Zentralübung Rechnerstrukturen im SS2007

Prozessorarchitektur

Dr. Rainer Buchty
buchty@ira.uka.de

Universität Karlsruhe (TH) – Forschungsuniversität
Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

16.05.2006

Einleitung

- **Abarbeitungsbeschleunigung durch**
  - Pipelining
  - Interne Parallelverarbeitung
- **Problem: Theoretisch maximale Beschleunigung unerreichbar**
  - Datenabhängigkeiten
  - Ressourcenkonflikte
  - Kontrollflußabhängigkeiten
- **Evaluierung: Wie schnell ist das System wirklich?**
  - Idealbedingungen
  - Einfluß von Konfliktsituationen
- **Behandlung auftretender Konfliktsituationen**

Pipelining

- **Beschleunigung $\sim$ Anzahl der Pipelinestufen**
- Bei gefüllter Pipeline: Single-cycle Execution (CPI=1)
- **CPI=1 unerreichbar** aufgrund von
  - **Anlauf / Leerlauf**
  - Wartezeiten (**Stalling**)
    durch Abhängigkeiten bzw. Konflikte (**Hazards**)
  - **Wiederanlauf** nach Pipeline-Flush

Formelwerk (Idealbedingungen)

- **Ausführungszeit:**

$$t_{exec} = \frac{t_{execNP}}{\#Pipeline-Stufen}$$

- **Beschleunigung (Speed-up):**

$$s = \frac{t_{avgNP}}{t_{avgP}}$$

Rechenbeispiel 1

Die Ausführungszeit eines Befehls betrage 5 Taktzyklen. Vergleichen Sie die Auswirkungen durch die Einführung einer 5- bzw. 10-stufigen Befehlspipeline.

Rechenbeispiel 1

Die Ausführungszeit eines Befehls betrage 5 Taktzyklen. Vergleichen Sie die Auswirkungen durch die Einführung einer 5- bzw. 10-stufigen Befehlspipeline.

- Ausführungszeit: $t_{exec} = \frac{t_{execNP}}{\#Pipeline-Stufen}$

  $t_5 = \frac{t_1}{5} = \frac{5}{5} = 1$

  $t_{10} = \frac{t_1}{10} = \frac{5}{10} = 0.5$

Rechenbeispiel 1

Die Ausführungszeit eines Befehls betrage 5 Taktzyklen. Vergleichen Sie die Auswirkungen durch die Einführung einer 5- bzw. 10-stufigen Befehlspipeline.

- Ausführungszeit: $t_{exec} = \frac{t_{execNP}}{\#Pipeline-Stufen}$

  $t_5 = \frac{t_1}{5} = \frac{5}{5} = 1$

  $t_{10} = \frac{t_1}{10} = \frac{5}{10} = 0.5$

- Beschleunigung (Speed-up): $s = \frac{t_{avgNP}}{t_{avgP}}$

  $s_5 = \frac{t_1}{t_5} = \frac{5}{1} = 5$

  $s_{10} = \frac{t_1}{t_{10}} = \frac{5}{0.5} = 10$

Rechenbeispiel 2

Die 5 Pipelinestufen einer Architektur könnten theoretisch mit folgenden Zykluszeiten betrieben werden: 8ns (IF), 5ns (ID), 10ns (MA), 7ns (EX), 2ns (WB). Welches ist die resultierende Taktgeschwindigkeit für die Architektur?

Rechenbeispiel 2

Die 5 Pipelinestufen einer Architektur könnten theoretisch mit folgenden Zykluszeiten betrieben werden: 8ns (IF), 5ns (ID), 10ns (MA), 7ns (EX), 2ns (WB). Welches ist die resultierende Taktgeschwindigkeit für die Architektur?

- Die **langsamste Pipelinestufe** bestimmt den Gesamttakt.
- MA ist langsamste Pipelinestufe mit 10ns, daher 100MHz Gesamttakt.

Pipelining (forts.)

- **Idealbedingungen in realen Systemen nicht erzielbar** (Hazards)
- Anhalten der Pipeline verringert CPI spürbar
- Auswirkungen auf Speedup

Einfluß von Hazards

- $CPI_P = CPI_{ideal} + \#Stall\ Clock\ Cycles/Instruction$
- Speedup $s = \frac{t_{avgNP}}{t_{avgP}} = \frac{CPI_{NP}}{CPI_P} * \frac{t_{cycNP}}{t_{cycP}}$

  $\rightarrow s = \frac{CPI_{NP}}{1+Stall\ Clock\ Cycles/Instr.}$
- Für perfekt ausbalanzierte Pipeline:

  Pipeline-Tiefe: $pd = \frac{t_{cycNP}}{t_{cycP}}$

  $\rightarrow s = \frac{1}{1+Stall\ Clock\ Cycles/Instr.} * pd$

Rechenbeispiel 3

Ein Instruktionsmix beinhalte 44% Datenzugriffe. Berechnen Sie ausgehend von einer idealen, konfliktfreien Architektur mit einem CPI von 1 die durchschnittliche Ausführungszeit t_{avg} für diesen Befehlsmix, wenn die konfliktbehaftete – d.h. Datenzugriffe verursachen eine Latenz von einem Taktzyklus – Architektur eine gegenüber der idealen Maschine 1,2-fach geringere Zykluszeit t_{cct} aufweist.

