TECHNISCHE UNIVERSITÄT DRESDEN

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

Untersuchungen zur Trace-basierten iterativen Rekonstruktion von Registerinhalten durch Simulation

Belegverteidigung

Dresden, 19.07.2011

Robert Ramm

s9363979@mail.zih.tu-dresden.de

Aufgabenstellung

1. Literaturstudium zu Trace-Analyse, verfügbaren Trace-Daten und –Formaten, sowie Programm- und Datenflussanalyse aus Sicht des Compilers
2. Ermittlung von typischen dynamischen Registerzugriffsprofilen anhand repräsentativer, selbst gewählter Beispiele
3. Konzeption und Implementierung eines Algorithmus zu iterativen Belegung von Registerlebenszyklen in Abhängigkeit der Belegung der Quelllebenszyklen, Bereitstellung einer Simulatorschnittstelle
4. Untersuchung des Einflusses von inverser Ausführung einfacher arithmetischer Operationen und Adressrechnung auf den Gesamtrekonstruktionsgrad, insbesondere bezüglich langer Registerlebenszyklen
5. Darstellung des Rekonstruktionsgrades und Aufwandes (Iterationszyklen) in Abhängigkeit verschiedener Registerzugriffsprofile anhand der o.g. Testprogramme, Dokumentation der Ergebnisse.

Gliederung

1. Tracing
2. Motivation
3. verfügbare Tracedaten
4. Transformation der Tracedaten
5. Belegung der Lebenszyklen
6. Messergebnisse
7. Zusammenfassung

Tracing

- Aufzeichnung und Speicherung des Programmflusses (Speicheroperationen, Registerzugriffe und Verzweigungen)
- Unterscheidung zwischen vollständigen und unvollständigen Trace
- Integrierung zusätzlicher Hardware zur Traceerfassung notwendig
- Rekonstruktion des Programmablaufes bis auf Hochsprachenebene möglich

Motivation

- steigende Integrationsdichte eingebetteter Systeme
  → steigende Komplexität der Soft- und Hardware
  → steigender Aufwand für Fehlerbehebung und Verifikation

- zunehmende Verbreitung nebenläufiger und/oder zeitkritischer Anwendungen

- keine effektive Nutzung herkömmlicher Debugging-Techniken möglich

Flow Trace

- Aufzeichnung des gesamten Programmflusses inkl. aller Registertransaktionen und Speicheroperationen
  → sehr hohes Datenvolumen

- Speicherungen in externem Speicher (Mbyte - Gbyte)
  → schnelle Speicheranbindung notwendig
  → Datenraten bis zu ~1 Gbyte/s notwendig

- Echtzeitanalyse ermöglicht Triggerung der Aufzeichnung

On-Chip-Trace

- Problem: schnelle Schnittstellen sehr kostenintensiv (viele Pins + Logik)
- Lösung : lokale Speicherung der Daten
- Auslesen der Daten nach Programmbeendigung
- nur einfache Triggerung möglich
- On-Chip-RAMs in ihrer Größe beschränkt
- Speicherkapazität maßgeblich für Dauer der Aufzeichnung

Verfügbare Tracedaten

- Unterscheidung in Befehls- und Datentrace

- Befehlstrace =
  statische und dynamische Sprünge, sowie nicht ausgeführte Instruktionen

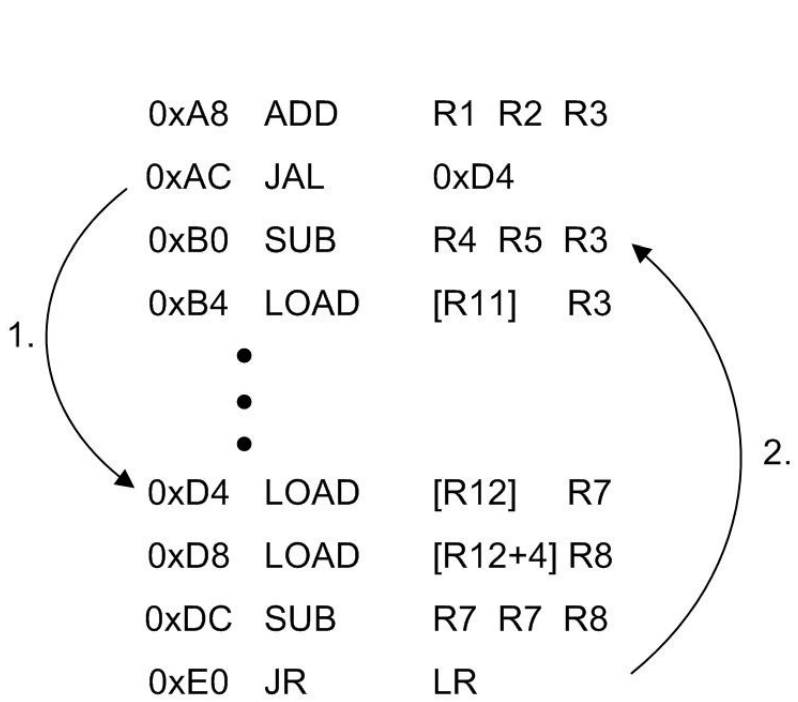
- Datentrace =
  lesende und schreibende Speicherzugriffe

