Speicherkonsistenz

Bastian Ramm

17. Dezember 2002

Zusammenfassung

Diese Seminararbeit bietet eine Einführung in Konsistenzmodelle für Speicher in Mehrprozessorumgebungen. Es soll erklärt werden was Konsistenzmodelle sind, warum man sie braucht und was für Modelle es gibt. Besondere Beachtung findet die sequentielle Konsistenz und ihre gelockerten Varianten. Anhand von Beispielen sollen die Probleme verdeutlicht werden, die Konsistenzmodelle nötig machen. Die spezielle Problematik im Zusammenhang mit der Verwendung von Caches wird erläutert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Speicher in Mehrprozessorumgebungen

Parallele Systeme bei denen es sich um MIMD-Maschinen handelt lassen sich in Mehrprozessorsysteme und Mehrrechnersysteme unterteilen. [Tanenbaum, S.641, Abb. 8.14] Sie unterscheiden sich darin, das Mehrprozessorsysteme sich einen gemeinsamen Speicher teilen, während Mehrrechnersysteme über Nachrichten miteinander kommunizieren. Im folgenden wird beschrieben wie der Speicherzugriff in einem Mehrprozessorsystem funktioniert.

1.2 Auftretende Probleme

Obwohl man von einem gemeinsamen Speicher in einem Mehrprozessorsystem spricht, ist dies eine idealisierte Vorstellung. Die tatsächlich vorhandenen Speichermodule können an verschiedenen Stellen liegen und die Verbindung zwischen CPUs und Speichermodulen kann überaus komplex sein. So kann es vorkommen, das verschiedene Prozessoren gleichzeitig dieselbe Speicheradresse auslesen oder beschreiben wollen. Dabei kann nicht gewährleistet werden, das die Operationen in derselben Reihenfolge ausgeführt werden wie sie angefordert wurden. Ein zusätzliches Problem stellt sich, wenn dieselben Daten in mehreren Kopien, etwa in Caches vorliegen.

1.3 Konsistenzmodelle

Diesen Problemen begegnet man indem man sogenannte Konsistenzmodelle für Speicherzugriffe entwirft. Man spricht auch von einem Vertrag zwischen Software und Speicherhardware. Allgemein lässt sich sagen, das bei Einhaltung der Konsistenzregeln durch die Software bestimmte Ergebnisse durch den Speicher zugesichert werden. [Tanenbaum, S.650]

1.3.1 Begriffsdefinition Konsistenz

Die Bedeutung des Begriffes Konsistenz lässt sich in verschiedenen Zusammenhängen mit Einheitlichkeit, Zusammenhang, Widerspruchsfreiheit, Struktur- und Formbeschaffenheit oder Zuverlässigkeit beschreiben. Der direkte Gegensatz dazu ist die Inkonsistenz oder auch Uneinheitlichkeit, Zusammenhanglosigkeit, Widerspruch. [Langenscheidts Fremdwörterbuch, online]

Speicherkonsistenz in einem Mehrprozessorsystem bedeutet also, das die Daten in diesem System widerspruchsfrei und einheitlich sind. Die verschiedenen Prozessoren müssen bei Ihren jeweiligen Speicheroperationen berücksichtigen, das die Konsistenz für das gesamte System gewährleistet ist. Was für genaue Anforderungen hier gestellt werden ist abhängig von dem Konsistenzmodell, von denen es eine Fülle gibt. In den folgenden Abschnitten werden verschiedene Konsistenzmodelle aus [Tanenbaum,

] kurz vorgestellt. Auf einige von ihnen wird in einer anderen Seminararbeit genauer eingegangen. [Seminararbeit Th. Wild, Vortrag]

1.3.2 Strikte Konsistenz

Strikte Konsistenz ist das einfachste Konsistenzmodell. In diesem Modell werden alle Speicheroperationen nach dem „first come - first serve“-Prinzip abgearbeitet, so das jede Leseoperation den zuletzt geschriebenen Wert zurück gibt. Dadurch stellt der Speicher aber einen enormen Engpass im System dar.

1.3.3 Sequentielle Konsistenz

Die sequentielle Konsistenz ist das Hauptthema dieser Seminararbeit. Ihr ist ein eigenes Kapitel gewidmet. (siehe Seite 4) An dieser Stelle daher nur eine kleine Übersicht.

Die sequentielle Konsistenz ist bereits ein etwas schwächeres Modell als die strikte Konsistenz. Sie besagt im wesentlichen, das eine Reihenfolge für verschiedene, gleichzeitige Operationen gewählt wird. Diese Reihenfolge ist aber global und wird von allen Prozessoren beachtet.

