Concurrency Paradigmen

Roman Roelofsen

Twitter: romanroe
GitHub: romanroe

Managing Director - Weigle Wilczek GmbH

Java Forum Stuttgart, 7. Juli 2011

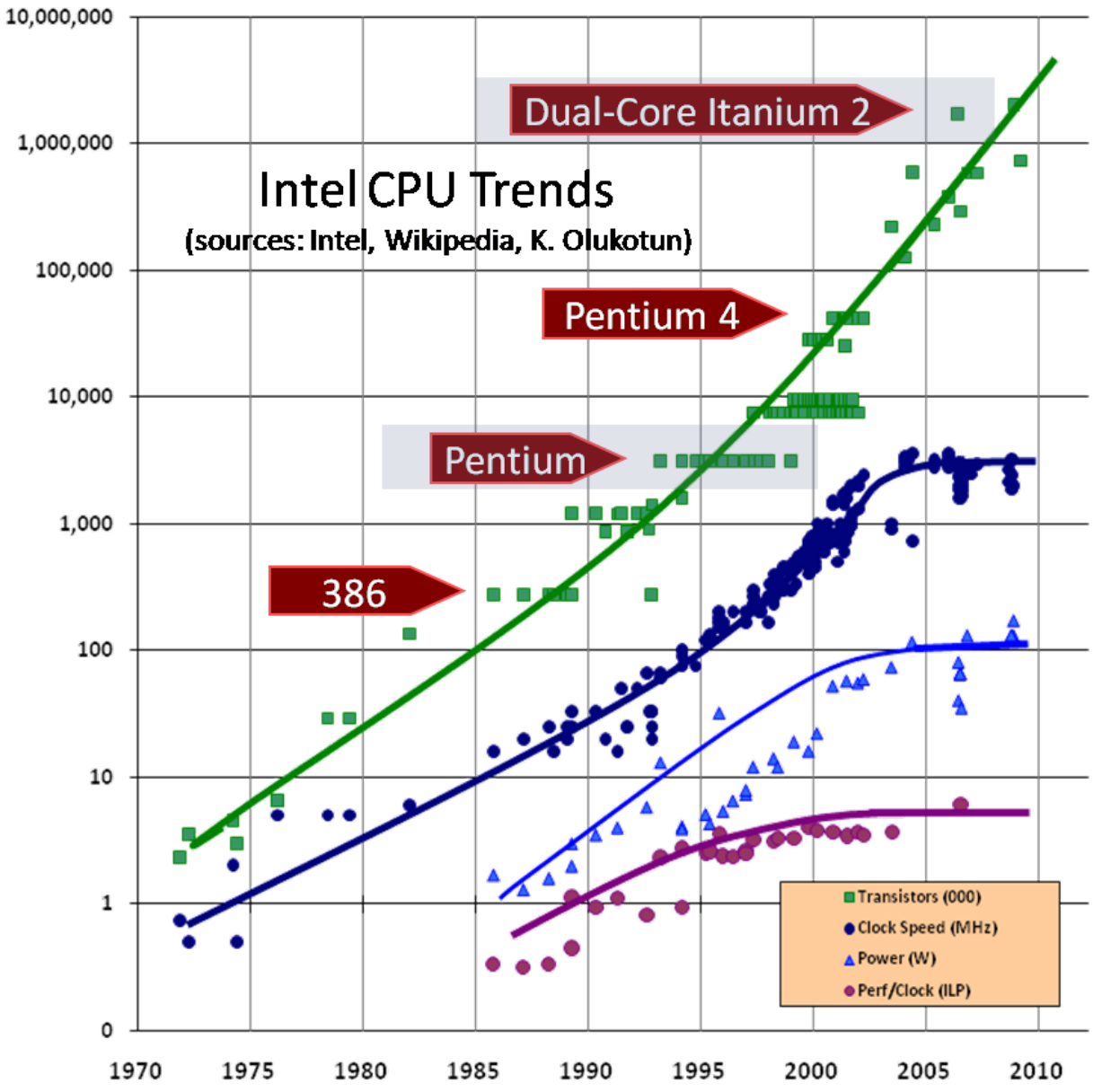
Concurrency - Nebenläufigkeit

- Beispiele für Nebenläufigkeit
  - Eine Anwendung erzeugt und startet neue Thread
  - Web-Server

Nebenläufigkeit - Relevanz Heute

- Programmierer müssen sich zunehmend mit Nebenläufigkeit befassen
- Anzahl Rich-Web-Client steigt
  - Browser
  - Smart Phones
  - Tablets
  - ... Autos, Fernseher, ...
- Steigende Anzahl von Threads auf dem Server
  - ... zwischen denen eine Kommunikation stattfinden muss
- Nebenläufigkeit war schon immer schwierig und wichtig
  - ... hat aber bisher selten die notwendige Aufmerksamkeit erhalten

