TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Erschließung von Just-in-Time-Compilierungstechniken in der Realisierung eines retargierbaren Architektursimulators

Vortrag zum Diplom

Marco Kaufmann
s9186072@mail.inf.tu-dresden.de

Dresden, 29.07.2009

Gliederung

1 Einführung

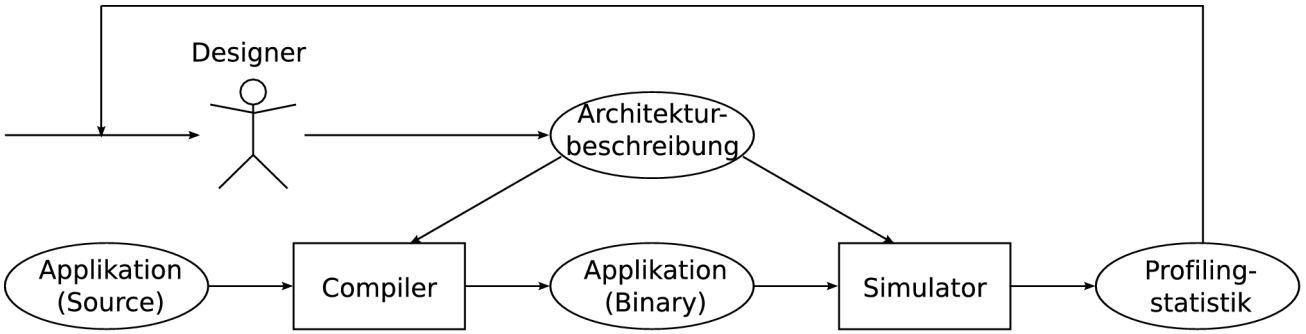
Aufgabenstellung

- Literaturstudium zu retargierbaren Architektursimulatoren und ihrer Beschreibungssprachen unter Berücksichtigung von JIT-Techniken
- Abgrenzung des Simulators und der von ihm unterstützten Techniken
- Entwurf des Simulators und Spezifikation der notwendigen Änderungen an TADL
- Implementierung des entworfenen Simulators
- Nachweis der Funktion und Bewertung der Implementierung
- Dokumentation des Entwurfs und der erzielten Ergebnisse

1 Einführung

Zweck der Simulation

- Validierung von Architekturdesigns
- Vorab Testen von Compiler / Software
- Evaluierung von Entwurfsentscheidungen



Entwurfszyklus nach [8]

Mächtiges Werkzeug für Architekturentwurf, DSE und Hardware / Software Codesign

1 Einführung

Anforderungen an Simulatoren

- Hohe Simulationsgeschwindigkeit
- Flexibilität
- Retargierbarkeit
- Portierbarkeit

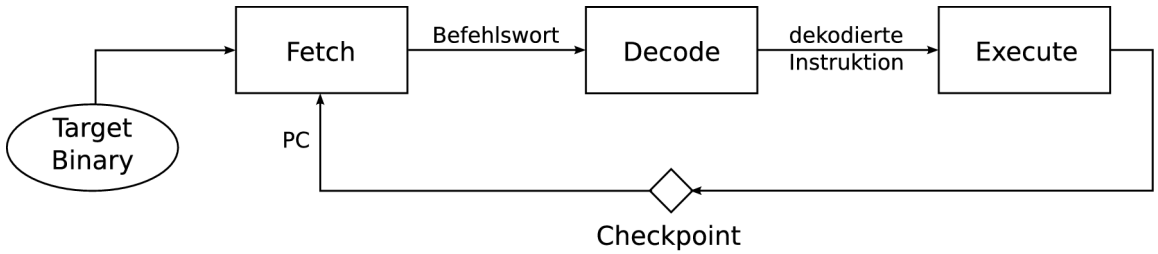
Im Unterschied zu Emulatoren:

- Beobachtbarkeit
- Grenze zwischen Simulation und Emulation verläuft jedoch unscharf

2 Simulationstechniken

Interpretierende Simulation

- Flexibelster, aber langsamster Ansatz
- Beispiel: DITO / TADL



Die meiste Zeit wird mit dem Holen und Dekodieren von Befehlen verbracht, anstatt der eigentlichen Befehlsausführung!