Rechenbeispiel 3

Ein Instruktionsmix beinhalte 44% Datenzugriffe. Berechnen Sie ausgehend von einer idealen, konfliktfreien Architektur mit einem CPI von 1 die durchschnittliche Ausführungszeit t_{avg} für diesen Befehlsmix, wenn die konfliktbehaftete – d.h. Datenzugriffe verursachen eine Latenz von einem Taktzyklus – Architektur eine gegenüber der idealen Maschine 1,2-fach geringere Zykluszeit t_{cct} aufweist.

- $t_{avg} = CPI * t_{cct}$
  $= (1 + 0.44 * 1) * \frac{t_{cct}(ideal)}{1.2}$
  $= 1.2 * t_{cct}(ideal)$

Rechenbeispiel 4

Eine Architektur ohne Pipeline habe eine Taktgeschwindigkeit von 200MHz. Sie verwende 5 Zyklen pro ALU-Instruktion und Speicherzugriffe sowie 7 Zyklen für Sprünge. In einem Instruktionsmix betrage der jeweilige Anteil 50%, 30% und 20%.
Diese Architektur werde in eine Architektur mit Pipeline umgewandelt, hierdurch wird die Taktrate um 4ns gesenkt, d.h. der einzelne Taktzyklus verlangsamt sich. Berechnen Sie den zu erwartenden Geschwindigkeitszuwachs.

Rechenbeispiel 4

Eine Architektur ohne Pipeline habe eine Taktgeschwindigkeit von 200MHz. Sie verwende 5 Zyklen pro ALU-Instruktion und Speicherzugriffe sowie 7 Zyklen für Sprünge. In einem Instruktionsmix betrage der jeweilige Anteil 50%, 30% und 20%.
Diese Architektur werde in eine Architektur mit Pipeline umgewandelt, hierdurch wird die Taktrate um 4ns gesenkt, d.h. der einzelne Taktzyklus verlangsamt sich. Berechnen Sie den zu erwartenden Geschwindigkeitszuwachs.

- $t_{avg} = t_{cyc} * CPI_{avg}$
  $= 5ns * (0.8 * 5 + 0.2 * 7) = (5 * 5.4)ns = 27ns$
- $s = \frac{t_{avgNP}}{t_{avgP}} = \frac{27ns}{5ns+4ns} = 3$

Beispiel: Stalling

Die Funktion a=b+c soll auf einer Architektur mit bekannter 5-stufiger DLX-Pipeline implementiert werden. Erarbeiten Sie eine Lösung mit 4 Befehlen, wobei Operanden und Ergebnis im Speicher stehen sollen. Wir wird diese Befehlsfolge abgearbeitet unter der Annahme, daß Speicherzugriffe einen Taktzyklus Latenz bewirken?

Beispiel: Stalling

Die Funktion a=b+c soll auf einer Architektur mit bekannter 5-stufiger DLX-Pipeline implementiert werden. Erarbeiten Sie eine Lösung mit 4 Befehlen, wobei Operanden und Ergebnis im Speicher stehen sollen. Wir wird diese Befehlsfolge abgearbeitet unter der Annahme, daß Speicherzugriffe einen Taktzyklus Latenz bewirken?

Befehl	**Pipeline**								
`LOAD R1,B`	IF	ID	EX	MA	WB				
`LOAD R2,C`		IF	ID	EX	MA	WB			
`ADD R3,R1,R2`			IF	ID	**	EX	MA	WB	
`STORE A,R3`				IF	**	ID	EX	MA	WB

Trugschlüsse

falsch: ***Prozessoren mit niedrigem CPI sind grundsätzlich schneller.***

- Ein kleiner CPI-Wert ist sicherlich positiv, aber wird in aller Regel durch schwergewichtige Pipelinestufen erkauft.
- Solche Pipelines haben typischerweise eine niedrigere Taktfrequenz als Pipelines mit einfachen Pipelinestufen.
- Nur bei geringer Parallelitätsausnutzung ist Taktfrequenz maßgeblich.
- Verlängerung der Pipeline zwecks Frequenzerhöhung birgt Risiken (Abhängigkeiten, Pipeline Flush)

Konfliktvermeidung – Dealing with Hazards

- Pipelining zur überlappten Ausführung von Befehlen
- Konflikte verringern theoretischen Maximaldurchsatz

- **Datenabhängigkeiten** auflösbar durch
  - Result Forwarding (data dependence)
  - Umbenennung (name dependence: kein Datentransfer, nur Ressourcenkonflikt)
- **Strukturkonflikte** auflösbar durch
  - Code-Umordnung (soweit möglich)
  - Hardware-Erweiterung wie Multiport-Register bzw. Vervielfachung von Einheiten (teuer)
- Problem: **Steuerkonflikte**