Gelockerte Modelle

Es gibt verschiedene Modelle, die auf dem Modell der sequentiellen Konsistenz basieren und verschiedene Lockerungen definieren. Diese Modelle werden ebenfalls im Kapitel über sequentielle Konsistenz erläutert.

1.3.4 Prozessorkonsistenz

Bei der Prozessorkonsistenz gibt es zwei Regeln für Schreiboperationen auf den Speicher. Zum ersten sind die Schreiboperation eines bestimmten Prozessors für alle anderen Prozessoren in genau der Reihenfolge sichtbar in der sie ausgegeben worden. Die zweite Regel besagt, das alle Schreiboperationen auf einem Speicherwort für alle Prozessoren in genau der Reihenfolge sichtbar sind, in der sie ausgegeben wurden. Dies impliziert aber ausdrücklich *nicht* das Schreiboperationen verschiedener Prozessoren auf verschiedenen Speicherworten von allen Prozessoren in derselben Reihenfolge gesehen werden.

1.3.5 Schwache Konsistenz

In diesem Modell gibt es keine einheitliche Sichtbarkeit von Speicheroperationen. In Unterschied zu den anderen Modellen gibt es hier einen Synchronisationsoperation, die von einem beliebigen Prozessor ausgelöst werden kann. Diese Synchronisationsoperation bewirkt, das alle Schreiboperationen zum Anschluss gebracht werden. Es dürfen keine neuen Schreiboperationen begonnen werden bevor die Synchronisation nicht selbst beendet ist. Dieses Prinzip ist vergleichbar mir „Barriers“ in der Programmierung von Threads.

1.3.6 Freigabekonsistenz

Die Freigabekonsistenz ist eine Weiterentwicklung der schwachen Konsistenz. Es gibt zwei Synchronisationsoperationen anstatt nur einer. Die „acquire“-Operation überprüft ob alle Schreiboperationen auf den gemeinsamen Daten beendet sind. Ist dies der Fall gibt sie einem Prozessor exklusiven Zugriff auf diese Daten. Mit der „release“-Operation kann der Prozessor die Daten nun wieder für alle frei zugänglich machen. Dazu müssen seine Schreiboperationen allerdings noch nicht beendet sein. Sollte ein anderer Prozessor auf die Daten zugreifen wollen, so überprüft er ja zunächst mittels „acquire“ ob alle Schreiboperationen bereits beendet wurden.

Kapitel 2

Sequentielle Konsistenz

2.1 Definition

Das Modell der sequentiellen Konsistenz ist ein recht einfaches Konsistenzmodell. Die Einschränkungen die es definiert sind noch relativ hart. Es wird wie folgt definiert:

> Ein Mehrprozessorsystem kann als sequentiell konsistent bezeichnet werden, wenn das Ergebnis jeder Befehlsausführung dasselbe ist wie wenn sämtliche Operationen der Prozessoren in einer sequentiellen Reihenfolge ausgeführt würden und alle Befehle eines einzelnen Prozessors in der Reihenfolge, die vom jeweiligen Programm des Prozessors vorgegeben wird, erscheinen. [Tutorial, S.5, frei übersetzt]

Etwas verständlicher formuliert, bedeutet das, das das sequentielle Konsistenzmodell verlangt, das Schreiboperationen für alle Prozessoren in derselben Reihenfolge sichtbar werden. Es ist also nicht möglich, das ein Prozessor eine andere Reihenfolge feststellt als ein zweiter Prozessor.

Aus Sicht des Programmierers ist dies ein sehr einsichtiges Modell. Für ihn hat es immer den Anschein, das eine bestimmte Reihenfolge ausgeführt wird. Diese Sichtweise entspricht (annähernd) der Sichtweise bei der Programmierung sequentieller Programme.

Hardwareoptimierungen, die in Einzelprozessorsystemen eingesetzt werden und dort keinerlei Probleme bereiten, können in einem Mehrprozessorsystem leicht zu Vrletzungen des Konsistenzmodelles führen. Im folgenden werden drei einfache Beispiele für Hardwareoptimierungen aus [Tutorial,] aufgegriffen und erläutert, um die Probleme mit dem sequentiellen Konsistenzmodell zu veranschaulichen.

