Abstrakte Interpretation (AbsInt)

Viktoria Pleintinger

24. Juni 2009

Inhaltsverzeichnis

1 Einleitung

Eine Laufzeitanalyse dient zur Sicherung von Funktion und Effizienz von Programmen. Besonders bei Echt-Zeit-Programmen (zum Beispiel ABS) ist eine möglichst exakte Berechnung der Worst-Case Laufzeit zwingend, da bei sicherheitsrelevanten Programmen eine einwandfreie Funktion garantiert werden muss. Der Worst-Case wird durch eine obere Schranke definiert, das heißt über den Wert der Schranke darf die Laufzeit nicht steigen. Dabei ist zu beachten, dass das Ziel eine möglichst niedrige obere Schranke ist, da das effizienter ist, weil nicht so viel zusätzliche Pufferzeit bei der Laufzeit eingerechnet werden muss.

2 Worst-Case Laufzeitanalyse aufgrund von Messung

Es ist kaum möglich eine gute Worst-Case Laufzeitanalyse durch Messung durchzuführen. Probleme liegen dabei sowohl bei der Software als auch bei der Hardware. Bei der Software gibt es Schwierigkeiten, weil oft zusätzliche Software von Dritten auf das Programm mit einwirkt, dessen Analyse zusätzlichen Aufwand und Probleme verursacht. Das ursprünglich zu analysierende Programm wird aber so stark durch das dritte Programm beeinflusst, dass die Worst-Case Laufzeit ohne Berücksichtigung von zusätzlicher Software nicht berechnet werden kann.

Bei der Hardware liegt die Schwierigkeit in der Voraussetzung der Messung. Es wird immer von einem frisch gestarteten System ausgegangen, was in der Realität aber nie der Fall ist. Meistens beeinflussen vorher ausgeführte Programme beziehungsweise Programmschritte die Messung, da zum Beispiel kein leerer Cache vorhanden ist. Zusätzliche Probleme bei der Messung können durch parallel laufende Programme verursacht werden.

Bei der Berechnung der Worst-Case Laufzeit durch Messung werden die Programmabschnitte einzeln gemessen und dann zusammengerechnet. Allerdings werden bei den einzelnen Messungen die vorausgegangenen Programmschritte nicht berücksichtigt. So wird zum Beispiel der Cache immer als leer behandelt, obwohl das fast nie der Fall ist. Das wird beim Addieren auch nicht mit einberechnet. Dadurch sind die Ergebnisse einer Worst-Case Laufzeitanalyse nicht korrekt. Um diese Fehler zu beheben gibt es die Worst-Case Laufzeit Analyse durch Static Programm Analysis

3 WCET Analyse durch Static Programm Analysis

3.1 Phasen der Worst-Case Laufzeitberechnung im Überblick

3.1.1 Das Tool aiT

AbsInt benutzt zur Worst-Case Laufzeitanalyse das Tool aiT, dessen Vorteile sind:

- Die von aiT berechneten engen Schranken spiegeln die tatsächliche Performance Ihres Systems wider. Das ermöglicht eine bessere Hardware-Auslastung – ohne Kompromisse bei derSicherheit.
- Die aiT-Analyseergebnisse sind für alle Eingaben und alle Ausführungsszenarien gültig. Die Berechnung läuft vollautomatisch; es ist nicht erforderlich, die schlimmstmöglichen Eingaben manuell herauszufinden. Die verbreiteten, aber fehlerträchtigen und zeitraubenden Meßverfahren können verkürzt oder ersetzt werden. Das erhöht die Sicherheit, spart Zeit und senkt Systemkosten.
- aiT analysiert Binärdateien und ist daher weitgehend unabhängig von der verwendeten Programmiersprache und dem eingesetzten Compiler. Sie müssen also keine Änderungen an Ihrer Software oder an Ihren Produktionsabläufen vornehmen.