Steuerkonflikte

- **Steuerkonflikt: Resultat einer nicht-sequentiellen Veränderung des Programmzählers**
  - Unbedingte Sprünge
  - Bedingte Sprünge
  - auch: externe Interrupts
- **Unbedingte Sprünge**
  - `JMP` / `BRA`, `JSR` / `BSR` / `CALL`,
  - Software-Traps (Aufruf von Systemroutinen) (`BRK`, `SWIx`, `INT x`)
  - Leicht auflösbar durch sogenannten *delayed branch*
  - Beispiel:


```
LOAD  r1,#5                LOAD  r1,#5
LOAD  r2,#8                JUMP  somewhere
ADDC  r3,r1,r2     ⟶      LOAD  r2,#8
JUMP  somewhere            ADDC  r3,r1,r2
...                        ...
```

Elimination von Sprüngen

- Einfachste Lösung: **Sprünge vermeiden**
- Ausrollen von Schleifen
- **Nachteil:** Codegröße, statisches Verfahren
- Neues Problem: **Schleifengetragene Abhängigkeiten über indexierte Speicherzugriffe**

```
for(i=0;i<100;i++)
{
                        A[i]=B[i]+C[i];
                        D[i]=A[i]*E[i];
}
```

Elimination von Sprüngen (forts.)

```
for(i=0;i<100;i++)
{
                              A[i]=B[i]+C[i];
                              D[i]=A[i]*E[i];
}
```

- A[i] kann ausgerollt werden, da keine Abhängigkeiten
- D[i] basiert direkt auf A[i], Datenabhängigkeit
- Sonderfall: Y[i]=Y[i-1]+Y[i] (**recurrence**)
- Abhängigkeitsdistanz (**dependency distance**) definiert ausnutzbaren Parallelismus (wichtig für parallele Architekturen)

Elimination von Sprüngen (forts.)

GCD-Test

- Erkennen potentieller Abhängigkeiten
- Annahme: Array-Indizes sind affin $\rightarrow$ `a*i+b` mit `i` als Laufindex
- Vergleich zweier Zugriffe `a*i+b` und `c*i+d` mit Test, ob `(d-b) mod GCD(c,a)=0`
- Falls Bedingung erfüllt, existiert potentielle Abhängigkeit

- **Caveat emptor!**
  - Pointer statt Indizes, indirekte Zugriffe (Erkennbarkeit)
  - Test nur für den negativen Fall aussagekräftig
  - Erkennen multipler Abhängigkeiten

Elimination von Sprüngen (forts.)

Beispiel: GCD-Test

Ermitteln Sie mithilfe des GCD-Verfahrens, ob nachfolgender Code schleifengetragene Abhängigkeiten (loop-carried dependences) enthält oder nicht. Wie ist das Ergebnis zu werten?

```
for(i=0;i<100;i++)
{
  X[3*i+4]=X[2*i+5]*3;
}
```

Elimination von Sprüngen (forts.)

Beispiel: GCD-Test

Ermitteln Sie mithilfe des GCD-Verfahrens, ob nachfolgender Code schleifengetragene Abhängigkeiten (loop-carried dependences) enthält oder nicht. Wie ist das Ergebnis zu werten?

```
for(i=0;i<100;i++)
{
  X[3*i+4]=X[2*i+5]*3;
}
```

- X[3*i+4] $\rightarrow$ a=3, b=4
- X[2*i+5] $\rightarrow$ c=2, d=5
- GCD(3,2)=1, (d-b)=1 $\rightarrow$ 1 dividiert 1, also ist Abhängigkeit möglicherweise enthalten.

Elimination von Sprüngen (forts.)

Beispiel: GCD-Test (forts.)

- Das GCD-Verfahren sagt nur definitiv aus, ob Abhängigkeiten **nicht** enthalten sind: Schlägt der Test fehl, sind garantiert keine Abhängigkeiten enthalten.
- Gelingt der Test, können Abhängigkeiten enthalten sein, es ist aber nicht zwingend, da GCD z.B. die Schleifengrenzen nicht überprüft.

Sprungvorhersage

- Problem: **Bedingte Sprünge**
  - Ergebnis der Auswertung erst nach EX-Phase
  - Pipeline-Stalling bis Auswertung
  - Minimierung des Pipeline-Durchsatzes
  - Alternative: Sprungvorhersage
- **Zwei grundsätzliche Arten** von Sprungvorhersage
  - Statische Vorhersage (feste Sprungannahme)
  - Dynamische Sprungvorhersage
- Ziel: **Umschiffung des Steuerkonflikts** durch korrekte Sprungvorhersage
  - Maximierung des Pipeline-Durchsatzes
  - Minimierung von Flush/Wiederanlauf

Statische Sprungvorhersage

- **Keine Vorhersage im eigentlichen Sinn, sondern Definition des Sprungverhaltens:**
  - always not taken (i386)
  - always taken (i486)
  - Abhängig von Sprungrichtung (PPC405)
- Sprungvorhersage **basiert auf Schleifenannahme**
  - Rücksprung in den Schleifenkörper nehmen
  - Aussprung durch Abbruchkriterium nicht nehmen
  - Bei einfachem Modell somit nur Fehlvorhersage für Abbruchkriterium
- Für besten Erfolg **Compilerunterstützung notwendig**
  - Umformung der hochsprachlichen Schleifen analog zur verwendeten Sprungvorhersage
  - Ggf. Umformulierung von Fallunterscheidungen
  - Anpassung des Programms an Sprungvorhersage