2.2 Optimierter Schreibpuffer

Die Rede ist hier von Systemen mit einem einfachen Schreibpuffer. Soll eine Schreiboperation durchgeführt werden, so wird diese in einem Puffer platziert. Das Programm kann dann weiter ausgeführt werden, ohne die Beendigung der Schreiboperation abzuwarten. Lesezugriffe finden wie gehabt statt, solange die gewünschte Leseadresse nicht Ziele einer sich im Puffer befindlichen Schreiboperation ist.

Es ist leicht, sich ein Beispiel vorzustellen, das das sequentielle Konsistenzmodell verletzt. Wenn ein Prozessor von einer Speicheradresse lesen will, so bemerkt er nur die vorangegangenen Schreibzugriffe auf diese Adresse, die bereits beendet sind, oder die in seinem eigenen Puffer liegen. Die Schreiboperationen in den Puffern der anderen Prozessoren bemerkt er nicht. Somit ist ihm eine andere Reihenfolge der Speicherzugriffe bekannt als den anderen Prozessoren, was einen Verstoss gegen das sequentielle Konsistenzmodell darstellt.

2.3 Überlappende Schreibzugriffe

In diesem Beispiel wir das vorhergenannte Beispiel noch um einen Aspekt erweitert. Zusätzlich zu den Schreibpuffern, gehen wir hier davon aus, das der Speicher seinerseits wieder aus verteilten Speichermodulen besteht. Verschiedene Schreiboperationen desselben Prozessors, können also bei unterschiedlichen Speichermodulen landen.

Das Konsistenzmodell kann hier verletzt werden, wenn sich zwei Prozessoren in diesem Beispiel über eine Variable synchronisieren, die den Zugriff auf eine andere Variable sperrt bzw. freigibt. Überholt beispielsweise die Schreiboperation für die Freigabe diejenige die die eigentliche Varaiable beschreiben soll, so ist es für andere Prozessoren möglich auf das Datum zuzugreifen, bevor es überhaupt gesetzt wurde.

2.4 Blockadefreie Lesezugriffe

Bei dieser Optimierung wartet das Programm nicht auf das Ergebnis einer Leseoperation. Weitere Operationen können bereits ausgeführt werden. Dadurch ist es möglich, das sich die Lesezugriffe eines Prozessors überlappen, das ihre Reihenfolge nicht eingehalten wird.

Wie im vorangegangenen Beispiel, können hier Vereltzungen des Konsistenzmodells im Zusammenhang mit der Verwendung von Synchronisationsvariabeln vorkommen. Ein Prozessor könnte versuchen das Datum zu lesen, bevor er die Freigabe überprüft hat. Dabei kann es dann passieren, das zwischen den beiden Leseoperationen von einem anderen Prozessor das Datum geändert und eine Freigabe erteilt wird. Der lesende Prozessor würde also die Datn verarbeiten, da er eine Freigabe erhalten hat. Jedoch würde er ein veraltetes Datum benutzen. Die sequentielle Reihenfolge wird hier also nicht gewährleistet.

Kapitel 3

Sequentielle Konsistenz und Caches

Den Themen Caches und Cache Kohärenz waren eigene Seminararbeiten zugeordnet:

- [Seminararbeit Chr. Ersfeld, Vortrag]
- [Seminararbeit R.Leisen, Vortrag]

Daher wird hier nicht weiter auf diese Themen eingegangen, sondern nur die Wechselwirkungen mit dem sequentiellen Konsistenzmodell betrachtet.

Alle bisherigen Aussagen zum sequentiellen Konsistenzmodell gingen von der vereinfachten Ausnahmen eines Mehrprozessorsystems ohne Caches und Datenreplikation aus. In der Realität sind diese natürlich nicht wegzudenken. Das wiederum hat Auswirkungen auf die Speicherkonsistenz, ähnlich wie in den bereits vorgestellten Kapiteln. Um weiterhin den Anschein eines sequentiellen Programmablaufs zu gewährleisten sind bei der Implementation eines Mehrprozessorsystems einige Dinge zu beachten, die im folgenden kurz vorgestellt werden sollen.

3.1 Sequentielle Konsistenz und Cache Kohärenz

Es gibt verschiedene Definitionen für Cache-Kohärenz. Einige davon sind so stark, das sie sequentielle Konsistenz erfüllen können. Dies lässt sich aber nicht allgemein für alle Cache Kohärenzprotokolle behaupten. Allgemeingültig für Kohärenzprotokolle sind die beiden folgenden Bedingungen:

- Schreibzugriffe sind für alle Prozessoren sichtbar
- verschiedene Schreibzugriffe auf die selbe Speicheradresse erscheinen für alle Prozessoren in derselben Reihenfolge

Die letztere Bedingung ist so nicht ausreichend um sequentielle Konsistenz zu gewährleisten. Sequentielle Konsistenz würde vorraussetzen, das *alle* Schreibzugriffe für alle Prozessoren in derselben Reihenfolge erscheinen. Ferner macht ein Kohärenzprotokoll noch keine Zusagen über die Ablaufreihenfolge der Operationen eines einzelnen Prozessors (die für sequentielle Konsistenz in der vom Programm vorgegebenen Reihenfolge erscheinen müssen).

