TECHNISCHE UNIVERSITÄT DRESDEN

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

Automatische Dekodersynthese für den retargierbaren Befehlssatzsimulator Jahris

Diplomarbeit

Ronald Rist

s0200738@mail.zih.tu-dresden.de

Dresden, 05.12.2013

Inhalt

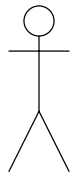
1 Einleitung

1.1 Hintergrund – Befehlssatzsimulation

Test/Debug auf Zielsystem

Einsehbarkeit?

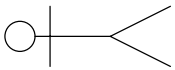
???



Entwickler

Test/Debug im Simulator

Geschwindigkeit?



Entwickler

- Vereinbarkeit von Inspizierbarkeit und hoher Simulationsperformanz?

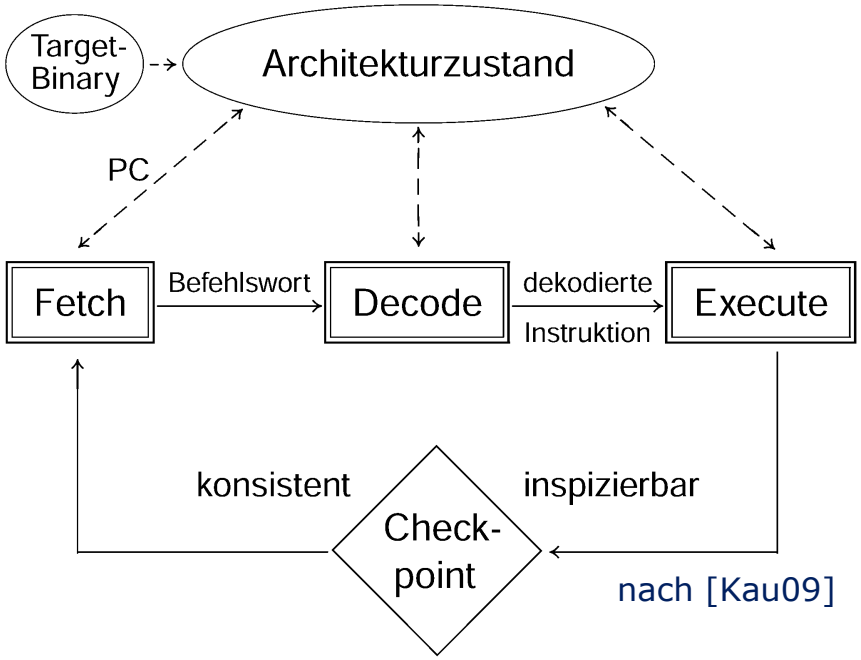
  → *Befehlssatzsimulator* ***Jahris***

1 Einleitung

1.2 Ausgangspunkt – Retargierbarer Befehlssatzsimulator Jahris

- von Marco Kaufmann in Java entwickelt, plattformunabhängig
- Simulation verschiedenster Architekturen/Befehlssätze, diese werden in HPADL beschrieben

- Beschränkung auf befehlsgenaue Simulation
- ausgelegt auf hohe Performanz durch Just-in-Time-Compilierung größerer Codeabschnitte
- interpretierende schrittweise Simulation ebenfalls möglich



nach [Kau09]

1 Einleitung

1.3 Motivation

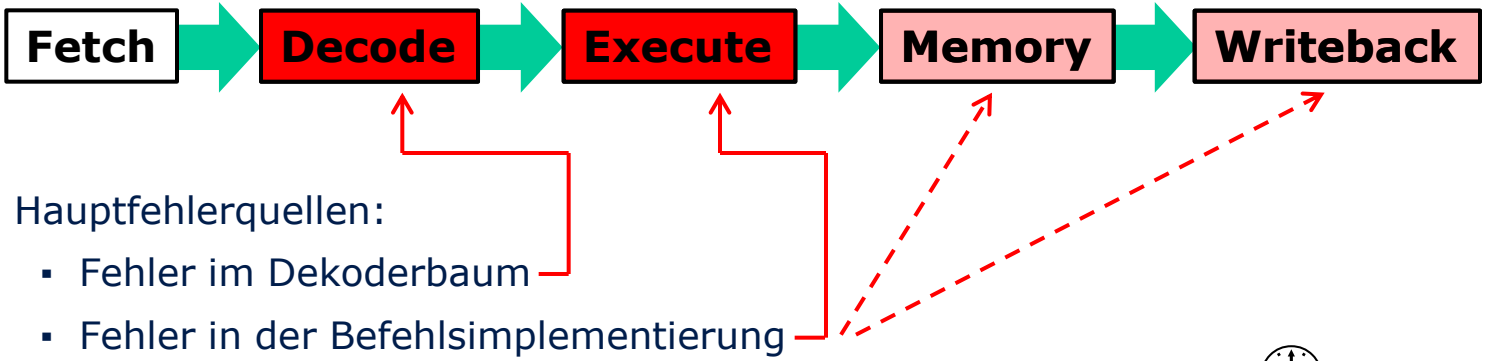
Neue Problemstellung:

Wie kann Entwicklung von Modellen für Jahris vereinfacht werden?

