Übung 7

Armin Kleinert und Ruth Höner zu Siederdissen

Aufgabe 1: Bedingte Ausführung

1. *Warum ist Sprungvorhersage sinnvoll?*

Sprungvorhersagen werden verwendet, um in der Pipeline entstehende “Control Hazards” zu minimieren. Diese entstehen, wenn ein Programm nicht linear abläuft, sondern bedingte Sprünge im Programm vorkommen, bei dem der nächste geladene Befehl von der Bedingung abhängig ist. In diesem Fall müsste man mit dem Reinladen der nächsten Befehle in die Pipeline warten, bis entschieden ist, ob die Bedingung erfüllt ist oder nicht. Dies ist allerdings nicht besonders effizient. Zusätzlich können Befehle umsortiert werden, um Control Hazards zu vermeiden. Das bedeutet unabhängige Befehle werden zwischen den Befehl, der die Bedingung auswertet und folgende Befehle gepackt, sodass der Befehl Zeit hat durch die Pipeline zu laufen, bevor das Ergebnis benötigt wird. Dies ist allerdings nicht immer möglich, da nicht immer unabhängige Befehle zur Verfügung stehen, um diese vorzuziehen. Bei der Sprungvorhersage versucht man stattdessen das Wissen über die letzten Sprünge zu nutzen, um zukünftige Sprünge vorherzusagen. Im Idealfall kann der Prozessor aus der Sprunghistorie sehen, ob der Sprung wahrscheinlich genommen wird oder nicht und dementsprechend je nachdem wie er entscheidet, die nächsten benötigten Befehle in die Pipeline laden. Nur wenn er sich falsch entscheidet, muss die Pipeline geflusht, also geleert, werden und eine Verzögerung bei der Bearbeitung entsteht (zusätzlich zu Verzögerungen bei echten Datenabhängigkeiten). Dadurch kann die Pipeline besser ausgenutzt und mehr Befehle abgearbeitet werden, als wenn man auf das Ergebnis der Bedingung wartet.

2. *Was ist der Unterschied zwischen lokalen und globalen Prädiktoren? Welche Gründe könnten für die eine oder andere Variante sprechen?*

Bei einem lokalen Prädiktor werden separate Buffer mit der Sprunggeschichte für jeden einzelnen bedingten Sprung gespeichert. Das bedeutet pro Sprung gibt es andere Sprungvorhersagen und diese ändern sich auch erst, wenn genau bei diesem Sprung eine Fehlentscheidung passiert oder eine schwache Vorhersage, durch Richtigkeit bestätigt wurde. Der Fokus liegt dabei auf der korrekten Sprungvorhersage des jeweiligen Sprungs, aber das Zusammenspiel der bedingten Sprünge innerhalb eines Programmabschnitts wird nicht beachtet, auch wenn die Sprünge eventuell zusammenhängen. Bei einem globalen Prädiktor wird der Fokus auf eben dieses Zusammenspiel von Sprüngen in einer Programmfolge gelegt. Hier gibt es einen gemeinsamen Buffer für die Sprungvorhersage von allen Sprüngen, um die Zusammenhänge zwischen unterschiedlichen bedingten Sprüngen zu erkennen und zu nutzen. Der Nachteil ist dabei, das wichtige Informationen zu einem bedingten Sprung auch verloren gehen können, wenn dieser unabhängig von den anderen Sprüngen ist. Außerdem kann der Buffer mit Informationen zu Sprüngen aus einem anderen Programmzweig gefüllt sein, als dem in dem gerade die bedingten Sprünge vorhergesagt werden müssen.