3.2 Erkennung beendeter Schreibzugriffe

Um die Ablaufreihenfolge eines Prozessors zu gewährleisten, muss dieser wissen, wann eine Schreiboperation beendet ist. Eine Art von Bestätigung kann dabei gesendet werden, sobald der Wert im Speichermodul eingetragen wurde. In einer Architektur mit Caches ist die Schreiboperation aber erst endgültig abgeschlossen, wenn auch alle Kopien eines Speichereintrages aktualisiert oder invalidiert

wurden. Um also das Ende einer Schreiboperation feststellen zu können, muss die erfolgreiche Aktualisierung/Invalidierung der Caches kommuniziert werden. Die Bestätigungen müssen eingesammelt und dem schreibenden Prozesoor mitgeteilt werden.

3.3 Scheinbare Atomarität von Schreibzugriffen

Mit sequentieller Konsistenz wird verlangt das das Ergebnis einer Schreiboperation erst gelesen werden kann, wenn sie abgeschlossen ist. Verwendet man Caches, so kann die Schreiboperation erst als abgeschlossen gelten, wenn auch alle Kopien entsprechend invalidiert oder aktualisiert wurden. Das alles gehört zu ein und derselben Schreiboperation, die als atomar erscheinen muss. Während einer Schreiboperation dürfen keine anderen Operationen geschehen. Das Problem ist, ähnlich wie im vorangegangenen Abschnitt, das verschiedene Schreib- und Leseoperationen nicht miteinander intervenieren dürfen, da sie sonst die Schreiboperationen verfälschen könnten, bzw. ein ungültiges Ergegnis liefern könnten.

Kapitel 4

Modelle mit gelockerter sequentieller Konsistenz

4.1 Allgemein

Für das Modell der sequentiellen Konsistenz existieren eine Menge von Vorschlägen, wie man das Modell auflockern kann. Man verspricht sich davon Performancesteigerungen indem Optimierungen möglich werden, die eigentlich das Konsistenzmodell verletzen. Nichtsdestotrotz muss die Konsistenz an sich gewahrt bleiben. Das Konsistenzmodell wird in mehereren Stufen an die Optimierungsmöglichkeiten von Einzelprozessorsystemen angepasst.

Im folgenden werden einige Lockerungen besprochen, die sich in Hardware implementieren lassen. Diese Lockerungen lassen sich nach zwei Kategorien beurteilen:

1. Lockerungen der Atomarität von Schreiboperationen
2. Lockerungen bezüglich der Ablaufreihenfolge eines Programmes

In diesen beiden Kategorien existieren wiederum verschiedene Möglichkeiten zur Lockerung. Für gewöhnlich werden diese in Kombination miteinander implementiert.

4.2 Gelockerte Atomarität des Schreibzugriffs

Wie bereits in 3.3 beschrieben, bedeutet die Atomarität des Schreibzugriffes, das das Ergebnis einer Schreiboperation erst gelesen werden kann, wenn diese Schreiboperation bereits abgeschlossen ist. In einer Architektur mit Caches gehört zu einer Schreiboperation aber mehr als das einfache ändern eines Wertes. Die Schreiboperation gilt erst als abgeschlossen, wenn einerseits der Wert tatsächlich geändert wurde und andererseits alle möglicherweise vorhandenen Kopien (Caches) dieses Wertes invalidiert oder aktualisiert wurden. Deshalb ist es möglich, Lockerungen vorzunehmen, die den Schreibzugriff nicht mehr als atomar betrachten.

Es gibt zwei Varianten, wie man die Atomarität der Schreibzugriffe auflockert:

4.2.1 Früher Zugriff auf eigene Schreibaktion

Hierbei ist es dem Prozessor möglich das Ergebnis seiner eigenen Schreibzugriffe zu lesen, bevor sie abgeschlossen und systemweit bekannt gemacht wurden.

4.2.2 Früher Zugriff auf fremde Schreibaktion

Hierbei ist es dem Prozessor möglich das Ergebnis von Schreiboperationen anderer Prozessoren zu lesen, bevor diese abgeschlossen und systemweit bekannt gemacht wurden.