1.3 Motivation – Fortsetzung

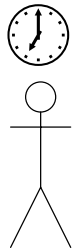
- Entwicklung von Architektursimulatoren bzw. Architekturbeschreibungen für retargierbare Architektursimulatoren:

  (abstrakte) Abbildung der physischen Pipeline-Stufen



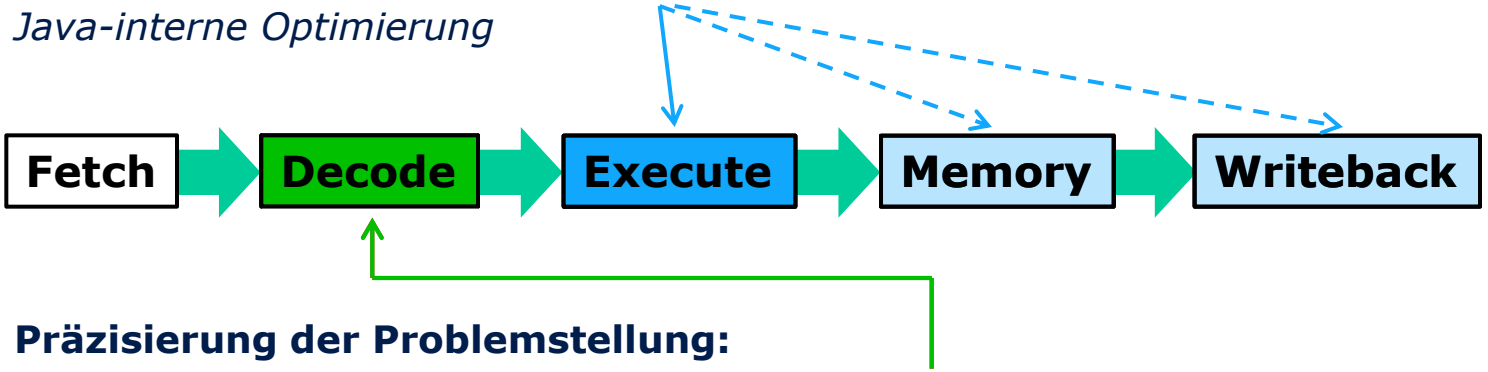
- Hauptfehlerquellen:
  - Fehler im Dekoderbaum
  - Fehler in der Befehlsimplementierung

- stellen gleichzeitig den Hauptzeitaufwand des Entwicklers dar (Erstellung sowie Optimierung von Hand)



1.3 Motivation – Fortsetzung (2)

- Jahris: Befehlsimplementierung durch Entwickler, jedoch *maschinelle* Optimierung der Befehle (bzw. deren Mikrooperationen) durch *Java-interne Optimierung*



- **Präzisierung der Problemstellung:**
  Kann Entwickler auch bei Erstellung des Dekoderbaumes entlastet werden?
  - maschinelle Erzeugung
  - ggf. automatische Optimierung

  → Ausschluss des „Fehlerfaktors Mensch"

2 Grundlagen

2.1 Erstellung eines Dekoders

- normierte Beschreibung aller verfügbaren Befehle der zu simulierenden Architektur

Aufgabe:

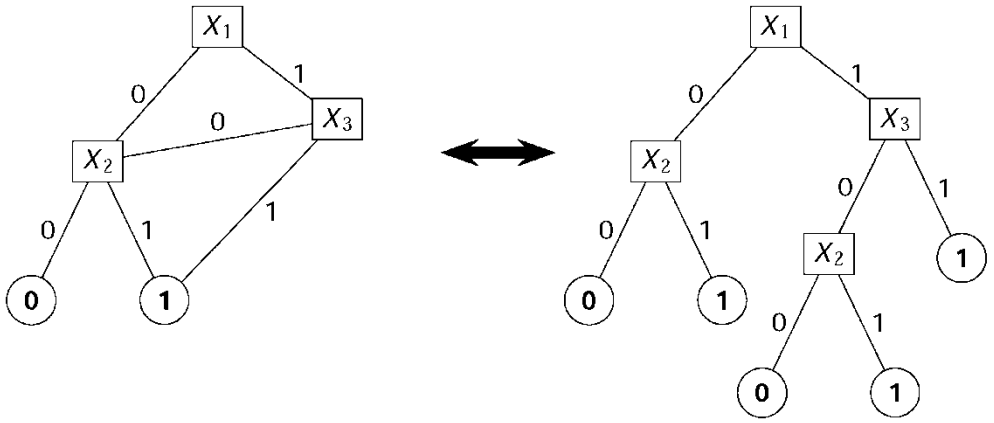
- Repräsentation als „Quelltext" und Repräsentation im Speicher?
  → Definition/Adaptierung geeigneter Formate
- Entwicklung eines Algorithmus zur Überführung der Befehlsliste in einen Befehlsdekoder

2 Grundlagen

2.2 Formale Grundlage – Entscheidungsbäume

- Abbildung von Befehlsdekodern in sog. *Entscheidungsbäumen*
- Zuordnung von Befehlsworten zu Operationen schrittweise hierarchisch