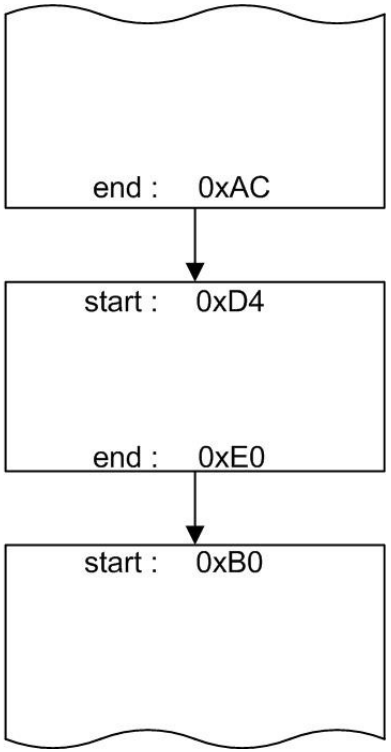
- zusammen ausreichend für vollständige Rekonstruktion, wenn beide vollständig
  → Was, wenn nicht vollständig?

Transformation der Tracedaten

- Zuweisung der Tracevents zu Befehlen im Quellcode
- Erstellung eines Kontrollflussgraphen aus dem Befehlstrace
- Knoten durch Basisblöcke dargestellt
- Basisblock : Sequenz von Instruktionen ohne Verzweigungen
- Kanten symbolisieren Sprünge und Verzweigungen im Befehlsfluss
- Zuordnung der Datentraceevents zu entsprechenden Basisblock
- Einfügen von Zeitstempeln → eine Zeiteinheit pro Befehl
  → Benötigt für Ermittlung der Registerlebenszyklen

