Karlsruher Institut für Technologie
Institut für Technische Informatik
Prof. Dr. Wolfgang Karl

Klausur Rechnerstrukturen
Sommersemester 2012
Musterlösung

Aushang der Ergebnisse: ab Mitte September 2012

Musterlösung 1: Verbindungsstrukturen & Vektorrechner **10P**

Verbindungsstrukturen **5P**

a) *1P*
- Einfache Erweiterbarkeit um wenige Knoten (eine Schicht)
- Der Verbindungsgrad der Knoten bleibt konstant 4, d.h. es müssen an schon existierenden Knoten keine zusätzlichen Verbindungen hinzugefügt werden.
- Der Durchmesser des Netzwerks steigt nur linear.
- Keine Änderung am Routing
- Die wesentlichen Eigenschaften des Netzwerks ändern sich nicht
- ...

b) *1P*
- Knotenzahl: $N = 2^n$
  $\Rightarrow$ Verdopplung der Knotenzahl mit jeder Erweiterung
- Verbindungsgrad: $log_2 N = log_2\, 2^n = n$
  $\Rightarrow$ der Verbindungsgrad aller Knoten nimmt bei jeder Erweiterung um 1 zu

c) Ein Knoten in der i-ten Dimension mit der Adresse $a_0, a_1, \ldots, a_{n-1}$ kann erreicht werden von Nachbarknoten mit den Adressen $a_0, a_1, \ldots, a_{i\pm1} mod\, k, \ldots, a_{n-1}$ *1P*

d) p Eingänge, p Ausgänge, k Stufen mit p/2 Zweierschalter *1,5P*

e) Sie weisen im Vergleich zu regulären Permutationsnetzen Lücken auf. *0,5P*

Vektorverarbeitung: **5P**

f) *3P*

```
MOV  R1, 64          # R1 mit 64 initialisieren
MTC1 VLR, R1         # vector-length register := 64
LV   V1, Ra          # int a[n] in V1 laden
LV   V2, Rb          # int b[n] in V2 laden
MOV  R2, 0           # R2 mit 0 initialisieren
ADDVS.I  V3, V1, R2  # Kopiere V1 in V3
MOV  R1, 0xff        # R1 mit 0xff initialisieren
SEQVS.I  V1, R1      # Vergleich mittels 'equals'
                     # if true 1 in Vektor Mask Register else 0
SUBV.I   V3, V3, V3  # Komponenten von V3 mit 1
                     # im VMR werden auf 0 gesetzt
ADDV.I   V3, V3, V2  # diese Komponenten werden mit
                     # den Werten aus V2 aufgefuellt c[i] = b[i];
CVM                  # Clear Vektor Mask -- VMR auf 1 setzen
SV  Rc, V3           # Schreiben von c[i]
```

g) Es handelt sich um eine abgewandelte Form des Blue-Box-Verfahrens, wenn in Vektor a[n] Pixel eines Bildes vor blauem Hintergrund gespeichert sind, wobei die blauen Pixel den Wert 0xff haben. *1P*

h) *1P*

- Stride von 1:

  Latenz von 6 Zyklen für erstes Element und 31 Zyklen für alle weiteren zu holenden Elemente, da bei 8 Speicherbänken jeder nächste Zugriff auf die jeweils nächste Bank geht und somit die Latenz von 6 Takten versteckt werden kann. Folglich benötigt man 37 Zyklen.

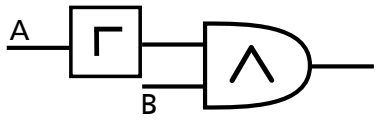
- Stride von 8:

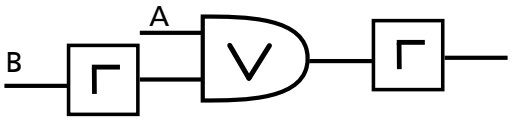
  Da 8 gleich 8 (der Anzahl der Bänke) ist, handelt es sich hier um den schlechtesten Fall. Jeder Zugriff des Strides geht auf die gleiche Memory Bank und kollidiert mit dem vorhergehenden. Damit benötigt jeder Speicherzugriff die Latenz von 6 Takten und man benötigt insgesamt: $6 * 32 = 192$ Takte

Musterlösung 2: Low-Power-Entwurf & Rechnerbewertung **10P**

Low-Power-Entwurf **4P**

a) Schaltbilder: *1P*





b) Signalwahrscheinlichkeiten:Variante 1: *1P*

$\mathbb{P}_{\neg A\ Ausgang=1} = 1 - \frac{3}{4} = \frac{1}{4}$
$\mathbb{P}_{\neg A\ \wedge B\ Ausgang=1} = \frac{1}{4} * \frac{1}{2} = \frac{1}{8}$