3.1.2 Graphik zu den Worst-Case Laufzeit Phasen

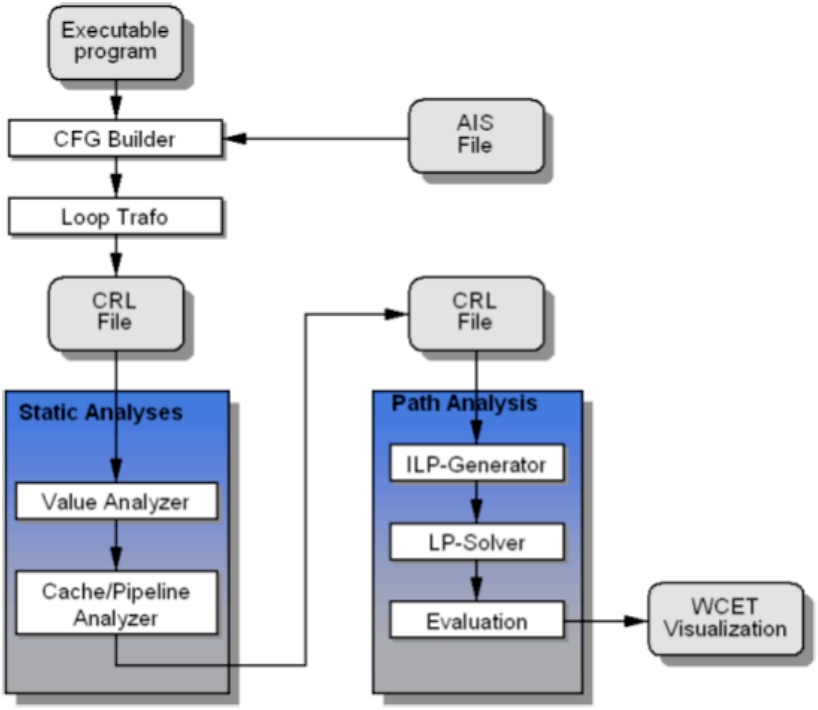

Figure 1: Phases of WCET computation

Abbildung 1: Graphik über Phasen der WCET

3.1.3 Die Phasen der Worst-Case Laufzeitanalyse im Überblick

Die Worst-Case Laufzeitanalyse besteht aus mehreren Schritten.

- Zunächst entsteht mit Hilfe des CFG-Builder der Kontroll-Fluss-Graph. Dieser wird aus einem binären Programm dekodiert und rekonstruiert.
- Die Value Analysis berechnet die Wertebereich und den Adressbereich für Speicherzugriffs Befehle.
- Die Loop Bound Analysis legt obere Schranken für die Anzahl von Iterationen von einfachen Schleifen fest.
- Die Cache Analysis klassifiziert Speicherzugriffe als Cache hits und Cache misses.
- Die Pipeline Analysis prognostiziert das Verhalten des Programms auf der Prozessor-Pipeline.
- Die Path Analysis bestimmt den Ausführungspfad, der den schlechtesten Fall realisiert.

3.2 CFG Builder

Der CFG-Builder konstruiert den Kontroll-Fluss-Graphen. Das AIS-File enthält zusätzliche Informationen in Form von Annotierungen, beispielsweise obere Schranken für Rekursionen. Ein Syntaxanalysierer liest die Ausführungen und rekonstruiert den Kontroll-Fluss-Graphen.

3.3 Value Analysis

Das Ziel der Value Analysis ist die Berechnung der Werte der Prozessor Register zu jedem Zeitpunkt und für jeden Anwendungskontext des Programms. In der Realität ist dies jedoch nicht möglich. Da die Werte nicht exakt berechnet werden können beschränkt man sich auf das Festlegen von oberen und unteren Schranken.

3.4 Loop Bound Analysis

Eine Worst-Case Laufzeitanalyse benötigt für alle Schleifen obere Schranken für die Anzahl an Iterationen, das heißt die maximale Anzahl an Iterationen muss schon gegeben sein. Die Loop Bound Analysis funktioniert nur bei einfachen, nicht verschachtelten Schleifen. Für die anderen Schleifen werden die oberen Schranken in der Annotierung angegeben. Die Loop Bound Analysis kombiniert Value Analysis mit Pattern Matching um typische Schleifenmuster zu finden.

