

# Einleitung

Prof. Dr. David Sabel

LFE Theoretische Informatik



- 1 Warum nebenläufige Programmierung?
- 2 Begriffe der nebenläufigen Programmierung

# Warum ist Nebenläufige Programmierung interessant?

Unterschied: sequentiell – nebenläufig  
Anwendungsbereiche nebenläufiger Programmierung  
Schwierigkeiten bei nebenläufiger Programmierung

## Sequentielle Programme:

- Folge von (Maschinen-)Befehlen
- Befehle werden nacheinander ausgeführt
- Jeder Befehl ändert Hauptspeicher, Register, etc.
- Ausführung **deterministisch**

### Nebenläufige (parallele) Programme:

- Mehrere Befehle werden **gleichzeitig** durchgeführt.
- gleichzeitig kann auch **quasi-parallel** bedeuten, d.h. in Realität sequentiell
- Ausführung i.a. **nichtdeterministisch**
- formale Definition: später

## **Beispiel Betriebssysteme:**

- aus Benutzersicht: verschiedene Aufgaben werden gleichzeitig durchgeführt
- z.B. Drucken, Surfen, Mausbewegen, Musik hören (alles gleichzeitig)
- Umsetzung: ohne nebenläufige Programmierung undenkbar

## **Beispiel Web-Programming:**

- Client-Server Modell: Client fordert Dienst beim Server an
- z.B. WWW-Server: Client fordert Webseite an
- Notwendig: Server bedient mehrere Clients gleichzeitig
- Absturz eines Clients, führt nicht zum Hängenbleiben anderer Clients

## Auch auf Einprozessor-Systemen:

- Durch Nebenläufigkeit können Ressourcen manchmal besser genutzt werden
- Beispiel: 2 Aufgaben
  - Lange Berechnung
  - Große Datei schreiben
- Datei schreiben belastet nicht die CPU (Geschwindigkeit wird von der Festplatte / Bus bestimmt)
- Nebenläufigkeit: Wenn Festplatte busy, dann rechne.
- Nebenläufige Programmierung als **Strukturierungsprinzip**

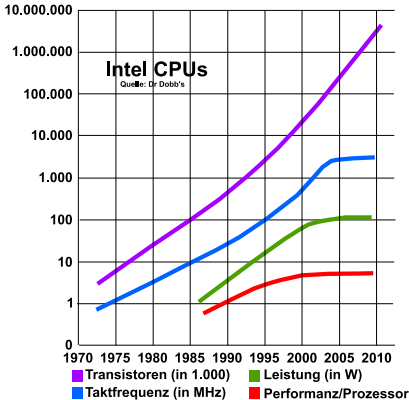


### **Hartnäckige Probleme und große Problem instanzen:**

- Berechnung durch massive Parallelisierung / verteilte Berechnung
- Hochleistungsrechner, “GRID-Computing”

# Nebenläufige Programmierung wird gebraucht (3)

## Entwicklung des Hardwaredesigns:



- Taktfrequenzen am Rande ihre physikalischen Möglichkeiten
- Statt Takterhöhung Erhöhung der Prozessoranzahl
- Multicore-Architekturen
- Architektur verlangt angepasste (parallelisierte) Programme
- Herb Sutter: Multiprozessorsysteme verlangen einen Paradigmenwechsel für Programmiersprachen

# Was macht nebenl. Programmierung so schwer?

---

- Parallelisierung sequentieller Algorithmen oft nicht offensichtlich.
- Hauptproblem: Die parallel ablaufenden Programme müssen korrekt zusammenarbeiten.
- Daten austauschen (**Kommunikation**)
- Aufeinander warten (**Synchronisation**)

# Beispiel: Kontoführung

---

(Der Wert von `konto` vor der Ausführung sei 100)

Prozess  $P$ :

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess  $Q$ :

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$

## Beispiel: Kontoführung (2)

---

Prozess P:

(P1) X:= konto;  
(P2) konto:= X-10;

Prozess Q:

(Q1) X := konto;  
(Q2) konto := X+10;

Ausführung (P1),(P2),(Q1),(Q2)  
konto = 100

## Beispiel: Kontoführung (2)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

Ausführung (P1), (P2), (Q1), (Q2)

$\text{konto} = 100$

## Beispiel: Kontoführung (2)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

Ausführung (P1), (P2), (Q1), (Q2)

$\text{konto} = 90$

## Beispiel: Kontoführung (2)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$   
 $X = 90$

Ausführung (P1),(P2),(Q1),(Q2)  
 $\text{konto} = 90$



## Beispiel: Kontoführung (2)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$   
 $X = 90$

Ausführung (P1),(P2),(Q1),(Q2)  
 $\text{konto} = 100$

## Beispiel: Kontoführung (3)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$

Ausführung (Q1),(P1),(P2),(Q2)  
 $\text{konto} = 100$

## Beispiel: Kontoführung (3)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (P2), (Q2)

$\text{konto} = 100$

## Beispiel: Kontoführung (3)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (P2), (Q2)

$\text{konto} = 100$

## Beispiel: Kontoführung (3)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (P2), (Q2)

$\text{konto} = 90$

## Beispiel: Kontoführung (3)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$   
 $X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$   
 $X = 100$

Ausführung (Q1),(P1),(P2),(Q2)  
 $\text{konto} = 110$

## Beispiel: Kontoführung (4)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$

Ausführung (Q1),(P1),(Q2),(P2)  
 $\text{konto} = 100$

## Beispiel: Kontoführung (4)

---

Prozess P:

(P1)  $X := \text{konto};$   
(P2)  $\text{konto} := X - 10;$

Prozess Q:

(Q1)  $X := \text{konto};$   
(Q2)  $\text{konto} := X + 10;$   
 $X = 100$

Ausführung (Q1), (P1), (Q2), (P2)  
 $\text{konto} = 100$



## Beispiel: Kontoführung (4)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (Q2), (P2)

$\text{konto} = 100$

## Beispiel: Kontoführung (4)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (Q2), (P2)

$\text{konto} = 110$

## Beispiel: Kontoführung (4)

---

Prozess P:

(P1)  $X := \text{konto};$

(P2)  $\text{konto} := X - 10;$

$X = 100$

Prozess Q:

(Q1)  $X := \text{konto};$

(Q2)  $\text{konto} := X + 10;$

$X = 100$

Ausführung (Q1), (P1), (Q2), (P2)

$\text{konto} = 90$

## Beispiel: Kontoführung (5)

---

### Analyse ergibt:

Reihenfolge	Wert von konto danach
(P1),(P2),(Q1),(Q2)	100
(Q1),(Q2),(P1),(P2)	100
(P1),(Q1),(P2),(Q2)	110
(Q1),(P1),(P2),(Q2)	110
(P1),(Q1),(Q2),(P2)	90
(Q1),(P1),(Q2),(P2)	90

### Fazit:

- Falsch programmiert.
- Traditionelles Debuggen funktioniert nicht
- Ergebnis kann von Ausführung zu Ausführung abweichen

# Verbraucher-Erzeuger Probleme

---

Annahme: Es gibt (Daten) erzeugende Prozesse und (Daten) verbrauchende Prozesse

Verschiedene Varianten:

- Mehrere Verbraucher und ein Erzeuger
- Mehrere Erzeuger und ein Verbraucher
- Mehrere Erzeuger und mehrere Verbraucher

# Verbraucher-Erzeuger Probleme

---

Annahme: Es gibt (Daten) erzeugende Prozesse und (Daten) verbrauchende Prozesse

Verschiedene Varianten:

- Mehrere Verbraucher und ein Erzeuger
- Mehrere Erzeuger und ein Verbraucher
- Mehrere Erzeuger und mehrere Verbraucher

## Gesucht:

Datenstrukturen um **sicheren** Austausch der Daten zwischen Verbrauchern und Erzeugern zu gewährleisten.

