Rechnerarchitektur SS 2012

Speicherkonsistenz

Michael Engel

TU Dortmund, Fakultät für Informatik

Teilweise basierend auf Material von Gernot A. Fink und R. Yahyapour

13. Juni 2013

Speicherkonsistenz

Kohärenz wichtig für Transfer von Daten zwischen Prozessoren

- Eine Speicherstelle wird geschrieben, anschließend gelesen
- Wert wird nach best. Zeit für lesenden Prozessor sichtbar
- Kohärenz sagt *nicht* aus, wann!

Ziel bei paralleler Programmierung oft, dass Lesen Ergebnis *eines bestimmten* Schreibvorgangs liefert Wieso?

- Ordnung von Lese- und Schreib-Op. soll hergestellt werden
- Zur Synchronisation werden i.a. mehrere Speicherstellen verwendet (z.B.“Flag” das Daten für gültig erklärt)

⇒ Konsistenz: Relative Ordnung zwischen Speichertransferbefehlen auf *verschiedenen* Zellen

Speicherkonsistenz II

Beispiel 1:

P_1	P_2
/* Annahme: A = flag = 0 */	
`A := 1;`	`while (flag == 0);`
`flag := 1;`	`print A;`

- Annahme: Schreibzugriffe zu `A`, `flag` werden in gleicher Reihenfolge bei anderem Prozessor sichtbar
- Aber: nicht durch Kohärenz sichergestellt!

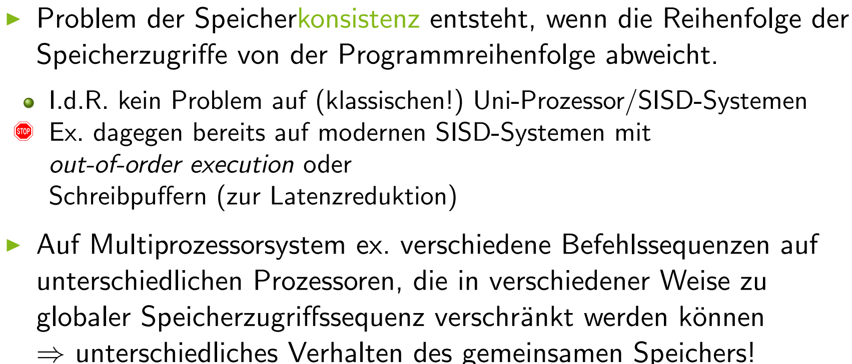
Beispiel 2:

P_1	P_2
/* Annahme: A = B = 0 */	
(1a) `A := 1;`	(2a) `print B;`
(1b) `B := 2;`	(2b) `print A;`

- Hier: Intuitiv “korrektes” Verhalten weniger klar, kein Flag o.Ä.
- Aber: Ausgabe von B=2 und A=0 widerspricht Annahme, dass Schreiboperationen in Programmreihenfolge sichtbar werden

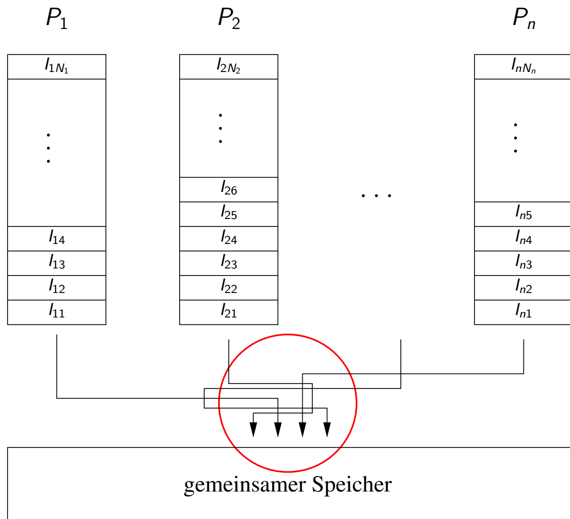
$\Rightarrow$ keine “sequentielle” Konsistenz

Speicherkonsistenz III



- Problem der Speicherkonsistenz entsteht, wenn die Reihenfolge der Speicherzugriffe von der Programmreihenfolge abweicht.
  - I.d.R. kein Problem auf (klassischen!) Uni-Prozessor/SISD-Systemen
  - Ex. dagegen bereits auf modernen SISD-Systemen mit *out-of-order execution* oder Schreibpuffern (zur Latenzreduktion)
- Auf Multiprozessorsystem ex. verschiedene Befehlssequenzen auf unterschiedlichen Prozessoren, die in verschiedener Weise zu globaler Speicherzugriffssequenz verschränkt werden können
  $\Rightarrow$ unterschiedliches Verhalten des gemeinsamen Speichers!

