TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Maximilian Hajduk

Synchronisationsmechanismen bei Multicore-Architekturen

Lehrstuhlseminar // Dresden, 07. Juni 2018

Gliederung

- Motivation: Wozu Synchronisation?
- Software- vs. Hardwaresynchronisation
- Hardwaresynchronisation
  - Atomare Zugriffe
  - Probleme bei Synchronisationsmechanismen
  - Bewertungskriterien
  - Beispielprotokoll
- Fazit

Wozu Synchronisation?

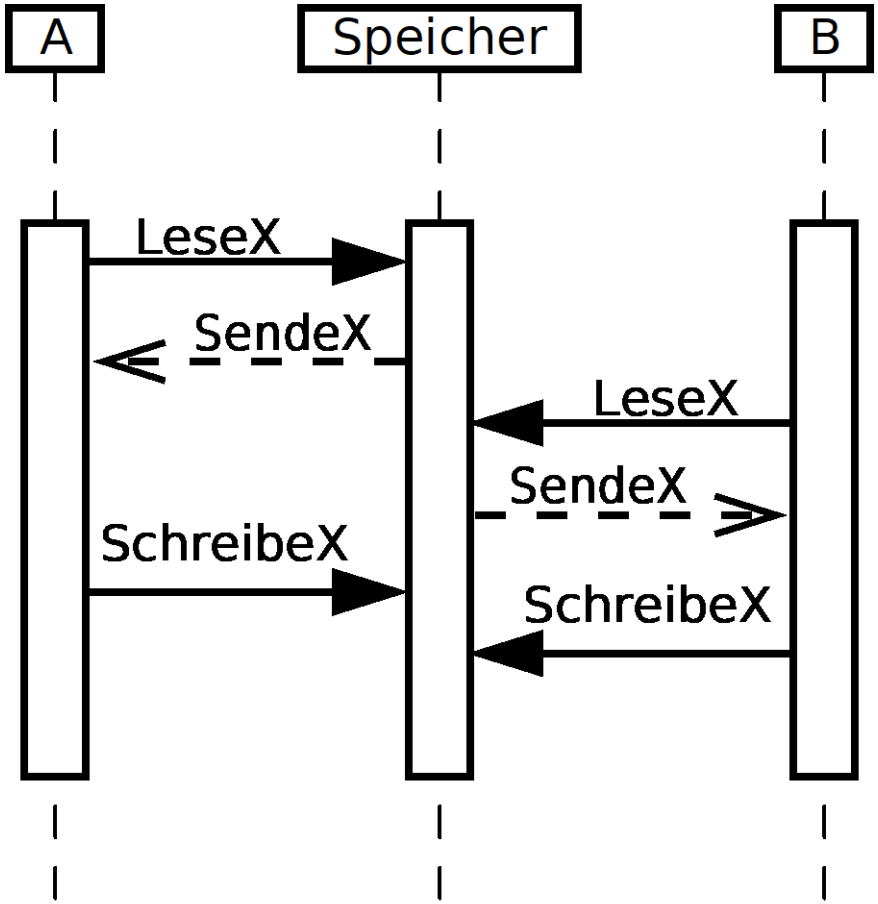

Abb. 1: Zugriff auf geteilte Variable

Software- vs. Hardwaresynchronisation

- Softwaremechanismen immer mehr in Hardware
- Hardwaresynchronisation Voraussetzung für Softwaresynchronisation
- Hardwaresynchronisation durch atomare Zugriffe und Nachrichten
- Softwaresynchronisation durch komplexe Algorithmen
  - Beispiele: wechselseitiger Ausschluss oder Synchronisationsschranken

Hardwaresynchronisation

- Abhängig von Speicherstruktur
  - Einteilung in gemeinsamen und verteilten Speicher
- Gemeinsamer Speicher:
  - Geteilt von mehreren Prozessoren
  - Kommunikation via Speicher und geteilten Variablen
- Verteilter Speicher:
  - Über gesamtes System verteilt
  - Kommunikation via Nachrichten