Dynamische Sprungvorhersage

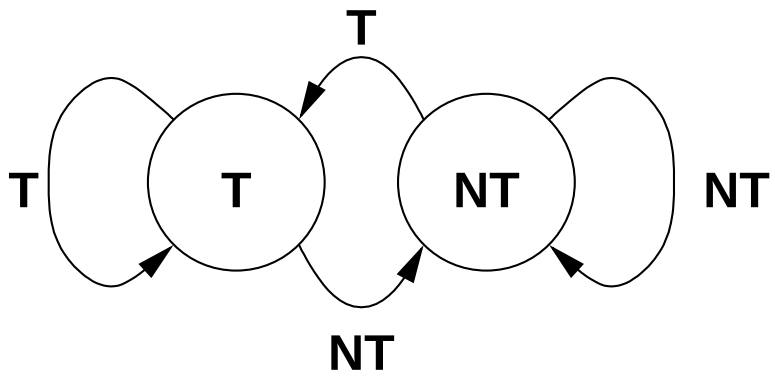
- **Anpassung der Sprungvorhersage an laufendes Programm**
- Einführung einer **Sprunghistorie**

- Etliche **Konzepte mit variierender Komplexität**
  - 1-Bit-Prädiktor
  - 2-Bit-Prädiktor (Sättigungszähler und Hysteresemodell)
  - Korrelationsprädiktoren
  - Zweistufig adaptive Prädiktoren
  - gshare, gselect
  - Hybridprädiktoren

- Vorhersagegenauigkeit in realen Programmen weit besser als statische Vorhersage
- **Vorhersagegenauigkeit** typischerweise $>$95%

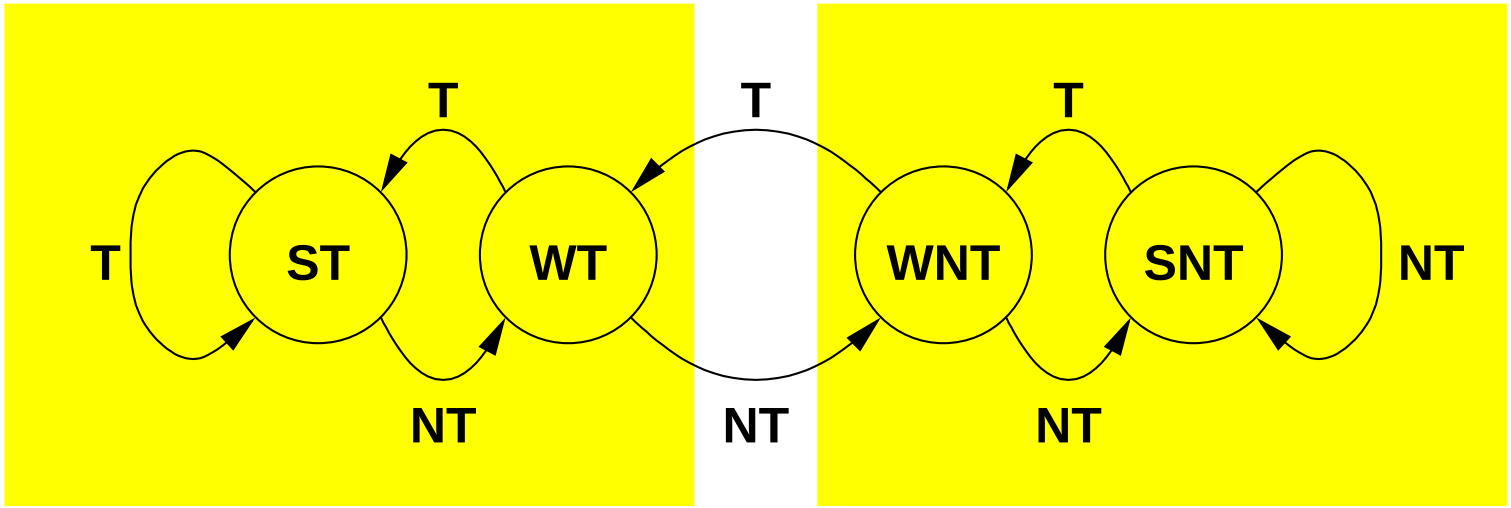
- Aber: **Hardwaretechnisch aufwendiger**

1-Bit-Prädiktor



- Zustandsautomat mit **2 Zuständen**:
  taken (T), not taken (NT)
- Aktueller **Zustand definiert Sprungvorhersage**
- Zustandsübergang anhand tatsächlichem Sprungverlauf
- daher: **Initialisierung entscheidet Anlaufverhalten**
- Problem: **Kurze Historie**
  - Zwei Fehlvorhersagen bei verschachtelten Schleifen:
    Schleifenende und Wiedereintritt

