(Bild: Ovidiu Iordachi – Fotolia)

Echtzeitanalyse für Software-gesteuerte Systeme

Echtzeitsysteme haben nur beschränkt Zeit, auf Signale zu reagieren. Sind Reaktionsfristen vorgegeben, so muss der Software-Entwickler nachweisen, dass sein System diese einhält. Wann man Tests vertrauen kann und wann man auf zuverlässigere Analysemethoden setzen sollte und wie diese funktionieren, zeigt dieser Überblick über Techniken der Worst-Case-Execution-Time-Analyse.

Von Prof. Reinhard Wilhelm

Angenommen, man misst die Antwortzeiten eines von Software gesteuerten Systems und weist als Testabdeckung 100 % Entscheidungsüberdeckung für die Software nach. 100 % Erreichung dieser Überdeckung ist für Software mit hohen Ansprüchen an die Software-Integrität üblich [1]. Was bedeutet diese strukturelle Testabdeckung für den Nachweis der Echtzeiteigenschaften? Leider gar nichts. Um eine obere Schranke für die Ausführungszeit zu berechnen, reicht es zum Beispiel nicht, alle Anweisungen in einer Schleife zweimal auszuführen. Man muss sie so oft ausführen, wie es in irgendeiner Programmausführung möglich ist. Doch auch mit nur zwei Schleifeniterationen lässt sich 100 % Entscheidungsüberdeckung erreichen.

Man kennt im Allgemeinen nicht die Eingabedaten, welche den schlechtesten Fall, also die längste Ausführungszeit verursachen. Es ist daher schon besser, Ausführungszeiten so zu messen, dass man das betrachtete Programm mehrfach mit verschiedenen Eingabedaten, Grenzwerten und Grenzwertkombinationen ausführt. Dabei geht man von der Vermutung aus, dass Grenzwerte an der Systemgrenze auch die längste Ausführungszeit verursachen. Doch was ist, wenn diese Vermutung falsch ist?

Selbst wenn man die im Bezug auf die Ausführungszeit schlimmsten Eingabedaten wüsste, würde das in den meisten Projekten das Problem nicht lösen. Denn bei den heute in eingebetteten Systemen verwendeten Hochleistungsprozessoren hängt die Ausführungszeit stark von der Architektur und dem initialen Ausführungszustand ab, also z.B. dem Zustand des Cache-Speichers und der Pipeline. Den schlechtesten initialen Zustand zu identifizieren ist so schwierig wie das Zeitanalyseproblem durch eine erschöpfende Durchmusterung des gigantisch großen Zustandsraums zu lösen.

Wenn man nicht viel Aufwand betreiben will oder kann, muss man zufrieden sein, endlich viele Ausführungszeiten durch das Ausführen des Programms mit verschiedenen Eingaben und Anfangsbedingungen zu sammeln und dann deren Maximum als „wahrscheinliche Ausführungszeit im schlechtesten Fall“ zu akzeptieren. Allerdings weiß man nicht, wann man mit dem Messen aufhören sollte. Wenn sich einmal keine Zeit oberhalb des Maximums ergeben hat? Zweimal hintereinander? Es gibt kein technisch begründbares Kriterium.

WCET-Analyse

Mit dem Messen von Ausführungszeiten lässt sich im Allgemeinen keine garantierte Ausführungszeit im schlechtesten Fall bestimmen. Es besteht aber kein Anlass zur Verzweiflung, denn für die Analyse des „Worst Case Execution Timing“ (WCET) gibt es Methoden und Werkzeuge, die sowohl korrekt und effizient als auch benutzbar sind.

Die Ausführungszeiten eines Programms hängen zunächst von der Eingabe in das Programm ab. Sie bestimmt den Pfad, den die Ausführung durch das Programm nimmt. Verschiedene Eingaben führen i.A. zu verschiedenen Pfaden. Es ist klar, dass die Iterationszahlen von Schleifen und die Rekursionstiefe von Funktionen begrenzt sein müssen. Sonst lassen sich keine endlichen oberen Schranken für die Ausführungszeiten bestimmen.