Speicherkonsistenz IV



Konsistenzmodell definiert, wie Instruktionssequenz und Speicherzugriffsequenz "kompatibel" (d.h. konsistent) gemacht werden

$\Rightarrow$ Ex. *starke* und *schwache* Modelle

Speicherkonsistenz V

- Speichermodell definiert, welches Verhalten eines shared-memory Systems von Prozessoren beobachtet wird
- Kompromiss erforderlich:
  - Starkes Modell $\Rightarrow$ minimale Einschränkungen an Software
  - Schwaches Modell $\Rightarrow$ effiziente Implementierung
- Formale Definition über partielle Ordnung von Speicherzugriffsereignissen
- Betrachten im folgenden 3 Typen von elementaren Speicherzugriffsereignissen für Multiprozessorsysteme:
  - Lesen/Laden (*load*)
  - Schreiben/Speichern (*store*)
  - Synchronisation (z.B. *swap*, d.h. atomare Sequenz von Lese- und Schreiboperation)

Speicherkonsistenz VI

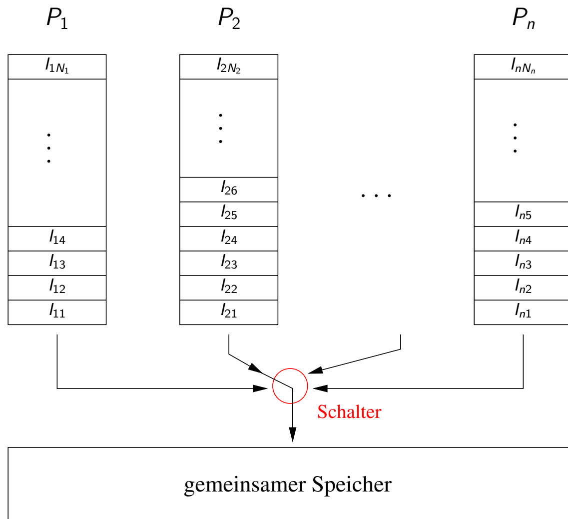
- Speicherkonsistenzmodell definiert *Einschränkungen* bzgl. der Ordnung, in der Speichertransferoperationen ausgeführt werden müssen (d.h. “sichtbar” werden)
  - Betrifft Operationen auf gleichen und unterschiedlichen Speicherzellen
  - Konsistenz *subsumiert* damit Kohärenz
- Größtmögliche Restriktivität: Sequentielle Konsistenz

  Multiprozessor ist sequentiell konsistent, wenn:

1. das Ergebnis jeder Programmausführung dasselbe ist, wie wenn alle Operationen des parallelen Programms in beliebiger sequentieller Reihenfolge ausgeführt würden, und
2. die Operationen einzelner Prozessoren in dieser Ordnung in Programmreihenfolge auftreten.

Was impliziert dies?

Sequentielle Konsistenz



Umschalter erzwingt Serialisierung der Speicherzugriffe verschiedener Prozessoren.

Achtung: Instruktionsordnung pro Prozessor in Programmreihenfolge!

Sequentielle Konsistenz II

Beispiel:

P_1	P_2
/* Annahme: A = B = 0 */	
`A := 1;`	`B := 1;`
`C := B;`	`D := A;`

Mögliche Serialisierungen der MP-Befehlsfolgen:

	gültig	gültig	gültig	*un*gültig!
	`A := 1;`	`B := 1;`	`A := 1;`	`C := B;`
	`C := B;`	`D := A;`	`B := 1;`	...
	`B := 1;`	`A := 1;`	`D := A;`	...
	`D := A;`	`C := B;`	`C := B;`	...
A,B,C,D =	1,1,0,1	1,1,1,0	1,1,1,1	?,?,?,?

Sequentielle Konsistenz III

Definition nach Lamport, 1979:

Ein MP-System ist sequentiell konsistent, wenn das Ergebnis jeder Ausführung dasselbe ist, als ob alle Instruktionen in einer sequenziellen Ordnung ausgeführt würden und in dieser die Operationen eines Prozessors in Programmreihenfolge auftreten.

