Seminar

Analyse von Programmausführungszeiten
Prof. Dr. Reinhard von Hanxleben

Vortrag über Pfad Analysen
WS 2002/2003
16.12.2002
Oliver Otte

Quellen:

Yau-Tsun Steven Li and Sharad Malik,
Performance analysis of embedded software using implicit path enumeration,
Proceedings of the ACM SIGPLAN 1995 Workshop on Languages,
Compilers, & Tools for Real-Time Systems, June 21 - 22,
1995, La Jolla, CA USA, Pages 88 - 98

Thomas Lundqvist and Per Stenström,
Integrating Path and Timing Analysis Using Instruction-Level Simulation Techniques,
Proceedings of ACM SIGPLAN Workshop on Languages,
Compilers, and Tools for Embedded Systems, pages 1-15, June 1998,
Lecture Notes in Computer Science 1474, Springer

Überblick

- Einleitung
- Ansatz von Li und Malik (ILP)
- Ansatz von Lundqvist und Stenström (Simulation auf Instruktionsebene)
- Vergleich der beiden Ansätze

Definition WCET

Die Zeit, die der längste Programmpfad
unabhängig von Eingabedaten und Initialzustand
zum Durchlaufen eines Programmes benötigt, wird

W(orst) **C**(ase) **E**(xecution) **T**(ime)

genannt.

Je ungenauer die Analyse der WCET ist desto mehr Rechenleistung muß zur Verfügung gestellt werden, um die Einhaltung der Deadlines garantieren zu können. Daher ist es sehr wünschenswert, eine möglichst genaue Näherung der WCET zu bekommen.

Mögliche Berechnungsarten WCET

- Alle Kombinationen der Eingabedaten durchprobieren und die Zeit messen
- Finden des längsten Pfades und anschließende Berechnung der Ausführungszeit (explizite Pfadnumerierung)
- Finden der Ausführungszeit des längsten Pfades, ohne den Pfad explizit ausgeben zu können (implizite Pfadnumerierung)

Im Allgemeinen ist dieses Problem unentscheidbar und daher wird oftmals eine sehr restriktive Programmierung verwendet (keine dynamischen Datenstrukturen, keine Rekursionen) , um das Problem beherrschbar zu machen.

Da es normalerweise sehr viele Kombinationsmöglichkeiten für die Eingabedaten gibt, ist der erste Ansatz üblicherweise nicht einsetzbar.

Auch der zweite Ansatz ist meistens nicht praktikabel, da die Anzahl der zu überprüfenden Pfade sehr schnell sehr hoch wird.

Die implizite Pfadnumerierung wird sowohl im Ansatz von Li und Malik als auch im Ansatz von Lundqvist und Stenström benutzt.

Pessimismus bei WCET

- Zu ungenaues Zeitmodell
  → Modellierung eines genaueren Zeitmodells
- Unmögliche Programmpfade → Pfadannotationen oder statische Pfad Analysen

5

Die Modellierung eines genaueren Zeitmodells (beispielsweise durch Berücksichtigung von Caches und Pipelines) wurde bereits in den vorigen Vorträgen behandelt, so daß an dieser Stelle darauf nicht mehr eingegangen werden soll.

Im Ansatz von Li und Malik werden Pfadannotationen benutzt (diese werden auf Programmiersprachebene durchgeführt, die Analyse jedoch auf Instruktionsebene) und im Ansatz von Lundqvist und Stenström wird eine statische Pfad Analyse durchgeführt .

Ansatz Li und Malik (ILP)

- Basisblock B_i
- Anzahl N der Basisblöcke des Programmes
- Ausführungsanzahl x_i
- Ausführungszeit c_i
- Maximiere $\sum_{i=1}^{N} c_i x_i$
- Hier: c_i fest, x_i werden genauer analysiert

6

Ein Basisblock ist eine maximale Instruktionssequenz mit der ersten Anweisung als einzigem Eingangspunkt und der letzten Anweisung als einzigem Ausgangspunkt.

Die x_i zeigen an, wie oft Basisblock B_i bei der maximalen Ausführungszeit durchlaufen wird.

Die c_i zeigen an, wie lange im schlechtesten Fall für einen Durchlauf des Basisblocks B_i gebraucht wird.