Beispiel Kontrollflussgraph



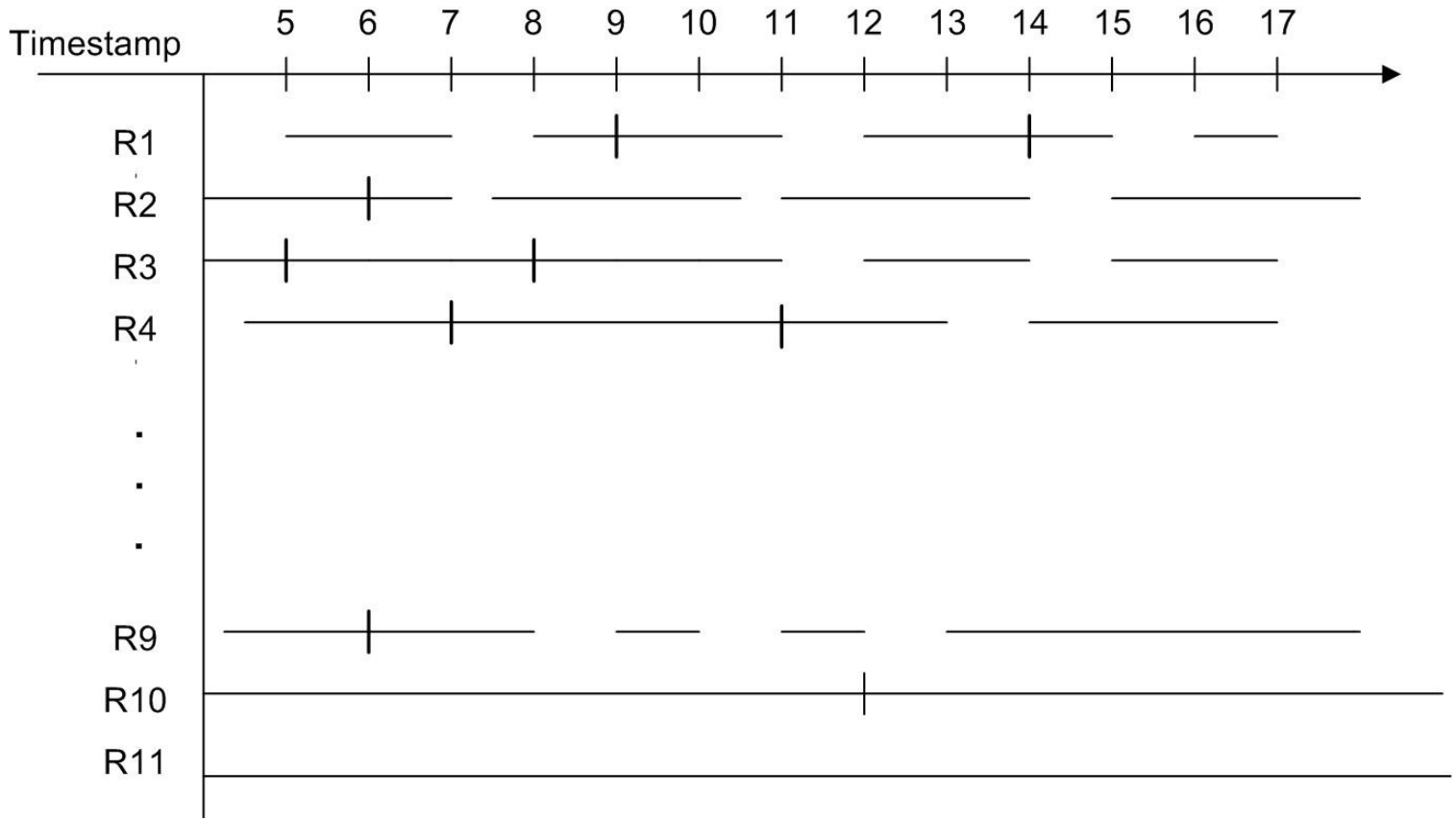


Registerlebenszyklen

„Ein Register *r* ist lebendig an einem Programmpunkt *p*, wenn es auf einem Programmpfad vom Eintrittsknoten der Prozedur nach *p* eine Setzung von *r* gibt und einen Pfad von *p* zu einer Benutzung von *r*, auf dem *r* nicht gesetzt wird. Die Lebensspanne eines Registers *r* ist die Menge der Programmpunkte, an denen *r* lebendig ist“

Wilhelm, R. ; Maurer, D.: Übersetzerbau, Theorie, Konstruktion, Generierung

Beispiel Registerlebenszyklen



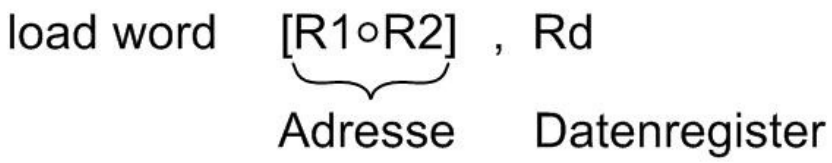
Belegung der Lebenszyklen

- Belegung der Inhalte in Abhängigkeit der Quellzyklen
  → wenn Quellregister bekannt, dann Zielregister ermittelbar
- Auswertung des Datentrace
- Adresse und Datum bekannt
- Datenregister bei Load/Store-Befehlen kann direkt belegt werden

Problem : **langlebige Registerinhalte, wie Stackpointer oder Schleifenzähler, so nur schwer ermittelbar**

Inverse Belegung

- Umkehrung einfacher arithmetischer und logischer Operationen (Addition, Subtraktion, Multiplikation, XOR , MOVE)
- bei Gleitkommaarithmetik nicht anwendbar
- Vorrausetzung : Zielregister bekannt, genau ein Quellregister unbekannt
- sehr gut anwendbar auf Adressrechnungen



Ermittlung des minimalen Belegungsgrades

- Betrachtung des Worst Case Szenario

1. Festlegung der Puffergröße
2. Festlegung des Startpunktes der „Traceaufzeichnung“
3. Bestimmung des damit möglichen Zeitintervalls
4. Rekonstruktion dieses Intervalls
5. Verschiebung des Startpunktes → Zurück zu 1.

Tracepuffermodell

Startpunkt :

Statusbits/Typ	PC	**Summe**
1 Byte	4 Bytes	**5 Bytes**

direkter Sprung :

Statusbits/Typ	PC	**Summe**
1 Byte	1 Byte	**2 Bytes**

indirekter Sprung :

Statusbits/Typ	Zieladresse	**Summe**
1 Byte	2 Bytes	**3 Bytes**

nicht ausgeführte Instr. :

Statusbits/Typ	PC	**Summe**
1 Byte	2 Bytes	**3 Bytes**

Load / Store :

