CS3210 Cheat Sheet

Chapter 2

Synchronization

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

Producer-Consumer

Readers-Writers

Lightswitch

Provides sort of an exclusive control over the semaphore while there exists counts of this type.

Turnstile

Chapter 3

Levels of Parallelism

Single Processor: {Bit, Instruction, Thread, Process} Level

Multiple Processors: {Shared, Distributed} Memory

Bit Level

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

Instruction Level

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

Thread Level

Simultaneous Multi-Threading: Hyperthreading, Run multiple (2) threads at the same time

Process Level

Multiple processes can be mapped to multiple processor cores

Flynn's Parallel Architecture Taxonomy

- SISD
- SIMD - SSE, AVX instructions
- MISD - Space shuttle
- MIMD - Multiprocessor

Memory Organization

Distributed-Memory Multicomputers - Memory in node is private, message-passing to exchange data

Hybrid (Distributed-Shared Memory)

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) [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)
  - Only cache present, no memory
  - Data migrates dynamically and continuously according to cache coherence scheme

Multicore Architecture

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

Chapter 4

Limits parallelism: Dependencies, Overheads of parallelism (switching), synchronization

Instruction Parallelism

Flow dependency (RAW), Anti-dependency (WAR), Output dependency (WAW)

Loop Parallelism

If each loop iteration is independent, can execute in parallel (i.e. OpenMP)

Data Parallelism

Partition data, similar operations on each part

SIMD, SPMD (i.e. MPI)

Task Parallelism

Partition tasks, same data on through all tasks (i.e. different components of an SQL statement)

Task Dependence Graph

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

Chapter 5

CPU Time (No memory miss)

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

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

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

CPU Time (With memory miss)

Memory Access Time

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

Consider a one-level cache:

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

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

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

Refinement with Memory Access Time

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

Average Memory Access Time

Average read access time = Time for read hit + Adjusted / Average Time for read miss penalty

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

Two-level Cache example:

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

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

Global Miss Rate:

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

MIPS, MFLOPS

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

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

Parallel Execution Time

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

$$C_p(n) = p \times T_p(n) \tag{15}$$

- $C_p(n)$ - 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

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

- $S_p(n)$ 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

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

- 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

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

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

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:

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

- Thus speedup:

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

- In such cases, we can have

$$S_p(n) \leq p \tag{22}$$

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

Assume parallel program is perfectly parallelizable (without overheads):

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

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

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

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

Chapter 6

Memory Consistency Models

Relaxed Consistency: Only if instructions operate on different memory locations

TSO [W → R]: All processors see updates in the same order

PC [W → R]: Different processors can see updates in different orders; **Note:** Ordering should still be consistent for updates coming from the same processor. If 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 [W → R; 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	$n-1$	$(\frac{n}{2})^2$
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 ($n = k^d$)	$2d$	$d\lfloor \frac{k}{2} \rfloor$	$2d$	$2k^{d-1}$

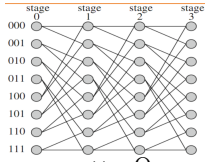
Indirect Interconnect

Bus Network - only 1 pair communicate at a time

Crossbar Network - $n \times m$ switches

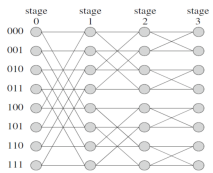
Multistage Switching Network (Om, Bu, Base)

Omega network



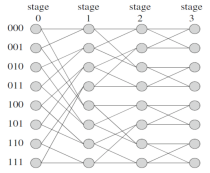
- $n \times n$ Omega network has $\log n$ stages
- $\frac{n}{2}$ switches per stage
- Switch position: (α, i)
- α: position of switch within a stage
- i: stage number
- Edge between (α, i) and (β, $i+1$) where
- $\beta = \alpha$ by a cyclic left bit shift
- $\beta = \alpha$ by a cyclic left bit shift + inversion of LSBit

Butterfly network



- Should be same number of switches and stages as Omega
- Node (α, i) connects to:
- (α, $i+1$), 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

Path length: Minimal or **Non-minimal**

routing: whether shortest path is always chosen

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

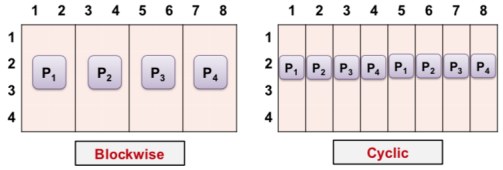
- $(\alpha_{n-1}, \alpha_{n-2}, \ldots, \alpha_1, \alpha_0) \rightarrow (\beta_{n-1}, \beta_{n-2}, \ldots, \beta_1, \beta_0)$
- Start from MSB to LSB (or LSB to MSB)
- Find first different bit
- Go to the neighboring node with the bit corrected
- **At most n hops**

