Utilizing Multiple Hardware Threads with Pipeline Parallelism

M. Anton Ertl
TU Wien

Motivation

- Multi-core Prozessoren und Simultaneous Multi-Threading (SMT) ... auf PCs
- Jahrzehnte Forschung an Parallelen Programmen
- ... für numerische Anwendungen
- Viele PC-Programme verhalten sich anders
  z.B. Anzahl der Schleifeniterationen
- Wie können wir die Möglichkeiten der Hardware
  für solche Programme nutzen?

Warum ist Parallelisierung schwer?

- Neue Fehlerklassen: race conditions, deadlocks
- Shared memory: Unintuitive Speichermodelle
- Synchronisierung vs. Modularität
- Ziel: Kürzere Laufzeit
- Overheads: Thread-Verwaltung, Synchronisieren
- Optimierung der Zahl der Tasks vs. Modularität

Transaktionen

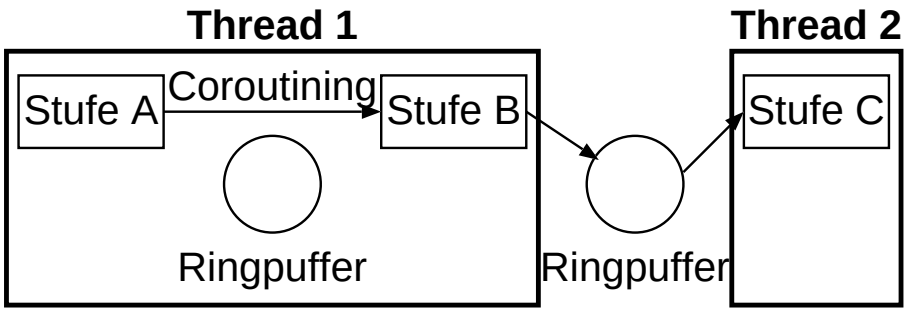
- Besser bezüglich Modularität, aber:
- Shared memory
- Teuer

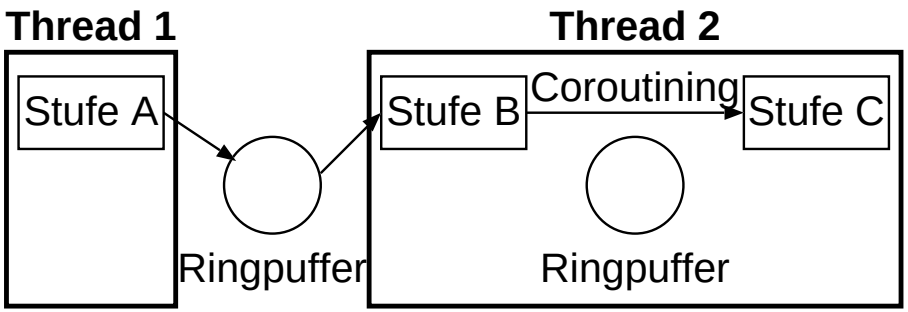
Pipelines

- Einfach zu benutzen
- Zusätzliche Form der Modularisierung
- Nicht-numerische Anwendungen
- Stream-Sprachen für DSP-Anwendungen
- XJava
  Implementation nur parallel für große Datenmengen
- S-Net

Skalierbare Implementation

- Wenig Kommunikationsoverhead erlaubt viele Tasks
- Ringpuffer zwischen Threads
- Coroutining zwischen Tasks in einem Thread
- Dynamisches Umschalten zum Lastausgleich





Zusammenfassung

- Parallelisierung ist schwierig
- Konflikt mit Modularität
- Pipelines: Parallelität und mehr Modularisierung
- Implementation per Ringpuffer und Coroutining
- Dynamische Umschaltung zur Lastverteilung