Statusbits/Typ	Datum	Adresse	**Summe**
1 Byte	4 Bytes	2 Bytes	**7 Bytes**

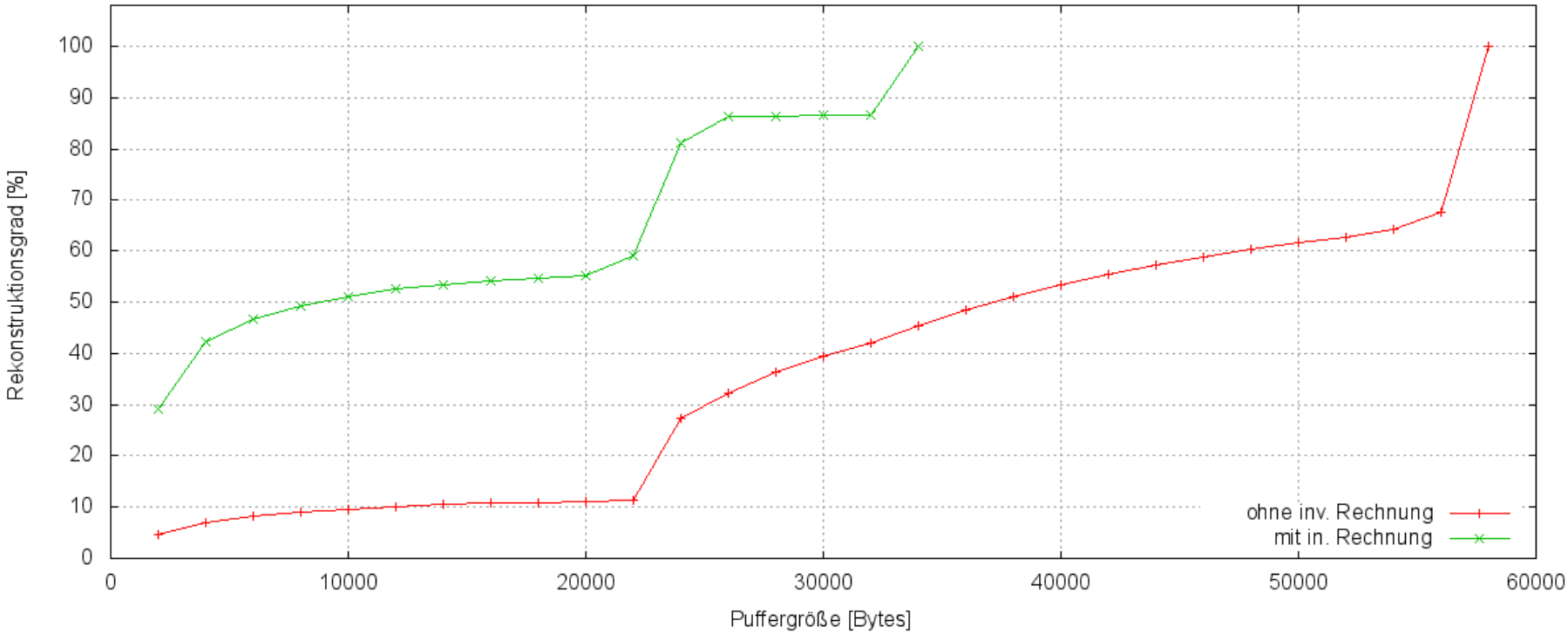
Testszenarien

- Nutzung der eembc-Benchmarksuite
  (Embedded Microprocessor Benchmark Consortium)

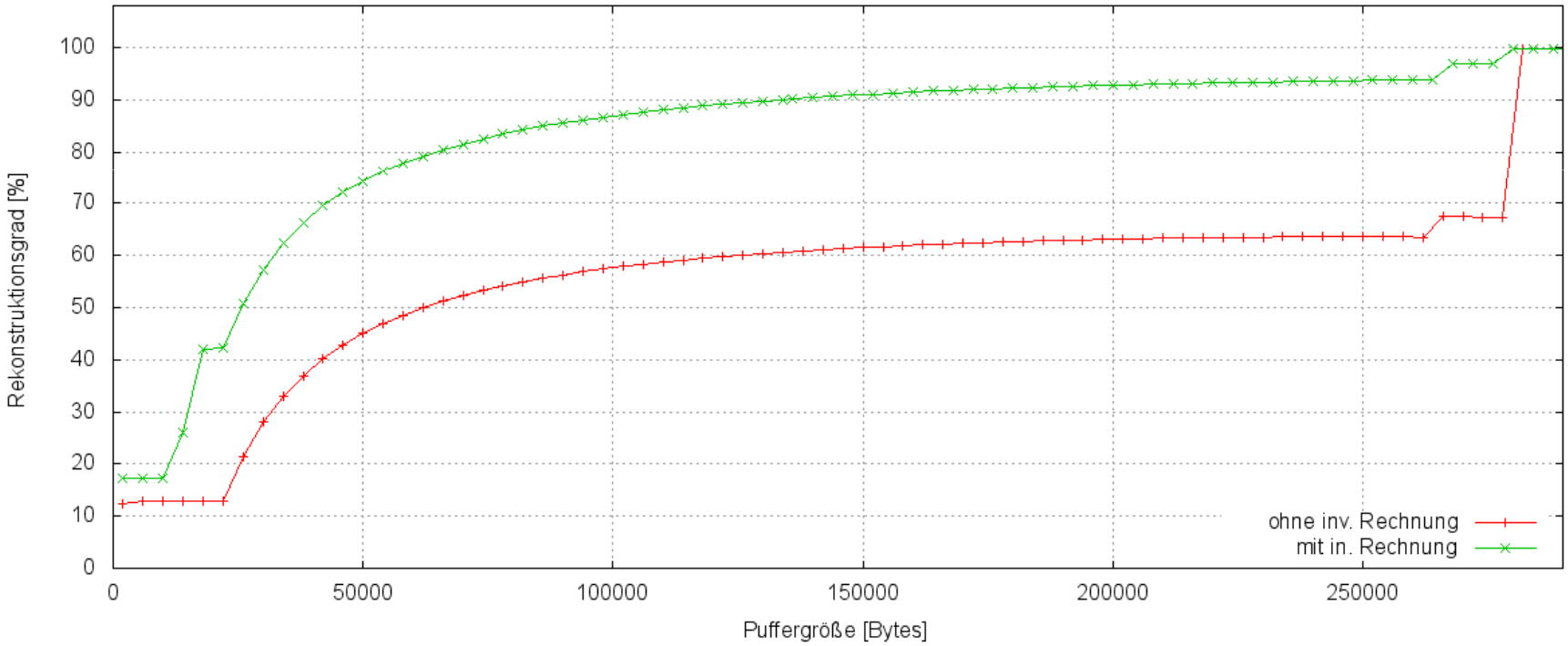
- Testsuite mit Schwerpunkt auf automotive Applikationen

