Ausnutzen von Multi-Cores mittels Pipelines

Forschungsplan

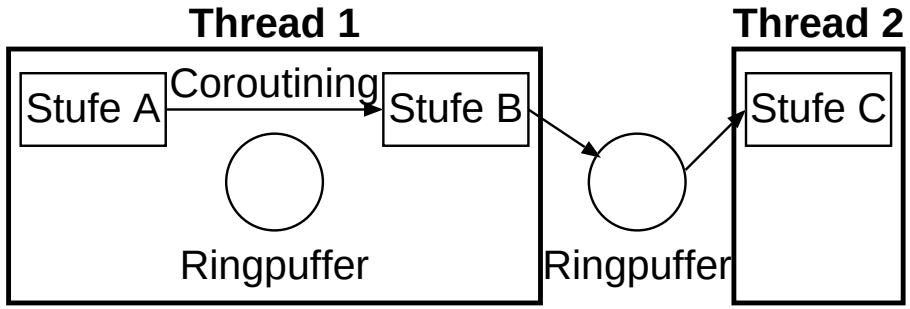
M. Anton Ertl
TU Wien

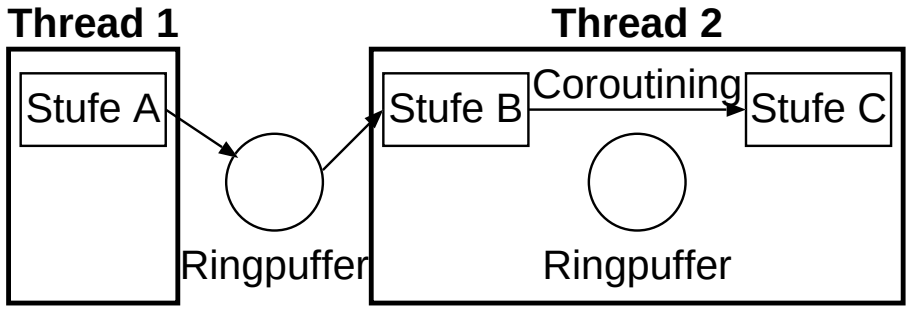
Problem

- Mehr Cores statt mehr GHz
- Concurrent Programming (mutexes etc.) zu schwierig
- Parallel Programming (Datenparallelismus) zu eingeschränkt

Lösung

- Pipeline Parallelismus mit vielen Tasks
- Implementierung: Effizienter (lockless) Ringpuffer
- Implementierung: Lastausgleich durch
  dynamisches Umschalten zwischen Ringpuffer und Coroutining





Wie publiziere ich das?

- Pipeline-Parallelismus ist nicht neu
  Lockless FIFO auch nicht

- Aber es ist auch kein etabliertes Thema

- Zu zeigen
  Pipeline-Parellelismus bringt's
  Meine Implementierungsidee bringt's

Zeige: Pipeline-Parellelismus bringt's

- Beispielprogramme schreiben und Parallelismus messen
- Implementation/Programmiersprache nötig
- Sehr viel Aufwand
  Nichts für mich
- Xjava (Walter Tichy et al., Karlsruhe)

Zeige: Implementierungsidee bringt's

- Z.B. in Xjava implementieren und mit XJava-Programmen messen
- evtl. langfristig, auch zuviel Aufwand für erstes Paper
- Ein paar Microbenchmarks entwerfen und messen.
- Separate Arbeiten für Ringpuffer und Dynamisches Umschalten

Welche Mikrobenchmarks für Ringpuffer

- Kommunikationsdurchsatz über Ringpuffer
  im Idealfall und bei leerem oder vollen Puffer
  Vergleich mit publiziertem lockless FIFO

- Skalierbarer Rechenaufwand
  Graph: Rechenaufwand vs. Durchsatz

- Messungen auf verschiedener Hardware (1-Sockel und 2-Sockel)

Welche Mikrobenchmarks für Ringpuffer mit Coroutining

- Ringpuffer allein mit zuvielen Threads fuer Cores
  vs. mit Umschalten auf Coroutining
- Overhead von Coroutining
- Eine Last, die dynamisches Hin- und Herschalten erfordert
- Wie skaliert das dynamische Umschalten mit der Anzahl der Tasks und der Threads