Atomare Zugriffe

- Bereitstellung von Hardware
- Beispiele:
  - compare-and-swap, load-link/store-conditional
- compare-and-swap:
  - Vergleich des geladenen Werts mit bestimmten, eigenen Wert
  - Tausch des Werts bei Gleichheit

- load-link/store-conditional:
  - Laden des Werts (load-link)
  - Arbeiten auf Wert (load-link)
  - Speichern des Werts, wenn Speicherinhalt unverändert ist (store-conditional)

- Bei RISC Architekturen load-link/store-conditional einsetzbar und compare-and-swap nicht einsetzbar

Probleme bei Synchronisationsmechnismen

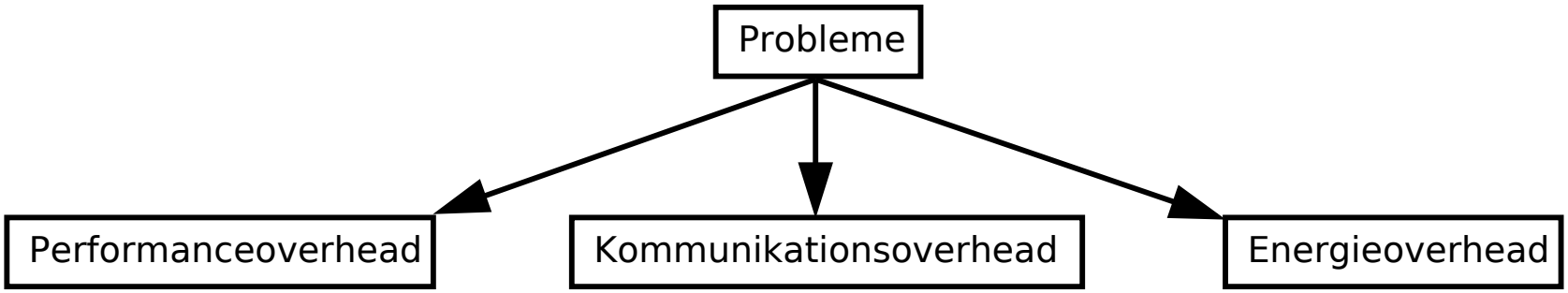

Abb. 2: Übersicht Probleme

- Performanceoverhead:
  - Prozessor mit Sendeanfragen beschäftigt
  - Sinkende Performance
- Kommunikationsoverhead:
  - Kontinuierliches Nachrichten senden bis Lock erhalten
  - Buslast vergrößert
- Energieoverhead:
  - Nebeneffekt von Performanceoverhead und Kommunikationsoverhead

Bewertungskriterien

- Granularität
  - Möglichst feingranular
- Latenz
  - Möglichst niedriglatent
- Skalierbarkeit:
  - Möglichst linear mit Anzahl der Kerne
- Flexibilität:
  - Möglichst anwendungsunabhängig
- Konfliktfrei

Beispielprotokoll

- Verwendung von Tagged Shared Variable Memory (TSVM)
  - Im Speicher an Wort spezielles Feld angehangen
  - In lokalen Speicher integriert
- Kohärenz durch Einsatz von Kohärenzbus
- Steuerung des Prozessors durch speziell definierte Befehle

- Befehle:
  - sws (store word with synchronization)
  - lws (load word with synchronization)
  - rxsws (relaxed sws)
  - rxlws (relaxed lws)
  - wlws (wait lws)

Zustandsdiagramm

- 3 Zustände:
  - E – empty
  - NE – near empty
  - FE – far empty
- *Instruktion*/*VerhaltenZurInstruktion*/*VerhaltenZuSynchronisationen*
- Gepunktete Pfeile: Aktionen, die Blockierung aufheben
- Einteilung Zustände in zwei Kategorien
  - full mit NE und FE
  - empty mit E