  - Fast Fourier Transformation
  - Filteranwendungen
  - Floatingpointemulation

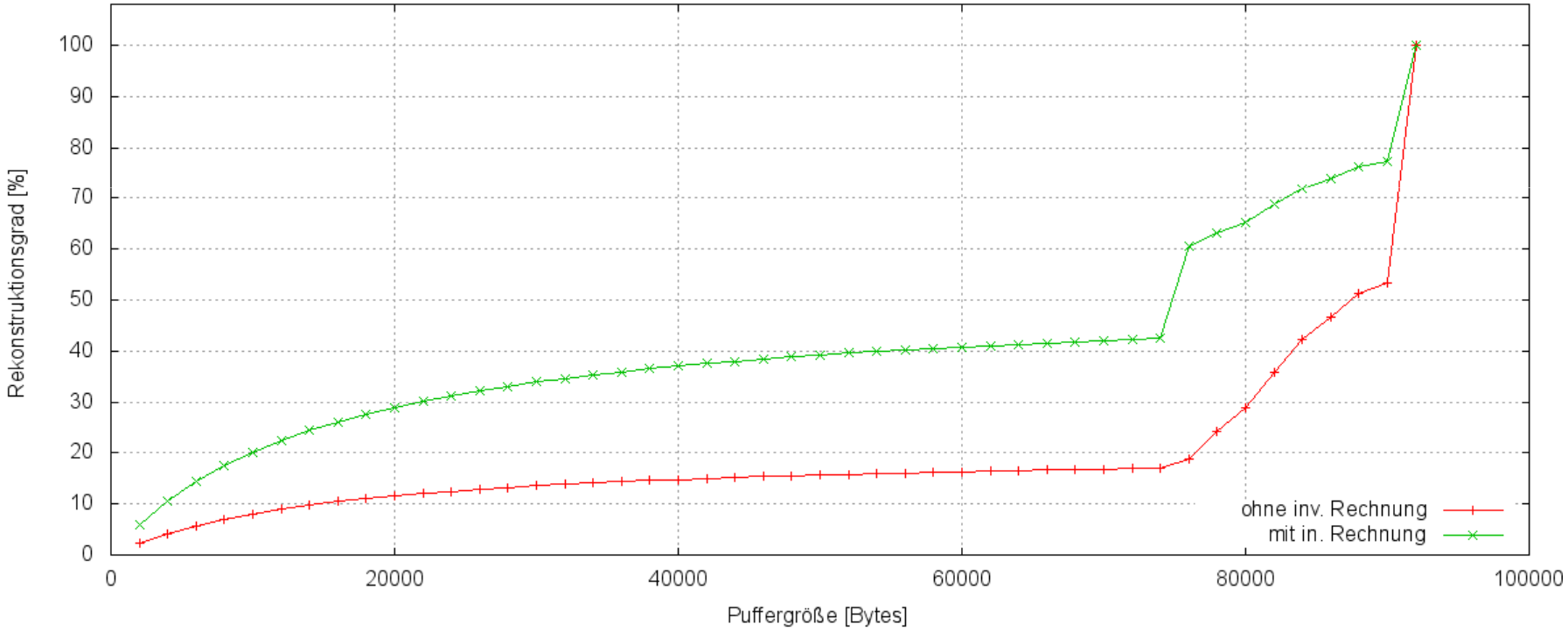
Finite Impuls Respons Filter



Fast Fourier Transformation



Floatingpoint Emulation



Auswertung

- Größe des benötigten Pufferspeicher stark applikationsabhängig
- höherer Rekonstruktionsgrad durch Nutzung inverser Rekonstruktion
  → Steigerung stark applikationsspezifisch
- Steigerung des Rekonstruktionsgrades durch „clevere“ Einstellung des Triggerzeitpunktes möglich

Quellenverzeichnis

Alex, Stefan: Entwurf und Implementierung einer parametrierbaren Trace-Hardware am Beispiel der Shap-Mikroarchitektur, Technische Universität Dresden

Berns, Karsten ; Schürmann, Bernd ; Trapp, Mario: Eingebettete Systeme, Systemgrundlagen und Entwicklung eingebetteter Software

Hopkins, Andrew B. ; McDonald-Maier, Klaus D.: Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores. In: IEEE Transactions on Computers 55 (2006), February

Leatherman, N. R.; S. R.; Stollon: An embedding debugging architecture for SOCs.

Menne, Torsten: Vergleich von CLP und ILP basierten Optimierunsstrategien am Beispiel der Codegenerierung für DSPs, Universität Dortmund, Fakultät Informatik