2 Simulationstechniken

Statisch Compilierende Simulation

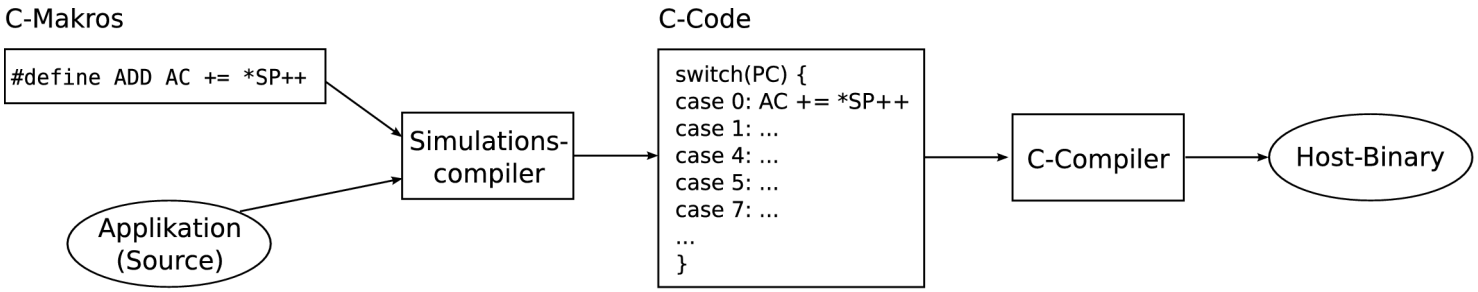
Idee:

- Simulationscompiler
- Möglichst viel Overhead von Laufzeit auf Compilierzeit verlagern
- Befehle vorab holen und dekodieren
- Zur Laufzeit lediglich Befehlsausführung

Sehr schnell, aber unflexibel!

2 Simulationstechniken

Statisch Compilierende Simulation mittels C-Makros



- Einzelne Instruktionen durch C-Makros repräsentiert
- Simulationscompiler generiert C-Code
- Optimierter Simulator durch C-Compiler
- Nicht takt- und pipelinegenau

2 Simulationstechniken

Dynamisch Compilierende Simulation

Intention:

- Flexibilität interpretierender mit Performance compilierender Simulation vereinen

Idee:

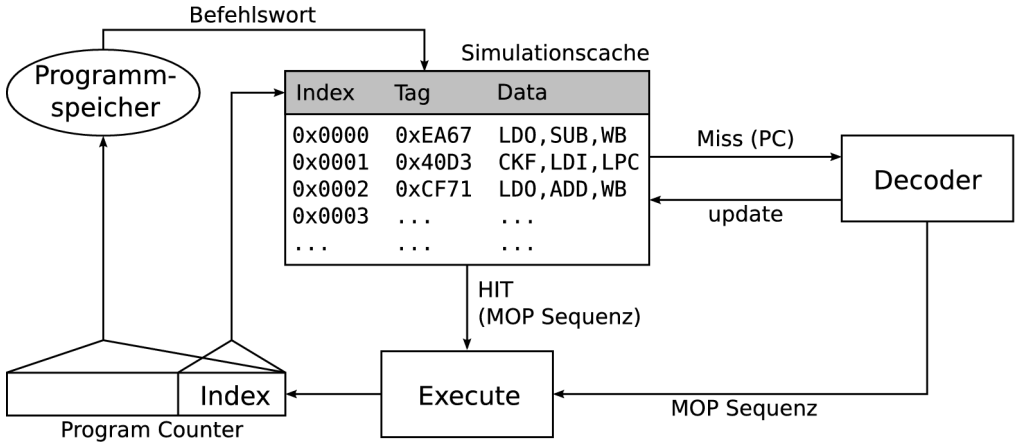
- Integration von Simulationscompiler und Simulator
- JIT-Compiler: Holen, Dekodieren und Übersetzen von Instruktionen Just-in-Time vor der ersten Ausführung

Schnell und flexibel, aber schlecht portierbar!