Definition nach Sindhu *et al.*, 1992:

1. Jeder *load* liefert immer den Wert des letzten *store* in dieselbe Zelle von anderen Prozessoren.
2. Die Speicherordnung ist eine vollständige binäre Ordnung über alle Speicherzugriffspaare (Prozessoren u. Speicherzellen).
3. Wenn zwei Operationen in einer best. Programmreihenfolge erscheinen, erscheinen diese in derselben Speicherreihenfolge.
4. Die *swap* Operation ist atomar bzgl. anderer *stores*.
5. Alle *store* und *swap* Operationen müssen in endlicher Zeit terminieren.

nach Huang 1993, S. 252f

Sequentielle Konsistenz: Zusammenfassung

- Erfordert *eine* sequentielle Ordnung *aller* Operationen, in der *lokale* Programmreihenfolge (d.h. bezogen auf einzelnen Prozessor) erhalten bleibt
- Ist daher nicht deterministisch!
- Erfordert atomare Schreiboperationen!

Sequentielle Konsistenz: Diskussion

- Restriktivstes Konsistenzmodell
  - ✓ Intuitiv (relativ gut) verständlich
  - STOP Garantiert trotzdem *kein* eindeutiges Verhalten eines MP-Systems!
- Implementierung von Sequentieller Konsistenz
  - Einfachste Möglichkeit:
    Jeder Prozessor verzögert Abschließen (completion) eines Speicherzugriffs, bis alle Invalidierungen (d.h. ungültig erklären von Cache-Inhalten), die durch Zugriff ausgelöst werden, abgeschlossen sind.

    $\Rightarrow$ a.d. Basis von snooping Cache-Kohärenz-Protokoll möglich
    ↯ Leistungseinbußen erheblich!
  - Weitere (prinzipielle) Möglichkeit:
    Kosten seq. Konsistenz durch Verstecken von Latenz zu reduzieren versuchen
- Alternative: Verwendung eines weniger restriktiven Konsistenzmodells $\Rightarrow$ *relaxed consistency*

Speicherkonsistenz V

Sicht des Programmierers: Einfachheit des Progammiermodells

- Erfüllt durch seq. Konsistenz trotz Leistungseinbußen
- Alternatives Programmiermodell: Synchronisierte Programme
  - Zugriffe auf alle gemeinsam verwendeten Daten werden durch Synchronisationsoperationen geordnet
  - Bezeichnet als "data-race free"
    (data race: Speicherstellen können ohne Synchronisation aktualisiert werden $\Rightarrow$ Ergebnis von relativer Geschwindigkeit der Prozessoren abhängig)

$\Rightarrow$ Synchronisierte Programme verhalten sich, als ob MP-System sequentiell konsistent wäre (auch wenn einfacheres Konsistenzmodell implementiert!)

Speicherkonsistenz VI

Synchronisierte Programme

- Synchronisationsoperationen:
  - *acquire*-Operation: fordert Zugriffsrecht an (Spezialfall: lock)
  - *release*-Operation: gibt Zugriffsrecht frei (Speziallfall: unlock)
- Grundstruktur eines synchronisierten Programms:
  - Schreiboperation gefolgt von Sync.-Op. und ...
  - korrespondierender Leseop. geht Sync.-Op voraus

Beispiel:

```
X :=            /* Schreiben */
release(s);
acquire(s);
:= X            /* Lesen */
```

⇒ Reihenfolge der Schreib- und Leseoperationen garantiert

Speicherkonsistenz: Eingeschränkte Modelle

Grundidee:

- Abschließen von Lese-/Schreib-Op. entgegen Programmreihenfolge
- Synchronisation für *notwendige* Ordnung verwenden

Kategorisierbar nach aufgehobenen Restriktionen

- Aufheben der W→R Reihenfolge (total store order)
  - Kann Latenz von Schreibzugriffen verstecken
  - Ordnung von Schreibzugriffen bleibt erhalten
  - "fast" wie sequentielle Konsistenz
- Aufheben v. W→R u. W→S Ordnung (partial store order)
  - Schreiboperationen zusammenfassbar (write merging)
  - Verletzt intuitive Semantik von seq. Konsistenz stark!
- Aufheben aller Reihenfolgen (weak ordering)
  - Keine Ordnung garantiert, speziell für Prozessoren mit dyn. Scheduling
  - Ordnung nur durch Synchronisationsoperationen

Speicherkonsistenz: Eingeschränkte Modelle II

Total Store Order (TSO) entwickelt für SUN SPARC-Architektur

Definition nach Sindhu *et al.*, 1992:

1. Jeder *load* liefert immer den Wert des letzten *store* in dieselbe Zelle von *beliebigen* Prozessoren.
2. Die Speicherordnung ist eine vollständige binäre Ordnung über alle Paare von *stores*.

