Schiefe binäre Suchbäume

Rabie Hammoud

Institut für Informatik, Albert Ludwigs Universität Freiburg
Prof. Susanne Albers & Tobias Jacobs
Seminar: Algorithm Engineering.

Abstrakt. Das Konzept dieser Arbeit besteht aus der Suche nach einer Datenstruktur, in der man die Verzögerung bestimmter Hardwarevorgänge reduzieren kann. Es ist theoretisch bekannt, dass die minimale Anzahl von Vergleichen in einem binären Suchbaum erst erreicht werden kann, wenn der Baum eine perfekte Balancierung nachweist. In der Praxis ist es jedoch erwiesen, dass das *Branching* links und rechts an einem bestimmten Knoten nicht immer die gleichen Kosten hervorrufen muss. Dies ist vor allem an den *Branch predictios schemes* belegbar. Anhand dieser Tatsache, soll eine analytische Diskussion über die Klasse der schiefen binären Suchbäume vorgenommen werden. Darüber hinaus werden auch verschiedene *Memory Layouts* untersucht, wobei jedes Element im Baum mit einer unterschiedlichen Wahrscheinlichkeit zugänglich ist. Das Ergebnis dieser Analyse zeigt, dass für eine gewisse Anzahl von *Layouts*, die schiefen binären Suchbäume eine bessere Laufzeit als die perfekt balancierten Suchbäume erreichen können.

1 Einführung

In der Arbeit soll der Aufbau von schiefen binären Suchbäume erläutert werden, die in der Praxis eine optimale Laufzeit für eine beliebige Suche liefern. Theoretisch kann die minimale Anzahl von Vergleichen dann erreicht werden, wenn der Baum perfekt balanciert ist, d.h. für jeden Knoten im Baum ist die Anzahl der Knoten im linken Teilbaum ungefähr gleich der Anzahl der Knoten im rechten Teilbaum. Im Gegensatz dazu wird in der Praxis gezeigt, dass eine optimalere Laufzeit erreicht werden kann, wenn der Baum schief ist; d.h. einer der Teilbäume enthält mehr Knoten als die andere Teilbaum.
Die Laufzeit eines Algorithmus wird meistens durch die Anzahl der Befehle, die durch das CPU gerechnet werden, analysiert. In der Tat sind es aber andere Hardware-Faktoren, die die Laufzeit immens beeinflussen. Es ist bekannt, dass während der Suche in einem binären Suchbaum die Abzweigung nach rechts oder nach links an einem bestimmten Knoten die gleichen Kosten der Laufzeit verursacht. Dieser Theorie ist zu widersprechen, da moderne Prozessoren die Befehle in einem *Pipeline* schieben, welches zu einem *Branch misprediction* führen kann.Im Falle, dass dieses Problem auftreten

sollte, muss die ganze *Pipeline* geleert werden, was zu einem enormen Zeitverlust führt.
Es werden verschiedene *Memory-layouts* und ihren Einfluss auf der Laufzeit von Speicherzugriffen diskutiert.

2 Hardware Diskussion

Die Laufzeit eines Algorithmus ist die Länge der Zeitspanne, die zur Bewältigung einer Aufgabe benötigt wird. Die Laufzeit ist meistens durch die Anzahl von Maschienenbefehle analysiert. In der Praxis jedoch, die Laufzeit eines Algorithmus kann durch andere Hardware Faktoren beeinflusst werden, wie im Falle von *Branch misprediction* und *Cache Faults*.

Pipeline:
Die Pipeline bezeichnet bei Mikroprozessoren eine Art "Fliessband", mit dem die Abarbeitung der Maschienenbefehle in Teilaufgaben zerlegt wird, die für mehrere Befehle parallel durchgeführt werden. Dieses Prinzip wird Pipelining genannt. Pipelining stellt eine weit verbreitete Mikroarchitektur heutiger Prozessoren dar.
Das Performanz von Rechnern lässt sich durch Pipelining steigern. Statt eines gesamten Befehls wird während eins Taktzyklus des Prozessors nur jeweils eine Teilaufgabe abgearbeitet, allerdings werden die verschiedenen Teilaufgaben mehrerer Befehle dabei gleichzeitig bearbeitet. Da diese Teilaufgaben einfacher (und somit schneller) sind als die Abarbeitung des gesamten Befehls am stück, kann durch Pipelining die Taktfrequenz des Mikroprozessors gesteigert werden. Insgesamt benötigt ein einzelner Befehl nun mehrere Takte zur Ausführung, da aber durch die quasi parallele Bearbeitung mehrerer Befehle in einem Zyklus ein Befehl fertig gestellt wird, wird der Gesamtdurchsatz dieses Verfahren erhöht.
Um Pipelining im Datenpfad ausnutzen zu können, muss die Abarbeitung eines Maschienenbefehls in mehrere Phasen mit möglichst gleicher Dauer aufgeteilt werden. Eine sinnvolle Aufteilung ist abhängig vom Befehlsatz und der verwendeten Hardware.
Beispiel: Abarbeitung in 5 Schritten:

- Befehlsholphase (IF Instruction Fetch)
- Dekodierphase (Lesen von Operanden aus Registern)
- Ausführung (Adressberechnung)
- Speicherzugriff(Memory Access)
- Abspeicherphase (Result write Back cycle)