2 Simulationstechniken

Dynamisch Compilierende Simulation - JIT-CCS

Simulationscache für bereits dekodierte Instruktionen

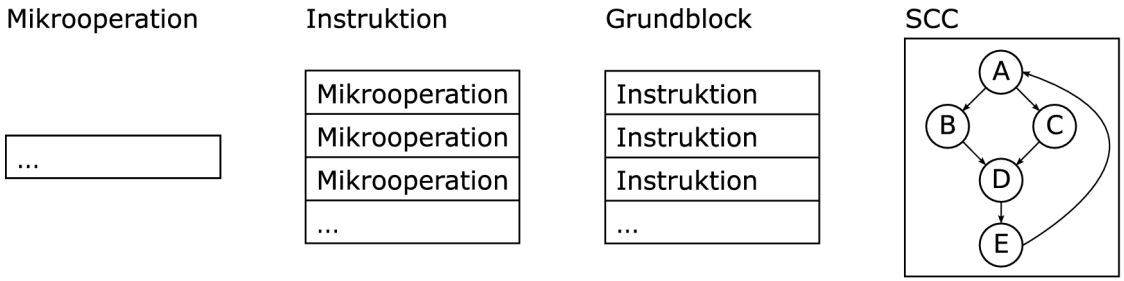


Simuliert takt- und pipelinegenau.

2 Simulationstechniken

Dynamisch Compilierende Simulation - LTU

- Bisher: TU = einzelner Befehl oder Mikrooperation
- Vorteil größerer Übersetzungseinheiten (LTU):
  - ➔ Weniger Zeit in Simulationsschleife
  - ➔ Größerer Bereich für Optimierung durch Compiler
- Beispiele: EHS, QEMU



Verringert jedoch die Beobachtbarkeit und geht somit bereits in Richtung Emulation!

3 Simulatorentwurf

Entwurfsziele

- Hochperformanter Simulator mit JIT-Eigenschaften
- Portierbarkeit
- Keine Einschränkung der Retargierbarkeit gegenüber DITO
- Flexibilität und Beobachtbarkeit dagegen nebensächliches Ziel
  - ➔ Keine pipelinegenaue Beschreibung und Simulation
  - ➔ Vorerst kein selbstmodifizierender Code
  - ➔ Dynamisch nachgeladener Code nur mit Einschränkungen möglich

3 Simulatorentwurf

Java als Quasi-Hostplattform

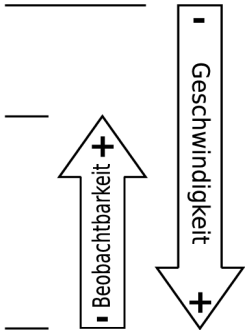
Schlechte Portierbarkeit von JIT-Simulatoren. Lösung:

- Verwendung der JRE als Quasi-Host!
  - ➔ JIT-Simulationscompiler generiert Java-Bytecode
  - ➔ Transformation in nativen Code für Host durch JVM
  - ➔ Backend somit für Vielzahl von Plattformen schon verfügbar
- Weiterer Vorteil: Optimierer bereits „gratis“ in JVM enthalten
  - ➔ Simulationscompiler kann auf eigenen Optimierer verzichten!
- Nachteil: eigene Java-Klasse für jede Translated Function
  - ➔ Neuladen und Entladen von Klassen in Java nur umständlich möglich