Zustandsdiagramm

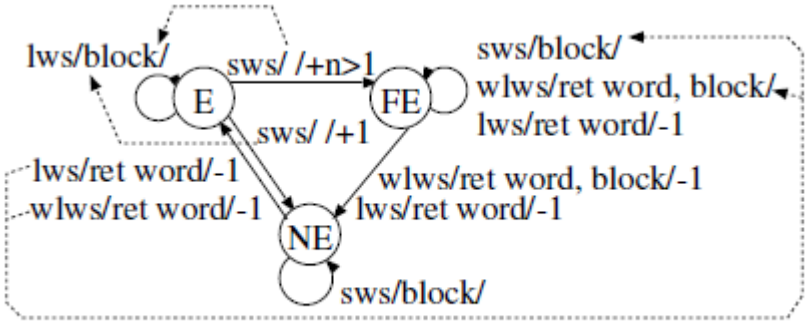

Abb. 3: Zustandsdiagramm [1]

- Wort in E:
  - lws-Zugriff: Blockierung des Prozesses
  - sws-Zugriff: Wechsel in Zustand FE oder NE
- Wort in FE:
  - lws-Zugriff: Wechsel in Zustand NE
  - sws-Zugriff: Blockierung des Prozesses
- Wort in NE:
  - lws-Zugriff: Wechsel in Zustand E
  - sws-Zugriff: Blockierung des Prozesses

Kohärenz

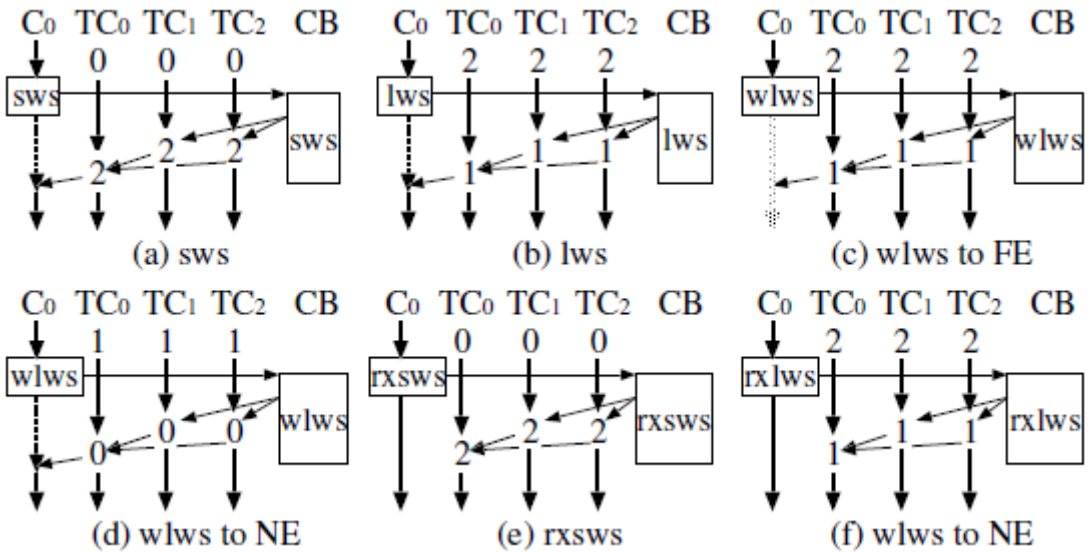

Abb. 4: Kohärenzabläufe [1]

- Alle Caches benötigen identische Daten
- Senden eines Befehls: Broadcast via Kohärenzbus
  - Aktualisierung aller Caches
- Blockierung der Prozesse bis alle Caches aktuell
  - Ausnahme: rxlws und rxsws