Bei einfachen Microcontrollern wäre diese Forderung ausreichend für eine einfache Methode, Zeitschranken zu bestimmen. Diese haben konstante Ausführungszeiten für ihre Befehle. Man suche also einen längsten Pfad durch das Programm, setze für alle Befehle auf diesem Pfad die entsprechenden Ausführungszeiten ein, summiere auf, fertig.

Eventuell ist der Suchraum aller Pfade zu groß. Dann bedient man sich der Methode der Zeitschemata (timing schemata) [2]. Sie gehen bottom-up über die Struktur des Programms und berechnen jeweils obere Zeitschranken für ein Programmkonstrukt aus den Zeitschranken (upper bounds, ub) seiner Komponenten. Hier sind zwei Beispiele:

➜ Anweisung (A1):
if b then u else v
Upper Bound (ub): ub(A1) = ub(b) + max(ub(u), ub(v))

➜ Anweisung (A2):
for i := 1 to 17 do u
Upper Bound (ub): ub(A2) = ub(i:=1) + 17(ub(i:=i + 1) + ub(?(i == 17)) + ub(u))

Diese Abschätzung rechnet mit konstanten Ausführungszeiten der im Programm vorkommenden Befehle, die meist in einer Tabelle zur Verfügung stehen. Die Methode ist einfach und intuitiv klar. Was man aber sieht, ist, dass nicht tatsächliche Ausführungszeiten im schlechtesten Fall, sondern obere Schranken für die Ausführungszeiten berechnet werden. Denn im Zeitschema für bedingte Anweisungen A1 bilden wir das Maximum der oberen Zeitschranken der beiden Fälle. Wenn nur einmal der schnellere Fall durchlaufen wird, haben wir den Beitrag dieser bedingten Anweisung zur Gesamtlaufzeit des Programms überschätzt. Anstatt die Menge aller Pfade erschöpfend auszuführen, um die exakte Ausführungszeit im schlechtesten Fall zu bestimmen, wäre es wünschenswert, Zeitschemata durch Methoden abzulösen, die die Ausführungszeiten defensiver, aber trotzdem sicher nach oben abschätzen.

Diese Ablösung wurde durch technischen Fortschritt bei Systemarchitekturen dringend notwendig gemacht: Die eben geschilderte Methode der Zeitschemata war nicht mehr einsetzbar, als eingebettete Systeme auf Hochleistungsprozessoren mit Caches, tiefen Pipelines, Out-of-Order-Ausführung von Befehlen und Spekulation realisiert wurden. Diese Prozessoren wurden von den Rechnerarchitekten für die Verwendung in Rechnern auf oder unter Tischen entworfen. Das Ziel war und ist hohe Leistung im Durchschnitt. Die eben geschilderten Errungenschaften der Rechnerarchitektur wie Caches und Pipelines führten dazu, dass die Ausführungszeiten von Befehlen jetzt nicht mehr konstant sind, sondern vom jeweiligen Ausführungszustand des Prozessors abhängen. Die Ausführungszeit eines Speicherzugriffs hängt heutzutage entscheidend davon ab, ob der Inhalt der zugegriffenen Speicherzelle gerade im Cache liegt oder nicht. Der Unterschied zwischen dem schnellsten Zugriff – der Fall eines Cache Hit – und dem langsamsten Zugriff – dem Cache Miss – kann mehr als den Faktor 100 betragen. Ähnlich ist die Größenordnung der Variabilität der Ausführungszeiten von Befehlen. In den meisten Fällen werden die Ausführungen schnell sein. Aber eine solche stochastische Argumentation reicht für kritische Echtzeitsysteme nicht aus. Man benötigt eine Garantie für jede Ausführung.