Variante 2:

$\mathbb{P}_{(A \vee \neg B)\ Ausgang=1} = 1 - \frac{1}{4} * (1 - \frac{1}{2}) = 1 - \frac{1}{8} = \frac{7}{8}$
$\mathbb{P}_{\neg(A \vee \neg B)\ Ausgang=1} = 1 - \frac{7}{8} = \frac{1}{8}$

c) Schaltwahrscheinlichkeiten:Für beide Schaltungsvarianten ergibt sich mit der bekannten Formel: $\mathbb{P}_{Schalt} = 2 * \mathbb{P}(1)(1 - \mathbb{P}(1)) = 2 * \frac{1}{8} * \frac{7}{8} = \frac{7}{32}$ *1P*

d) Bewertung:Problem: Formel berücksichtigt Inverter nicht! Da jeder Inverter das anliegende Signal invertiert und somit bei jedem Signalwechsel unbedingt schaltet, müssen diese Schaltvorgänge mit berücksichtigt werden. Die bisherige Berechnung anhand der Signalwahrscheinlichkeiten gibt dies nicht her. Somit ist trotz gleicher Berechnung der Schaltwahrscheinlichkeiten, die Variante 1 vorzuziehen, da sie die geringere Zahl an Invertern aufweist. *1P*

Es wurde auch als richtig bewertet, wenn man die Schaltwahrscheinlichkeit für jede Stufe des Schaltbilds einzeln berechnet und zur Bewertung aufsummiert hat. Die Ergebnisse sind dann analog zu oben:

Variante 1: $\mathbb{P}_{\neg A} = \frac{3}{8}$, $\mathbb{P}_{\neg A \wedge B} = \frac{7}{32}$, $\mathbb{P}_{sum} = \frac{19}{32}$.

Variante 2: $\mathbb{P}_{\neg B} = \frac{1}{2}$, $\mathbb{P}_{A \vee \neg B} = \frac{7}{32}$, $\mathbb{P}_{\neg(A \vee \neg B)} = \frac{7}{32}$, $\mathbb{P}_{sum} = \frac{30}{32}$

Leistungsbewertung **2,5P**

e) Rechnung: Es ändern sich nur die Zyklen pro Instruktion, Taktrate und Anzahl der *1,5P*

Instruktionen bleiben gleich. Der unoptimierte CPI-Wert errechnet sich nach:

$$CPI_{base} = \sum_{i=1}^{n} CPI_i * \frac{IC_i}{InstructionCount_{total}} = (2 * 0,75) + (4 * 0,25) = 2,5.$$

Die Zyklen pro Instruktion mit neuem ADD: CPI_{newADD} kann durch Abziehen der gesparten Zyklen erfolgen:

$$\begin{aligned} CPI_{newADD} &= CPI_{base} - p_{ADD} * (CPI_{oldADD} - CPI_{newADD}) \\ &\Rightarrow 2,5 - 0,50 * (1,5 - 1) = 2,25. \end{aligned}$$

Die Alternative mit dem CPI_{newInt}-Wert errechnet sich analog zum CPI_{base}:

$$CPI_{newInt} = (0,75 * 1,5) + (0,25 * 4) = 2,125 = 2\frac{1}{8}.$$

f) Begründung:Aufgrund des geringeren CPI-Werts bietet sich die Alternative 2 mit den verbesserten Zyklen pro Integeroperation an. *1P*

Fehlertoleranz **3,5P**

Berechnen Sie λ, so dass die MTTF $\frac{2}{3}$ beträgt. Eingesetzt:

$$\int_0^\infty (6 * R(t)^2 - 8 * R(t)^3)\,\mathrm{dt} = \frac{2}{3}$$
$$\leftrightarrow \int_0^\infty (6 * (e^{-\lambda t})^2 - 8 * (e^{-\lambda t})^3)\,\mathrm{dt} = \frac{2}{3}$$
$$\leftrightarrow \int_0^\infty (6 * e^{-2\cdot\lambda t} - 8 * e^{-3\cdot\lambda t})\,\mathrm{dt} = \frac{2}{3}$$
$$\leftrightarrow \Big[-\frac{6}{2\lambda} * e^{-2\cdot\lambda t} + \frac{8}{3\lambda} * e^{-3\cdot\lambda t} \Big]_0^\infty = \frac{2}{3}$$
$$\leftrightarrow \Big[-\frac{3}{\lambda} * e^{-2\cdot\lambda t} + \frac{8}{3\lambda} * e^{-3\cdot\lambda t} \Big]_0^\infty = \frac{2}{3}$$
$$\leftrightarrow \Big[-\frac{9}{3\lambda} * e^{-2\cdot\lambda t} \Big]_0^\infty + \Big[\frac{8}{3\lambda} * e^{-3\cdot\lambda t} \Big]_0^\infty = \frac{2}{3}$$
$$\leftrightarrow \Big[0 + \frac{9}{3\lambda} * 1 \Big] + \Big[0 - \frac{8}{3\lambda} * 1 \Big] = \frac{2}{3}$$
$$\leftrightarrow \frac{9}{3\lambda} - \frac{8}{3\lambda} = \frac{2}{3} \rightarrow \underline{\lambda = \tfrac{1}{2}}$$