Bedingte Synchronisation

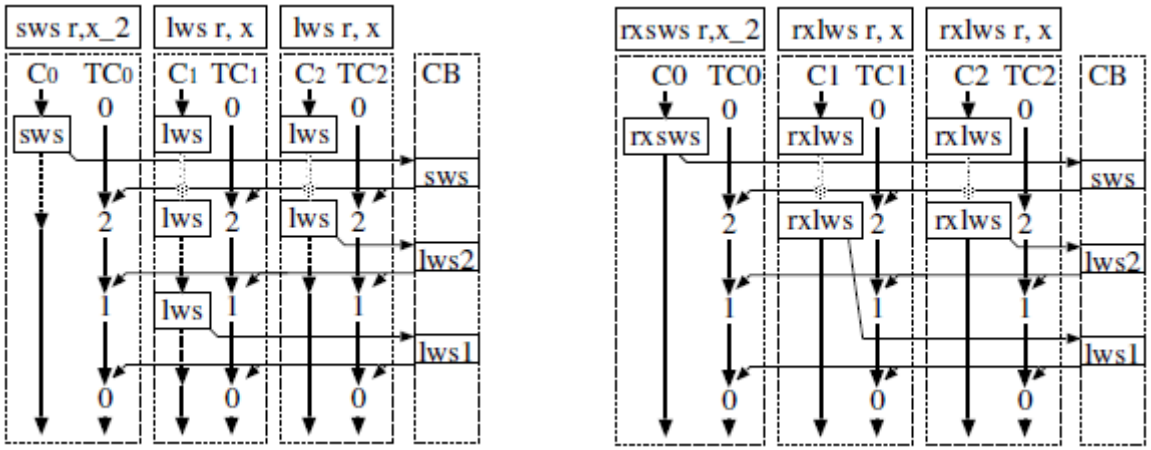

Abb. 5: Bedingte Synchronisation [1]

- Kombination von Befehlen
  - sws/lws Befehlspaar
  - Synchronisation via Anzahl der Synchronisationen

  - rxsws/rxlws Befehlspaar
  - Identisch zu sws/lws
  - Unterschied: kein Warten auf Cache-Aktualisierung

Wechselseitiger Ausschluss

- Kombination von Befehlen
  - lws/rxsws Befehlspaar
  - Zustand des Objekts zur Koordinierung der Synchronisation: NE
  - Betreten kritischer Abschnitt: lws
  - Verlassen kritischer Abschnitt: rxsws

  - sws/rxlws Befehlspaar
  - Zustand des Objekts zur Koordinierung der Synchronisation: E
  - Ablauf komplementär

  - Befehlspaare rxsws/rxlws und rxlws/rxsws nicht möglich

Wechselseitiger Ausschluss

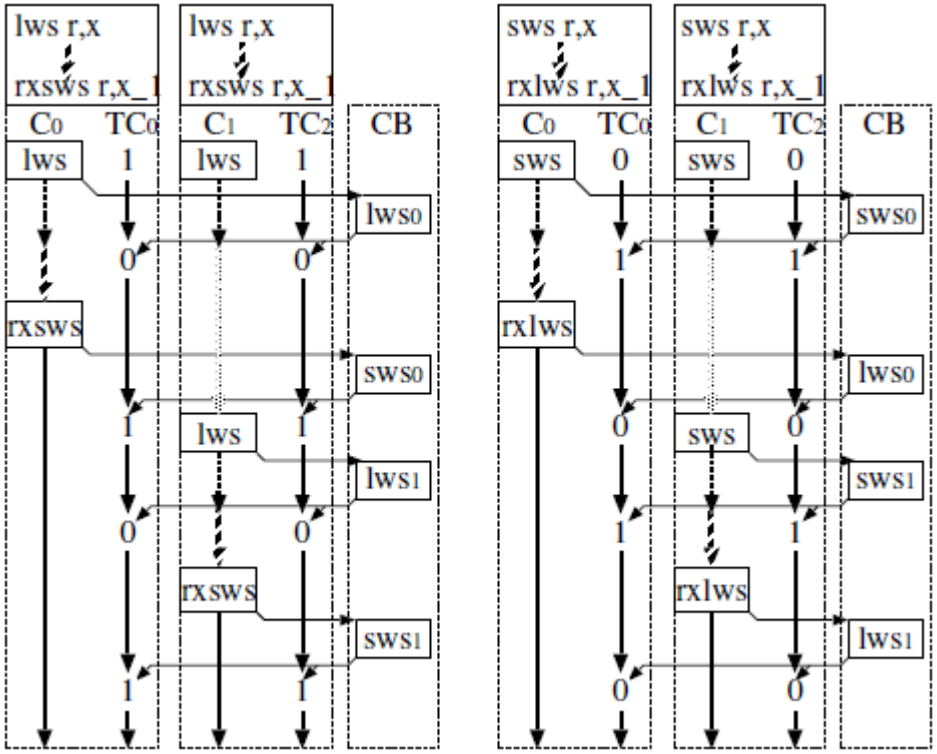

Abb. 6: Wechselseitiger Ausschluss [1]