3. *Was ist ein Vor- / Nachteil von 2-bit-Zustandsautomaten ggü. 1-bit-Zustandsautomaten?*

Ein Vorteil von 2-bit-Zustandsautomaten gegenüber dem 1-bit-Zustandsautomaten ist, dass die Vorhersage zweimal falsch sein muss, um von einem starken Zustand in den anderen Vorhersagezustand zu wechseln. Das ist besonders wichtig bei verschachtelten Schleifen, wo direkt nach einem Austritt aus einer inneren Schleife wieder der Eintritt in die äußere Schleife folgt. Wenn hier beim Austritt aus der inneren Schleife eine Fehlvorhersage stattfindet würde ein 1-bit-

Zustandsautomat in der Vorhersage direkt zu “Not Taken” überwechseln und gleich darauf wieder falsch liegen. Der 2-bit-Zustandsautomat wechselt zuerst von einem starken Vorhersagezustand in einen schwachen Vorhersagezustand, bleibt aber erstmal bei der Vorhersage “Taken” und geht deshalb wieder in die äußere Schleife rein.
Ein Nachteil von 2-bit-Zustandsautomaten ist, dass man eventuell bei einer Schleife erstmal zwei Fehlvorhersagen am Anfang hat, wenn man in die Schleife reingeht und der Zustand der Vorhersage auf “strongly not taken” ist. Der Vorteil gegenüber dem 1-bit-Zustandsautomaten wird erst beim längeren Lauf durch die Schleife sichtbar.

4. *Was ist der Vorteil vom Hysteresis Scheme ggü. dem Saturation Counter Scheme?*

Bei dem Hysteresis Scheme wird nach einer Fehlvorhersage nicht von einem schwachen Vorhersagezustand in den anderen schwachen zustand gewechselt, sondern gleich zum starken gegensätzlichen Vorhersagezustand. Anders als beim Saturation Counter Scheme, kann hier die Vorhersage nicht zu schnell zwischen den beiden schwachen Vorhersagezuständen hin und her wechseln. Das ist gerade das Verhalten von einem 1-bit-Zustandsautomaten, was verbessert werden sollte durch die Nutzung von zwei Bits. Bei einer Sprungfolge, die die ganze Zeit wechselt zwischen Sprung nehmen und Sprung nicht nehmen, kann es beim Saturation Counter Scheme passieren, dass man in allen Fällen falsch vorhersagt, weil man sich immer genau in der entgegengesetzten Vorhersage befindet. Bei dem Hysteresis Scheme würde man hier nur in der Hälfte der Fälle falsch liegen.

5. *Was ist der Branch Target Buffer (BTB) und was tut er?*

Der Branch target Buffer ist ein kleiner Speicherbereich im Prozessor, in dem die einzelnen bedingten Sprünge mit ihren Adressen und deren Zieladressen gespeichert werden. Der BTB wird in der “Instruction Fetch”- Phase in einer Pipeline zurate gezogen, um zu sehen, ob die aktuelle Befehlsadresse zu einem Befehl mit bedingtem Sprung gehört, bevor der Befehl dekodiert wird. Außerdem wird im BTB zu jedem Sprung die Vorhersage gespeichert, ob der Sprung genommen werden soll oder nicht.

6. *Warum ist Not Taken bzw. Strongly Not Taken der einzig sinnvolle Startzustand für die Automaten (auch im Zusammenhang mit dem BTB)?*

Zum einen ist es anfangs meist wahrscheinlicher, durch den Code durchzugehen, als direkt zu springen. Zum Beispiel, wenn man im Programm zum ersten Mal eine Schleife betritt und wenn dort eine Abbruchbedingung mit einem bedingten Sprung aus der Schleife wäre, würde bei der Vorhersage “Taken” oder “Strongly Taken” die Schleife sofort wieder verlassen werden, was meist nicht gewollt ist. Außerdem ist der BTB am Anfang des Programmes noch leer und wird erst zur Laufzeit gefüllt. Das bedeutet, beim ersten Mal an einem bedingten Sprung würden bei der Vorhersage “Taken” oder “Strongly Taken” eigentlich die Befehle von der Zieladresse in die Pipeline geladen werden, aber diese steht noch gar nicht im BTB drin. Das heißt der Prozessor wüsste gar nicht, wohin er eigentlich springen soll.