3 Simulatorentwurf

Simulationsmodi

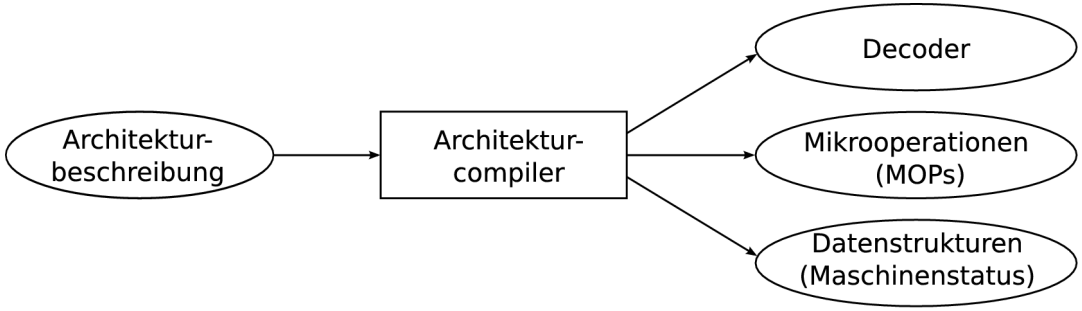
- Interpretermodus
- Precompiled-Interpretermodus
- Pseudo-JIT – Modus (P-JIT)
- Single-Instruction-JIT – Modus (SI-JIT)
- Basic-Block-JIT – Modus (BB-JIT)



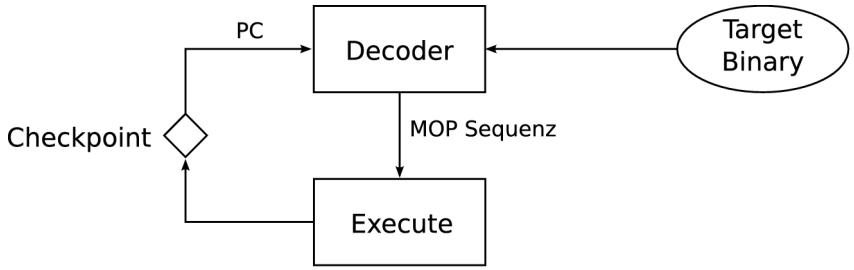
- Sinvoll zur Anwendung bei der Simulation: JIT-Modi
- Interpretermodi zu Test- und Vergleichszwecken