3a. Wenn zwei *stores* in einer best. Programmreihenfolge erscheinen, erscheinen diese in derselben Speicherreihenfolge.

3b. Wenn Speicheroperation in Programmreihenfolge auf *load* folgt, muss sie auch in Speicherreihenfolge auf das *load* folgen.

4. Die *swap* Operation ist atomar bzgl. anderer *stores*.
5. Alle *store* und *swap* Operationen müssen in endlicher Zeit terminieren.

✓ Leseoperationen können Schreiboperationen überholen!

✓ "Reads own write early" (d.h. gleiche Speicherzelle)

Speicherkonsistenz: Eingeschränkte Modelle III

Processor Consistency (PC) [Goodman 1989]

- Schreiboperationen desselben Prozessors sind immer in Programmreihenfolge.
- Keine Beschränkung für Leseoperationen eines Prozessors
- Leseoperationen, die auf Schreiboperationen folgen, können diese überholen.
- Schreiboperationen müssen lediglich in endlicher Zeit terminieren.

Sehr ähnlich zu TSO, zusätzlich: "Reads others' writes early"
(bei nicht-bus-gebundenem Verbindungsnetzwerk)

Partial Store Order (PSO) für SUN SPARC
wie TSO, aber spätere Schreibzugriffe können frühere überholen
(solange nicht gleiche Speicherzelle betroffen)

Wann kann das sinnvoll sein?

Speicherkonsistenz: Eingeschränkte Modelle III

Weak Ordering [Dubois *et al.* 1996]

Erforderlich: Explizite Synchronisierungsoperation

- Synchronisierungsoperationen sind sequenziell konsistent
- Alle Synchronisierungsoperationen müssen vor einer Speichertransferoperation abgeschlossen sein
- Alle Speichertransferoperationen müssen vor einer Synchronisierungsoperation abgeschlossen sein

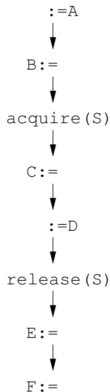
Release Consistency (RC) [Gharachorloo *et al.* 1990]

Erforderlich: Paar von Synchronisationsoperationen (*acquire* und *release*, realisiert z.B. als *lock* und *unlock*)

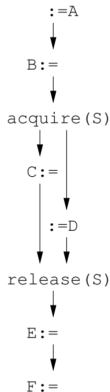
- Synchronisierungsoperationen sind *prozessor*-konsistent
- Prozessor wird angehalten, bis *acquire* beendet und
- Abschließen eines *release* wird verzögert, bis alle vorangeg. Speicheroperationen abgeschlossen.

Eingeschränkte Konsistenzmodelle: Überblick

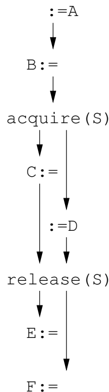
Sequentielle Konsistenz



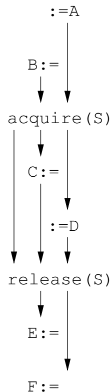
Total Store Order (TSO)



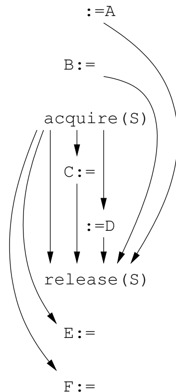
Partial Store Order (PSO)



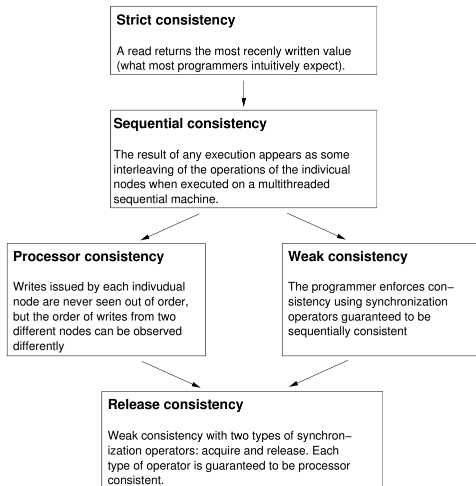
Weak Ordering



Release Consistency



Eing. Konsistenzmodelle: Zusammenfassung



nach Nitzberg/Lo:
Distributed Shared Memory:
A Survey of Issues and Algorithms,
IEEE Computer, 24(1), 1991.