Karlsruher Institut für Technologie
Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Klausur Rechnerstrukturen Sommersemester 2017 Musterlösung

Voraussichtliche Bekanntgabe der vorläufigen Ergebnisse:
September 2017

Aufgabe 1: Sprungvorhersage, Fehlertoleranz und Verbindungsstrukturen — 10 P

Sprungvorhersage — 5 P

a) Die Sprungvorhersage verhindert das Leerlaufen der Pipeline bei Kontrollflussbefehlen. *1 P*

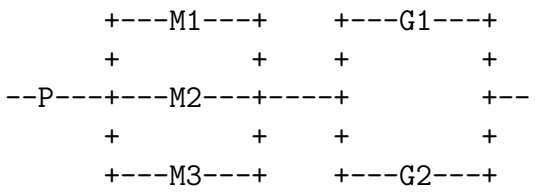
b) Der Branch Target Address Cache speichert als Tag die/einen Teil der Instruktionsadresse der Verzweigung und als Wert die Instruktionsadresse des Sprungziels. *1 P*

c) *3 P*

	Globaler Prädiktor	S1		Globaler Prädiktor	S2	
		Vhs.	Sprung		Vhs.	Sprung
1	(WT, **WT**)	**T**	T	(WT, **ST**)	**T**	T
2	(WT, **ST**)	T	NT	(**WT**, WT)	**T**	T
3	(ST, **WT**)	T	NT	(**ST**, WNT)	T	NT
4	(**WT**, WNT)	**T**	T	(ST, **WNT**)	**NT**	NT
5	(**ST**, SNT)	**T**	T	(ST, **SNT**)	NT	T
6	(ST, **WNT**)	**NT**	NT	(**ST**, SNT)	T	NT

Fehlertoleranz — 3 P

d) Zuverlässigkeitsblockdiagramm: *1 P*



e) Die allgemeine Bezeichnung ist n-aus-m Systeme. *0,5 P*

f) Fehlerbereiche: $B_1 = M_1$, $B_2 = M_2$, $B_3 = M_3$, $B_4 = M_1, M_2$, $B_5 = M_1, M_3$, $B_6 = M_2, M_3$ *1,5 P*

Verbindungsstrukturen — 2 P

g) Flit ist die Abkürzung für Flow Control Unit. Flits werden auf der Link Ebene *1 P*

des QPI Protokolls verwendet.

h) Die Übertragungsmodi Cut Through und Wormhole Routing sind identisch, wenn keine Blockierung vorliegt. *1 P*

Aufgabe 2: Vektorrechner, VLIW und Tomasulo 10 P

Vektorrechner 2 P

a) Möglichkeiten der Parallelarbeit: *2 P*

- Vektor-Pipeline-Parallelität
- Mehrere (unterschiedliche) Vektor-Pipelines in einer Vektoreinheit
- Vervielfachung der Pipelines (mehrere gleiche Pipelines)
- Mehrere Vektoreinheiten

VLIW 3 P

b) VLIW-Prozessoren *3 P*

Slot 1	Slot 2
2) ld r2, [r1]	3) ld r4, [r3]
1) add r5, r3, r1	5) add r1, r4, r2
4) mul r5, r5, r4	6) st [r6], r1

Tomasulo 5 P

c) *5 P*

Feld	R1	R2	R3	R4	R5
Value	(R2 + R3)				
Valid	1	0	0	0	0
RS		Add/Sub 1	Add/Sub 2	Mul 1	Div 1

Unit	Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
Add/Sub 1	0	0	add	R2		0	Add 2		0	Mul 1
Add/Sub 2	0	1	sub	R3	(R2+R3)	1	–	(R2)	1	–
Mul 1	0	1	mul	R4	(R2+R3)	1	–	(R5)	1	–
Div 1	0	0	div	R5	(R5)	1	–	(R2+R3)	1	–

Aufgabe 3: Caches — 10 P

a) Inhalt des auf höchster Stufe stehenden Cache-Speichers befindet sich auch in den Cache-Speichern niedrigerer Ebene. (L1$)<(L2$)< ... *1 P*

b) Wenn Prozessoren jeweils unabhängig voneinander auf Speicherwörter des Hauptspeichers zugreifen können, dann müssen mehrere Kopien des gleichen Speicherwortes miteinander in Einklang gebracht werden. *1 P*

c) *1 P*

- Sequentielle Konsistenz
- Prozessorkonsistenz
- Schwache Konsistenz
- Freigabekonsistenz
- Entry-Konsistenz

d) *2 P*

Proz.	Aktion	Proz. 1		Proz. 2		Proz. 3	
		Zeile 1	Zeile 2	Zeile 1	Zeile 2	Zeile 1	Zeile 2
-	init	-	-	-	-	-	-
2	wr 4			4/M			
1	rd 4	4/S		4/O			
3	wr 4	4/I		4/I		4/M	
3	rd 5						5/E
3	wr 1					1/M	