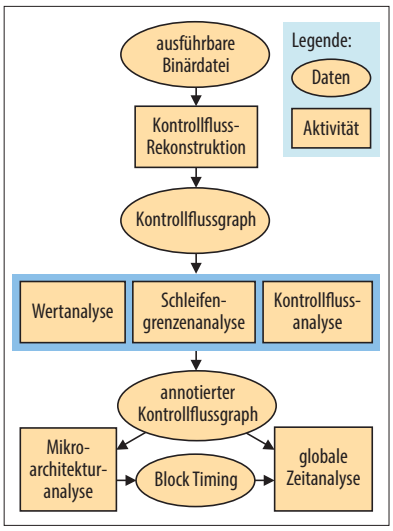

Bild 1. Eine Werkzeugarchitektur zur Zeitanalyse.

Wenn man für alle Befehle jeweils ihre längste Ausführungszeit annimmt, hätte man natürlich eine solche Garantie. Nur leider würde diese die tatsächliche schlechteste Ausführungszeit i.A. sehr stark überschätzen. Die so erhaltenen Schranken wären bei Weitem nicht präzise genug. Die Herausforderung ist also, auf Prozessoren mit variablen Ausführungszeiten für Befehle eine effiziente Methode zu finden, mit der zuverlässige und präzise obere Laufzeitschranken bestimmt werden können.

Laufzeitanalyse

Dieser Abschnitt des Artikels stellt nun eine statische Methode vor, die zuverlässig und präzise Laufzeitschranken berechnet. Statisch heißt, sie beruht nicht auf dem Messen von Ausführungszeiten, sondern auf der Analyse des Programm(text)s. Da sie keine spezielle Eingabe und auch keinen speziellen Anfangszustand annimmt, gelten ihre Ergebnisse für alle Ausführungen. Die Methode ist effizient genug für die industrielle Praxis.

Die Ergebnisse einer Laufzeitanalyse, also eine obere Schranke für alle Ausführungszeiten eines Programms, werden benutzt, um sie mit der vorgegebenen Reaktionsfrist zu vergleichen, wenn das Programm allein auf einem Prozessor läuft. Laufen mehrere zeitkritische Programme auf einem Prozessor, so müssen für alle diese Programme solche Laufzeitschranken bestimmt werden. Diese Schranken sind dann Eingaben in eine Planbarkeitsanalyse (schedulability analysis). Sie stellt fest, ob alle Programme so auf dem gegebenen Prozessor eingeplant werden können, dass sie alle ihre Reaktionsfristen einhalten.

Die verwendete Methode zur Laufzeitanalyse beruht auf der Theorie der abstrakten Interpretation. Ein abstrakter Interpreter für eine Klasse von Eigenschaften von Programmzuständen untersucht ein Programm daraufhin, welche dieser Eigenschaften an den Programmpunkten für alle dort möglichen Ausführungszustände gelten. Ein Beispiel ist die Cache-Analyse. Eine mögliche Werkzeugarchitektur ist in **Bild 1** dargestellt. Die Aufgabe der einzelnen Phasen und die verwendeten Techniken werden im Folgenden beschrieben.

Die **Kontrollflussrekonstruktion** (control-flow reconstruction) analysiert ein ausführbares Binärprogramm und erstellt ein Modell für den Kontrollfluss. Sie identifiziert insbesondere die Schleifen und die Funktionen im Programm. Schwierigkeiten machen eventuell die Befehlssequenzen für Switch-Anweisungen, indirekte Funktionsaufrufe durch Funktionszeiger und die Identifizierung von Rücksprüngen aus Funktionen, wenn der Prozessor dafür keinen Befehl hat und der Rücksprung deshalb indirekt durch ein Register erfolgt. Das Ergebnis dieser Phase ist ein Kontrollflussgraph und ein Aufrufgraph mit Assembler-Befehlsfolgen in Basisblöcken.

Die **Wertanalyse** (value analysis) versucht, an allen Programmpunkten

herauszufinden, welche Inhalte die verwendeten Prozessorregister und wichtige Programmvariablen haben. Da ein Register im Laufe einer oder mehrerer Programmausführungen mehrere verschiedene Werte enthalten kann, berechnet die Wertanalyse ein alle möglichen Werte umfassendes Intervall. Die Ergebnisse werden für verschiedene Zwecke benutzt, insbesondere die Schleifengrenzenanalyse, die Kontrollflussanalyse und die Daten-Cache-Analyse.