4.3 Gelockerte Reihenfolge der Speicherzugriffe

Diese Lockerungen beziehen sich auf Paare von Speicheroperationen. Die Speicheroperationen müssen sich dabei auf unterschiedliche Adressen beziehen. Mit Implementierung dieser Lockerungen werden die Hardwareoptimierungen aus Kapitel 2 ermöglicht.

Für folgende Kombinationen existieren Möglichkeiten die sequentielle Reihenfolge des Programmablaufs aufzulockern:

- Schreibzugriff gefolgt von Lesezugriff
- Schreibzugriff gefolgt von Schreibzugriff
- Lesezugriff gefolgt von beliebigem Zugriff

Bei jedem Verfahren werden üblicherweise die vorher genannten Verfahren mit eingeschlossen.

4.3.1 Gelockerte Schreib-Lese -zugriffe

Schreibzugriffe dauern für gewöhnlich deutlich länger als Lesezugriffe. Um diese Latenz zu umgehen, erlaubt man es, die Reihenfolge von auf Schreiboperationen folgende Leseoperationen anzupassen. Es wird Leseoperationen ermöglicht, das Ergebnis der Schreiboperation eines beliebigen Prozessors zurükzugeben, bevor diese Schfreiboperation abgeschlossen wurde. Er „sieht“ die Schreiboperation also, bevor die anderen Prozessoren sie beobachten können. Dies verletzt die Bedingung des Konsistenzmodells, kann auf Hardwareebene aber eine deutliche Beschleunigung bringen.

4.3.2 Gelockerte Schreib-Schreib -zugriffe

Schreibt ein Prozessor mehrfach hintereinander auf unterschiedliche Speicheradressen, so ist es ohne Bedeutung (was die Schreiboperationen angeht), in welcher Reihenfolge diese Operationen tatsächlich ausgeführt werden. Daher wird es erlaubt, das solche Schreibzugriffe einander überlappen. Auch die Reihenfolge muss nicht eingehalten werden. In diesem Punkt wird die Bedingung des Konsistenzmodells einer festen Ablaufreihenfolge verletzt.

4.3.3 Gelockerte Lese-Lese und Lese-Schreib -zugriffe

Mit diesem Verfahren wird es nun zusätzlich zulässig, die Ablaufreihenfolge in Anschluss an eine Leseoperation zu verletzen. Die Operationen dürfen sich überlappen oder umsortiert werden. Leseoperationen blockieren dadurch nicht mehr.

Wie bereits erwähnt schliesst dieses Verfahren die anderen mit ein. Da das Konsistenzmodell somit sehr flexibel bezüglich der Ablaufreihenfolge geworden ist, ist es nunmehr auch sinnvoll geworden die Optimierungen von einem Compiler vornehmen zu lassen. Dieser benötigt maximale Flexibilität, da die Speicheroperationen oftmals sehr stark mit einenader verflochten sind. Die anderen Konsistenzmodelle wäre für eine Compileroptimierung nicht ausreichend.

4.4 Sicherheits-Modelle

Es gibt verschiedene Implementierungen der erwähnten gelockerten Konsistenzmodelle. Diese beziehen unterschiedliche Auffangverfahren mit ein, um die Einhaltung des Konsistenzmodelles zu gewährleisten. Dazu gehören beispielsweise:

- Weak Ordering
- Release Consistency
- Alpha, RMO, PowerPC

Auf die konkreten Implementierungen soll hier aber nicht weiter eingegangen werden. Die verschiedenen Lockerungen des sequentiellen Konsistenzmodells und ihre Funktionsweisen wurden bereit erläutert. Für tiefergehende Informationen zu den Sicherheitsmodellen der konkreten Implementierungen sei auf [Tutorial,] verwiesen.

Literaturverzeichnis

[Langenscheidts Fremdwörterbuch] „Langenscheidts Fremdwörterbuch“, www.langenscheidt.aol.de

[Seminararbeit Chr. Ersfeld] Ersfeld, C.: Seminararbeit „Organisation von Caches“

[Seminararbeit R.Leisen] Leisen, R.: Seminararbeit „Cache Coherence“

[Seminararbeit Th. Wild] Wild, T.: Seminararbeit „Relaxed Memory Consistency (Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems)“

[Tanenbaum] Tanenbaum, A.S., Goodman, J.: „Computerarchitektur“, Pearson Studium, 2001

[Tutorial] Adve, S.V., Gharachorloo K.: „Shared Memory Consistency Models: A Tutorial“, September 1995