e) *4 P*

	Proz. 1	Proz. 2	Proz. 3	Art des Cache-Misses
1	Read A			Cold Miss/Compulsory Miss
2		Read B		Cold Miss/Compulsory Miss
3			Read C	Cold Miss/Compulsory Miss
4	Write A			
5			Read D	Cold Miss/Compulsory Miss
6		Read B		False-Sharing Miss
7	Write B			
8			Read C	Capacity Miss/Conflict Miss
9		Read B		True-Sharing Miss

f) *1 P*

Der Miss zum Zeitpunkt 6, der False-Sharing Miss, kann ignoriert werden.

Aufgabe 4: Hardware-Entwurf und VHDL 10 P

Hardware-Entwurf 4 P

a) Formel der Beschleunigung nach Gesetz von Amdahl *1,5 P*

$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{\frac{1-a}{n} + a}$$

mit $a =$ der ausschließlich sequentiell ausführbare Programmanteil und $n =$ die Anzahl der Prozessoren.

b) *2,5 P*

- $$S_{sqrt} = \frac{1}{\frac{1-0.8}{10} + (0.8)} = \frac{1}{0,82} = 1.22$$
- $$S_{fp} = \frac{1}{\frac{0.5}{2} + (1 - 0.5)} = \frac{1}{0.75} = 1.33$$
- Die Perfomanzverbesserung aller Gleitkommaoperationen ist die bessere Alternative, aufgrund der höheren Nutzbarkeit.

Produktion 1 P

c) A ist die theoretisch erzielbare maximale Anzahl von Dies (0.5 P), B der durch die Einbeschreibung von Rechtecken entstehende Verschnitt (0.5 P). *1 P*

VHDL 5 P

d) Die Schaltungsbeschreibung modelliert einen 4-Bit-Zähler, d.h. einen Zähler von 0-15 (0,5 P). `clk`: Löst beim Übergang von 0 nach 1 (0,5P) einen Zählschritt aus (0,5 P) `rst`: Löst bei 0-Pegel (=low-aktiv, 0,5 P) ein Rücksetzen des Zählers auf den Wert 15 aus (0,5 P) `count`: aktueller Zählerstand (0,5 P) *3 P*

e) `count` hat die Signalrichtung *out* (0,5 P), d.h. `count` kann nur geschrieben, aber nicht gelesen werden, wie es für die Addition notwendig ist (0,5 P). Es ist zur Behebung ohne Änderung der Architecture die Signalrichtung in der Entity zu ändern (0,5 P) und zwar auf *inout* oder *buffer* (0,5 P). *2 P*

Aufgabe 5: Leistungsbewertung **10 P**

a) *4 P*

(a.1) falsch (0,5p)
Begründung: Es ist ist leicht nachzuweißen, dass eine Verbesserung einer einzelnen Systemkomponente (wenn diese beispielsweise nie zum Einsatz kommt) nicht unmittelbar zu einem bestimmten Verbesserungsfaktor für das gesamte System führt. (0,5p)

(a.2) falsch (0,5p)
Begründung: Die Verwendung eines einzelnen Wertes wie Frequenz, CPI oder Befehlszahl ist nie als Maß zur Leistungsbewertung ausreichend, da diese immer von der Anwendung und der kompletten Systemkonfiguration abhängig ist. (0,5p)

(a.3) falsch (0,5p)
Begründung: Anhand des MIPS-Wertes können beispielsweise Systeme mit unterschiedlichen Befehlssätzen nicht verglichen werden. Damit kann hiermit kein Maßstab für den Vergleich von Systemen definiert werden.(0,5p)

(a.4) wahr (0,5p)
Begründung: Amdahls Gesetzt beschreibt den folgenden Zusammenhang:

$$\text{Zeit nach der Verbesserung} = \frac{\text{von der Verbesserung betroffene Ausführungszeit}}{\text{Verbesserungsfaktor}} + \text{nicht betroffene Ausführungszeit.}$$