Musterlösung 3: Speicherhierarchie **9P**

Cache-Kohärenzprotokoll MOESI **6P**

a) *4P*

Zeile	Prozessor	Aktion	Prozessor 1		Prozessor 2		Prozessor 3	
			Line 1	Line 2	Line 1	Line 2	Line 1	Line 2
0.		init	-	-	-	-	-	-
1.	3	wr 1					1/M	
2.	2	wr 2			2/M			
3.	1	rd 3	3/E					
4.	2	wr 5				5/M		
5.	3	wr 2			2/I			2/M
6.	1	rd 1		1/S			1/O	
7.	1	rd 5	5/S			5/O		
8.	3	wr 4					4/M	
9.	3	rd 1		1/S				1/S
10.	3	rd 2					2/E	
11.	2	rd 4			4/E			

b) Das MOESI-Protokoll führt zu einer beschleunigten Abarbeitung: in Zeile 6 und 7 werden die Daten direkt durch einen Cache-zu-Cache-Transfer übertragen. Dies spart jeweils 2 Speicherzugriffe (1 lesend/1 schreibend). *1P*

c) Falls die LRU-Strategie MOESI-Zustandsänderungen als Zugriffe berüchsichtigt, würde der Zustandsübergang (M->O) in Zeile 6 von Prozessor 3 den LRU-Zähler für Zeile 1 zurücksetzen. Damit würde in Zeile 8 die Line 2 ersetzt anstatt der Line 1. Damit erreicht man in Zeile 9 als auch Zeile 10 einen lokalen Cache-Hit statt eines Cache-Misses. Im Fall von Zeile 10 erspart man sich einen Speicherzugriff. *1P*

Cache-Leistung **3P**

d) *2P*

Allgemein: $t_a = r_{H-L1} * t_{H-L1} + (1 - r_{H-L1}) * (r_{H-L2} * t_{H-L2} + ((1 - r_{H-L2}) * t_{Mem})))$

fester Teil: $(r_{H-L2} * t_{H-L2} + ((1 - r_{H-L2}) * t_{Mem}))$

eingesetzt: $0,8 * 25\,ns + 0,2 * 125\,ns = 20\,ns + 25\,ns = 45\,ns$

- Alternative A: $0,7 * 4\,ns + 0,3 * 45\,ns = 16,3\,ns$,
- Alternative B: $0,8 * 6\,ns + 0,2 * 45\,ns = 13,8\,ns$,
- Alternative C: $0,4 * 2\,ns + 0,6 * 45\,ns = 27,8\,ns$,

Damit erreichen Alternative A und B die Vorgabe.

e) Da auch im Fall eines Cache-Hits in den höheren Ebenen Anfragen an den Hauptspeicher gestartet werden, die dann später wieder abgebrochen werden, blockieren diese den Bus und verhindern so, dass notwendige Daten zu anderen Prozessoren transferiert werden. Dadurch hat dieses System eine beschränkte Skalierbarkeit. *1P*

Musterlösung 4: Fertigung und Hardwareentwurf **10P**

Fertigungskosten **5P**

a) $yield_{die} = yield_{wafer} * (1 + \frac{dpua * a_{die}}{\alpha})^{-\alpha}$ *1P*

b) $dpw = A - B = \frac{\pi * (d_{wafer}/2)^2}{a_{die}} - \frac{\pi * d_{wafer}}{\sqrt{2 * a_{die}}}$ *1P*
A: theoretisches Maximum
B: Verschnitt

c) $cost_{die} = \frac{cost_{wafer}}{dpw * yield_{die}}$ ($\frac{1}{2}$P) *2.5P*
$\frac{cost_{wafer-300mm}}{dpw_{300mm} * yield_{die-300mm}} = \frac{cost_{wafer-400mm}}{dpw_{400mm} * yield_{die-400mm}}$ ($\frac{1}{2}$P)

$\Rightarrow$
$yield_{die-400mm} = \frac{cost_{wafer-400mm} * dpw_{300mm} * yield_{die-300mm}}{cost_{wafer-300mm} * dpw_{400mm}}$ ($\frac{1}{2}$P)