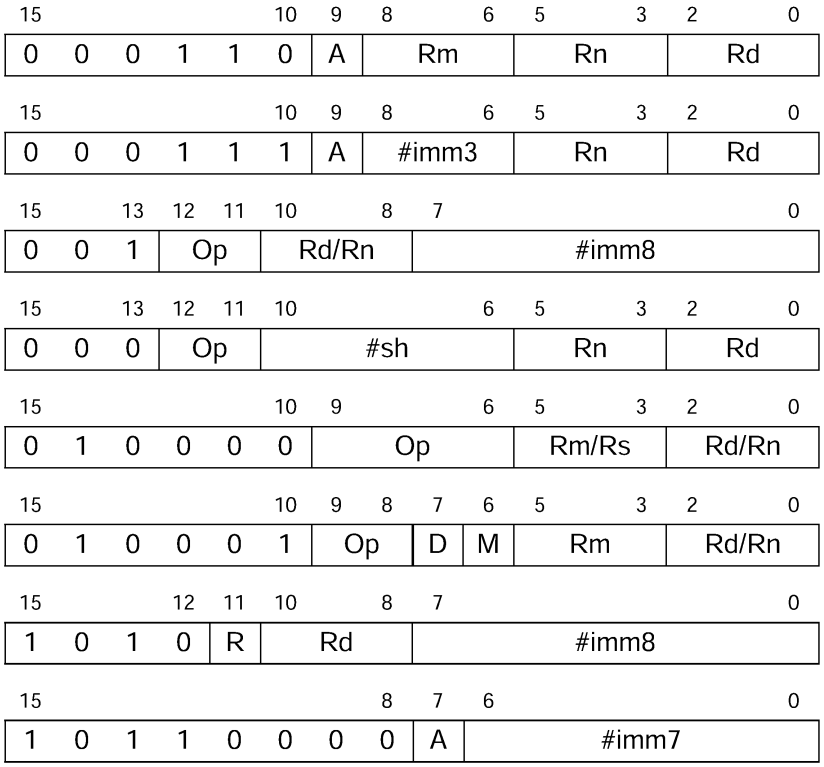
- *Baumstruktur*



nach [Mor82]

2.2 Entscheidungsbäume – Fortsetzung

- *Was soll abbildbar sein?*



(1) `ADD|SUB Rd,Rn,Rm`

(2) `ADD|SUB Rd,Rn,#imm3`

(3) `<Op> Rd,Rn,#imm8`

(4) `LSL|LSR|ASR Rd,Rn,#sh`

(5) `<Op> Rd/Rn,Rm/Rs`

(6) `ADD|SUB|MOV Rd/Rn,Rm`

(7) `ADD Rd,SP|PC,#imm8`

(8) `ADD|SUB SP,SP,#imm7`

nach [Fur00]

2 Grundlagen

2.2 Entscheidungsbäume – Fortsetzung (2)

- Komplexe Befehlssätze erzeugen komplexe Dekoder.
- Wie kann ein optimaler Dekoder gefunden oder ein vorhandener optimiert werden?
- *Lösungsraum*

$$A_B(k) = \prod_{i=0}^{k-1} (k-i)^{2^i}$$

$$A_B(k+1) = (k+1) * (A_B(k))^2$$

Decisions	# of Trees
1	1
2	2
3	12
4	576
5	1.658.880
6	16.511.297.126.400
…	…

2.3 Optimierung eines Dekoderbaumes

- Art und Weise der Optimierung?
  - **brute-force** (ggf. m. **backtracking**): praktisch nicht realisierbar
  - **greedy-Algorithmen** (engl. gierig): Beschleunigung durch gezielte Übernahme günstiger Folgezustände nach *Auswahlkriterien*

    → ggf. Kombination mit **hill-climbing**

- Festlegen von Bewertungskriterien für *Effizienz*
  - **maximale Baumtiefe** (längster Pfad, ungünstigster Überprüfungsaufwand)
  - **mittlere Baumtiefe** (ggf. Berücksichtigung der Auftrittswahrscheinlichkeit)
  - **Speichergröße des Baumes**

  → ggf. *einstellbare* Optimierungskriterien

3 Entwurf: Erweiterung der HPADL-Syntax

3.1 Definition von verschiedenen Befehlswortformaten

```
format DATAPROC(opc)    { opc:4; op1:4; op2:4; opD:4              }
format LD_ST(opc,opc2) { opc:4; op1D:5; opc2:2; op2:3; :2=0b11   }
format MEDIA(opc)       { opc:6; op1D:6; op2:4=0xF; signed imm:16 }
```

- der Identifikation dienende Opcode-Teile eindeutig vorgeben (vgl. Parameter)
- Definitionen {...} angelehnt an C-Bitfeld-Syntax, inkl. anonymer Felder
- Festlegung von Bitbreite, Wert und Vorzeichenerweiterung möglich

- Angabe von aufrufbaren Hilfsfunktionen
- Befehlsformate sind optional

3.2 Beschreibung von Befehlen

```
ADD8 DATAPROC(0xA)    { rn=op1; rm=op2; rd=opD; ls1rn; ls2rm; ADD; sdrd; } //ADD8
SUB8 DATAPROC( 12)    { rn=op1; rm=op2; rd=opD; ls1rn; ls2rm; SUB; sdrd; } //SUB8

ADD3 DATAPROC(  5)    { if (opD==15) B; else ADD;                         }
```

- direkte Verwendung von (bisherigem) HPADL-Code in der Befehlsdefinition

```
LD_ST(7,1)       { ... }
```

- effektive Definition von geteilten Opcodes (vgl. vorige Folie)

```
MEDIA(0x1C)      { ... }
MEDIA(0b01_1101) { ... }
```

- vorherige Definition der Befehlswortabschnitte, Binärzahlunterstützung und Separatoren → vereinfachte Spezifikation von nichtausgerichteten Feldern