3.5 Cache Analysis

Die Cache Analysis klassifiziert die Zugriffe auf den Arbeitsspeicher. Dabei wird zwischen einem sicheren Cache hit und einem nicht klassifizierten Zugriff unterschieden.

3.5.1 Der Cache

Cache-Architektur: Der Cache ist ein spezieller Puffer-Speicher, der zwischen dem Arbeitsspeicher und dem Prozessor liegt. Damit der Prozessor nicht jeden Programm-Befehl aus dem langsamen Arbeitsspeicher holen muss, wird gleich ein ganzer Befehls- oder Datenblock in den Cache geladen. Die Wahrscheinlichkeit, dass die nachfolgenden Programmbefehle im Cache liegen, ist sehr groß, da die Programm-Befehle nacheinander abgearbeitet werden. Erst wenn alle Programm-Befehle abgearbeitet sind oder ein Sprungbefehl zu einer Sprungadresse außerhalb des Caches erfolgt, dann muss der Prozessor auf den Arbeitsspeicher zugreifen. Deshalb sollte der Cache möglichst groß sein, damit der Prozessor die Programm-Befehle ohne Pause hintereinander ausführen kann.

- L1-Cache / First-Level-Cache

  In der Regel ist der L1-Cache nicht besonders groß. . Meistens ist der Speicherbereich für Befehle und Daten voneinander getrennt. Die Bedeutung des L1-Caches wächst mit der höheren Geschwindigkeit der CPU. Im L1-Cache werden die am häufigsten benötigten Befehle und Daten zwischengespeichert, damit möglichst wenige Zugriffe auf den langsamen Arbeitsspeicher erforderlich sind. Dieser Cache vermeidet Verzögerungen in der Datenübermittlung und hilft eine CPU optimal auszulasten.

- L2-Cache / Second-Level-Cache

  Im L2-Cache werden die Daten des Arbeitsspeichers (RAM) zwischengespeichert. Über die Größe dieses Caches versorgen die Prozessorhersteller die unterschiedlichen Marktsegmente mit speziell modifizierten Prozessoren. Ein Prozessor mit sehr großen L2-Cache ist teuer herzustellen. Deshalb war der L2-Cache auch schon mal außerhalb des Prozessor-Kerns angeordnet.

- L3-Cache / Third-Level-Cache

  Mit dem L3-Cache kann das Cache-Koheränz-Protokoll (*Ein Cache-Kohärenz-Protokoll hat die Aufgabe, den Status eines gecachten Speicherblocks zu verfolgen.*)von Multicore-Prozessoren viel schneller arbeiten. Dieses Protokoll gleicht die Caches aller Kerne ab, damit die Datenkonsistenz erhalten bleibt. Der L3-Cache hat also weniger die Funktion

eines Caches, sondern soll das Cache-Koheränz-Protokoll und den Betrieb des Switches vereinfachen und beschleunigen.

3.5.2 Cache-Speicher

Ein Cache kann über drei Parameter charakterisiert werden: Die *capacity* ist die Anzahl an Bytes die ein Cache enthält. Die *line size* (oder block size)ist die Anzahl der zusammenhängenden Bytes die bei einem Cache miss vom Speicher transferiert werden. Ein Cache hat höchstens *capacity/line size* Blöcke. Die associativity ist die Anzahl von Cache Location wo sich ein bestimmter Block befinden kann. *capacity/(line size * associativity)* ist die Anzahl an sets eines Caches. Unter einem set versteht man einen vollständig assoziativen subcache.

Kann ein Block überall im Cache stehen, so nennt man den Cache fully associative. Kann ein Block nur an genau einer Stelle stehen nennt man den Cache direct mapped. Kann ein Block an genau A Stellen stehen, so ist der Cache A-way set associative. Der fully associative Cache und der direct mapped Cache sind Spezialfälle des A-way set associative Cache.