Die x_i müssen nach oben begrenzt werden, um dieses Problem lösbar zu machen. Das gleichzeitige Erreichen aller oberer Schranken wird durch zwei Faktoren verhindert: durch die Programmstruktur und durch die Funktionalität des Programmes.

Bedingungen bei ILP

```
x1  for (i=0; i<k; i++)
x2    if (OK)
x3      do_something();
        else  {
x4      do_something_else();
          OK=true;
        }
      }
```


Diese Schleife hat eine obere Grenze k, die aber aufgrund des wechselseitigen Ausschlusses der Basisblöcke B_3 und B_4 niemals gleichzeitig in beiden Basisblöcken erreicht werden kann. Dies ist eine Einschränkung, die sich aus der Programmstruktur ergibt.

Der Basisblock B_4 wird im Höchstfall genau einmal ausgeführt, so daß niemals die obere Schranke k erreicht wird. Dies ist eine Einschränkung, die sich aus der Programmfunktionalität ergibt.

Programmstrukturbedingungen

- Werden automatisch aus dem Kontrollflußgraphen gewonnen
- Entsprechen der Anzahl der Durchläufe des Kontrollflusses durch die entsprechenden Kanten bzw. Basisblöcke
- Summe eingehende Kanten = Ausführungsanzahl des Basisblocks = Summe austretende Kanten

Beispiele

- $d_1 = 1$ für die Startkante der Hauptfunktion
- $x_1 = d_1 = d_2$ für einen Basisblock mit einer eingehenden und einer ausgehenden Kante
- $x_1 = d_1 = f_1$ für einen Basisblock, der durch einen Funktionsaufruf verlassen wird
- $d_2 = f_1 + f_2$ für die Startkante einer Funktion, die von zwei Stellen aus aufgerufen wird

Funktionsaufrufe bekommen im Gegensatz zum normalen Programmfluß eine f-Kante und keine d-Kante. Diese Kante wird wie eine d-Kante behandelt, sie beinhaltet allerdings einen Zeiger auf den Kontrollflußgraphen der aufgerufenen Funktion

.

Programmfunktionalitätsbedingungen

- Müssen vom User eingegeben werden
- Hauptsächlich Schleifengrenzen
- Zur Verbesserung der WCET-Analyse auch zusätzliche Pfadinformationen angebbar

Es ist geplant, einige Funktionalitätsbedingungen automatisch zu finden, so daß dem User Arbeit erspart wird.

Bevor die Analyse ausgeführt werden kann, müssen alle Schleifen nach oben und nach unten begrenzt werden. Alle anderen Funktionalitätsbedingungen können zusätzlich angegeben werden, um eine noch genauere Analyse zu ermöglichen.

Beispiel

```
    check_data()
    {
     int i, morecheck, wrongone;
x1   morecheck=1; i=0; wrongone=-1;
     while (morecheck)  {
x2     if (data[i] < 0)  {
x3       wrongone=i; morecheck=0;
       }
       else
x4      if (++i >= DATASIZE)
x5        morecheck = 0;
x6   }
x7   if (wrongone >= 0)
x8     return 0;
      else
x9     return 1;
    }
```

Falls DATASIZE=10 angenommen wird, gilt $x_1 R x_2$ und $x_2 R 10x_1$

Weiter gilt

$(x_3=0 \& x_5=1) | (x_3=1 \& x_5=0)$

$x_3=x_8$

Bei Schleifen werden die x_i automatisch gefunden (es ist einmal der Zähler für den Basisblock direkt vor der Schleife und einmal der Zähler für den ersten Basisblock innerhalb der Schleife), der User muß nur noch die untere und die obere Grenze eingeben.

Bedingungen, die ein | enthalten, führen zu einer Verdopplung der Funktionalitätsbedingungsmengen!