Einfaches Einsetzen von Beispielzahlen (z.B. Verbesserungsfaktor$\rightarrow \infty$) ergibt den entsprechenden Zusammenhang. (0,5p)

b) Berechnung: *6 P*

Unter der Annahme, dass Einflüsse des Betriebssystems keine Rolle spielen, gilt:

$$CPI_A = 0,2 \cdot 2 + 0,8 \cdot 1 = 1,2 \qquad (1p)$$

da 20% des Codes aus Verzweigungen mit 2 Zyklen bestehen. Die Zeit ergibt sich damit zu:

$$Zeit_A = 1,2 \cdot IC_A \cdot t_A. \qquad (1p)$$

Da auf CPU B Vergleiche nicht explizit ausgeführt werden, fehlen 20% des Codes im Vergleich zu CPU A. Der Anteil an Verzweigungen im Gesamtcode beträgt daher

$$\frac{20\%}{80\%} = 25\%. \qquad (1p)$$

Damit gilt:

$$CPI_B = 0,25 \cdot 2 + 0,75 \cdot 1 = 1,25. \qquad (0,5p)$$

Die Zeit ergibt sich damit unter Beachtung der unterschiedlichen Codelänge zu:

$$\begin{aligned} Zeit_B &= 1,25 \cdot IC_B \cdot t_B \\ &= 1,25 \cdot (0,8 \cdot IC_A) \cdot (1,25 \cdot t_A) \\ &= 1,25 \cdot IC_A \cdot t_A. \qquad (1p) \end{aligned}$$

Somit wird CPU A den Code schneller abgearbeitet haben, als CPU B. (0,5p)
Für den geänderten Faktor ergibt sich die Zeit von CPU B zu:

$$\begin{aligned} Zeit_B &= 1,25 \cdot IC_B \cdot t_B \\ &= 1,25 \cdot (0,8 \cdot IC_A) \cdot (1,1 \cdot t_A) \\ &= 1,1 \cdot IC_A \cdot t_A. \qquad (0,5p) \end{aligned}$$

Somit wird CPU B den Code schneller abgearbeitet haben, als CPU A. (0,5p)

Aufgabe 6: Parallelverarbeitung und Pipelining 10 P

Parallelverarbeitung 6 P

a) Vergleich OpenMP und MPI: *3 P*

- OpenMP: Fork-Join-Model (0,5 p), Kommunikation über geteilte Variablen (0,5 p), Einsatz insbesondere im Bereich der UMA-Architekturen (0,5 p)
- MPI: Single Program Multible Data (0,5 p), Kommunikation über Nachrichten (0,5 p), Einsatz insbesondere im Bereich der NORMA-Architekturen (0,5 p)

b) Weitere parallele Programmiermodelle: *3 P*

- Multiprogramming (0,5 p), eine Menge von unabhängigen Programmen, die nicht miteinander kommunizieren (0,5 p), Einsatzbereich: multitasking-fähiges Betriebssystem (0,5 p)
- Datenparallelismus (0,5 p), gleichzeitiges Ausführen von Operationen auf getrennten Elementen einer Datenmenge (0,5 p), Einsatzbereich: Vektorrechner (0,5 p)

Pipelining 4 P

Der MIPS-Code für das gegebene Segment lautet unter der Vorraussetzung, dass sich alle Variablen bereits im Speicher befinden und als Offset von Register $\$t0$ adressierbar sind, wie folgt:

```
1        lw   $t1, 0($t0)
2        lw   $t2, 4($t0)
3        add  $t3, $t1, $t2
4        sw   $t3, 12($t0)
5        lw   $t4, 8($t0)
6        sub  $t5, $t3, $t4
7        sw   $t5, 16($t0)
```

c) Konflikte, Lösungen und Folgekonflikte: *2 P*

- Konflikte: Datenkonflikte zwischen Zeile 2 und 3 sowie 5 und 6 (1p),
  Optional: Die Storebefehle (Z. 4 und 7) erzeugen auch Datenkonflikte, die mit Pipeline-Bubbles gelöst werden können
- Lösung: Umsortieren der Befehle: Zeile 5 zwischen Zeile 2 und 3 einschieben(0,5p)
- Folgekonflikte: Zugriff der Subtraktion auf das Ergebniss der Addition bevor dieses vollständig in den Speicher/die Register geschrieben wurde (0,5p)

d) Definition forwarding und Beitrag zur Konfliktlösung: *2 P*

- Definition: Eine Methode zum Lösen eines Datenkonflikts, bei der das fehlende Datenelement aus internen Pufferspeichern abgerufen und nicht darauf gewartet wird, bis dieses aus den für den Programmierer sichtbaren Registern oder aus dem Speicher kommt. (1p)
- Beitrag: Durch das Umsortieren der Befehle im vorangegangenen Aufgabenteil rücken Addition und Subtraktion näher zusammen, was zu einem weiteren Datenkonflikt führt. Forwarding ermöglicht ein Übertragen des Ergebnisses der Addition aus einem Puffer direkt in die Subtraktion. Der dazwischen liegende Store-Befehl liefert den erforderlichen Zeitrahmen, den die Pipeline für diesen Vorgang benötigt. Somit ist ein Piple-Bubble unnötig und der Datenkonflikt ist gelöst. (1p)