3.5.3 Konkrete Semantiken - Mapping Relation

Im Folgenden wird ein Cache durch eine Menge von Cache Lines (Adressbereiche) betrachtet. Eine Mapping Relation definiert die Cache Lines, die einen bestimmten Speicherblock halten können. Fully associative mapping bedeutet, dass ein Speicherblock von jeder Cache Line gehalten werden kann. Direct Mapping bedeutet, dass ein Speicherblock in genau einer Cache Line liegen kann. A-way set associative mapping bedeutet, dass ein Speicherblock in genau A Cache Lines liegen kann.

$$\mathcal{U}_{\mathcal{M}_{assoc}}(c,s) = \begin{cases} [\, l_0 \mapsto s, \\ \quad l_i \mapsto c(l_{i-1}) \mid i = 1 \ldots h, \\ \quad l_i \mapsto c(l_i) \mid i = h+1 \ldots n-1 \;\,]; \text{if } \exists l_h : c(l_h) = s \\ [\, l_0 \mapsto s, \\ \quad l_i \mapsto c(l_{i-1}) \text{ for } i = 1 \ldots n-1 \,]; \text{otherwise} \end{cases}$$

Abbildung 2: Cache Update Funktion

3.5.4 Cache Update Funktion

Wir betrachten die Cache Update Funktion anhand eines fully associative Cache. Die Reihenfolge der Lines repräsentiert das relativer Alter der Speicherblöcke. Der Speicherblock auf den zuletzt zugegriffen wurde ist an erster Stelle.

Wird bei einem Speicherzugriff auf einen Speicherblock der schon im Cache vorhanden ist zugegriffen, wird dieser auf die erste Postion verschoben. Die Speicherblöcke von der ersten Position bis zur Position vor dem ausgewählten Speicherblock werden um eine Position nach hinten verschoben. Die Speicherblöcke nach dem ausgewählten Speicherblock bleiben stehen.

3.5.5 Abstrakte Semantiken

Ein abstract cache state ist die Abbildung von Cache Lines auf die Potenzmenge von Speicherblöcken. Die abstract semantic Funktion beschreib den Effekt eines Kontroll-Fluss-Knotens auf ein Element der Potenzmenge. Bei einem Kontroll-Fluss-Knoten mit wenigstens zwei Vorgängern werden join-Funktionen benutzt, um die abstrakten Cache Zustände zu kombinieren.

– *fully associative cache with LRU replacement strategy:* $\mathcal{U}_{\mathcal{M}_{assoc}}(\hat{c}, s) = \hat{c}'$

$$\hat{c}' = \begin{cases} [\, l_0 \mapsto \{s\}, \\ \quad l_i \mapsto \hat{c}(l_{i-1}) - \{s\} \mid i = 1 \ldots h, \\ \quad l_i \mapsto \hat{c}(l_i) - \{s\} \mid i = h+1 \ldots n-1]; & \text{if } \exists l_h : \hat{c}(l_h) = \{s\} \\ [\, l_0 \mapsto \{s\}, l_i \mapsto \hat{c}(l_{i-1}) - \{s\} \text{ for } i = 1 \ldots n-1]; & \text{otherwise} \end{cases}$$

Abbildung 3: Abstract Cache Update Funktion

3.5.6 Abstract Cache Update Funktion

Wir betrachten eine Abstract Cache Update Funktion anhand eines fully associative Cache. Die Reihenfolge der Lines repräsentiert das relativer Alter der Speicherblöcke. Der Speicherblock auf den zuletzt zugegriffen wurde ist an erster Stelle.

Wie bei der Cache Update Funktion wird bei einem Speicherzugriff auf einen Speicherblock, der Speicherblock schon im Cache vorhanden ist , dieser auf die erste Postion verschoben. Die Speicherblöcke von der ersten Position bis zur Position vor dem ausgewählten Speicherblock werden um eine Position nach hinten verschoben. Zusätzlich wird noch der Speicherblock auf den zugegriffen wurde aus der Potenzmenge entfernt.

