Validierung des Zeitverhaltens von kritischer Echtzeit-Software

Dr.-Ing. Christian Ferdinand, Dr.-Ing. Daniel Kästner, Dr.-Ing. Florian Martin, Marc Langenbach, Martin Sicks, Stephan Wilhelm, Dr. Reinhold Heckmann, Nico Fritz, Stephan Thesing, Frank Fontaine, Dr.-Ing. Henrik Theiling, Michael Schmidt, Alexander Evstiougov-Babaev, Prof. Dr. Reinhard Wilhelm

AbsInt Angewandte Informatik GmbH

Schlüsselworte: Sicherheit, Echtzeit, zeitgesteuerte Software, TTA, FlexRay, X-by-Wire, Planbarkeits-Analyse (Schedulability Analysis), Validierung, Laufzeitbestimmung im schlimmsten Fall (WCET worst case execution time prediction)

1 Validierung des Zeitverhaltens

Viele Steuergeräte in modernen Automobilen sind sicherheitsrelevant, wie ABS-, Airbag-, Motor- oder Getriebesteuerung. Ein Fehler in diesen Anwendungen kann fatale Folgen für die Insassen und andere Verkehrsteilnehmer haben.

Neue Anwendungen (X-by-Wire, z.B. Lenkung) werden immer komplexer und stellen sehr hohe Ansprüche an das zuverlässige Funktionieren der Steuerprogramme. Voraussetzung ist allerdings immer, daß das Steuerprogramm korrekt arbeitet. Bei Echtzeitanwendungen hängt die Korrektheit nicht nur vom logischen Ergebnis ab, sondern auch vom Zeitpunkt, wann das Ergebnis geliefert wird. Wenn die richtige Antwort zu spät berechnet wird, kann das schwerwiegende Folgen haben.

Die verbreiteten Methoden zur Verbesserung der Programmqualität wie Definition von Programmentwicklungsprozessen, Einsatz von Testwerkzeugen, Code-Reviews, oder Einsatz von Modellierungswerkzeugen und Code-Generatoren, zielen in der Regel nur auf die logische Korrektheit. Zur Validierung des Zeitverhaltens muß deshalb meistens auf die mühsame und schwierige direkte Laufzeitmessung zurückgegriffen werden. Die Validierung des Zeitverhaltens ist aus verschiedenen Gründen eine Herausforderung:

- Die Bestimmung der maximalen Laufzeit durch (wiederholtes) Messen der Laufzeit auch von kurzen Programmstücken ist oft nicht sicher. Beispielsweise ist es oft unmöglich zu beweisen, daß die Bedingungen, die zur höchsten Laufzeit führen, berücksichtigt wurden.
- Die Rechenleistung von Mikrocontrollern wird immer höher. Es gibt einen Trend, deshalb immer mehr Funktionen auf ein einziges Steuergerät zu integrieren. Ein vollständiges Testen des Systems mit allen möglichen Eingaben und allen möglichen Verteilungen über die Zeit der "externen Ereignisse", die zu einem Interrupt führen, ist in der Regel nicht möglich.
- Heutzutage wird Software in der Regel in höheren Programmiersprachen wie C, C++, Java oder Ada entwickelt. Compiler-Optimierungen machen es für den Entwickler schwer, die Laufzeit ihres Codes abzuschätzen.
- Software wird bei komplexeren Anwendungen von Teams entwickelt. Der Anteil von Fremd-Software wie Betriebssystemen oder Kommunikationsbibliotheken steigt. Das Zeitverhalten der interagierenden Komponenten ist nur selten genau bekannt.

– Moderne Prozessorarchitekturkomponenten wie Pipelines und Caches erschweren die Bestimmung der Laufzeit erheblich, da die Laufzeit einer einzelnen Instruktion von der Ausführungsgeschichte abhängen kann.

Echtzeitsysteme bestehen meist aus einer Menge von Tasks mit festen "Deadlines", die sich aus der physikalischen Umgebung ergeben. Mit einer sogenannten Planbarkeitsanalyse (Schedulability Analysis) muß bewiesen werden, daß alle Zeitanforderungen unter allen Umständen eingehalten werden. Zur Planbarkeitsanalyse gibt es viele Forschungsresultate. Im praktischen Einsatz verbreitet sind Methoden, die auf RMA (Rate-Monotonic Approach, siehe [LL73]) basieren. Ein weiterer Ansatz, um komplexe sicherheitskritische Echtzeitsysteme zu entwerfen, ist der zeitgesteuerte Ansatz (time-triggered approach). Der zeitgesteuerte Ansatz wurde beispielsweise erfolgreich bei der Implementierung von Fly-by-Wire-Systemen in Flugzeugen eingesetzt. In Kombination mit zeitgesteuerten Kommunikations-Bussen (TTA, FlexRay, TTCAN, SAFEbus, ...) und zeitgesteuerten Echtzeitbetriebssystemen (z.B. OSEKtime) ist es möglich, verteilte sicherheitskritische Echtzeitsysteme zu entwerfen. Der zeitgesteuerte Ansatz findet in jüngerer Zeit auch im Automobilbereich mehr und mehr Anwendung (z.B. X-by-Wire).