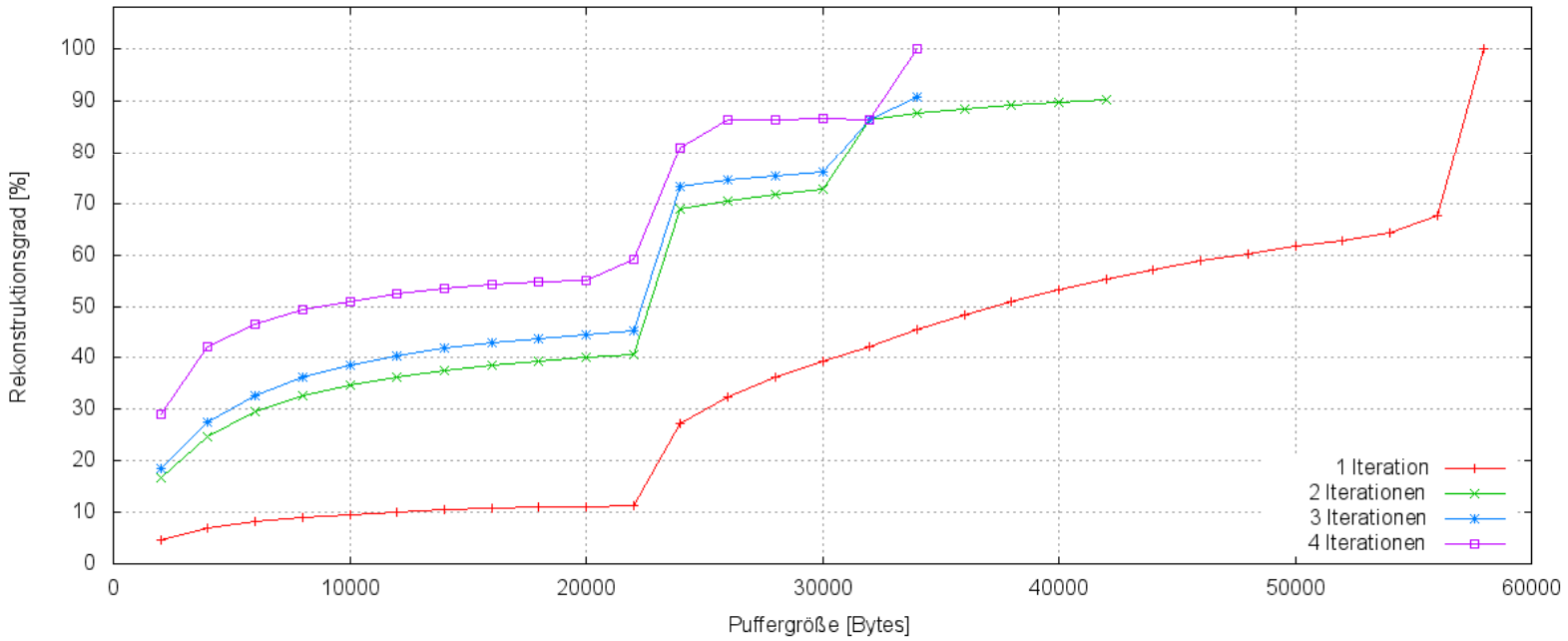
Myers, Glenford J.: Methodisches Testen von Programmen.

Stahleder, Elmar: Debugger mit R uckspiegel. Trace-Techniken im Überblick.

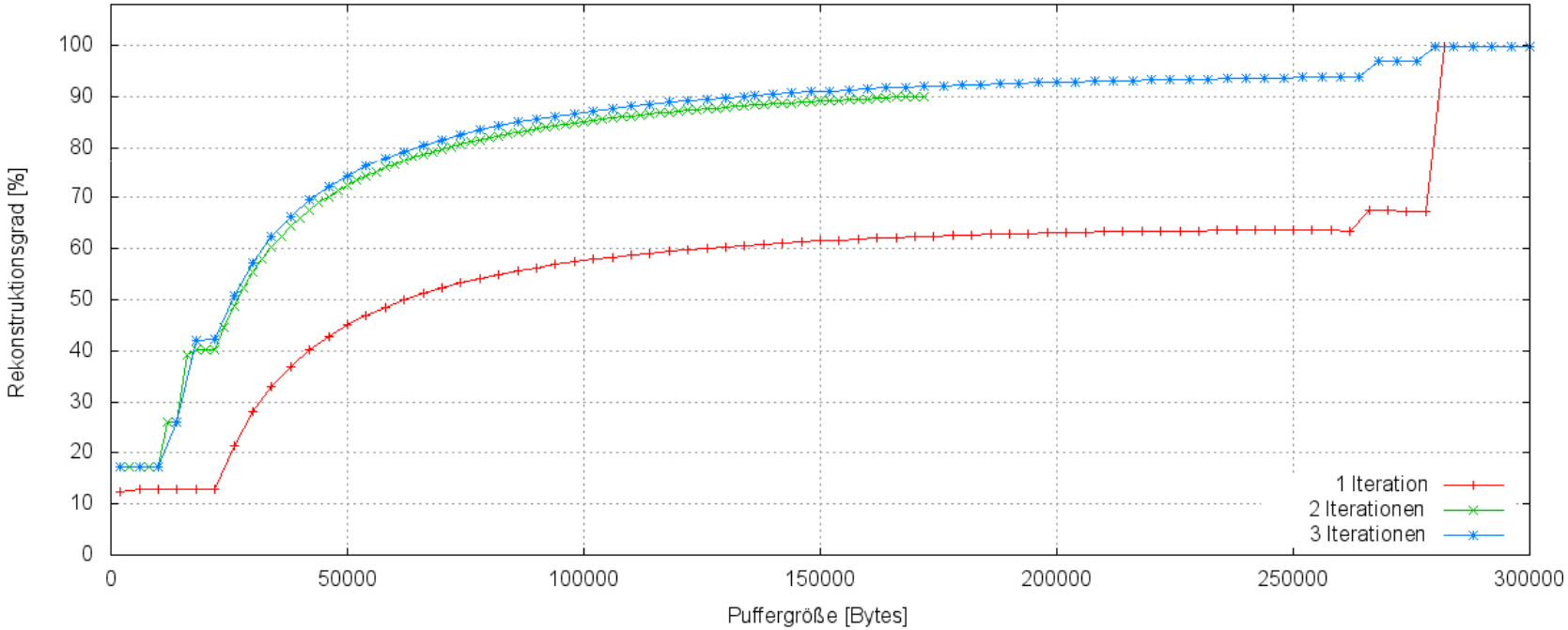
Wilhelm, R. ; Maurer, D.: Ubersetzerbau, Theorie, Konstruktion, Generierung.