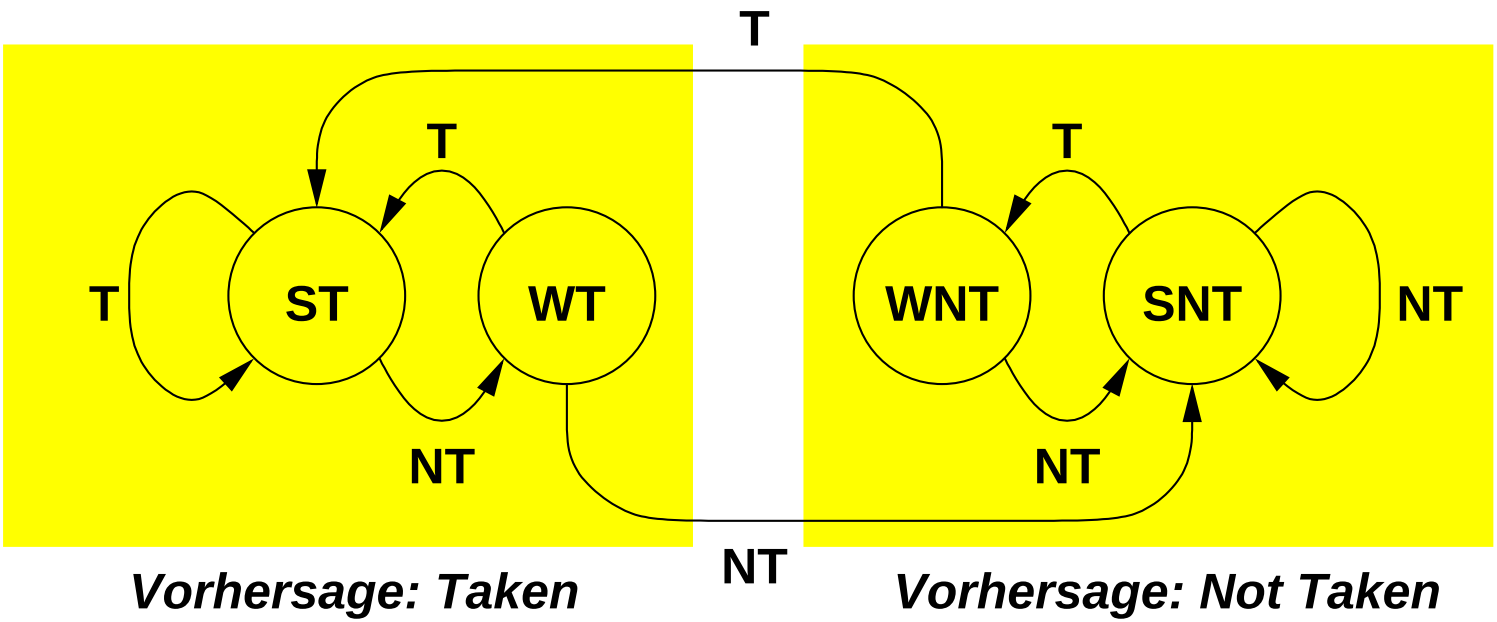
2-Bit-Prädiktor mit Sättigungszähler



Vorhersage: Taken ***Vorhersage: Not Taken***

- Zustandsautomat mit **4 Zustände**n:
  strongly taken (**ST**), weakly taken (**WT**), weakly not taken (**WNT**),
  strongly not taken (**SNT**)
- Wechsel der Vorhersage erst nach zwei Fehlvorhersagen
- Optimiert Schleifenaustritts- und Wiederanlaufverhalten

2-Bit-Prädiktor mit Hysterese



- Zustandsautomat mit **4 Zuständen**:
  strongly taken (**ST**), weakly taken (**WT**),
  weakly not taken (**WNT**), strongly not taken (**SNT**)
- Wechsel der Vorhersage erst nach zwei Fehlvorhersagen
- Aggressiveres Umschaltverhalten, vermeidet “Flattern”
  zwischen Weakly-Zuständen

n-Bit-Prädiktoren: Diskussion

- **Qualität der 2-Bit-Prädiktoren variiert** je nach Anwendungsgebiet
- **Sehr gute Leistung bei numerischen Problemen** (große Schleifen, kaum if/then/else-Konstrukte)
- **Anzahl Fehlspekulationen bei allgemeinen Anwendungen höher:** kurze Schleifen, Programmcode dominiert von Fallunterscheidungen
- Fehlspekulation bei SPEC89: zwischen 1% und 18% (gcc: 12%)
- **Weitere Vergrößerung bringt praktisch keine Verbesserung** der Sprungvorhersage
- **Ursache:** if/then/else-Konstrukte

Problemfall: Korrelation

Code:

```
if(d==0)        /* Sprung 1 */
    d=1;
if(d==1)        /* Sprung 2 */
    ...
```

d vor S1	d==0?	S1	d vor S2	d==1?	S2
0	Ja	NT	1	Ja	NT
1	Nein	T	1	Ja	NT
2	Nein	T	2	Nein	T

- Beispiel zeigt korrelierten Sprung (S1∼S2) mit 1-Bit-Vorhersage
- Korrelationsinformation nicht von n-Bit-Prädiktoren auswertbar
- Keine Kenntnis über Sprungumgebung