Nebenläufigkeit - Relevanz Heute



Nebenläufigkeit - Relevanz Heute

- Multi-Core Krise
  - "The free lunch is over"
  - CPU Core Leistung stagniert
  - Anzahl der Cores steigt
  - Erhöhter Bedarf von Nebenläufigkeit
    - Prozesse
    - Threads

Nebenläufigkeit - Relevanz Heute

- Multi-Core Krise?
- Artikel, Blogs, etc. sprechen von riesigen Problemen
  - Multi Core Crisis, **Software Crisis**, ...
- Parallelisierungs-Frameworks sind die neuen "Framework-Stars"
  - Java: Kilim, Actors Guild, Actor Foundry
  - Scala: Scala Actors, Lift, Akka
  - Python: ParallelProcessing
  - Groovy: GPars

Nebenläufigkeit - Relevanz Heute

- Analyse Web Anwendungen
  - Web Anwendungen sind oft eine "Data Processing Pipeline" zwischen Browser und Datenbank
  - Jeder HTTP Request wird auf einen Thread übertragen
  - Anzahl der Request i.d.R. größer als Anzahl CPU Cores
- Entscheidene Fragen:
  - HTTP Requests $>=$ Cores?
  - Besteht die Notwendigkeit, zwischen den Requests zu kommunizieren/Daten auszutauschen?
  - Wird mit externen Systemen kommuniziert?
    - Datenbank
    - Web Service
    - ...

Nebenläufigkeit - Relevanz Heute

- Analyse Desktop Anwendungen
  - Swing/SWT GUI Elemente sind single-threaded $=>$ Autsch!
  - Events werden vermehrt parallel ausgeführt werden müssen
  - Desktop Anwendungen müssen einen Weg finden, mehere Cores gleichzeitig zu nutzen

Nebenläufigkeit - Relevanz Heute

- Analyse Desktop Anwendungen
  - Swing/SWT GUI Elemente sind single-threaded => Autsch!
  - Events werden vermehrt parallel ausgeführt werden müssen
  - Desktop Anwendungen müssen einen Weg finden, mehere Cores gleichzeitig zu nutzen
- Entscheidene Fragen:
  - Gibt es Performance Probleme?
  - Können die Algorithmen parallelisiert werden?
  - Wird mit externen Systemen kommuniziert?
    - Datenbank
    - Web Service
    - ...

Nebenläufigkeit - Herrausforderungen

- Einfach wenn...
  - Threads zu 100% isoliert laufen
  - Die Ausführungsumgebung single-threaded ist
  - Die Anwendung zustandslos ist
  - Das Ergebnis uninteressant ist (fire-and-forget)

Nebenläufigkeit - Herrausforderungen

- Identity, Value, State und Mutable State werden beim Einsatz aktueller Programmiertechniken zu oft vermischt[1]
- Definition:
  - Value: Ein stabiler, sich nicht verändernder Wert
  - Identity: Eine Entität, der im Laufe der Zeit verschiedene Werte zugeordnet werden
  - State: Der Wert, den eine Entität zu einem bestimmten Zeitpunkt hat
  - Mutable State: Wenn eine Entität verschiedene Werte annimmt
- Veränderbare Zustände sind ein Problem, wenn Nebenläufigkeit vorhanden ist

[1] Diskussionsgrundlage: http://www.clojure.org/state

Value/Identity/State - Johns Alter

Beschreibung:

- Identity: Das Alter von John McCarthy
- Values: Die Zahlen 50, 51, 52, ...
- State 1: John McCarthys Alter 1977
- State 2: John McCarthys Alter 1978
- Mutable State: Definitiv :-)

Value/Identity/State - Johns Alter

In Java:

- Identity:

```
int johnsAge;
```

- Values:

```
50; 51; 52;
```

- State 1: John McCarthys Alter 1977

```
johnsAge = 50;
```

- State 2: John McCarthys Alter 1978

```
johnsAge = 51;
```

Value/Identity/State - Johns Alter

- Identity:

```
int johnsAge = 50;
```

- Threads:

```
Thread 1 { johnsAge = 51; }
Thread 2 { johnsAge = 52; }
Thread 3 { johnsAge = 53; }
```

- Servlet Code

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res) {
    render(johnsAge);
    render(johnsAge);
    render(johnsAge);
}
```

Nebenläufigkeit - Relevanz Heute

Neues Beispiel

Geld von Konto A auf Konto B überweisen

Value/Identity/State - Geld überweisen

- Konto-Entitäten

