Threaded Code in der Architektursimulation

Gabriel Kittel
e9027972@student.tuwien.ac.at

2007-01-07

Diese Seminararbeit – eine Zusammenfassung von [3], [5] und [8] – bietet eine Einführung in das Konzept des direkten und indirekten Threaded Code, der schnelleren und mitunter kleineren Code für Interpreter und virtuelle Maschinen ermöglicht. Weiters wird der Architektursimulator SimICS vorgestellt, der auf einem Threaded-Code-Interpreter basiert.

1 Einleitung

In dieser Arbeit wird das Konzept eines Threaded-Code-Rechners [3] vorgestellt. Threaded-Code-Rechner führen ein Programm aus, das aus einer Liste von Sprungadressen zu Unterprogrammen besteht, wobei am Ende des Unterprogramms ein Sprung zum nächsten Unterprogramm – zum Beispiel mittels eines indirekten Sprungbefehls – erfolgt. Dieses Design ermöglicht gegenüber dem maschineneigenen Befehlssatz eine größere Flexibilität und verglichen mit interpretiertem Code eine schnellere Laufzeit. Meistens sind Verbesserungen sowohl bei der Codegröße als auch bei der Geschwindigkeit möglich.

Im Falle von direktem Threaded Code zeigen die obengenannten Sprungadressen direkt auf eine Serviceroutine, bei indirektem Threaded Code [5] auf eine Speicherstelle mit der Adresse einer Serviceroutine. Letzteres Design ermöglicht eine größere Plattformunabhängigkeit.

Threaded Code wird unter anderem als Interpretiertechnik [7] der Programmiersprachen FORTH [10], Smalltalk [4], Scheme [1] und Java [6] angewandt. Auch die erste Implementierung der Programmiersprache B – ein Vorläufer von C – generierte Threaded Code. [11]

Diese Arbeit stellt nach einer Erläuterung des Konzepts des Threaded Code den Architektursimulator SimICS [8] vor, dessen Kern ein Threaded-Code-Interpreter ist. Zuvor wird eine Einführung in die Architektursimulation gegeben.

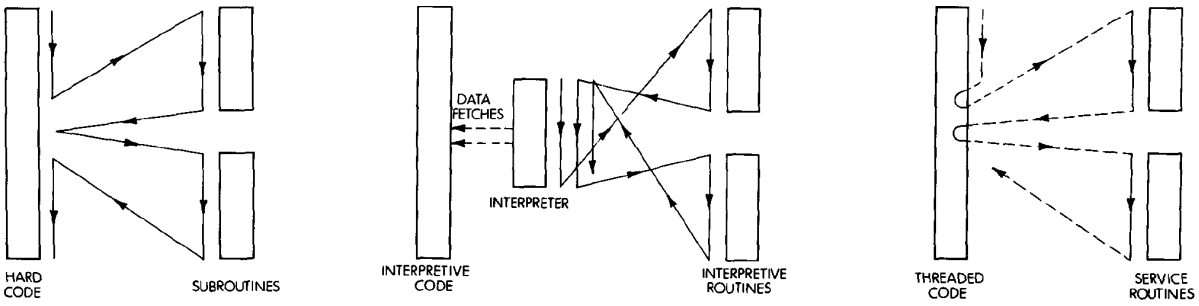

Abbildung 1: Kontrollfluss bei maschineneigenem Code, interpretiertem Code und Threaded Code

2 Threaded Code

2.1 Direkter Threaded Code

Grundsätzlich kann ein Programm mittels maschineneigenem Code *(hard code)* oder interpretiertem Code *(interpretive code)* implementiert werden. Erstere Variante führt zu guter Performance, jedoch zu hoher Codegröße, da der maschineneigene Befehlssatz nicht für den Anwendungsbereich optimiert ist. Bei interpretiertem Code kann durch Wahl einer geeigneten Programmiersprache ein Befehlssatz verwendet werden, der gut an die Problemstellung angepasst ist. Daraus resultiert kleiner Code, der jedoch interpretiert werden muss und daher langsam ausgeführt wird.