Korrelationsprädiktoren

- **n-Bit-Prädiktoren sind sprungzentrisch**
- Keine Betrachtung des umgebenden Sprungverlaufs
- **Einführung von Korrelationsprädiktoren** Anfang der 1990er-Jahre

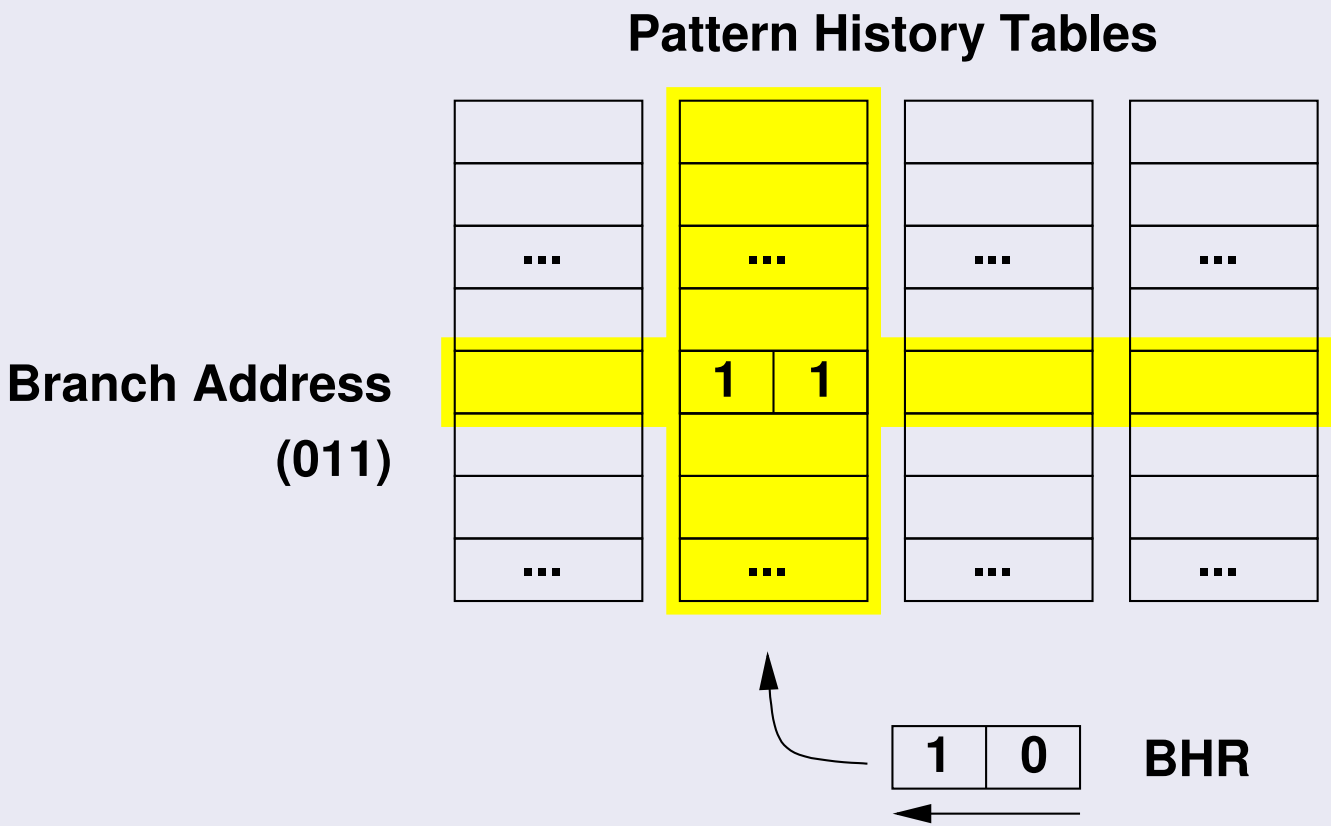
Korrelationsprädiktoren

- Pan et al., 1992: **(m,n)-Korrelationsprädiktoren** *(correlation-based predictors, correlating predictors)*
- Yeh und Patt, 1993: **Zweistufig adaptive Prädiktoren** *(bi-level adaptive predictors)*
- McFarling, 1994: **gselect** und **gshare**
- McFarling, 1993: **Hybridprädiktoren**

(m,n)-Korrelationsprädiktoren

- Einführung einer m Sprünge umfassenden Historie
- Speicherung in Sprungverlaufsregister bzw. **Branch History Register** (BHR)
- BHR ist m-Bit breites Schieberegister, hält den Ausgang der letzten m Sprünge
- Auswahl eines n-Bit-Prädiktors aus Sprungverlaufstabelle (**Pattern History Table**, PHT) anhand von Sprungadresse und BHR
- (Teil der) Sprungadresse selektiert Reihe
- BHR selektiert eine von 2^m Spalten
- Ausgewähltes Speicherfeld enthält n-Bit-Prädiktor

(2,2)-Korrelationsprädiktor



Beispielprogramm:

```
        LOAD R1,#0   ; R1=0
        LOAD R2,#2   ; R2=2
start:  CMP R1,R0    ; R1==0? (R0 ist ‘Zero”-Register)
        BRNZ l1      ; if (R1!=R0[0]) goto l1
        LOAD R1,#1   ; R1=1
l1:     CMP R1,R2    ; R1==R2?
        BRNZ l2      ; if (R1!=R2[2]) goto l2
        LOAD R1,#0   ; R1=0
        BRA start    ; goto start
l2:     LOAD R1,#2   ; R1=2
        BRA start    ; goto start
```

Sprungvorhersage (forts.)