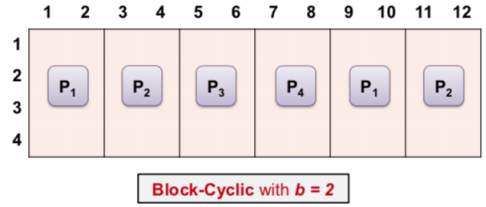
XOR-Tag Routing for Omega Network

- Let T = Source Id $\oplus$ Destination Id
- At stage-k (from left to right):
- Go straight if bit k of T is 0
- Crossover if bit k of T is 1

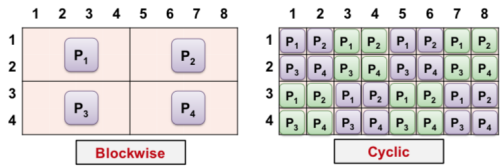
Chapter 7

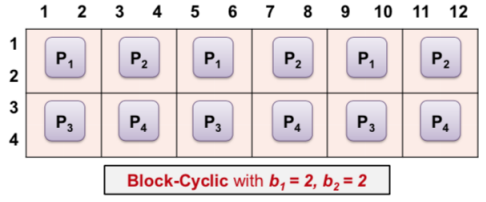
Data Distribution





Checkerboard

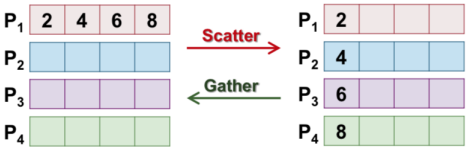




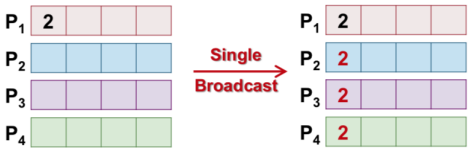
Communication Operations

Single Transfer

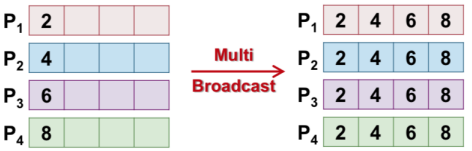
Gather and Scatter



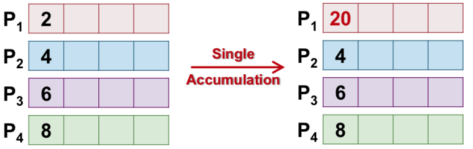
Single Broadcast



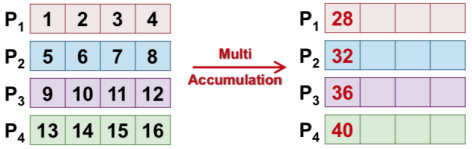
Multi-Broadcast



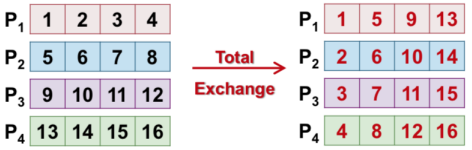
Single-accumulation (gath, reduction)



Multi-accumulation

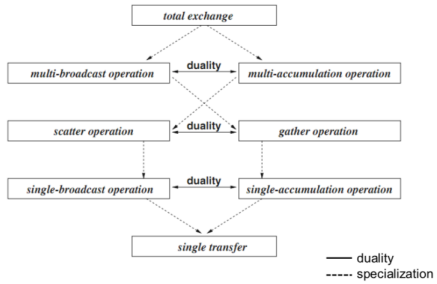


Total Exchange



Duality

Two communication operations is a duality if the same spanning tree can be used for both operations.



Chapter 8

Blocking: Does not return until resource is safe to be reused or modified (Safe and easier programming)

Non-blocking: Returns before it may be safe. (Hide communication overhead)

Buffered: System buffer present. If blocking and sending, blocks until written to system send buffer. (Finite buffer management)

Non-Buffered: System buffer absent. If blocking and sending, blocks until data is sent over (wait for a corresponding receive if no read buffer). (Idling)

MPI Operation Semantics

Local view

Blocking: Return indicates safe to reuse resources

Non-blocking: May return before operation completes, and before safe to reuse

Global view

Synchronous: Communication operation does not complete before both processes have started their communication operation (needs sync / agreement to transfer data, only commences upon both "start")

Asynchronous: Sender can execute communication operation without any coordination from receiver (e.g. data transfer can commence and place inside receiver system buffer before any explicit "receive")

Chapter 9

```
size_t total_num_threads = gridDim.x
* gridDim.y * gridDim.z * blockDim.x
* blockDim.y * blockDim.z;

size_t block_index_in_grid = blockIdx.x
* (gridDim.y * gridDim.z) + blockIdx.y
* (gridDim.z) + blockIdx.z;

size_t thread_index_in_block =
threadIdx.x * (blockDim.y * blockDim.z)
+ threadIdx.y * (blockDim.z) + threadIdx.z;

size_t thread_id = block_index_in_grid
* (blockDim.x * blockDim.y * blockDim.z)
+ thread_index_in_block;
```

Coalesce Access to Global Memory into minimal memory transactions

Chapter 10

Foster's Design Methodology: Partionining, Communication, Agglomeration, Mapping