Synopsis: Concurrent Programming

Bernard Sufrin

Hilary Term, 2017

Part 1: Introduction

Background (1)

Concurrent program organization will soon be the norm (1)

Real-World Concurrency (2)

Example: a restaurant (3)

Concurrency: dealing with lots of things (4)

Interleaving of subtasks (5)

Concurrency without parallelism (6)

Concurrency with parallelism (7)

Terminology (8)

Programming Constructs (11)

Threads vs. Operating-System Processes (13)

The Scheduler (14)

Infrastructure (16)

CSO processes are Scala values (17)

Running a CSO Process (18)

Interleaving of program actions is nondeterministic (19)

Race conditions and atomic actions (21)

Race Conditions (21)

Races for shared variables (22)

Various runs of Race (23)

Abstract Model (24)

Additional Complication #1: Cache incoherence on multiprocessors (27)

Complication #2: compiler optimisations (29)

Synchronization (30)

Programming atomic actions using locks (30)

Synchronization fixes Race2 (31)

Cache memory and synchronized blocks (32)

Deadlock (33)
Summary (34)
Part 2: Low Level Mechanisms - Monitors and Semaphores
Monitors (2)
Monitors in Scala (3)
A producer-consumer problem (4)
A non-Monitor solution: busy waiting (5)
wait and notify (8)
Semantics of wait and notify (9)
Thread States (10)
Synchronization and the JVM memory model (11)
wait and notify (12)
Variants of wait and notify (13)
notify or notifyAll? (14)
Slot revisited: avoiding unecessary notifications (15)
CSO Monitors + Conditions (16)
Correctness vs. Performance Under Load (19)
The Readers and Writers Problem (20)
Readers and Writers: Controller Specification (21)
Readers and Writers: Solution using wait/notify/notifyAll (23)
Should endRead and endWrite use notify or notifyAll? (24)
Problem: Read/Write have identical priority (25)
Problem: Fairness and Overtaking Writes (27)
Fair CSO Monitors (28)
Monitors - Case Study: H_2O (29)
Hydrogen$_2$ + Oxygen = Water (29)
Semaphores (33)
Binary Semaphores (34)
Example: Protecting a resource from a race (36)
Signalling with Binary/Boolean Semaphores (37)
Aside: once-only signalling semaphores (41)
Counting Semaphores (42)
A bounded-buffered multi-producer/multi-consumer slot (43)
The Memory Allocation Problem (45)
Case Study: The Memory Allocation Problem (45)
Monitor-based (47)
Semaphore-based and prioritized (48)
Supplement: Building Monitors from Boolean Semaphores (50)
Building Monitors from Boolean Semaphores (50)
Part 3: Synchronous Data Parallel Programming

Introduction (2)
Applications (3)
Barrier synchronisation (4)
Simulating a simple cellular automaton (5)
Termination using a Combining Barrier (6)
Example: smoothing an image (7)
Particle Computations (9)
Particle computations (9)
Simple physics (10)
Calculating the total force on each particle sequentially (12)
Towards a concurrent algorithm (13)
Avoiding race conditions (14)
Calculating the total forces (15)
Detailed implementation of a Worker process (16)
The pattern of synchronisation (17)
Load balancing (18)
Displaying the state of the simulation (19)
Slowing the drawing-rate down (20)
Barrier Implementation (21)
Barrier Implementation with Semaphores (21)
Supplement: Jacobi Iteration (26)
Supplement: Jacobi Iteration – Specification (26)
Towards a sequential implementation (29)
Using a second array (31)
A two-stage algorithm (32)
Termination (33)
A complete sequential implementation (34)
Towards a concurrent implementation (35)
The concurrent algorithm (39)
Experimental results (39)

Part 4: Message Passing

Introduction (1)
Channels (1)
The CSO Programming Model (2)
Channels in CSO (4)
Example: copy (5)
InPorts and OutPorts (6)
Synchronous Channel implementations in CSO (9)
Misuse of unshared ports (10)
N2N channels (11)

Towards Fine-grained concurrency (12)
Naively-designed components (12)
Example: printing multiples of four (14)
Channel Closing and Termination (15)
Closing Channels (15)
Repetition (17)
Termination of Process Networks (19)
canInput, canOutput and attempting reads and writes (20)
Concurrent reads and writes (21)
Avoiding potential deadlock with concurrent reads and writes (21)
Deadlock-avoiding redesign of tee (23)
Termination of Concurrent Compositions (24)
Case Study: the sameLeaves problem (25)
A sameStreams Component (26)
The Tree Trait (27)
Buffers (29)
Component Generators (30)
Component generators (30)
zipWith (31)
map (32)
Demonstration: Generating naturals (33)
Aside: zipwith revisited for efficiency (35)
Demonstration: an integrator component (36)
Aside: process-generation v. process generation and start (37)
Demonstration: Sorting networks (38)
Case Study: Generating Primes (39)
Discipline for concurrent programming with channels (46)
Sharing variables (46)
Transmitting mutable objects over channels (47)
Cacheing and Channels (48)

