Fachhochschule Nordwestschweiz

Module Concurrent Programming

The Free Lunch is Over![1]

- **Past 30 years**
  - Clock speed
- **Now**
  - Multicore

But multi-core CPUs will have nearly no impact on most current applications !

[1] http://www.gotw.ca/publications/concurrency-ddj.htm

Multicore CPUs are here!

Octa Core!

Galaxy A40 Black

64 Cores!

AMD Ryzen
Threadripper 3990X

5120 Cuda Cores

PNY

Grafikkarte NVIDIA Tesla V100
32GB

68 Cores
272 HW Threads

Özgür Akgün @ozgurakgun · Apr 11

this is indeed how most programs work on a multicore computer!

Amdahl's Law

- **Formula to predict the maximum speedup using multiple processors (N) based on the proportion of parallelizable (p) and serial (1-p) component of a program.**

$$\text{Speedup} \leq \frac{1}{(1-p) + \frac{p}{N}}$$

- **Speedup is limited by the time needed for the sequential fraction of the program**

All concurrent applications have some sources of serialization!

Amdahl's Law: Example (p = 0.5)

$$\frac{1}{(1-p) + \frac{p}{N}}$$

N = 1

$$S = \frac{1}{(1 - 0.5) + 0.5} = 1$$

N = 2

$$S = \frac{1}{(1 - 0.5) + 0.25} = 1.33$$

N = ∞

$$S = \frac{1}{(1 - 0.5) + 0} = 2$$

Amdahl's Law: Example (p = 0.9)

$$\frac{1}{(1-p) + \frac{p}{N}}$$

N = 1

$$S = \frac{1}{(1 - 0.9) + 0.9} = 1$$

N = 2

$$S = \frac{1}{(1 - 0.9) + 0.45} = 1.82$$

N = ∞

$$S = \frac{1}{(1 - 0.9) + 0} = 10$$

Amdahl's Law

$$\frac{1}{(1-p) + \frac{p}{N}}$$

Source: http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Concurrent Programming

Concurrent vs Parallel

- **Concurrent Program**
  - has multiple logical threads of control
  - Concurrency is about dealing with lots of things at once
  - To solve a concurrent problem you have to handle events which can happen simultaneously
  - Concurrent programs are often non-deterministic (result depends on timing of events)
- **Parallel Program**
  - executes different parts of the computation simultaneously (in parallel)
  - Parallelism is about doing lots of things at once
  - To solve a parallel problem, the problem has to be broken down into pieces
  - Parallelism doesn't imply non-determinism

Parallelism: Omnipresence

- **Modern computers are parallel on many different levels**
  - Bit-level parallelism
    - 64bit computer is faster than an 8bit computer
  - Instruction-level parallelism (with the illusion that everything is sequential)
    - Pipelining
    - Out of order execution
    - Speculative execution
  - Data parallelism (SIMD)
    - GPUs have evolved into extremely powerful data parallel processors
  - Task-level parallelism
    - Multi-core machines, typically shared memory
    - => memory model

Concurrency: Omnipresence

- **Threads are everywhere**
  - Even if a programmer does not explicitly create threads, frameworks may create threads, and code called from these threads must be thread-safe!

- **Examples in Java**
  - Threads are used by every Java application
    - GC-Thread
    - Finalization-Thread
  - Java FX applications:
    - JavaFX Application Thread (Event dispatcher)
    - QuantumRenderer Thread
  - Web application frameworks:
    - Different requests are executed by different threads

Consequences

- **Applications need to be concurrent if you want to exploit multicores**
  - Existing applications have to be redesigned
- **Good support of programming languages/libraries**
  - Java [java.util.concurrent]
    - Biased towards shared memory and the required coordination mechanisms
  - Special language concepts become mainstream
    - Functional Programming (Immutability, side effect avoidance)
    - Actor model (Erlang, Scala)
    - Software transactions (STM: Clojure, Scala)

The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency. ***[Herb Sutter]***

Risks / Problems

- **Safety hazards**
  - Nothing bad ever happens
  - Race conditions
- **Liveness hazards**
  - Something good eventually happens
  - Deadlock, Livelock, Starvation
- **Performance hazards**
  - Over-Synchronization: Amdahl's Law
  - Context switches are not free: Saving and restoring execution context
- **Testing is not reliable**
  - Scheduling is not deterministic
- **Debugging is often impossible**
  - Heisenbugs (bug disappears when trying to observe it)

```
@NotThreadSafe
public class IdGenerator {
   private int currentId;

   public int nextId() {
      return currentId++;
   }
}
```

Goals

- **The Students**
  - … know typical problems and hassles in the context of concurrent programs
  - … know and can apply abstract solution patterns for such problems
  - … know different paradigms to solve concurrency problems such as actors and software transactions
  - … know the memory model used by Java programs
  - … can write correct concurrent programs

Reference for the first part

- ***Java Concurrency in Practice***
  - Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea
  - Addison-Wesley Professional (May, 2006)
  - ISBN-13: 978-0321349606

- ***Nebenläufige Programmierung mit Java***
  - Jörg Hettel, Manh Tien Tran
  - dpunkt Verlag (August, 2016)
  - ISBN-13: 978-3-86490-369-4

Reference for the second part

- ***Learning Concurrent Programming in Scala***
  - Aleksandar Prokopec
  - PACKT Publishing (November, 2014)
  - ISBN-13: 978-1783281411

Writing correct programs is hard;
writing correct concurrent programs is harder