Die **Schleifengrenzenanalyse** (loop-bound analysis) versucht, basierend auf den Ergebnissen der Wertanalyse, die Unter- und Obergrenzen für die Schleifeniterationen zu bestimmen. Wenn das nicht für alle Schleifen gelingt, wird die Angabe der fehlenden Grenzen vom Entwickler verlangt.

Die **Kontrollflussanalyse** (control flow analysis) versucht, nicht ausführbare Pfade zu identifizieren. Solche können die Präzision der Laufzeitbestimmung verschlechtern.

Die **Mikroarchitekturanalyse** (microarchitecture analysis) versucht, Informationen über die Ausführungszustände der Plattform zu berechnen. Wie oben gesagt, hängen die Ausführungszeiten der Maschinenbefehle von den jeweiligen Ausführungszuständen ab. Diese Ausführungszustände wiederum sind Ergebnis der bisherigen Ausführungsgeschichte. An einem Programmpunkt sind i.A. verschiedene Ausführungszustände möglich, wenn man nämlich die Programmausführung in verschiedenen Anfangszuständen beginnen und/oder auf verschiedenen Pfaden zu dem Programmpunkt kommen kann. Die Mikroarchitekturanalyse analysiert ein Programm daraufhin, welche Ausführungszustände an jedem Programmpunkt auftreten können. Sie berechnet diese Menge approximativ, genauer, sie berechnet eine Obermenge der Menge aller möglichen Zustände. Was hat man damit gewonnen? Betrachten wir zwei Komponenten der Ausführungsplattform, die Caches und die Pipeline.

Cache und Pipeline

Eine Komponente der Mikroarchitekturanalyse analysiert ein Programm daraufhin, welche Speicherblöcke an einem Programmpunkt auf jeden Fall im Cache sind, wann immer die Programmausführung diesen Punkt erreicht. Sie berechnet eine Unterapproximation der Menge der sicher im Cache befindlichen Speicherblöcke. Wenn an einem Programmpunkt auf einen Speicherblock *a* zugegriffen wird und die Cache-Analyse berechnet hat, dass *a* garantiert im Cache ist, dann kann die Zeitanalyse sicher annehmen, dass dieser Zugriff nur die Zeit für einen Zugriff auf den Cache-Speicher benötigt. Wenn die Cache-Analyse nicht herauskriegt, dass *a* garantiert im Cache ist, dann muss man im Prinzip mit beiden Fällen rechnen, einem Cache Hit und einem Cache Miss.

Man könnte meinen, dass es in diesem Fall reichte, den langsameren Fall, den Cache Miss, zu betrachten. Leider haben moderne Hochleistungsprozessoren sogenannte Zeitanomalien. Dies ist z.B. dann der Fall, wenn ein Cache Miss eine Fehlspekulation bei einem bedingten Sprung verhindert. Die Existenz von Zeitanomalien erzwingt eine vollständige Durchmusterung des Raums der Pfade durch die Architektur. Dies verursacht den größten Aufwand bei der Zeitanalyse.

Die Befehls-Pipeline eines Prozessors ist in der Lage, mehrere aufeinander folgende Befehle in einer überlappenden Weise parallel zueinander auszuführen und damit die Programmausführung zu beschleunigen. Die größte Beschleunigung ergibt sich bei einer perfekt gefüllten Pipeline. Allerdings gibt es eine

Elektronik
embedded
Fachmedium für die Entwicklung von Embedded-Systemen

Redaktion

Anschrift: Redaktion Elektronik
Richard-Reitzner-Allee 2, 85540 Haar
Assistenz: Andrea Seidel, Silvia Langford
Telefon: 089 25556-1332; Telefax: -1670
Internet: www.elektroniknet.de
E-Mail: redaktion@elektronik.de

Chefredakteur:
Dipl.-Ing. Gerhard Stelzer (gs) (verantwortlich)