Ein Threaded Code Computer stellt demgegenüber das Programm als eine Liste von Adressen von Serviceroutinen dar. Er arbeitet nach folgendem Algorithmus:

1. Hole S, das Speicherwort, auf das der Programmcounter (PC) derzeit zeigt.
2. (a) Führe das Unterprogramm aus, das auf S beginnt
   (b) Inkrementiere S
3. weiter mit Schritt 1

Dieser Algorithmus kann implementiert werden, indem am Ende jedes Unterprogramms ein indirekter Sprungbefehl mit Inkrement erfolgt. Auf einer PDP-11 lautet der entsprechende Befehl `JMP @(R)+`, wobei R das allgemeine Register ist, das dem Programmcounter (PC) des Threaded-Code-Computers entspricht.

Der Kontrollfluss von maschineneigenem Code, interpretiertem Code und Threaded Code wird in Abbildung 1 dargestellt.

Im folgenden Abschnitt wird das hier vorgestellte Konzept des (direkten) Threaded Code erweitert.

2.2 Indirekter Threaded Code

Bei indirektem Threaded Code wird das im vorigen Abschnitt beschriebene Konzept verändert, indem die Sprungadressen des Programms nicht direkt auf eine Serviceroutine

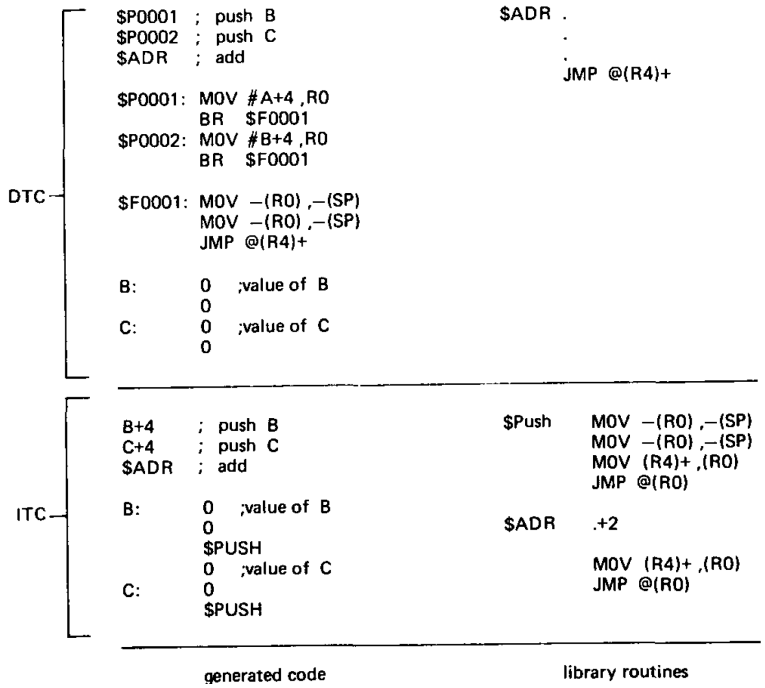

Abbildung 2: `return B + C` als direkter und indirekter Threaded Code

verweisen, sondern auf die Adresse einer Serviceroutine. Bei Ausführung eines indirekten Threaded-Code-Programms muss somit zweimal dereferenziert werden. Dieses Konzept bietet gegenüber direktem Threaded Code eine höhere Maschinenunabhängigkeit und kleinere Programme aufgrund besserer Codewiederverwendung. Abbildung 2 stellt Implementierungen des Programmes `return B + C` als direkter und indirekter Threaded Code gegenüber.

3 Architektursimulation

Architektursimulatoren dienen dazu, eine neu zu entwickelnde Systemarchitektur auf einem bestehenden System zu simulieren. In [2] wird beispielsweise ein Motorola-88000-Simulator vorgestellt, der auf einem Motorola-68020-basierten System läuft.

Neben der Entwicklung einer neuen Systemarchitektur dienen Simulatoren auch der Entwicklung von Systemsoftware (z.B. Betriebssystemen) für diese Architektur. Architektursimulatoren bestehen neben einer virtuellen Maschine auch aus Mechanismen für die Simulation der Speicher- und Interruptverwaltung, auf die hier jedoch nicht näher eingegangen werden soll. Eine wichtige Problemstellung in der Disziplin der Architektursimulatoren ist das Erreichen einer akzeptablen Performance.

