Concurrency and Parallel Computing

Ric Glassey
glassey@kth.se
Outline

• Concurrency and Parallel Computing
  – **Aim:** “Introduce the paradigms of concurrency and parallel computing and distinguish between them”
  – Course information
  – Motivation: compute many things, at the same time
  – Paradigm features and comparison
    • Sequential, concurrent and parallel computing
  – Complications of concurrency
  – Using Go for concurrent computing
COURSE INFORMATION
Course Information

• 3 week course
• 3 homework exercises
• Class consists of INDA students + incoming PALINDA students
• Aims:
  – Introduce the topic of concurrent and parallel computing
  – Introduce Go, a systems language with built-in support for concurrency
• All text books, exercises and homework here
  – http://www.csc.kth.se/utbildning/kth/kurser/DD1339/inda14/
MOTIVATION
Computation was a limited resource
Democratisation of computation
Different sub-tasks; different speeds

- User Input
- Task A: I/O
- Disk Operation
- Network Event
- Task A: Computation
- Short
- Long
Wasted potential

User Input

Task A: I/O

Disk Operation

Network Event

Short

Long

Task A: Computation

Computer

Core 1
Core 2
Core 3
Core 4
From scarcity to abundance

• The availability of computational resources has exploded in terms of
  – Number of devices
  – Number of processors per device

• Multiplexing mainframe and independent computers act neatly as metaphors for
  – Concurrency
  – Parallel programming

• Yet we develop most software in a predominantly sequential, imperative style
PARADIGMS
Sequential Paradigm

Assume a single processor, single core architecture
Concurrency and Parallelism

• Often misused as synonyms
• In computer science / programming they are distinctly different, but ultimately related
  – A **parallel program** will use a multiplicity of independent processing units to accomplish a task(s)
  – A **concurrent program** is one structured such that there are multiple threads of control, which give the impression of simultaneous execution
Parallel Paradigm

Time

Processing Unit 1
Task A

Processing Unit 2
Task B

Processing Unit 3
Task C
Concurrent Paradigm

Time

Task A  Task B  Task C

pause  pause  pause

some progress on all tasks
COMPLEXITIES
Design Effort

• The first complication is designing concurrent and parallel systems
  – What is the appropriate level of division of task
    • Decomposition across many levels
      – Blocks
      – Functions
      – Classes
      – Tasks
  – What communication is required between task
    • Ideally, non, deterministic outcomes
    • Normally, some, non-deterministic outcomes
  – How can correctness in task be maintained
    • No longer any guarantees of order
Debugging effort

• Execution is potentially non-deterministic
  – Can go wrong in multiple places
  – Can go wrong at different times
• No two executions yield the same sequence of operations that lead to a bug
• Creates another region for bugs to hide and go unnoticed until some critical moment
### Race Condition

Two Tasks that are coordinated in a sequential order

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Account Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Balance</td>
<td></td>
<td>1000 Kr</td>
</tr>
<tr>
<td>Decrease Balance</td>
<td></td>
<td>1000 Kr</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td>400 Kr</td>
</tr>
<tr>
<td>Read Balance</td>
<td></td>
<td>400 Kr</td>
</tr>
<tr>
<td>Decrease Balance</td>
<td></td>
<td>400 Kr</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td>100 Kr</td>
</tr>
</tbody>
</table>
## Race Condition

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Account Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Balance</td>
<td>Read Balance</td>
<td>1000 Kr</td>
</tr>
<tr>
<td>Decrease Balance 600 Kr</td>
<td></td>
<td>1000 Kr</td>
</tr>
<tr>
<td>Decrease Balance 700 Kr</td>
<td></td>
<td>1000 Kr</td>
</tr>
<tr>
<td>Update</td>
<td>Update</td>
<td>-300 Kr</td>
</tr>
</tbody>
</table>

Two Tasks that are not coordinated and executing concurrently
Deadlock

Task A wants Resource 1.

Resource 1 wants Memory, File, and Processor.

Task B wants Resource 2.

Resource 2 wants Memory and File.
Deadlock

Task A

Resource 1

Wants

Held By

Resource 2

Wants

Held By

Task B
Tasks A & B may have higher priority, or the scheduling algorithm employed is unfair.
CONCURRENCY IN GO
Concurrency in Go

- Relatively new systems language
- Designers recognised the need for concurrency primitives in a systems language
- Provides simple features to investigate concurrency

Later today
- Goroutines (light weight processes that can be created and execute functions concurrently)
  - `go f()`

- Channels (communication mechanism between goroutines)
  - `go func() { messages <- "ping" }()`
  - `msg := <-messages`
Readings

• Fundamentals of Concurrent Programming
  – by S. Nilsson
  – Required Reading
  – **Sections 1 + 2**

• Go for Java Programmers
  – by S. Nillson
  – **Work through text in line with course**

• Go website
  – Lots of useful reference and rationale for Go