Stellvertretender Chefredakteur:
Dipl.-Ing. Joachim Kroll

Chef vom Dienst:
Dipl.-Ing. Achim Grolman (-1333)

Redaktion: Stefanie Eckardt (eck/ -1342): Automotive, Kfz-Elektronik; Dipl.-Ing. (FH) Andrea Gillhuber (ag/ -1609): Antriebstechnik, Erneuerbare Energien, Energiespeicher + Ladekonzepte, Leistungselektronik, Sensorik, Stromversorgung; Dipl.-Ing. (FH) Alfred Goldbacher (go/ -1366): Aufbau- und Verbindungstechnik, CA-Techniken, Elektromechanik, passive Bauelemente, Produkte; Dipl.-Ing. (FH) Wolfgang Hascher (ha/ -1368): Chipkarten + RFID, Energy Harvesting, Kommunikation, Messen + Testen, Wireless (HF-Technik); Dipl.-Phys. Irina Hübner (ih/-1371): Analog-/Mixed-Signal-ICs, Bildverarbeitung, Displays, Optoelektronik; Dipl.-Ing. Joachim Kroll (jk/ -1335): Automatisierung, Computertechnik, Software-Entwicklung, Embedded Design; Dr. Ingo Kuss (ku/-1369): Automotive, Kfz-Elektronik; Dipl.-Phys. Helmuth Lemme (le/ -1332): HF-Technik, Optoelektronik, Sensorik; Dipl.-Ing. Frank Riemenschneider (fr/ -1714): Medizinelektronik, Mikroelektronik, Prozessoren, SoCs/ASICs/programmierbare Logik, Speicher-ICs; Dr. Jens Würtenberg (jw/ -1338): Beruf + Karriere, Distribution, Forschung + Entwicklung, Konsumelektronik + Multimedia, Stromversorgung, Wirtschaft

Korrespondent: W. Schulz, Berlin

Layout, Grafik: Hermann Schmitzberger, Anja Schumann, Norma Alfes-Bodinger

Titel: Norbert Preiß

Sonderdrucke: Alle in dieser Ausgabe erschienenen Beiträge können für Werbezwecke in Form von Sonderdrucken hergestellt werden. Anfragen an Dominik Popp, Tel.: 089 25556-1450, E-Mail: dpopp@wekanet.de

Technik: JournalMedia GmbH, Richard-Reitzner-Allee 4, 85540 Haar

Druck: L. N. Schaffrath, Marktweg 42–50, 47608 Geldern

Verlag

Anschrift: WEKA FACHMEDIEN GmbH
Richard-Reitzner-Allee 2, 85540 Haar
Telefon: 089 25556-1000
Telefax-Anzeigen: 089 25556-1670
www.weka-fachmedien.de

Gesamtanzeigenleitung:
Peter Eberhard (-1385), verantw.
peberhard@weka-fachmedien.de

Anzeigenverkaufsleitung, International Account Manager:
Sonja Winkler (-1383),
swinkler@weka-fachmedien.de

Mediaberatung:
media@weka-fachmedien.de
Nicole Müller (-1384),
nmueller@weka-fachmedien.de
Konrad Nadler (-1382),
knadler@weka-fachmedien.de
Bernhard Reinisch (-1381),
breinisch@weka-fachmedien.de

Anzeigen Karriere-Markt:
089 25556-1374,
stellenanzeigen@weka-fachmedien.de

Disposition: Hildegund Rößler (-1473),
hroessler@wekanet.de

Anzeigenpreise nach Preisliste 49, gültig ab 1. 1. 2014.

Publisher ITK: Matthäus Hose (-1302)
Vertriebsleitung: Marc Schneider (-1509),
mschneider@weka-fachmedien.de
Herstellungsleitung: Marion Stephan (-1442)

Verlagsleitung: Peter Eberhard, Matthäus Hose
Geschäftsführung: Kurt Skupin, Werner Mützel, Wolfgang Materna

Bestell- und Abonnement-Service:
WEKA FACHMEDIEN GmbH
c/o Zenit Pressevertrieb GmbH
Postfach 810640, 70523 Stuttgart
Tel. 0049 (0)711 7252-210
Fax 0049 (0)711 7252-333
E-Mail: abo@weka-fachmedien.de

Der Preis für dieses Sonderheft ist in der Abonnementgebühr für die Zeitschrift Elektronik enthalten.