$yield_{die-400mm} = \frac{1000EUR * 300 * 0.6}{900 * 500} = \frac{12}{30} = 0.4$ ($\frac{1}{2}$P)

$\Rightarrow$
$yield_{die-400mm}$ muß größer als 0.4 sein, damit die Umstellung profitabel ist. ($\frac{1}{2}$P)

d) $cost_{IC} = \frac{cost_{die} + cost_{die-test} + cost_{packaging}}{yield_{final}}$ *0.5P*

Entwurfsautomatisierung **5P**

e) Vorteile: *2P*

- Flexibilität
- Weniger Fehleranfällig

Nachteile:

- Randbedingungen können nur schwer eingehalten werden
- Syntheseergebnisse oft schlechter als manueller Entwurf

Hinweis: In der Vorlesung wurden noch weitere Vor- und Nachteile genannt.

f) Syntheseschritte: *3P*

- High-Level-Synthese
- Logik-Synthese
- Layout-Synthese

Beschreibungen:

- RT-Beschreibung
- Gatter-Beschreibung
- Geometrie-Beschreibung

Musterlösung 5: Parallelverarbeitung und Architekturen **10P**

a) *1P*
- $E(n) = \frac{S(n)}{n}$
- $U(n) = \frac{I(n)}{n} = R(n) * E(n) = \frac{P(n)}{n*T(n)}$

b) *1,5P*
- Die maximal erreichbare Beschleunigung wird durch den Anteil des Programms, der nur sequentiell ausgeführt werden kann, begrenzt. (0,5P)
- Amdahls Gesetz (Je 0,5P für Formel und Erklärung):

$$T(n) = \underbrace{\frac{T(1)}{n} * (1-a)}_{1} + \underbrace{T(1) * a}_{2}$$

Die Formel zerfällt in die Ausführungszeiten des parallel ausführbaren Programmteils 1 und des rein sequentiell ausführbaren Programmteils 2.

c) *1P*
- Prozessor: Physikalische Ressource
- Prozess: Abstraktion, Virtualisierung von einem Multiprozessor

d) Die Anzahl der Prozesse mus nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein *0,5P*

e) Eine zu feine Granularität führt zu einem hohen Veraltungsaufwand (Overhead) durch gesteigerte Kommunikation. *1P*

f) *1P*
- Shared-Memory-Programmiermodell: OpenMP
- Nachrichtenorientiertes Programmiermodell: MPI

g) *2P*
- Collective Network: Broadcast- and Reduktions-Funktionalität
- Multiprozessor mit verteiltem Speicher (Cluster), NoRMA

h) Energie, Speicher, Zuverlässigkeit, Parallelität und Lokalität *2P*

Musterlösung 6: Rechnerarchitektur 11P

Parallelismus auf Befehlsebene 6P

a) Completion bedeutet, dass der Befehl abgearbeitet wurde und darauf wartet in der ursprünglichen Befehlsfolge gültig gemacht zur werden (Commitment). Das Commitment kann durch die eine Exception($\frac{1}{2}$P), die durch einen vorhergehenden Befehl ausgelöst wird, oder aber durch einen Interrupt($\frac{1}{2}$P), der zeitlich vor dem betreffenden Befehl eingeordnet wird, verhindert werden. *1P*

b) *5P*

Feld	R1	R2	R3	R4
Value	-	-	-	(R1+R3)
Valid	0	0	0	1
RS	Mul 1	Int 2	Div 1	-

Unit	Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
Int 1	1						-			-
Int 2	0	1	add	R2	(R1)	1	-	(R2)	1	-
Mul 1	0	0	mul	R1		0	Int 2		0	Div 1
Div 1	0	1	div	R3	(R2)	1	-		1	Int 1

Sprungvorhersage 5P

c) Tabelle: *4P*

	S1				S2		
	Init	Vhs.	Sprung	Präd neu	Vhs.	Sprung	Präd neu
1	(**NT**, NT)	NT	T	(T, **NT**)	NT	T	(T, T)
2	(T, **T**)	T	NT	(**T**, NT)	T	NT	(NT, NT)
3	(**NT**, NT)	NT	T	(T, **NT**)	NT	T	(T, T)
4	(T, **T**)	**T**	T	(T, **T**)	**T**	T	(T, T)

d) Antwort: Da der Ausgang der beiden Sprünge jeweils auf den gleichen Prädiktorensatz abgebildet wird($\frac{1}{2}$P), hat man hier das Problem der gegenseitigen Beeinflussung, auch Aliasing genannt($\frac{1}{2}$P). *1P*