Pipelining arbeitet nicht immer einwandfrei, sondern es treten manchmal Probleme auf. Zu diesen Problemen gehörn *Branch misprediction. Branch mispredictoin* gehört zur Kategorie *Kontrollflusskonflikte*, wenn die Pipeline abwarten muss, ob ein bedingter Sprung ausgeführt wird oder nicht.
Betrachte folgendes Maschienenprogramm:

1. 40 beq \$1, \$3, 72
2. 44 and \$12,\$2, \$5
3. 48 or \$13, \$6, \$2
4. 52 add \$14, \$2, \$2
5. .....
6. 72 lw \$4,\$50,\$14

Die Frage, die sich nun automatisch stellt, welche Instruktion wird nach Instruktion 40 ausgeführt?
Es wird angenommen, dass der Sprung nicht ausgeführt werden muss. Die Folgeinstruktionen \$pc+4, \$pc+8, und \$pc+12 werden auf verdacht "Spekulativ" in die Pipeline geschoben. Wird die Abzweigung doch genommen, so muss die Pipeline geleert werden.
Solche Pipeline Konflikte führen zu einem enormen Zeitverlust der mit der Länge der Pipeline steigt. Da die Pipeline immer länger und länger werden, haben *Branch misprediction* einen ansteigenden Einfluss auf die Laufzeit von Algorithmen.

3 Schiefe binäre Suchbäume

Schiefe binäre Suchbäume sind binäre Bäume mit den folgenden Eigenschaften:

- Sei α eine Konstante zwischen 0 und 1/2.
- Für jeden Knoten V gibt es einen bestimmtes Verhältnis zwischen der Anzahl der Knoten im linken Teilbaum und Anzahl der Knoten in dem gesamten Teilbaum mit Wurzel V.
- Dieses Verhältnis kann man Formal als folgendes ausdrücken:

$$size(left(v)) = \alpha \bullet size(V)$$

Theorem: die mittlere Pfadlänge P eines binären Suchbaumes T ist höchstens durch

$$(1 + 1/n)\bullet\log(n + 1)/H(\alpha)$$

gegeben. In der Praxis, aufgrund von Hardware Angelegenheiten, ist die Laufzeit von dem nächsten Knoten abhängig; d.h. vom linken oder vom rechten Kind. Die Kosten für das rechte Kind eines gegebenen Knotens sind in dem Fall unterschiedlich.
Wir definiern nun einen schiefen binären Suchbaum T mit dem Faktor α. Weiterhin sei C_l und C_r die Kosten für das Zweigen nach links und rechts. Eine beliebige Suche im Baum T hat nun:

$$O((\alpha \bullet C_l + (1 - \alpha) \bullet C_r)\bullet logn/H(\alpha)$$

erwartete Kosten.
Nun betrachten wir einen *Static Branch Prediction Scheme* mit left cost $C_l = 1$ und right cost $C_r = 0$. Die erwartete Anzahl von *Branch misprediction* für eine beliebige Suche in einem schiefen binären Suchbaum von Faktor α ist :

$$O(\alpha\bullet logn/H(\alpha))$$

4 Simple Memory Layouts

Alle Layouts sind als ein Feld von n-Knoten gespeichert, die wiederum zwei Zeiger auf dem linken und dem rechten Kind enthalten. Der Anzahl der *Cache Faults* kann dramatisch durch die Memory Layouts beeinflusst werden. Für eine beliebige Suche werden verschiedene Layouts analysiert, um den Einfluss auf der Laufzeit zu verstehen.

Das BFS Layout:

Die Knoten an einer Ebene werden in einem *left-to-right* order bearbeitet, d.h. die ersten B-Knoten vom Feld enthalten den oberen Teilbaum. Die Pfadlänge in diesem Teilbaum ist dann O($logn$), und der obere Teilbaum wird in die Memory mit einer einzigen I/O operation durchgeführt, d.h. für die ersten O($logB$) Knoten werden O(1) I/O gebraucht, und für den Rest O(1) I/O per Knoten gebraucht. Die erwartete Anzahl vin I/Os für einen Suchpfad P ist dann:

$$\mathrm{O}(1 + P - \mathrm{logB})$$

Das DFS *(Deep First Search)* Layout:

Der Baum liegt im Array, sodass nach dem der Wurzel besucht ist, wird erstens das rechte Kind gespeichert. Die rechten Kinder werden dann in einem Block gespeichert. Also, *Branching* rechts kostet O(1/B) I/Os, und *Branching* links kostet O(1) I/O. Das ergibt für beliebige Suche Kosten von:

$$\mathrm{O}((\alpha/B + (1-\alpha))\bullet \log n/H(\alpha))$$

5 Schlussfolgerung

Schiefe binäre Suchbäume können eine optimalere Laufzeit als perfekt balancierte binäre Suchbäume erreichen, weil die Kosten für die Suche links und rechts unterschiedlich sind.

Literatur

1. *G. S. Brodal and G. Moruz. Skewed binary search trees. In Proceedings of the 14th Conference on Annual European Symposium (ESA'06), 2006*
2. *Computer Architecture. A Quantitative Approach von John L. Hennessy und David A. Patterson von Academic Press*