Bankverbindung:
HypoVereinsbank
BLZ: 700 202 70; Konto-Nr. 35 70 49 81
Einzelheftbestellung:
Einzelheft: € 12,80 zzgl. € 3,00 Versand

ISSN 0013-5658s
Vertriebskennzeichen B 2594

Verlagsvertretungen

Anzeigenvertretung Ausland:
Great Britain: Huson International Media, Gerald Rhoades-Brown, Cambridge House, 8 Gogmore Lane, Chertsey, Surrey, KT16 9AP, Tel. 0044 (0)1932 564 999, Fax 0044 (0)1932 564 998, gerry.rhoadesbrown@husonmedia.com

Benelux, Skandinavien, Frankreich: Huson International Media, Breitnerhof 3, 1628 XL Hoorn, The Netherlands, Tel. 0031 229-841882, Fax 0031 84 7488240, rodric.leerling@husonmedia.com

Taiwan: ACTEAM Int. Marketing Corp., Anita Chen, 11 F-6, No. 170, Sec. 4, Nanjing E. Rd., Taipei, Taiwan 105, R.O.C., Tel. 00886-2-2577-1000, Fax 00886 2-2577-7177

USA West: Huson International Media, Ralph Lockwood, Pruneyard Towers, 1999 South Bascom Avenue, Suite 450, Campbell, CA 95008, Tel. 001 408 879 6666, Fax 001 408 879 6669, ralph.lockwood@husonmedia.com

USA East: Huson International Media, Francesco Lascari, Empire State Building, 350 Fifth Avenue, Suite 2719, New York, NY 10118, Tel. 001 212 268 3344, Fax 001 212 268 3355, francesco.lascari@husonmedia.com

Japan: Shinano International, Inc., Akasaka Kyowa Bldg. 2F, 1-6-14 Akasaka, Minato-ku, Tokyo 107-0052 Japan, Tel.: 0081 3-3584-6420, Fax: 0081 3-3505-5628, E-Mail: scp@bunkoh.com

Reihe von Gründen, weshalb die Pipeline eventuell nicht perfekt gefüllt ist: Ein Befehl verursacht beim Laden aus dem Speicher einen Cache Miss, oder ein Befehl kann nicht in die arithmetische Einheit zur Ausführung seiner Operation kommen, weil alle von vorangehenden Befehlen belegt sind, oder ein Befehl kann seine Operation nicht ausführen, weil ein Operand noch nicht von einem vorangehenden Befehl berechnet worden ist. In all diesen Fällen muss die Pipeline für einige Zyklen anhalten.

Die Mikroarchitekturanalyse versucht, möglichst viele von diesen Gründen auszuschließen und damit die Zahl der Wartezyklen zu begrenzen, die auf die Ausführungszeit der Befehle addiert werden müssen.

Interrupts

Die Zeitanalyse von Programmen, die durch Interrupts unterbrochen werden können, ist aus mehreren Gründen schwierig: Zum ersten kann ein Interrupt Handler einen Effekt auf den Ausführungszustand haben, welchen die Mikroarchitekturanalyse berücksichtigen müsste. Wenn Interrupts an beliebiger Stelle zugelassen sind, müsste dieser Effekt überall einbezogen werden, um auf der sicheren Seite zu sein. Dadurch würden die Analyseergebnisse sehr unpräzise. Zweitens müsste der Zeitaufwand für die Behandlung aller möglichen Unterbrechungen in die obere Laufzeitschranke einkalkuliert werden. Gibt es keine Beschränkung in der Zahl der Unterbrechungen, ist die Berechnung einer sicheren Laufzeitschranke unmöglich.