Xiao Hu, Shuming C.: Applications of On-chip Trace on Debugging Embedded Processor.

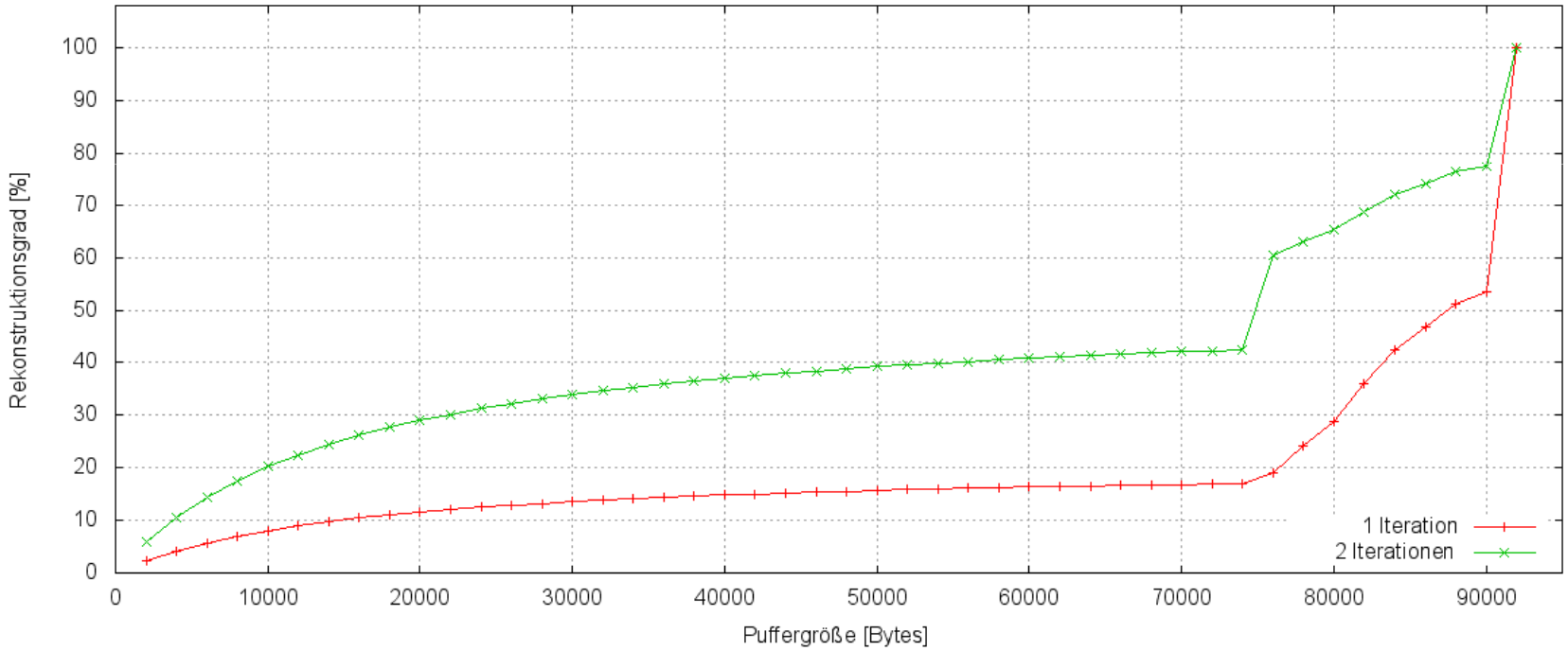
Iterationsaufwand FIR



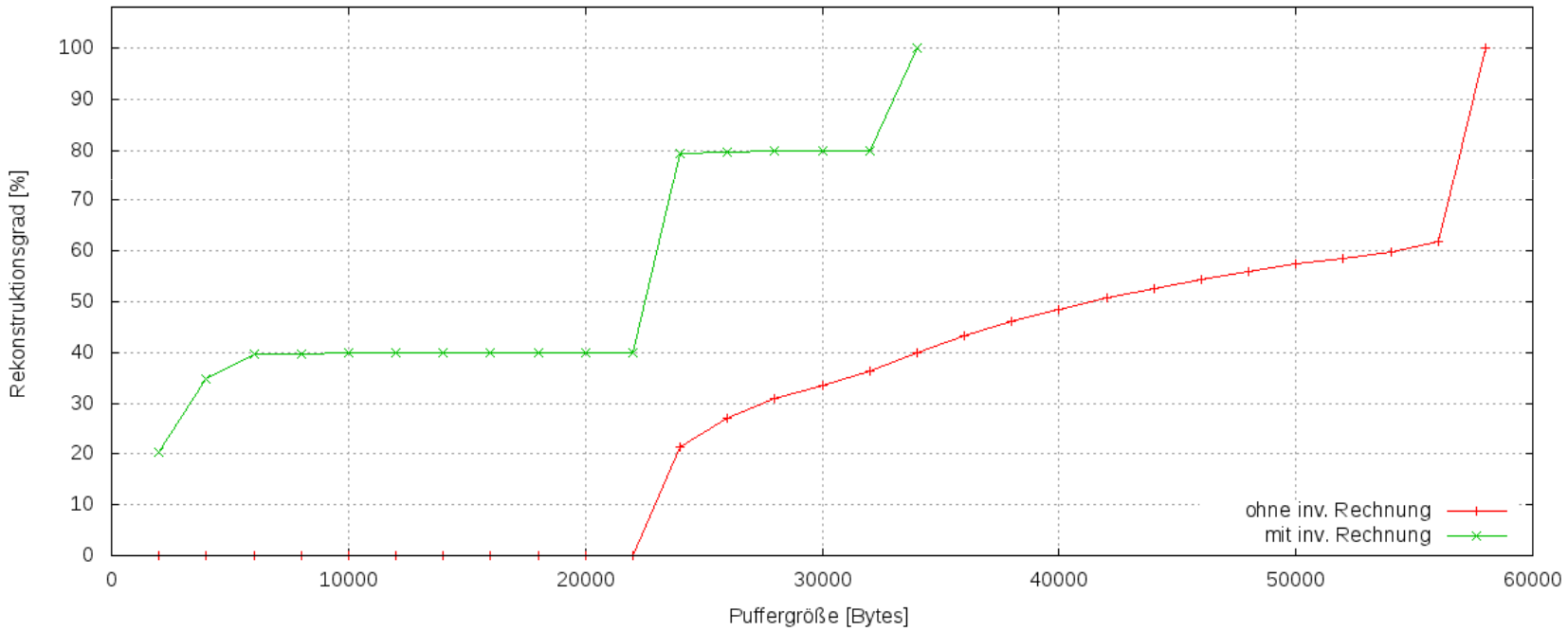