3.5.7 Join-Funktionen

Eine Join-Funktion ist eine binäre Funktion auf abstract Cache States. Ziel ist für jeden Kontroll-Fluss-Knoten zu berechnen, ob der Zugriff auf den Speicher einen cache hit oder cache miss hervorruft Die Join-Funktion für direct mapped Caches berechnet für jede Cache Line eine Menge von möglichen Inhalt. Die Join-Funktionen von vollständig associativen Caches berechnet ob ein Speicherblock an einem bestimmten Kontroll-Fluss-Knoten im Cache ist. Die Position des Speicherblocks im abstract cache state, d.h. die Nummer der Cache Line, steht für das relative Alter des Speicherblocks. Falls ein Speicherblock zwei verschiedene relative Alter in zwei abstrakten Cache Zuständen hat, nimmt die Join-Funktion das älteste relative Alter, das heißt die höchste Position. Die Funktion nimmt die Schnittmenge.

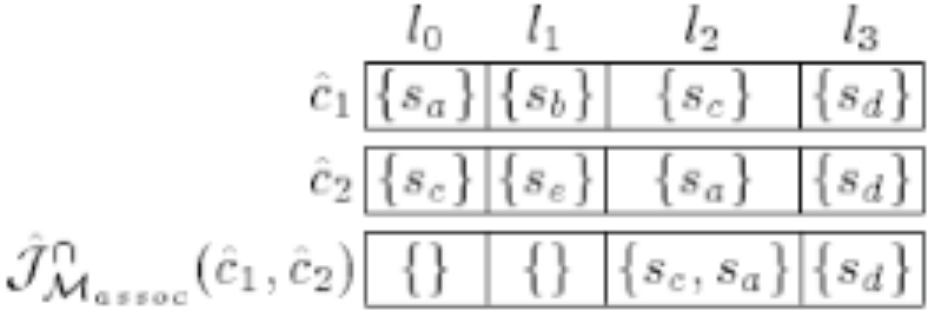

Abbildung 4: Join-Funktion fully associative Cache

3.5.8 Interpretation

Ein abstract Cache State $\hat{c}$ an einem Kontroll-Fluss-Knoten kann folgendermaßen interpretiert werden:

- Wenn der Speicherblock s Element von $\hat{c}(l)$ für eine Cache Line l ist, dann ist s definitiv im Cache. Ein Zugiff auf s wir immer ein Cache Hit sein.
- Wenn der Speicherblock s Element von $\hat{c}(lx)$ dann bleibt s mindestens (*capacity/line size* - x) Cache Updates im Cache, wenn die Cache Updates ein neues Element in den Cache setzen.

Um zu berechnen ob ein Speicherblock s an einem Kontroll-Fluss-Knoten nie im Cache enthalten ist nimmt man die Join-Funktion, welche die Vereinigungsmenge verwendet.

Ein abstract Cache State $\hat{c}$ an einem Kontroll-Fluss-Knoten kann folgendermaßen interpretiert werden:

- Wenn ein Speicherblock s kein Element von $\hat{c}(l)$ für ein beliebiges l, dann ist der Speicherblock definitiv in keiner Cache Line. Der Speicherzugriff wird immer ein Cache Miss.
- Wenn der Speicherblock s Element von $\hat{c}(lx)$ mit kleinstmöglichen x, dann wird s für höchstens (*capacity/line size* - x) Cache Updates, welche ein neues Element in den Cache setzen.

3.5.9 Schleifenanalyse

Im Kontroll-Fluss-Graphen wird eine Schleife als Kreislauf dargestellt. Der Anfangsknoten einer Schleife hat zwei ankommende Kanten. Eine Kante repräsentiert den Schleifenanfang, die andere den Kontroll-Fluss-Graphen vom Schleifenende zum Schleifenanfang. Da die Ausführung des Schleifenrumpfs meist eine Veränderung des Cache Zustands hervorruft, ist es sinnvoll die erste Iteration getrennt von den anderen zu betrachten. Das erreicht man durch das einmalige virtuelle Entrollen jeder Schleife. Bei verschachtelten Schleifen kann das Entrollen eine sehr teuere Transformation werden. Ein ähnliches Problem kann in der Prozedur-Analyse gelöst werden, deswegen werden Schleifen in Prozeduren umgewandelt, damit das Problem mit dieser Methode gelöst werden kann.