Fortsetzung Beispiel

Daraus ergeben sich folgende Funktionalitätsbedingungsmengen:

Menge 1	Menge 2
$x_1 - x_2 \mathrel{R} 0$	$x_1 - x_2 \mathrel{R} 0$
$10x_1 - x_2 \mathrel{S} 0$	$10x_1 - x_2 \mathrel{S} 0$
$x_3 = 0$	$x_3 = 1$
$x_5 = 1$	$x_5 = 0$
$x_3 - x_8 = 0$	$x_3 - x_8 = 0$

12

Bei jedem Hinzufügen einer Funktionalitätsbedingung wird überprüft, ob sich eine Nullmenge (beispielsweise $x_1 >= 1$ & $x_1 = 0$) gebildet hat, so daß man diese Menge nicht mehr überprüfen muß. Auf diese Weise wird vermieden, daß der ILP-Solver unnötig oft aufgerufen wird.

Lösen der Bedingungen

- Kombiniere die Funktionalitätsbedingungen mit den Strukturbedingungen
- Rufe den ILP-Solver mit jeder der Bedingungsmengen auf (mit dem Ziel zu maximieren)
- Das Maximum der Ergebnisse ist die WCET

Testergebnisse

Pessimism in path analysis

Function	Number of Constraint Sets	Estimated Bound	Calculated Bound	Pessimism
check_data	2	1039	1039	0,00
Fft	1	3346000	3312544	0,01
Piksrt	1	4333	4333	0,00
Des	2	604169	592559	0,02
Line	1	8485	8485	0,00
Circle	1	16652	16301	0,02
Longloop	1	14104	14104	0,00
jpeg_fdct_islow	1	16291	16291	0,00
jpeg_idct_islow	1	20665	20665	0,00
Recon	1	9319	9319	0,00
Fullsearch	1	244305	244025	0,00
Whetstone	2	13711800	13711800	0,00
Dhry	8 -> 3	1264430	1264430	0,00
Matgen	1	13933	13933	0,00

'Estimated Bound' ist das Ergebnis der Analyse, für 'Calculated Bound' wurden die Programme mit den schlechtesten Eingabedaten ausgeführt und die erhaltenen Daten in die ILP-Formel eingesetzt.

Im Prinzip könnten bei der Analyse beliebig viele Bedingungsmengen entstehen, aber in der Praxis sind diese Probleme nicht aufgetaucht. Es entstanden nur wenige Bedingungsmengen, die durch das Auffinden von Nullmengen noch weiter reduziert werden konnten.

Testergebnisse

- Analyst ≠ Programmierer → Probleme bei den Funktionalitätsbedingungen
- Je mehr Informationen gegeben werden desto genauer wird die Analyse

15

Wird das Verfahren nicht vom Programmierer selbst ausgeführt sondern von einer anderen Person, dann besteht die Gefahr, daß die Grenzen viel zu hoch angesetzt werden, da sich die analysierende Person nicht so gut mit dem Programm auskennt. Aus dem gleichen Grund werden dann häufig nicht so viele Funktionalitätsabhängigkeiten erkannt.

Ansatz von Lundqvist und Stenström (Simulation auf Instruktionsebene)

- Normalerweise Simulation eines einzigen Pfades
- Für Berechnung der WCET Simulation aller Pfade durch Hinzufügen eines Elementes "Unbekannt"
- Wenn ein Pfad simuliert wurde, dann sind alle von den Eingabedaten unabhängige Abhängigkeiten aufgelöst

Normalerweise sind bei einer Simulation die Eingabedaten bekannt und es wird genau ein Pfad simuliert. Zur Bestimmung der WCET müßten dann alle Kombinationen der Eingabedaten simuliert werden und dabei die benötigten Zeiten gemessen werden. Dies ist natürlich nicht praktikabel.

Das Element "Unbekannt"

- Semantik der Instruktionen muß geändert werden
- "Unbekannt" bei if-then-else-Bedingung → simuliere beide Pfade
- "Unbekannt" bei Adresse einer load-Anweisung → setze die Variable auf "Unbekannt"
- "Unbekannt" bei Adresse einer store-Anweisung → speichere in gesonderten Bereich

Bei store-Anweisungen mit "Unbekannt" in der Adresse müßte man eigentlich in alle Speicherpositionen "Unbekannt" schreiben, um den schlechtesten Fall zu betrachten, aber da dies eine zu große Einschränkung wäre, werden erst alle "unvorhersehbaren Datenstrukturen" identifiziert und durch den Linker auf eine Speicherzone abgebildet, die nur mit unbekannten Werten antworten kann. Daher können während der Simulation stores in diese Zone ignoriert werden und loads liefern immer ein "Unbekannt" zurück.