Iterationsaufwand FFT



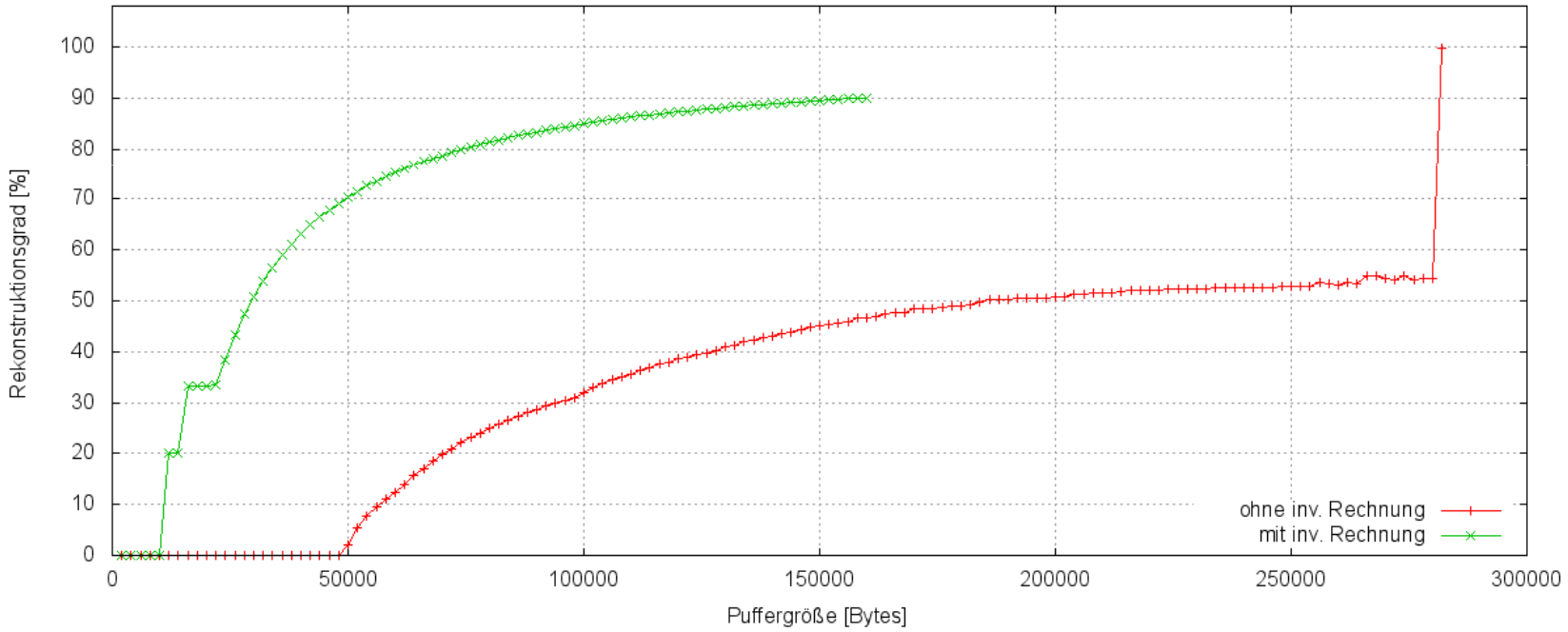
Iterationsaufwand Floatingpoint



FIR Zyklen > 50



FFT Zyklen > 50



FloatingPoint Zyklen > 50