Beispiel 1

Ermitteln Sie den Verlauf der Sprünge beim viermaligen Durchlaufen der Endlosschleife. Hierbei müssen die letzten beiden Sprünge (BRA start) nicht berücksichtigt zu werden: Ihre Bedingung ist immer wahr und somit wird immer gesprungen.

Sprungvorhersage (forts.)

Beispiel 1

Ermitteln Sie den Verlauf der Sprünge beim viermaligen Durchlaufen der Endlosschleife. Hierbei müssen die letzten beiden Sprünge (BRA start) nicht berücksichtigt zu werden: Ihre Bedingung ist immer wahr und somit wird immer gesprungen.

Sprung 1 BRNZ I1	Sprung 2 BRNZ I2	Sprung 3 JMP start	Sprung 4 JMP start
NT (R1=1)	T (R1=2)	-	T
T (R1=2)	NT (R1=0)	T	-
NT (R1=1)	T (R1=2)	-	T
T (R1=2)	NT (R1=0)	T	-

Analyse: Sprünge 1 und 2 sind voneinander abhängig!

Sprungvorhersage (forts.)

Beispiel 2

Benutzen Sie einen Ein-Bit-Prädiktor und initialisieren diesen mit NT für den ersten und T für den zweiten Sprung. Stellen Sie den Sprungverlauf dar und ermitteln Sie die Anzahl der Fehlannahmen.

Sprungvorhersage (forts.)

Beispiel 2

Benutzen Sie einen Ein-Bit-Prädiktor und initialisieren diesen mit NT für den ersten und T für den zweiten Sprung. Stellen Sie den Sprungverlauf dar und ermitteln Sie die Anzahl der Fehlannahmen.

Es werden **6** Fehlannahmen gemacht:

Sprung 1			Sprung 2		
Prädiktion	**Sprung**	**P. neu**	**Prädiktion**	**Sprung**	**P. neu**
NT	**NT**	NT	T	**T**	T
NT	**T**	T	T	**NT**	NT
T	**NT**	NT	NT	**T**	T
NT	**T**	T	T	**NT**	NT

Sprungvorhersage (forts.)

Beispiel 3

Wiederholen Sie obige Aufgabe mit einem Zwei-Bit-Prädiktor, welcher mit “weakly not taken” für den ersten und “weakly taken” für den zweiten Sprung initialisiert wird.

Beispiel 3

Wiederholen Sie obige Aufgabe mit einem Zwei-Bit-Prädiktor, welcher mit “weakly not taken” für den ersten und “weakly taken” für den zweiten Sprung initialisiert wird.

Es werden **4** Fehlannahmen gemacht:

Sprung 1			Sprung 2		
Prädiktion	**Sprung**	**P. neu**	**Prädiktion**	**Sprung**	**P. neu**
WNT	**NT**	SNT	WT	**T**	ST
SNT	**T**	WNT	ST	**NT**	WT
WN	**NT**	SNT	WT	**T**	ST
SNT	**T**	WNT	ST	**NT**	WT

Sprungvorhersage (forts.)

Beispiel 4

Führen Sie eine weitere Analyse unter Verwendung eines (1,1)-Korrelationsprädiktors durch. Gehen Sie hierbei davon aus, daß der Sprung vor Eintritt in die Endlosschleife nicht genommen wurde (NT) und die Initialisierung der Prädiktoren (NT,T) für den ersten und (T,NT) für den zweiten Sprung lautet.

Beispiel 4

Führen Sie eine weitere Analyse unter Verwendung eines (1,1)-Korrelationsprädiktors durch. Gehen Sie hierbei davon aus, daß der Sprung vor Eintritt in die Endlosschleife nicht genommen wurde (NT) und die Initialisierung der Prädiktoren (NT,T) für den ersten und (T,NT) für den zweiten Sprung lautet.

Es werden **keine** Fehlannahmen gemacht:

Sprung 1			Sprung 2		
Prädiktion	**Sprung**	**P. neu**	**Prädiktion**	**Sprung**	**P. neu**
(**NT**,T)	NT	(NT,T)	(**T**,NT)	T	(T,NT)
(NT,**T**)	T	(NT,T)	(T,**NT**)	NT	(T,NT)
(**NT**,T)	NT	(NT,T)	(**T**,NT)	T	(T,NT)
(NT,**T**)	T	(NT,T)	(T,**NT**)	NT	(T,NT)

Sprungvorhersage (forts.)

Analyse

Vergleichen Sie die erhaltenen Ergebnisse für Ein- und Zwei-Bit-Prädiktors mit dem (1,1)-Korrelations-Prädiktor. Begründen Sie die Unterschiede.