Probleme bei der Simulation

- Anzahl der zu simulierenden Pfade
- Sich wechselseitig ausschließende Pfade können zu einer Überschätzung der WCET führen
- Wenn alle Ausstiegsbedingungen für eine Schleife unbekannt sind, dann endet die Simulation nie

Die Anzahl der zu simulierenden Pfade wird durch ein Pfad-Verschmelzungs-Verfahren möglichst gering gehalten.

Sich wechselseitig ausschließende Pfade bekommt man besser in den Griff, wenn man die Domänen der Werte an das jeweilige Programm anpasst. Das hat nur eine langsamere Simulation und mehr Speicherbedarf zur Folge.

Wenn alle Ausstiegsbedingungen von unbekannten Eingabedaten abhängen, dann muß entweder das Programm geändert werden oder es müssen von Hand Grenzen hinzugefügt werden.

WCET-Algorithmus

- Es wird jeweils der Pfad mit den wenigsten Schleifeniterationen simuliert
- Bei gleicher Anzahl der Iterationen müssen die Pfade erst verschmolzen werden
- Dies wird solange durchgeführt bis alle Pfade simuliert wurden
- Die WCET ist dann die längste Zeit aller bis zum Ende des Programmes simulierten Pfade

Das Hauptproblem bei der Simulation mit unbekannten Werten ist die große Menge der zu simulierenden Pfade. Damit die Simulation praktikabel bleibt, werden bei jedem Schleifendurchlauf zwar alle Pfade betrachtet, aber bevor der nächste Durchlauf beginnt werden die Pfade zu einem einzigen verschmolzen, so daß statt n^k Pfade nur n simuliert werden müssen.

Pfad-Verschmelzung

- Nur Pfade mit gleichem Programmzähler können verschmolzen werden
- Es wird der Pfad mit der längsten WCET genommen
- Alle Register und Speicherpositionen werden vereinigt

Bei der Vereinigung der Register und Speicherpositionen wird bei unterschiedlichen Werten der Wert "Unbekannt" eingetragen, um somit einen möglichen Einfluß dieser Werte auf die WCET des ausgewählten Pfades zu berücksichtigen.

Testergebnisse

The estimated and true WCET of benchmark programs (WCET is expressed in clock cycles)

Program	Measured true WCET	Estimated WCET	Ratio	Estimated structural WCET	Ratio
Matmult	7063912	7063912	1,0	7063912	1,00
Bsort	292026	292026	1,0	572920	1,96
Isort	2594	2594	1,0	4430	1,71
Fib	697	697	1,0	697	1,00
DES	118675	118675	1,0	119877	1,01
Jfdctint	6010	6010	1,0	6010	1,00
Compress	9380	49046	5,2	161161	17,20

21

Bei den Tests wurden Caches und Pipelines ausgeschaltet, der Compiler durfte nicht optimieren und alle Aufrufe von Systemfuntionen waren ausgeschaltet, weil es kein Betriebssystem in der Testumgebung gab.

Die 'richtige' WCET wurde durch Aufruf der Programme mit den schlechtesten Eingabedaten und Messen der benötigten Zeit herausgefunden. Zum Vergleich wurde auch noch auf eine andere Weise die WCET berechnet (mit festen Schleifenbegrenzungen), bei der keine unmöglichen Pfade beseitigt werden. Auf diese Weise wurde festgestellt, daß mit der Simulation eben jene Pfade gefunden und beseitigt wurden. Dadurch kann sich die Zeitanalyse auf die möglichen Pfade konzentrieren und bessere Ergebnisse liefern.

Vergleich der beiden Ansätze

- Beide haben Pfad- und Zeitanalyse miteinander verbunden → Probleme mit Zeitmodell
- Mit dem Ansatz von Lundqvist und Stenström kann man auch fremde Programme recht gut analysieren, beim Ansatz von Li und Malik kann es problematisch werden
- Beim Ansatz von Li und Malik ist sehr viel mehr Handarbeit gefragt und daher ist der Zeitaufwand für den User größer