Part 5: Example: Numerical Integration by Process-Farming

Numerical integration (1)
Trapezium rule (2)
Sequential code (3)
Towards parallel code (4)
Communication channels (5)
A worker (6)
The controller (7)
Putting the system together (10)
Empirical Performance Measurements (11)

Diminishing the relative overhead of process creation (14)
A worker (15)
Towards a concurrent controller (16)
Concurrent Controller (17)
Empirical Performance Measurements (CSO) on 2 machines (19)
Empirical Performance Measurements (CSO) decreasing tasksize (20)
Empirical Performance Measurements (CSO) (21)
Empirical Performance Measurements (CSO) decreasing tasksize (22)
Observations (23)
Empirical Performance Measurements (Go) (24)
Tentative conclusions (25)

Part 6: Alternation

Alternation (1)
The need for alternation (1)
alt – simple syntax (2)
Example: a simple tagger (3)
Events with Boolean Guards (4)
Case study: almost-fair tagger (5)
Rebuilding the almost-fair tagger (6)
serve vs. repeated alt (7)
Case Study: The Dining Philosophers (9)
The Dining Philosophers – a parable of deadlock avoidance (9)
Dining Philosophers: towards a simulation (12)
A philosopher (14)
A fork (15)
Channels (16)
Putting it together (17)
Simulation results (18)
alt vs. ManyOne / N2N channels (22)
Alternation – semantics (23)
alt semantics (23)
serve (and repeated alt) termination (24)
alt with outport guards (25)
Example: tee revisited (26)
Mixing inport and outport events: a two place buffer (27)
Alternation vs. Monitors/Semaphores (29)
Restrictions on the alternation constructs (30)
Alternation - timeouts, failures, priorities (31)
Timeouts (31)
Case study: failure detection (32)

Dining philosophers with timeouts (33)
orelse (34)
prialt and priserve (35)
Generalized alt (36)
Extended Rendezvous (37)
Extended Rendezvous Read (38)
Summary of Alternation Notations (41)
Alternation Notation (41)
Event Notation (42)

Part 7: Interacting Peers

Interacting peers (2)
The happens-before relation, $\preceq$ (3)
Reasoning about patterns (5)
Centralized pattern (6)
The clients (7)
The controller (8)
Invariant (9)
Efficiency (10)
Symmetric pattern (11)
A node (12)
Invariant (14)
Removing contention (15)
Efficiency (16)
A ring topology (17)
The first ring topology protocol (18)
A normal node (19)
The initiator (20)
Efficiency (21)
A second ring protocol (22)
A node (23)
Buffering (24)
Tree-based protocol (25)
A node (27)
Correctness (29)
Efficiency (30)
Comparison (31)
Summary (32)

Part 7a: A Ring-Structured Database

Introduction (2)
Towards an algorithm: the problem (3)

Token Ring (4)
Observations (5)
Fixing the Token Ring Bug (6)
Cursory Performance Analysis (10)
Roscoe's Algorithm (11)

Part 8: Patterns of Concurrent Programming

Patterns of concurrent programming (1)
Process Farm (2)
Process Farm (aka Bag of tasks with replacements) (2)
Case-Study: Magic squares (3)
Partial solutions (4)
Representing partial solutions (5)
System Architecture (7)
A worker (8)
The Controller (9)
Distributed Bag of Tasks (13)
Removing the bottleneck from Bag of Tasks (13)
Working Node = Bag || Worker (14)
The Bag Serve-Loop (18)
Soliciting Tasks and the Termination Protocol (19)
Putting it all together: Magic Squares Revisited (22)
Recursive parallelism (24)
Case Study: Recursively-Parallel in-place Quicksort (25)
Limits of recursive parallelism (27)
Limiting the number of processes (28)
Performance: $ns/number$ vs. $log\ maxleaves$ array size 10^7, $K = 6$ (29)
Quicksort: process-farming appears to beat recursive parallelism (30)
Tentative conclusion (33)
Other Patterns (34)
Competition parallel (34)
Task parallel programming (37)
Futures (39)

Part 8a: Server Elimination

Server Elimination: Replumbing for efficiency (1)
Example: Tee2Gen (2)

Part 9: Clients and Servers

Introduction (2)
Requests and responses (3)
The type of requests (4)
Servicing the request – stateless servers (5)

Clients of Stateless Servers (6)
System Structures for Stateless Servers (7)
Servicing the request – sessional servers (8)
Refinements (10)
Multiple subclasses versus alternation (10)
Problems with sessional servers (11)
Using timeouts (12)
Descriptors, Cookies, Persistent Sessions (13)
Case Study: Memory Allocation Server (16)
Encapsulating Servers (18)
Monitors reimplemented as encapsulated Servers (20)
Part 10: CSO vs GO
Go – the language (1)
Integration in Go (2)
Implementation of PAR (Sufrin): first approximation (7)
Forking Hell! (9)
SELECT has a wart (10)