3.2 Beschreibung von Befehlen (Fortsetzung)

```
MEDIA(!0x0*) { ... }
MEDIA(0b1**) { ... }
```

- Unterstützung von Platzhalterangaben („Wildcards“) und Negation
  → vereinfachte Übernahme derart notierter Reference Manuals

```
MEDIA(0x3F)  { fetch.byte(16); ... }
```

- Automatisches Laden und Nachladen weiterer Befehlswortbestandteile

- Intermix von Formaten und Befehlen (minimiertes Scrolling)

```
LDRlit {:5=1001b; rd:3; imm:8} { LS1R15; imm<<=2; LS2_IMM; ADDS1S2; LDR; sdrd; }
```

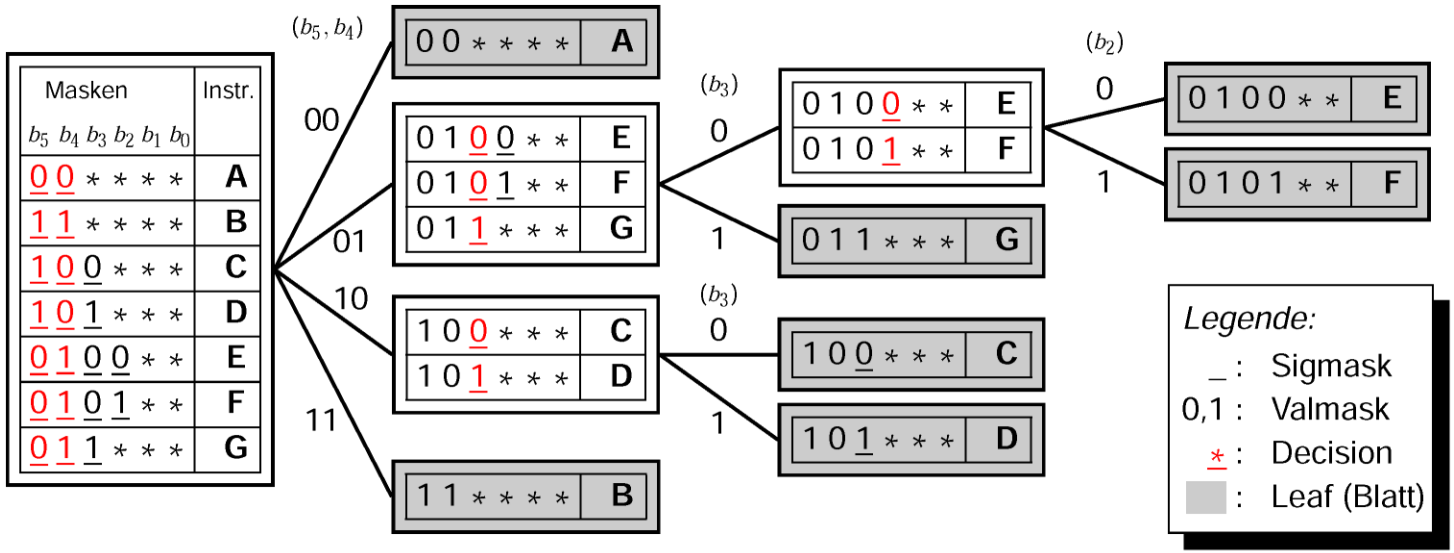
- Verzicht auf die Verwendung von Formaten („Single-Instruction“) durch integrierte Definition möglich; Mnemonic ebenfalls optional

4 Implementiertes System

4.1 Generelles

- Umsetzung als „Zwiebelschale“ (Onion) zu vorhandenem System, keine größeren Modifikationen am HPADL-Zwischencode
  - kompatibilitätserhaltend
  - Entkopplung von Dekodersynthese und Simulationsengine
- geeignete Überführung der Befehlsdefinitionen in auswertbare Bitmasken
- notwendige Nachladeautomatik impliziert leichte User-Einschränkungen
- korrekte Bereitstellung aller Operanden erforderlich
- Unterstützung impliziter sowie manueller default-Zweige zu Debug-Zwecken
- umfangreiches Protokollsystem über erzeugte Dekoder

4.2 Dekodersynthese – Allgemeine Vorgehensweise



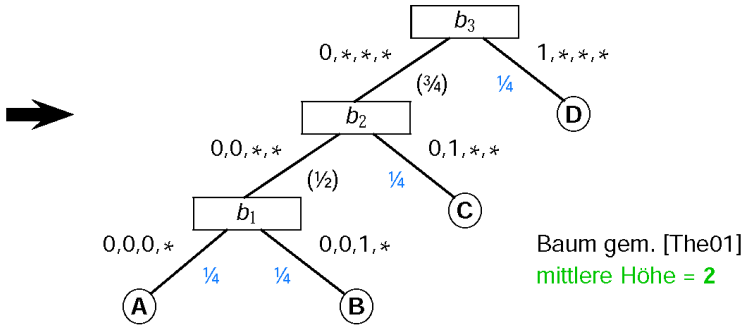
- rekursive Restmengenzerlegung (*recursive grouping*):
  Ermittlung der für alle Befehle der aktuellen Stufe signifikanten Bitstellen und Einteilung in identische Untergruppen; Wiederholung für alle Untergruppen

4.3 Optimierungsmaßnahmen

- *Aufspaltung von Befehlen*

Idee:

Befehl	Pattern b_3	b_2	b_1	b_0	$p(.)$
A	0	0	0	*	¼
B	0	0	1	*	¼
C	0	1	*	*	¼
D	1	*	*	*	¼