Globale Sprungvorhersage

```
        mov rax, 0
.loop1: mov rbx, 1
.loop2: ;do smth
        inc rbx
        cmp rbx, 10
        jl .loop2
        inc rax
        cmp rax, 3
        jl .loop1
```

Initial: PWT (10)
rbx = 1, rax = 0
rbx = 2, rax = 0
1. Aufruf "rbx < 10": True (Falsch geraten) => PST (11)
rbx = 3, rax = 0
2. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 4, rax = 0
3. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 5, rax = 0
4. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 6, rax = 0
5. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 7, rax = 0
6. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 8, rax = 0
7. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 9, rax = 0
8. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 10, rax = 0
9. Aufruf "rbx < 10": False (Falsch geraten) => PWT (10)
rbx = 10, rax = 1
1. Aufruf "rax < 3": True (Richtig geraten) => PST (11)

rbx = 2, rax = 1
10. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 3, rax = 1
11. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 4, rax = 1
12. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 5, rax = 1
13. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 6, rax = 1
14. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 7, rax = 1
15. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 8, rax = 1
16. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 9, rax = 1
17. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 10, rax = 1

18. Aufruf "rbx < 10": False (Falsch geraten) => PWT (10)
rbx = 10, rax = 2
2. Aufruf "rax < 3": True (Richtig geraten) => PST (11)

rbx = 1, rax = 2
19. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 2, rax = 2
20. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 3, rax = 2
21. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 4, rax = 2
22. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 5, rax = 2
23. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 6, rax = 2
24. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 7, rax = 2
25. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 8, rax = 2
26. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 9, rax = 2
27. Aufruf "rbx < 10": True (Richtig geraten) => PST (11)
rbx = 10, rax = 2
28. Aufruf "rbx < 10": False (Falsch geraten) => PWT (10)
rbx = 10, rax = 3
3. Aufruf "rax < 3": False (Falsch geraten) => PSNT (00)

Es gab insgesamt 31 Sprungvorhersagen. Davon waren 4 falsch und 27 richtig. Das entspricht einer Erfolgsrate von 87,09%.

Quellen

- Vorlesungsfolien + Tutorium
- https://en.wikipedia.org/wiki/Branch_predictor#Local_branch_prediction
- https://docplayer.org/83754699-Vorhersagemechanismen-in-fliessbandarchitekturen.html

Ohne Predicated Instructions:

		IF	ID	OF	EX	WB
	1	1	*	*	*	*
1inc a	2	2	1	*	*	*
2inc a	3	3	2	1	*	*
3inc a	4	4	3	2	1	*
4cmp a, 0	5	5	4	3	2	1
5jne .after	6	9	5	4	3	2
6div b, b, 2	7	10	9	5	4	3
7add b,b,2	8	11	10	9	5	4
.after:	9	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
9inc a	10	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
10inc a	11	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
11inc a	12	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	13	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	14	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	15	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	16	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	17	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	18	FLUSH	FLUSH	FLUSH	FLUSH	FLUSH
	19	6				
	20	7	6			
	21	9	7	6		
	22	10	9	7	6	
	23	11	10	9	7	6
	24	*	11	10	9	7
	25	*	*	11	10	9
	26	*	*	*	11	10
	27	*	*	*	*	11

Mit predicated Instructions:

		IF	ID	OF	EX	WB
1inc a	1	1	*	*	*	*
2inc a	2	2	1	*	*	*
3inc a	3	3	2	1	*	*
4cmp a, 0	4	4	3	2	1	*
5div_if_a_z ero b, b, 2	5	5	4	3	2	1
6add_if_a_z ero b, b, 2	6	6	5	4	3	2
7inc a	7	7	6	5	4	3
8inc a	8	8	7	6	5	4
9inc a	9	9	8	7	6	5
	10	*	9	8	7	6
	11	*	*	9	8	7
	12	*	*	*	9	8
	13	*	*	*	*	9

Laut seriösen Websites werden predicated Instructions gleich geladen und Ausgeführt, aber ein Write Back findet nur statt, wenn die Bedingung erfüllt ist.
https://userblogs.fu-berlin.de/asoeyler/archive/128