Synchronisationsschranken

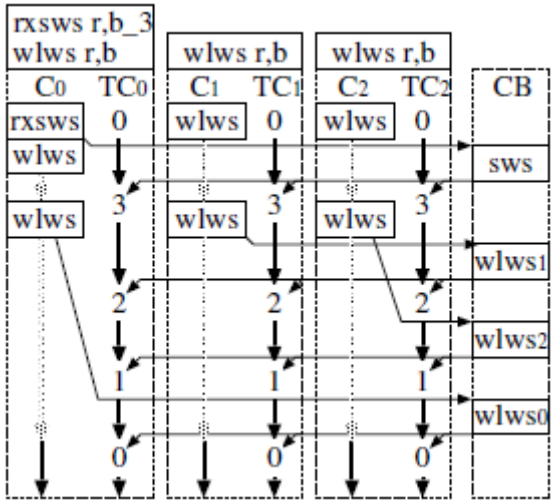

Abb. 7: Synchronisationsschranken [1]

- Kombination von Befehlen
  - rxsws/wlws Befehlspaar
  - Festlegung der Anzahl synchronisierender Prozesse: rxsws
  - Warten an Schranke: wlws

Protokollbewertung

Bewertungskriterium	Analyse
Granularität	feingranular
Latenz	niedriglatent
Konfliktfrei?	ja
Skalierbarkeit	nicht gut
Flexibilität	flexibel

Fazit

- Synchronisation zur Vermeidung von Konflikten
- Hardware: Bereitstellung von grundlegenden Befehlen (atomare Zugriffe, Nachrichten)
- Software: Implementierung komplexer Algorithmen (z.B. wechselseitiger Ausschluss)
- Bei Beispielprotokoll: Verwendung von Kohärenzbus und TSVM als Cache
- Befehle steuern Kohärenz und Synchronisation
- Komplexere Synchronisationen durch Kombination der Befehle möglich
- Skalierbarkeit des Protokolls und im Allgemeinen problematisch

Literaturverzeichnis

- John B Carter, Chen-chi Kuo, and Ravindra Kuramkote. A Comparison of Software and Hardware Synchronization Mechanisms for Distributed Shared Memory Multiprocessors UUCS-96-011 2 System Organization. pages 1–24, 1996.
- Maurice Herlihy. Wait-free synchronization. ACM Transactions on Programming Languages and Systems, 13(1):124–149, jan 1991.
- W. Jeffrey Mee and Gurdeep S. Hura. Synchronization techniques for distributed systems: An overview. Microelectronics Reliability, 32(1-2):175–197, jan 1992.
- Shaoshan Liu and Jean-luc Gaudiot. Synchronization Mechanisms on Modern Multi-core Architectures, volume 4697. Springer-Verlag, Seoul, 2007.
- Victor Frederico Silva, Cantidio De Oliveira Fontes, and Flavio Rech Wagner. The impact of synchronization in message passing while scaling multi-core MPSoC systems. In 2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC), volume 07-10-Octo, pages 267–270. IEEE, oct 2012.
- Akira Yamawaki and Masahiko Iwane. Coherence Maintenances to realize an efficient parallel processing for a Cache Memory with Synchronization on a Chip-Multiprocessor. In 8th International Symposium on Parallel Architectures,Algorithms and Networks (ISPAN'05), volume 2005, pages 324–333. IEEE, 2005.

Abbildungsverzeichnis

[1]: Akira Yamawaki and Masahiko Iwane. Coherence Maintenances to realize an efficient parallel processing for a Cache Memory with Synchronization on a Chip-Multiprocessor. In 8th International Symposium on Parallel Architectures,Algorithms and Networks (ISPAN'05), volume 2005, pages 324–333. IEEE, 2005.