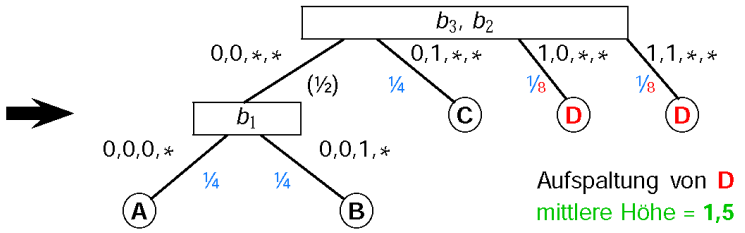


Befehl	Pattern b_3	b_2	b_1	b_0	$p(.)$
A	0	0	0	*	¼
B	0	0	1	*	¼
C	0	1	*	*	¼
D	1	*	*	*	¼

$(b_3\ b_2)$

0	A	0	0	0	*	¼
	B	0	0	1	*	¼
1	C	0	1	*	*	¼
2	D	1	0	*	*	⅛
3	D	1	1	*	*	⅛

D geteilt



Es wäre von Vorteil, wenn im ersten Schritt weit mehr Befehle direkt ausgewertet werden könnten.

nach [Qin+03], modfiziert

4.3 Optimierungsmaßnahmen – Fortsetzung

Einführung eines Bewertungskriteriums:

vorher (n)

Masken $b_5\,b_4\,b_3\,b_2\,b_1\,b_0$	Instr.
0 0 0 0 * *	A
0 0 0 1 * *	B
0 0 1 0 * *	C
0 0 1 1 * *	D
1 0 0 0 0 *	E
1 0 0 0 1 *	F
1 0 0 1 0 *	G
1 0 0 1 1 *	H
1 0 1 0 0 *	I
1 0 1 0 1 *	J

⟶

nachher (n+1)

Masken $b_5\,b_4\,b_3\,b_2\,b_1\,b_0$	Instr.
0 0 0 0 0 *	A_0
0 0 0 0 1 *	A_1
0 0 0 1 0 *	B_0
0 0 0 1 1 *	B_1
0 0 1 0 0 *	C_0
0 0 1 0 1 *	C_1
0 0 1 1 0 *	D_0
0 0 1 1 1 *	D_1
1 0 0 0 0 *	E
1 0 0 0 1 *	F
1 0 0 1 0 *	G
1 0 0 1 1 *	H
1 0 1 0 0 *	I
1 0 1 0 1 *	J

Bestand an Befehlen in diesem Knoten			Reduktion der Unbestimmtheit			Nutzeffekt?
B			**U**			$\lfloor \mathbf{Id}\ \Delta\mathbf{B} \rfloor \leq \Delta\mathbf{U}$
n	n+1	Δ	n	n+1	Δ	
Beispiel: Instruktionen A bis F						2 > 1
6	10	**+4**	1	0	**−1**	wird nicht übernommen
Beispiel: Instruktionen A bis H						2 = 2
8	12	**+4**	2	0	**−2**	Grenzfall
Beispiel: Instruktionen A bis J						2 < 3
10	14	**+4**	3	0	**−3**	wird übernommen

4.3 Optimierungsmaßnahmen – Fortsetzung (2)

Iterative Anwendung des Bewertungskriteriums:

vorher (n)

Masken $b_5\,b_4\,b_3\,b_2\,b_1$	Instr.
0 0 0 0 1	K
0 0 0 1 *	L
0 0 1 * *	M
0 1 0 * *	N
0 1 1 * *	O
1 * * * *	P

→

(n+1)

Masken $b_5\,b_4\,b_3\,b_2\,b_1$	Instr.
0 0 0 0 1	K
0 0 0 1 *	L
0 0 1 * *	M
0 1 0 * *	N
0 1 1 * *	O
1 0 * * *	P_0
1 1 * * *	P_1

→

(n+2)

Masken $b_5\,b_4\,b_3\,b_2\,b_1$	Instr.
0 0 0 0 1	K
0 0 0 1 *	L
0 0 1 * *	M
0 1 0 * *	N
0 1 1 * *	O
1 0 0 * *	P_{00}
1 0 1 * *	P_{01}
1 1 0 * *	P_{10}
1 1 1 * *	P_{11}

→

nachher (n+3)

Masken $b_5\,b_4\,b_3\,b_2\,b_1$	Instr.
0 0 0 0 0	K
0 0 0 1 *	L
0 0 1 0 *	M_0
0 0 1 1 *	M_1
0 1 0 0 *	N_0
0 1 0 1 *	N_1
0 1 0 0 *	O_0
0 1 0 1 *	O_1
1 0 0 0 *	P_{000}
1 0 0 1 *	P_{001}
1 0 1 0 *	P_{010}
1 0 1 1 *	P_{011}
1 1 0 0 *	P_{100}
1 1 0 1 *	P_{101}
1 1 1 0 *	P_{110}
1 1 1 1 *	P_{111}