Es gibt mehrere Abhilfen: In sicherheitskritischen Flugzeugsystemen werden häufig Unterbrechungen vollständig verboten. Stattdessen wird Polling verwendet. Um überhaupt eine Analyse zu ermöglichen, muss es Beschränkungen für die Zahl der Unterbrechungen geben, z.B. in der Form von Mindestabständen zwischen je zwei aufeinanderfolgenden Unterbrechungen. Alternativ kann man auch Bereiche des zu analysierenden Programms für Unterbrechungen sperren.

Werkzeuge für die WCET-Analyse

Die beschriebene Methode der WCET-Analyse ist in der Werkzeugfamilie aiT der Firma AbsInt realisiert. Wie man oben gesehen hat, hängt die Zeitanalyse stark von der Ausführungsplattform ab. Jede Ausführung eines aiT-Werkzeugs für eine spezielle Architektur enthält ein abstraktes Modell der Plattform. Die Modellierung einer neuen Plattform ist sehr aufwendig und benötigt eine präzise Beschreibung dieser Plattform. Der größte Teil dieser Beschreibung, z.B. die Beschreibung des Prozessors, wird vom Werkzeugentwickler in das Werkzeug integriert. Eventuell bleibt ein Rest übrig, welcher vom Systementwickler per Annotation dem Werkzeug bekannt gemacht werden muss. Dieser Rest betrifft die speziell benutzte Hardware-Konfiguration, z.B. die Speicher- und Cache-Konfiguration mit ihren Parametern. aiT wurde inzwischen für viele Plattformen realisiert. Die wichtigsten sind Freescale PowerPC 5xx/55xx, 7448, 750, 755, Renesas V850, Infineon TriCore 1796/1797, Infineon C16x/ST10 und Leon 2/3.

In Ergänzung zu statischen Zeitanalysen gibt es auch Mischformen aus Analysemethoden und Messung. Man misst eine gewisse Zahl von Programmläufen und bestimmt eine obere Schranke der beobachteten Ausführungszeiten (worst observed execution times). Im Gegensatz zum Messen von vollständigen Ausführungszeiten, also von Programmstart bis zur Terminierung, kann man bei diesem Ansatz das Programm in Programmstücke

zerlegen, die Ausführungszeiten dieser Programmstücke in einer gewissen Zahl von Startzuständen messen, obere Schranken von den gemessenen Ausführungszeiten bilden und diese gemäß der zuvor erwähnten Zeitschemata zusammensetzen. Diesen Ansatz verfolgt das Werkzeug Rapitime der Firma Rapita Systems.

Problematisch bei allen auf Messung basierenden Werkzeugen ist, dass man i.A. nicht alle Ausführungen messen kann und deshalb nie sicher sein kann, den Fall der längsten Ausführungszeit auch tatsächlich gemessen zu haben. Diese Methoden können also keine Garantien liefern.

Timing-Analyse ist Bestandteil mehrerer Normen

Einige internationale Normen legen fest, welche Methoden für die Verifikation von funktionalen wie nichtfunktionalen Eigenschaften sicherheitskritischer Systeme zugelassen sind oder sogar verwendet werden müssen. DO-178B und DO-178C sind internationale Normen für die Entwicklung von Software für zivile Flugzeuge. Beide fordern für die höchste Kritikalitätsstufe den Einsatz formaler Methoden zum Nachweis nichtfunktionaler Eigenschaften wie Pünktlichkeit, beschränkter Platzverbrauch und Abwesenheit von Laufzeitfehlern.

Die Norm EN 61508 empfiehlt dringend die Verwendung von statischer Zeitanalyse für alle Kritikalitätsstufen. Ein Kriterium für die Anwendbarkeit einer Methode ist ihre Vollständigkeit. Diese ist, wie wir gesehen haben, durch Testen i.A. nicht zu erreichen. Statische Analysen, die auf der Theorie der abstrakten Interpretation beruhen, akzeptiert die Norm bezüglich ihrer Vollständigkeit als mathematisch bewiesen. Ihre Zuverlässigkeit wird auf der höchsten Stufe angesiedelt.