3 Simulatorentwurf

Compilierzeit

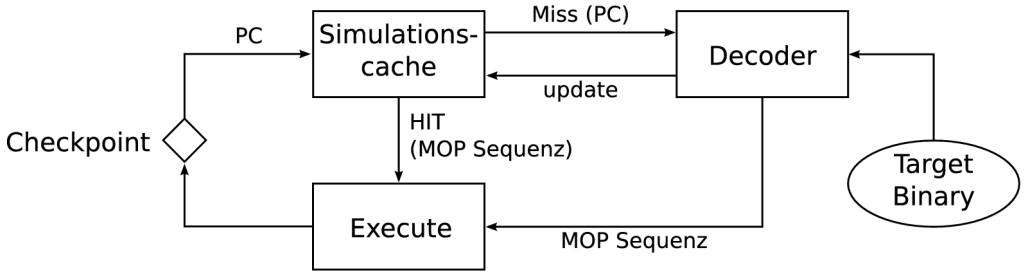


Laufzeit – (Precompiled)-Interpretermodus

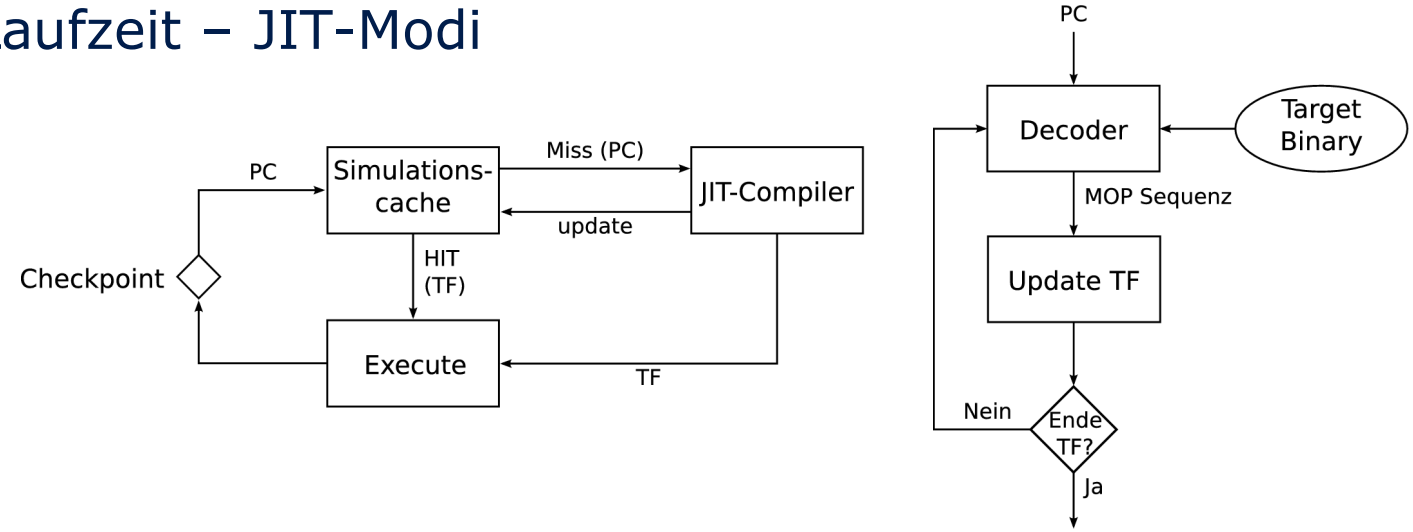


3 Simulatorentwurf

Laufzeit – Pseudo-JIT-Modus



Laufzeit – JIT-Modi



4 Architekturbeschreibung

Notwendige Modifikationen an TADL

- Verzicht auf Pipelinebeschreibung
  - ➔ Kein Zeitverhalten an Ressourcen (Stalls, Ports, Verzögerungen, Requests)
- Neues Dekoder- und Befehlsbeschreibungsmodell
  - ➔ Strikte Trennung von Dekoder – und Befehlsbeschreibung
  - ➔ Verhaltensbeschreibung für Dekoder als Bestandteil der Architekturbeschreibung
  - ➔ Dekoder generiert Sequenz von Mikrooperationen
    - Keine ganzen Instruktionen!
    - Keine Befehlshierarchie!
- Typsystem (`bool`, `int`, `long`, `big`)

5 Quellen

[1] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[2] Daniel Jones and Nigel Topham. High speed cpu simulation using ltu dynamic binary translation. In *HiPEAC '09: Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, pages 50–64, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] C. Mills, S. C. Ahalt, and J. Fowler. Compiled instruction set simulation. *Software, Practice and Experience*, 21(8):877–889, 1991.

[4] Achim Nohl, Gunnar Braun, Oliver Schliebusch, Rainer Leupers, Heinrich Meyr, and Andreas Hoffmann. A universal technique for fast and flexible instructionset architecture simulation. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pages 22–27, New York, NY, USA, 2002. ACM.

5 Quellen

[5] Mehrdad Reshadi, Prabhat Mishra, and Nikil Dutt. Instruction set compiled simulation: a technique for fast and flexible instruction set simulation. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 758–763, New York, NY, USA, 2003. ACM.

[6] Mehrdad Reshadi, Prabhat Mishra, and Nikil Dutt. Hybrid-compiled simulation: An efficient technique for instruction-set architecture simulation. *ACM Trans. Embed. Comput. Syst.*, 8(3):1–27, 2009.

[7] Jianwen Zhu and Daniel D. Gajski. A retargetable, ultra-fast instruction set simulator. In *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, page 62, New York, NY, USA, 1999. ACM.

[8] Thomas B. Preußer: Entwicklung eines Prozessorsimulators unter besonderer Berücksichtigung des Organic Computing. Diplomarbeit, *Technische Universität Dresden*, 2003.