In [8] wird der Architektursimulator SimICS vorgestellt, dessen Kern ein Threaded-Code-Interpreter ist. Als Befehlssatzsimulator übersetzt er einen Gastbefehl in mehrere Hostbefehle, d.h. eine Zusammenfassung von Gastbefehlen findet nicht statt. SimICS

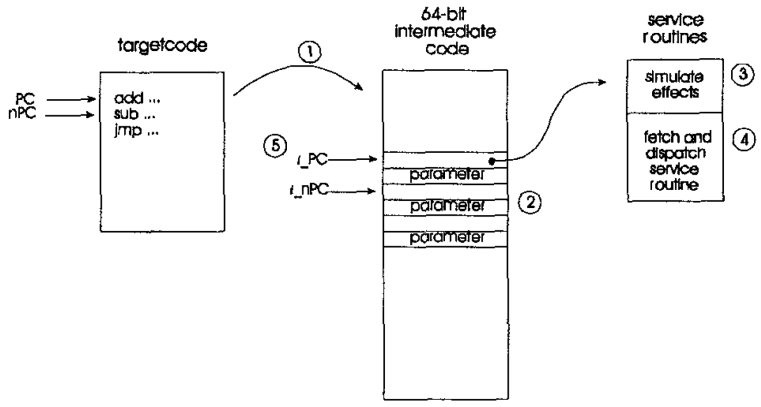

Abbildung 3: Arbeitsweise des Threaded-Code-Interpreters von SimICS

ist ein Architektursimulator der zweiten Generation [9], der zwecks Verbesserung der Performance bereits übersetzte Instruktionen in einem Cache zwischenspeichert und wiederverwendet.

Die Arbeitsweise des Threaded-Code-Interpreters von SimICS wird in Abbildung 3 dargestellt.

4 Zusammenfassung

Diese Seminararbeit hat das Konzept des Threaded-Code-Computers erläutert und den Unterschied zwischen direktem und indirektem Code erklärt. Nach einem Überblick über verschiedene Anwendungen von Threaded Code wurde eine dieser Anwendungen, der Architektursimulator SimICS vorgestellt.

Literatur

[1] Bartley, David H. und John C. Jensen: *The Implementation of PC Scheme.* In: Gabriel, Richard P. (Herausgeber): *Proceedings of the ACM Conference on LISP and Functional Programming*, Seiten 86–93, Cambridge, MA, August 1986. ACM Press.

[2] Bedichek, R.: *Some Efficient Architecture Simulation Techniques.* In: *Proceedings of the USENIX Winter 1990 Technical Conference*, Seiten 53–64, Berkeley, CA, 1990. USENIX Association.

[3] Bell, James R.: *Threaded Code.* Communications of the ACM, 16(6):370–372, Juni 1973.

[4] Deutsch, L. Peter und Allan M. Schiffman: *Efficient implementation of the smalltalk-80 system.* In: *POPL '84: Proceedings of the 11th ACM SIGACT-*

SIGPLAN symposium on Principles of programming languages, Seiten 297–302, New York, NY, USA, 1984. ACM Press.

[5] Dewar, Robert B. K.: *Indirect Threaded Code.* Communications of the ACM, 18(6):330–331, Juni 1975.

[6] Gregg, David, M. Anton Ertl und Andreas Krall: *Implementing an Efficient Java Interpreter.* In: *HPCN Europe 2001: Proceedings of the 9th International Conference on High-Performance Computing and Networking*, Seiten 613–620, London, UK, 2001. Springer-Verlag.

[7] Klint, Paul: *Interpretation Techniques.* Software—Practice and Experience, 11(9):963–973, September 1981.

[8] Magnusson, Peter S.: *Efficient instruction cache simulation and execution profiling with a threaded-code interpreter.* In: *WSC '97: Proceedings of the 29th conference on Winter simulation*, Seiten 1093–1100, 1997.

[9] May, C.: *Mimic: a fast system/370 simulator.* In: *PLDI*, Seiten 1–13. ACM, 1987.

[10] Rather, Elizabeth D., Donald R. Colburn und Charles H. Moore: *The evolution of Forth.* In: *HOPL-II: The second ACM SIGPLAN conference on History of programming languages*, Seiten 177–199, New York, NY, USA, 1993. ACM Press.

[11] Ritchie, Denis M.: *The Development of the C Language.* In: Wexelblat, Richard L. (Herausgeber): *Proceedings of the Conference on History of Programming Languages*, Band 28(3) der Reihe *ACM Sigplan Notices*, Seiten 201–208, New York, NY, USA, April 1993. ACM Press.