ISO 26262 verlangt für das Umfeld der Automobilelektronik, dass Echtzeitbedingungen Teil der Systemspezifikation sind, verlangt eine Abschätzung einer oberen Schranke der Ausführungszeit, empfiehlt formale Verifikation und verlangt Kontrollfluss- und Datenflussanalyse für die beiden höchsten Kritikalitätsstufen.

Werkzeugunterstützte WCET-Analyse wird für sicherheitsrelevante Software der Flugzeugindustrie routinemäßig eingesetzt [3]. Das Werkzeug aiT wurde 2011 im Rahmen der Untersuchung der amerikanischen Verkehrssicherheitsbehörde über die unbeabsichtigte Beschleunigung von Toyota-Fahrzeugen als Standardwerkzeug zur Zeitanalyse eingesetzt und konnte zeigen, dass keine WCET-Schranken überschritten wurden [4]. Diese Tatsachen legen nahe, dass statische WCET-Analysen als Stand der Technik angesehen werden und zur Klärung von Haftungs- und Schadenersatzfragen berücksichtigt werden können.

Mit Ausführungsmodell oder auf Basis von Messungen

Wir haben zwei Alternativen für die Zeitanalyse kennengelernt. Eine führt eine Reihe von statischen Analysen des ausführbaren Programms durch und benutzt dabei ein abstraktes Modell der Ausführungsplattform. Diese ist in der Lage, eine Garantie für das Zeitverhalten abzugeben. Allerdings ist das Erstellen von abstrakten Modellen komplexer Ausführungsplattformen sehr aufwendig.

Auf der anderen Seite gibt es messbasierte Verfahren, welche ohne das Modellieren der Ausführungsplattform auskommen. Sie können allerdings keine Garantien liefern. Bei jeder Zeitmessung muss beachtet werden, dass die Software im Feld völlig identisch zur vermessenen Software sein muss. Das heißt, eingefügte Instrumentierungspunkte zur Zeitmessung müssen im Code bleiben. Jede geringfügige Änderung, auch das Streichen von Code, macht neue Zeitmessungen notwendig, weil Cache-Effekte i.A. nicht vorhersehbar sind.

Beide Analysemethoden sind neben vielen Testmethoden im 2013 erschienenen Buch „Software-Test für Embedded Systems" [1] genauer beschrieben.

jk

Reinhard Wilhelm
ist Informatikprofessor an der Universität des Saarlandes, wo er den Lehrstuhl für Programmiersprachen und Übersetzerbau innehat. Er gilt als einer der führenden Wissenschafter auf dem Gebiet der Echtzeitanalyse. 2009 wurde er mit der Konrad-Zuse-Medaille für Verdienste in Forschung und Lehre auf dem Gebiet des Übersetzerbaus und der Echtzeitanalyse von Programmen sowie für seine Tätigkeit als wissenschaftlicher Direktor des Leibniz-Zentrums für Informatik in Schloss Dagstuhl ausgezeichnet.
wilhelm@cs.uni-saarland.de

Literatur

[1] *Grünfelder, S.:* Software-Test für Embedded Systems. Dpunkt-Verlag, 2013.
[2] *Shaw, A.:* Reasoning about Time in Higher-Level Language Software. IEEE Transactions on Software Engineering, Vol. 15, No. 7, S. 875–889. Juli 1989.
[3] *Thesing, S., Souyris, J., Heckmann, R. Randimbivololona, F. Langenbach, M. Wilhelm, R. Ferdinand, C.:* An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. Proceedings of the International Conference on Dependable Systems and Networks (DSN 2003), 22.–25. Juni 2003, S. 625–632. Online: www.informatik.uni-trier.de/~ley/db/conf/dsn/dsn2003.html.
[4] Technical Support to the National Highway Traffic Safety Administration (NHTSA) on the Reported Toyota Motor Corporation (TMC) Unintended Acceleration (UA) Investigation, 2011. Online: www.nhtsa.gov/staticfiles/nvs/ pdf/NASA-UA_report.pdf.