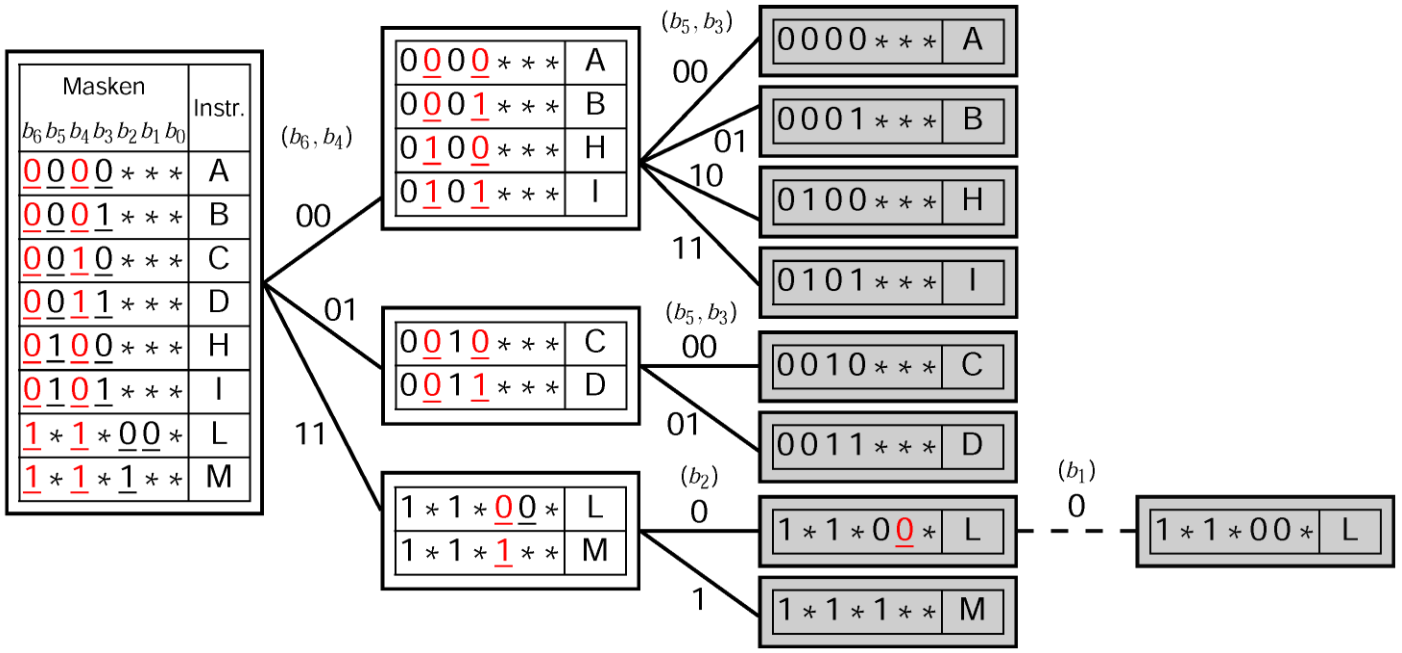
$\Delta B = +1 \quad \Delta U = -1 \quad \rightarrow \quad 0 < 1$
wird übernommen

$\Delta B = +2 \quad \Delta U = -2 \quad \rightarrow \quad 1 < 2$
wird übernommen

$\Delta B = +7 \quad \Delta U = -1 \quad \rightarrow \quad 2 > 1$
wird nicht übernommen

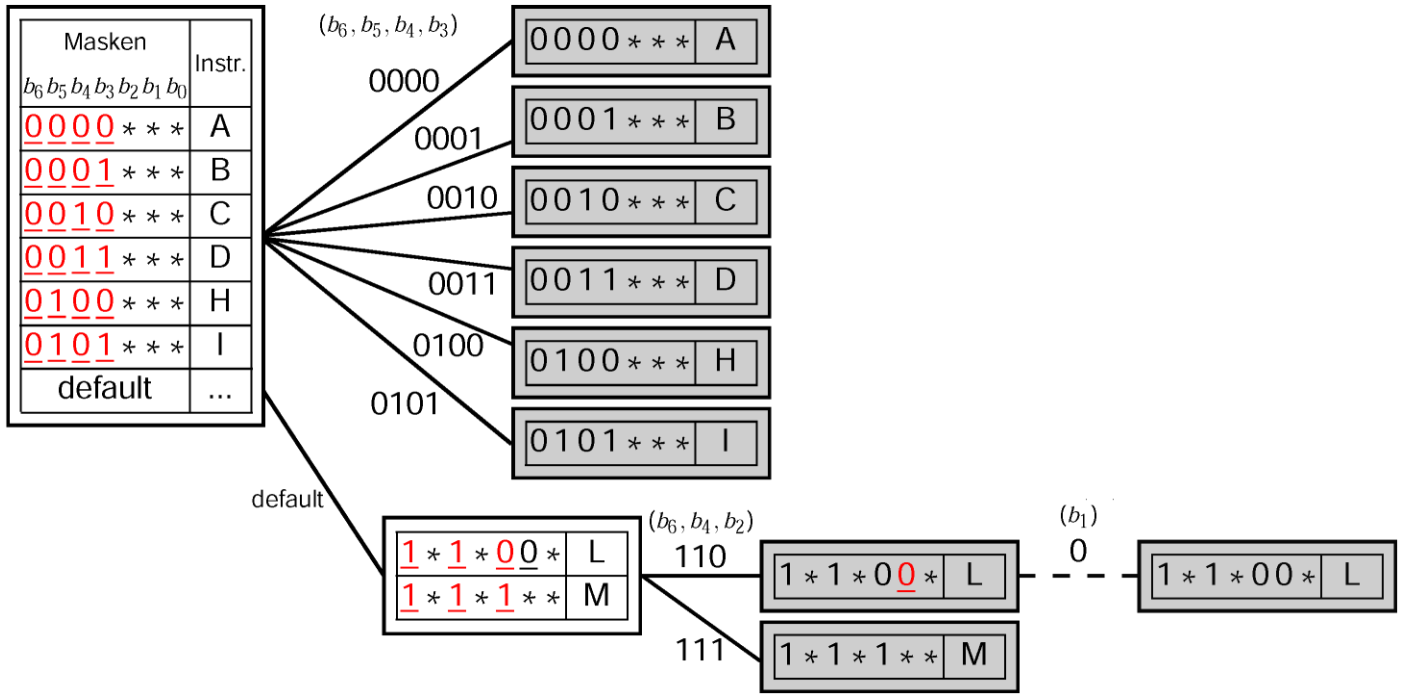
4.3 Optimierungsmaßnahmen – Fortsetzung (3)