```
Konto a = new Konto();
Konto b = new Konto();
```

- Geld überweisen

```
Thread {
    a.abheben(100);
    b.einzahlen(100);
}
```

- Kontostand prüfen

```
Thread {
   a.getKontostand();
   b.getKontostand();
}
```

Value/Identity/State - Geld überweisen

- Lösungsversuch 1

```
class Konto {
    public synchronized void abheben(long betrag) {
        ...
    }
    public synchronized void einzahlen(long betrag) {
        ...
    }
}
```

Value/Identity/State - Geld überweisen

- Lösungsversuch 2

```
synchronized (kontoA) {
    synchronized (kontoB) {
        kontoA.abheben(100);
        kontoB.einzahlen(100);
    }
}
```

Value/Identity/State - Arbeit parallelisieren

Java Code

Thread 1, Thread 2, Thread 3

Java's Bordmittel

- Java Syntax / API
  - synchronized
  - Thread.sleep()
  - Thread.interrupt()
  - Thread.join()
  - Thread.wait()
  - Thread.notify()
- Evaluierung
  - Sehr low-level
  - Lässt sich kaum auf Geschäftlogik übertragen
  - Code lässt sich schwer komponieren
  - Geringe Wiederverwendung
  - Gefahr von Deadlocks

Gentlemen: Choose Your Weapons!

- Executor
- Future/Promise
- Actors
- STM
- Data Flow Concurrency

Executor

Java Code

Executor

Future/Promise

- Bisher waren unsere Threads "fire-and-forget"
- Das Ergebnis wurde nicht verarbeitet
- Doch wieder Thread.join()? (Igitt!!)

Future/Promise

Java Code

Future

Actors

- Message-passing Concurrency
- Bekannt durch Erlang Programmiersprache
- Kein "Shared State"
- Actors kommunizieren über Messages
  - Asynchron & non-blocking
- Reactive
  - Actors senden selbstständig Messages
  - Actors können neue Actors erzeugen
  - Actors bestimmen selber ihr Verhalten beim Empfang

Actors

- Jeder Actor hat seinen eigenen "Lebensraum"
  - Actor $\Longleftrightarrow$ Thread, oder
  - Thread Pool
- Mailbox
  - Empfangene Nachrichten werden in der Mailbox gespeichert
  - Sequentielle Abarbeitung (wie single-threaded)
  - Actor entscheidet selber, wann Nachrichten aus der Mailbox verarbeitet werden
- Implementierungen
  - Scala Actors (Teil der Scala Distribution)
  - Akka (Scala & Java)
  - Kilim (Java)

Actors

Scala Code

Actors1 Actors2 Actors3

Software Transactional Memory

- Software Transactional Memory (STM)
- Der Speicher (Heap/Stack) wird wie eine Datenbank behandelt
- ACI(kein D)
  - Atomar: Sequenz von Daten-Operationen wird entweder ganz oder gar nicht ausgeführt
  - Konsistent: Eine Sequenz von Daten-Operationen hinterlässt nach Beendigung einen konsistenten Datenzustand
  - Isoliert: Nebenläufige Daten-Operationen beeinflussen sich nicht gegenseitig

Software Transactional Memory

- Transaktionsklammer nötig
  - Begin, Commit, Rollback
  - Verschachtelungen sind möglich (Yiepiee!)
- Transaktionen werden bei Kollisionen wiederholt
  - Transaktionen dürfen keine Seiteneffekte beinhalten!
- Implementierungen
  - Multiverse (Java)
  - Scala STM
  - Elementarer Bestandteil der Sprache Clojure

Clojure

- "Lisp für die Massen"
- Läuft auf der JVM
- Interoperabilität mit Java
- Variablen können ihren Wert nur im Rahmen des STM Systems verändern

Clojure - "Hello John!"

```
(ns hello-john)

(def hello "Hello")

(defn say-hello [firstname]
  (println hello firstname "!"))

(say-hello "John")

>> Hello John !
```

Software Transactional Memory

Clojure Code

stm1 stm2 stm3

Fazit

- Jeder sollte mit den vorgestellten Konzepten vertraut sein
  - Eine konkrete Implementierung kann ihm Rahmen eines Projektes evaluiert und gelernt werden
- Es eilt nicht...
  - ... sollte jedoch auch nicht ignoriert werden
- Es gibt kein "Silver Bullet"
  - Jedes vorgestellte Konzept hat Vor- und Nachteile
  - Nicht jedes Konzept ist für jedes Problem geeignet

Durchstarten mit Scala

Wir stellen ein!

- Wir suchen fähige Java/Scala/Clojure Entwickler
- Wir bieten eine gesunde Mischung aus
  - Programmierer
  - Berater
  - Kicker-Profi
- Bitte bei mir melden!
  - roelofsen@weiglewilczek.com

Fragen?