3.5.10 In einen Cache schreiben

Es gibt zwei übliche Cache Organisationen in Bezug auf das Schreiben in einen Cache:

- Es gibt zwei übliche Cache Organisationen in Bezug auf das Schreiben in einen Cache
- Write through: Die Daten werden auf den Speicherblock und die dazugehörige Cache Line geschrieben.
- Write back: Die Daten werden nur auf die Cache Line geschrieben. Die modifizierte Cache Line wird nur in den Arbeitsspeicher geschrieben, wenn sie ersetzt wird. Dies wird meistens duch ein Bit (auch dirty bit genannt) für jede Cache Line implementiert, das anzeigt, ob die Cache Line modifiziert wurde.

Die Laufzeit eines Speicherbefehls hängt oft davon ab, ob der Speicherblock der geschrieben wird im Cache ist (write hit) oder nicht (write miss). Es gibt zwei Cache Organisationen bezüglich write misses:

Die Laufzeit eines Speicherbefehls hängt oft davon ab, ob der Speicherblock der geschrieben wird im Cache ist (write hit) oder nicht (write miss). Es gibt zwei Cache Organisationen bezüglich write misses:

- Write-Allocation ist ein Cache-Verwaltungsstrategie. Bei Schreibvorgängen auf Adressen, die nicht im Cache vorhanden sind, führt das dazu, dass die Daten zuerst in den Cache gelesen werden. Der Schreibvorgang erfolgt dann nur noch auf den Cache. Write-Allocation muss im BIOS eingeschaltet sein.
- No Write-Allocation: Der Block wird nicht in den Cache geladen. Es wird nur der Arbeitsspeicher verändert.

3.6 Pipeline Analysis

Pipeline Analysis modelliert das Pipeline Verhalten um die Laufzeit von sequentiellen Flüssen (basic blocks) von Anwendungen zu berechnen. Daraus wird die Laufzeit für jeden basic block in jedem ausgezeichneten Anwendungskontext berechnet. Der tatsächliche Output der Pipeline Analysis ist die Anzahl der Abläufe, die ein basic block benötigt um ausgeführt zu werden.

3.7 Path Analysis

Die Path Analysis nutzt das Ergebnisse der vorausgegangen Analyse Schritte um einen möglichen Ausführungspfad des Worst-Case zu ermitteln.

3.8 Analyse von Schleifen und rekursiven Prozeduren

Mit Schleifen und rekursive Prozeduren verbraucht man einen Großteil der Laufzeit. Der erste Aufruf des Schleifen-Bodys lädt den Cache. Da spekulativ im Vorfeld geladen wird, kann es beim zweiten Aufruf zu einer Veränderung des Cache-Kontext kommen. Deswegen wird der erste Aufruf seperat von den weiteren betrachtet. Durch obere Schranken für Schleifendurchläufe ist es möglich alle Iterationen zu analysieren. Das erhöht die Genauigkeit des Ergebnis, allerdings auch die Zeit für die Analyse.

3.9 Annotierungen

Ein Tool wie aiT benötigt zusätzlich Informationen vom Benutzer um überhaupt ein Ergebnis zu erzielen. Wichtige Annotierungen sind die, welche den Zielbereich der berechneten Aufrufe und Verzweigungen festlegen, genauso wie die Anmerkungen, welche die Höchstzahl an Iterationen bei Schleifen enthalten.

4 Zusammenfassung

Worst-Case Laufzeit Analyse ermöglicht die Entwicklung von komplexen Echt-Zeit-Systemen auf hochmoderner Hardware, erhöht die Sicherheit und spart Zeit bei der Entwicklung. Präzise Zeitprognosen machen es möglich, die kosteneffizienteste Hardware auszuwählen. Tools wie aiT sind von hoher Bedeutung, da die neuesten Entwicklungen das Wissen der Worst-Case Laufzeit Analyse benötigen.