- *Suche nach Auslagerungskandidaten*



4.3 Optimierungsmaßnahmen – Fortsetzung (4)

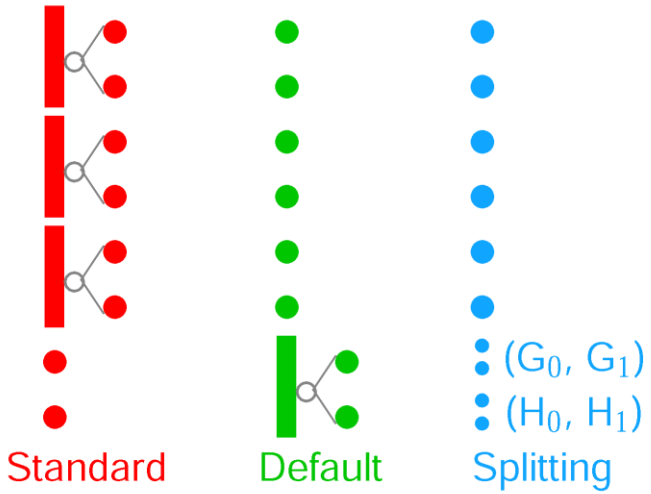
Auslagerung



4.3 Optimierungsmaßnahmen – Fortsetzung (5)

- Nutzen der Maßnahmen

Masken						Instr.
b_5	b_4	b_3	b_2	b_1	b_0	
0	0	0	0	0	*	A
0	0	0	0	1	*	B
0	0	0	1	0	*	C
0	0	0	1	1	*	D
0	0	1	0	0	*	E
0	0	1	0	1	*	F
0	1	0	0	*	*	G
0	1	0	1	*	*	H



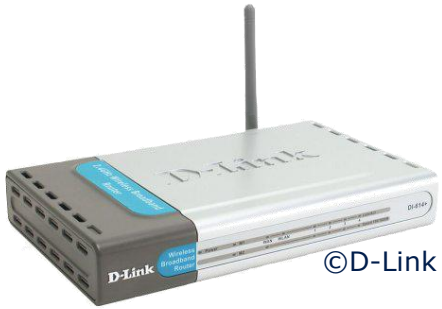
	Standard	Default	Splitting
mittlere Tiefe:	**1,75**	**1,25**	**1**
Knotenäquivalent:	11	9	10