- Der 1-Bit Prädiktor wechselt nach jedem Sprung und führt daher immer Fehlannahmen aus.
- Beim 2-Bit Prädiktor kann das Wechseln der Vorhersage verhindert werden, darum ist nur jede zweite Vorhersage falsch.
- Der (1,1)-Korrelations-Prädiktor schneidet optimal ab, da alle Sprünge miteinander korreliert sind.

gselect und gshare-Prädiktoren

- Gehören zur Klasse der **Korrelationsprädiktoren**
- BHR und PHT
- **gselect**
  - Adressierung durch Konkatenation von BHR und Sprungbefehlsadresse (bzw. Teilen hiervon)
- **gshare**
  - Statt Konkatenation bitweise XOR-Verknüpfung von BHR und Sprungbefehlsadresse
  - Resultat: weniger Interferenzen bei gleicher Länge

Adressteil	BHR	gselect 4/4	gshare 8/8
00000000	00000001	00000001	00000001
00000000	00000000	00000000	00000000
11111111	00000000	11110000	11111111
11111111	10000000	11110000	01111111

Hybridprädiktoren

- Kombination mehrerer separater Prädiktoren
- Prädiktoren haben **unterschiedliche Stärken**, d.h. sind auf jeweils andere **Klasse von Sprungbefehlen** zugeschnitten
- **Kombinations- bzw. Hybridprädiktor**
  - Besteht aus zwei unterschiedlichen Prädiktoren und Selektor-Prädiktor
  - Selektor-Prädiktor wählt für Vorhersage zu verwendenden Prädiktor aus
- **McFarling**: Zwei-Bit-Prädiktor mit gshare
- **McFarling**: PAp-Prädiktor *(“local predictor”)* mit gshare
- Kombination besser als der beste von beiden bzw. Einzelprädiktoren gleicher HW-Kosten
- Beispiel Alpha 21264: SAg(10,1024) als Teil eines Hybridprädiktors

Hybridprädiktoren (forts.)

- **Young & Smith**: Compilerbasierte statische Sprungvorhersage mit zweistufig adaptivem Prädiktor
- Vermeidet Fehlvorhersage in Anlaufphase des kompletten Prädiktors
- **Grunwald et al.**: Selektorprädiktor nicht eigenständiger Prädiktor sondern Wahrscheinlichkeit der Fehlspekulation

Applicaton	commited instructions (millions)	conditional branches (millions)	taken branches (%)	misprediction rate (%)		
				SAg	gshare	combining
compress	80.4	14.4	54.6	10.1	10.1	9.9
gcc	250.9	50.4	49.0	12.8	23.9	12.2
perl	228.2	43.8	52.6	9.2	25.9	11.4
go	548.1	80.3	54.5	25.6	34.4	24.1
m88ksim	416.5	89.8	71.7	4.7	8.6	4.7
xlisp	183.3	41.8	39.5	10.3	10.2	6.8
vortex	180.9	29.1	50.1	2.0	8.3	1.7
jpeg	252.0	20.0	70.0	10.3	12.5	10.4
mean	**267.6**	**46.2**	**54.3**	**8.6**	**14.5**	**8.1**

Architekturüberblick

Sprungvorhersage	Beispielarchitekturen
keine Sprungvorhersage	Intel i8086, praktisch alle 8- & 16-Bit-Prozessoren sowie Mikrocontroller
Statische Verfahren	
always not taken	Intel i386
always taken	Intel i486, Sun SuperSPARC
Rücksprung genommen, Aussprung nicht	HP PA-7x00, PPC405
Dynamische Verfahren	
1-Bit-Prädiktor	DEC Alpha 21064, AMD K5
2-Bit-Prädiktor	PPC604, MIPS R10000, Cyrix 6x86, M2, NexGen 585, Motorola 68060
Bi-level Adaptive	Intel PPro, PII, AMD K6
gshare	Intel PIII, AMD Athlon
Hybridprädiktoren	DEC Alpha 21264
Prädikation	Intel IA-64, ARM, TI TMS320C6201, weitere DSPs
Mehrpfadausführung	IBM 360/91, IBM 3090 (Großrechner)

Leistungssteigerung – Zusammenfassung

- **Leistungssteigerung** durch
  - Taktrate
  - Überlappende Ausführung (Pipelining)
  - Intelligente Behandlung von potentiellen Konfliktsituationen (Sprungvorhersage, Eliminierung von Sprüngen durch Loop-Unrolling)
- **Weitere Leistungssteigerung** durch
  - Ausnutzung von inhärentem Befehlsparallelismus (Instruction Level Parallelism, ILP)
  - Benötigt Architekturen mit mehreren, parallelen Funktionseinheiten
  - Aber: Parallelisierbarkeit begrenzt (Amdahl's Law), weitere Einflüsse durch Speichersubsystem und Verbindungsnetz
- **Parallelisierung** kann erfolgen
  - Explizit durch Programmierer oder Compiler (VLIW, EPIC)
  - Mittels dedizierter Scheduling-Hardware (Superskalare Prozessoren)