**Deswegen:** Nebenläufige Programmierung benötigt neue Datenstrukturen

# Begriffe der nebenläufigen Programmierung

Definition und Abgrenzung wichtiger Begriffe

## Parallelität und Nebenläufigkeit

### Paralleles Programm:

- Berechnung auf mehreren Prozessoren
- Gleichzeitig, überlappend

### Nebenläufige Programmierung (engl. concurrent programming):

- Ausführung auf mehreren Prozessoren ein Szenario
- Potentiell sind **alle** Ausführungsreihenfolgen möglich.



## Nebenläufige und verteilte Systeme

### Verteiltes System:

- System aus mehreren Prozessoren (oft auch örtlich getrennt)
- Kein gemeinsamer Speicher
- Datenaustausch: Alleinig über Senden und Empfangen von Nachrichten.

### Nebenläufiges System:

- Berechnung auf einem oder mehreren Prozessoren
- Gemeinsamer Speicher möglich

## Prozesse und Threads

- **Prozess** wird in der Theorie verwendet (z.B. im  $\pi$ -Kalkül)
- Oft Unterscheidung anhand der Kontrolle:
  - Betriebssystem verwaltet **Prozesse**
  - Programme verwalten **Threads**
  - wobei Programm ist Betriebssystem-Prozess

## Prozesse und Threads

- **Prozess** wird in der Theorie verwendet (z.B. im  $\pi$ -Kalkül)
- Oft Unterscheidung anhand der Kontrolle:
  - Betriebssystem verwaltet **Prozesse**
  - Programme verwalten **Threads**
  - wobei Programm ist Betriebssystem-Prozess
- Wir trennen nicht strikt zwischen beiden Begriffen.

## Prozesse und Threads

- **Prozess** wird in der Theorie verwendet (z.B. im  $\pi$ -Kalkül)
- Oft Unterscheidung anhand der Kontrolle:
  - Betriebssystem verwaltet **Prozesse**
  - Programme verwalten **Threads**
  - wobei Programm ist Betriebssystem-Prozess
- Wir trennen nicht strikt zwischen beiden Begriffen.
- **Multi-Threading** = Eigenschaft von Programmiersprachen: Konstrukte zu Verwaltung von Threads

## Prozesse und Threads

- **Prozess** wird in der Theorie verwendet (z.B. im  $\pi$ -Kalkül)
- Oft Unterscheidung anhand der Kontrolle:
  - Betriebssystem verwaltet **Prozesse**
  - Programme verwalten **Threads**
  - wobei Programm ist Betriebssystem-Prozess
- Wir trennen nicht strikt zwischen beiden Begriffen.
- **Multi-Threading** = Eigenschaft von Programmiersprachen: Konstrukte zu Verwaltung von Threads
- Verwandt: **Multi-Tasking** = Möglichkeit mehrere Aufgaben quasi-parallel durchzuführen

## Message-Passing- und Shared-Memory Modell

### Message-Passing-Modell:

- Prozesse haben keinen gemeinsamen Speicher.
- Kommunikation **ausschließlich** über Senden und Empfangen von Nachrichten.
- Passt zu verteilten Systemen

### Shared-Memory-Modell:

- Prozesse verwenden auch gemeinsamen Speicher.
- Kommunikation findet direkt über den gemeinsamen Speicher statt.

## Synchrone und asynchrone Kommunikation

### Synchron:

- Kommunikation zwischen Sender und Empfänger geschieht “ohne Zeit”
- Beispiel: Telefon

### Asynchron:

- Senden und Empfangen muss nicht gleichzeitig stattfinden.
- Oft natürlicher.
- Beispiel: Briefe oder Emails schreiben ist asynchron.