5 Umgesetzte Dekoder und Auswertung

5.1 Bezug

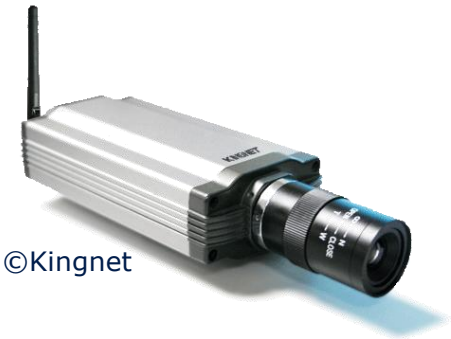
BeagleBoard Project

©Sony



©D-Link

©Samsung



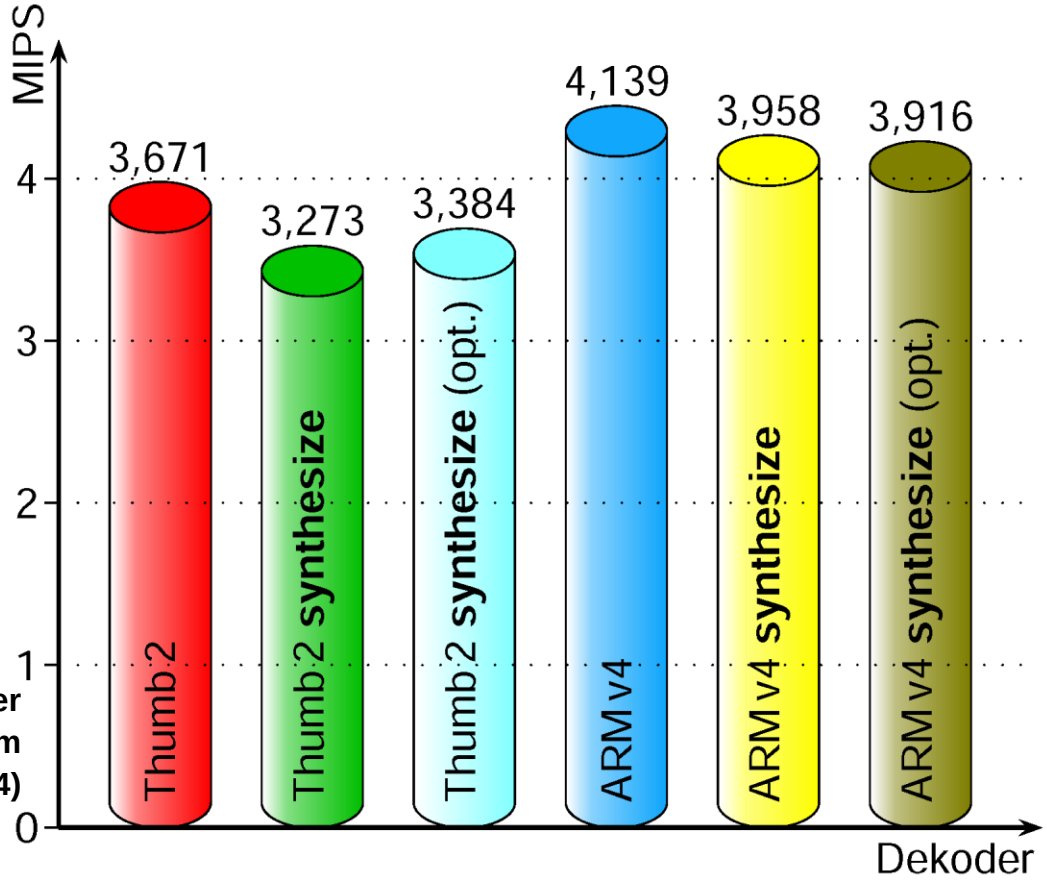
©Kingnet

5.2 Ergebnisse der erzeugten ARM- und Thumb-Dekoder

(all unsafe)	instr. cnt.	tree height		pct. of ref.	tree size	pct. of ref.
		worst	avg.			
ARM v4 (opt.)	233	6	**3,15**	**42 %**	385	102 %
ARM v4	189	7	4,38	58 %	325	86 %
ARM v7 (opt.)	410	6	**3,83**	**46 %**	688	104 %
ARM v7	331	10	6,44	77 %	580	88 %
Thumb (opt.)	163	5	**1,52**	**24 %**	255	165 %
Thumb	78	8	3,47	55 %	136	87 %
Thumb (man.)	78	6	3,54	56 %		
Thumb 2 (opt.)	463	8	**3,24**	**38 %**	777	100 %
Thumb 2	388	11	5,75	67 %	693	89 %

5.3 Benchmark

- nur bei interpretierender Simulation
- synthetisierte Dekoder derzeit ca. 8 bis 10% langsamer als manuell erzeugte Dekoder
- leider bis dato keine signifikanten Performanz-Gewinne durch die Optimierungen nachweisbar



Klassischer und synthetisierter Dekoder mit Testprogramm Sieve (O3, Thumb2 und ARMv4)

5.4 Ressourcenverbrauch der Dekodersynthese

- Zeit: derzeit vernachlässigbar – Dekodererzeugung stets im einstelligen Sekundenbereich auf Intel Core 2 Duo mit 2,66 GHz
- *Speicherverbrauch* („unsafe", ohne Log):

Testprogramm Sieve (O3) @	Jahris approx. peak mem.
ARM v4 manueller Dekoder	73 MB
ARM v7 synthetisierter Dekoder	170 MB
ARM v7 ~ mit Optimierungen	366 MB
Thumb 2 manueller Dekoder	75 MB
Thumb 2 synthetisierter Dekoder	152 MB
Thumb 2 ~ mit Optimierungen	261 MB

→ Aufwand/Nutzen der Optimierungen muss erwogen werden

6 Zusammenfassung und Ausblick

- HPADL wurde um eine Syntax zur Befehlsauflistung erweitert
- Dekodergenerator wurde erstellt
- Funktionsfähigkeit an ARM v7- und Thumb 2-Befehlssätzen demonstriert
- Umsetzung diverser Optimierungsansätze
- umfangreiche Logging- und Ausgabefunktion
- Modellentwicklung in HPADL vereinfacht

- Ausbaumöglichkeiten:
  - diverse Optimierungen des Speicherverbrauchs, Multi-Thread-Synthese (?)
  - weitere auf Wildcards anwendbare Operatoren: <,≤,>, ≥
  - Analyse von Befehlshäufigkeiten (Profiling)
  - HPADL: in Bytecode übersetzte Decoder

7 Ausgewählte Quellen

[Kau09] Kaufmann, M.: Erschließung von Just-in-Time-Compilierungstechniken in der Realisierung eines retargierbaren Architektursimulators, TU Dresden, 2009

[Mor82] Moret, B. M. E.: Decision Trees and Diagrams, University of New Mexico, ACM, CSUR 1982, S. 593-623, ISSN 0360-0300

[Nor11] Norvell, Th. S.: The JavaCC FAQ, http://www.engr.mun.ca/~theo/JavaCC-FAQ/javacc-faq-moz.htm

[Qin+03] Qin, W.; Malik, S.: Automated Synthesis of Efficient Binary Decoders for Retargetable Software Toolkits, Princeton University, ACM, DAC 2003, S. 764-769, ISBN 978-1-58113-688-3

[The01] Theiling, H.: Generating Decision Trees for Decoding Binaries, Universität des Saarlandes, ACM, LCTES 2001, S. 112-120, ISBN 978-1-58113-425-4