{\text{read\_miss}}(A) \times T^{L2}_{\text{read\_miss}}(A) \quad (11)$$

Global Miss Rate:

$$R^{L1}_{\text{read\_miss}}(A) \times R^{L2}_{\text{read\_miss}}(A) \quad (12)$$

MIPS, MFLOPS

$$MIPS(A) = \frac{N_{\text{instr}}(A)}{\text{Time}_{\text{user}}(A) \times 10^6} = \frac{\text{clock\_frequency}}{CPI(A) \times 10^6} \quad (13)$$

$$MFLOPS(A) = \frac{N_{\text{fl\_ops}}(A)}{\text{Time}_{\text{user}}(A) \times 10^6} \quad (14)$$

Parallel Execution Time

- $T_p(n)$ - time for p processors to work on problem of size n

$$C_p(n) = p \times T_p(n) \quad (15)$$

- $C_p(n)$ - 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

$$S_p(n) = \frac{T_{best\_seq}(n)}{T_p(n)} \quad (16)$$

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

$$E_p(n) = \frac{T_*(n)}{C_p(n)} = \frac{S_p(n)}{p} = \frac{T_*(n)}{p \times T_p(n)} \quad (17)$$

- Use $T_*(n)$ as a shorthand for $T_{best\_seq}(n)$
- Efficiency measures the actual degree of speedup performance achived compared to the maximum
- In an ideal speedup $S_p(n) = p \rightarrow E_p(n) = 1$

Parallel Laws

Amdahl's Law

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

$$S_p(n) = \frac{T_*(n)}{f \times T_*(n) + \frac{1-f}{p} T_*(n)} = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \quad (18)$$

$$S_p(n) = \frac{p}{1 + (p-1)f} \quad (19)$$

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:

$$\lim_{n \to \infty} f(n) = 0 \quad (20)$$

- Thus speedup:

$$\lim_{n \to \infty} S_p(n) = \frac{p}{1 + (p-1)f(n)} = p \quad (21)$$

- In such cases, we can have

$$S_p(n) \leq p \quad (22)$$

$$S_p(n) = \frac{\tau_f + \tau_v(n,1)}{\tau_f + \tau_v(n,p)} \quad (23)$$

Assume parallel program is perfectly parallelizable (without overheads):

$$\tau_v(n,1) = T^*(n) - \tau_f \text{ and } \tau_v(n,p) = \frac{T^*(n) - \tau_f}{p} \quad (24)$$

$$S_p(n) = \frac{\tau_f + T^*(n) - \tau_f}{\tau_f + \frac{T^*(n) - \tau_f}{p}} = \frac{\frac{\tau_f}{T*(n) - \tau_f} + 1}{\frac{\tau_f}{T^*(n) - \tau_f} + \frac{1}{p}} \quad (25)$$

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

$$\lim_{n \to \infty} S_p(n) = p \quad (26)$$

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 Consitency (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 bandiwth 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 $g(G)$	diameter $\delta(G)$	edge-connectivity ec(G)	bisection bandwidth $B(G)$
complete graph	$n-1$	1	$n-1$	$\left(\frac{n}{2}\right)^2$
linear array	2	$n-1$	1	1
ring	2	$\lfloor \frac{n}{2} \rfloor$	2	2
d-dimensional mesh ($n = r^d$)	$2d$	$d(\sqrt[d]{n} - 1)$	d	$n^{\frac{d-1}{d}}$
d-dimensional torus ($n = r^d$)	$2d$	$d \lfloor \frac{\sqrt[d]{n}}{2} \rfloor$	$2d$	$2n^{\frac{d-1}{d}}$
k-dimensional hyper-cube ($n = 2^k$)	$\log n$	$\log n$	$\log n$	$\frac{n}{2}$
k-dimensional CCC-network ($n = k2^k$ for $k \geq 3$)	3	$2k - 1 + \lfloor k/2 \rfloor$	3	$\frac{n}{2k}$
complete binary tree ($n = 2^k - 1$)	3	$2 \log \frac{n+1}{2}$	1	1
k-ary d-cube ($n = k^d$)	$2d$	$d \lfloor \frac{k}{2} \rfloor$	$2d$	$2k^{d-1}$

Indirect Interconnect

- Bus Network
- Crossbar Network - $n \times m$ switches
- Multistage Switching Network
  - Omega network
    - $n \times n$ Omega network has $\log n$ stages
    - $\frac{n}{2}$ switches per stage
    - Switch position: (α, i)
    - α: position of switch within a stage
    - i: stage number
    - Edge between (α, 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 (α, i) connects to:
    - $(\alpha, i+1)$, straight edge
    - (α', i), α and α' 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

- $(X_{src}, Y_{src}) \rightarrow (X_{dst}, Y_{dst})$
- Move in X direction until $X_{src} == X_{dst}$
- Move in Y direction until $Y_{src} == Y_{dst}$

E-Cube Routing for Hypercube

- $(\alpha_{n-1}, \alpha_{n-2}, \ldots, \alpha_1, \alpha_0) \rightarrow (\beta_{n-1}, \beta_{n-2}, \ldots, \beta_1, \beta_0)$
- 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 ⊕ Destination Id
- At stage-k:
- Go straight if bit k of T is 0
- Crossover if bit k of T is 1