Beide Ansätze erfordern aber in jedem Fall, daß die Laufzeit im schlimmsten Fall von jeder Task im voraus bekannt ist. Die Laufzeit im schlimmsten Fall einer Task ist nicht immer genau berechenbar. Deshalb werden meist Abschätzungen berechnet. Diese Abschätzungen müssen sicher sein, dürfen also die wirklich auftretenden Ausführungszeiten niemals unterschätzen. Weiterhin sollten sie präzise sein, d.h. die Überschätzung sollte so klein sein wie möglich.

Bei modernen Prozessorarchitekturen mit Cache-Speichern, Jump-Target-Puffer oder Pipelines hängt die Ausführungsgeschwindigkeit einer Instruktion von der Ausführungsgeschichte ab. Kleine Änderungen im Code wie beispielsweise das Verschieben einer Instruktion um ein paar Bytes im ausführbaren Programm können die Laufzeit stark beeinflussen. Vorhersagemethoden, die dies nicht berücksichtigen, überschätzen die Laufzeit in der Regel erheblich.

2 aiT - Vorhersage der Laufzeit im schlimmsten Fall

Das hier vorgestellte Werkzeug aiT zur automatischen Laufzeitbestimmung im schlimmsten Fall wurde im IST-Projekt DAEDALUS nach den Anforderungen von Airbus France zur Validierung des Zeitverhaltens kritischer Flugsteuersoftware entworfen.

aiT berechnet automatisch eine sichere obere Schranke der Laufzeit eines Programmnstücks. Diese Schranke gilt für alle möglichen Eingaben und wird bestimmt, ohne daß eine "worst-case" Eingabe geliefert werden muß.

Auf der Grundlage der abstrakten Interpretation kombiniert aiT die folgenden Komponenten zur Vorhersage der Laufzeit im schlimmsten Fall:

– **Rekonstruktion** des Kontrollflusses aus einem ausführbaren Programm [The00] [The01].
– **Wertebereichs-Analyse** zur automatischen Adress-Bestimmung von Speicherzugriffen: Die Kenntnis der Ziele von Speicherzugriffen ist bei Laufzeitanalysen von entscheidender Bedeutung. Zu berücksichtigen sind nicht nur unterschiedliche Zugriffsgeschwindigkeiten verschiedener Speichertypen, sondern auch das Cache-Verhalten. Die Wertebereichs-Analyse berechnet daher für jeden Programmpunkt und für alle

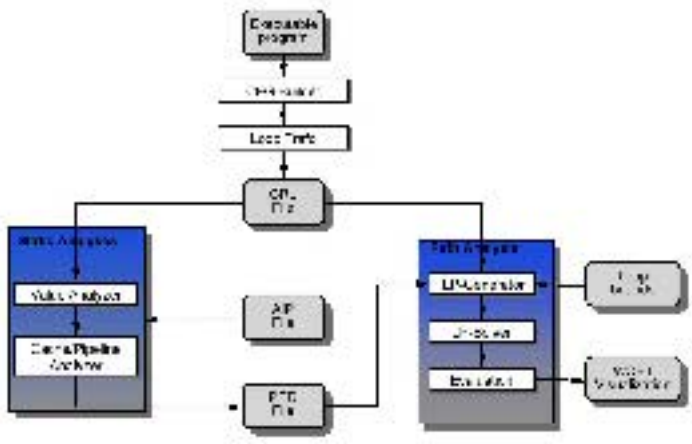

Abbildung 1. Struktur der Analyse: Der erste Schritt ist die Rekonstruktion des Kontrollflusses aus dem ausführbaren Programm (z.B. im ELF- oder COFF-Format). Dann erfolgt die Analyse des Cache- und Pipelineverhaltens durch eine statische Programmanalyse. Das Ergebnis sind die Ausführungszeiten im schlimmsten Fall der Basisblöcke in einem interprozeduralen Ausführungskontext, der die verschiedenen Pfade im Aufrufgraphen unterscheiden kann und je nach Benutzereinstellung auch verschiedene Iterationen von Schleifen. Daraus berechnet die Pfad-Analyse die Gesamtlaufzeit im schlimmsten Fall.

Register der Zielarchitektur ein Intervall, das die tatsächlichen Werte, die während der Ausführung des Programmes an diesem Programmpunkt angenommen werden können, möglichst gut und sicher approximiert.

- **Cache-Analyse** zur Vorhersage der möglichen Cache-Zustände an Programmpunkten: Der Einsatz von Cache-Speicher hat einen erheblichen Einfluß auf die Ausführungszeiten von Speicherzugriffen. Findet sich eine Kopie des adressierten Speicherblockes im Cache, so wird ein (langsamerer) Zugriff auf den Hauptspeicher vermieden. Um eine möglichst enge obere Grenze der maximalen Laufzeit zu erreichen, wird eine Cache-Analyse durchgeführt, die Speicherzugriffe als Cache-Hit (Treffer) oder Cache-Miss (Fehlzugriff) klassifiziert [Fer97].
- **Pipeline-Analyse** zur Vorhersage des zeitlichen Verhaltens der Prozessor-Pipeline: Aufgabe der Pipeline-Analyse [LTH02][SF99] ist die Berechnung der Ausführungszeiten einzelner Instruktionen für alle im Programmablauf möglichen Kontexte. Dazu werden Pipeline-Modelle verwendet, die das Zeitverhalten der Pipeline präzise modellieren. Parallel arbeitende funktionale Einheiten, spekulative Ausführung und Prefetching, wie sie beispielsweise bei dem Motorola PowerPC 755 vorkommen, machen eine präzise Analyse aufwendig. So wurde zur Verbesserung der Präzision die Cache-Analyse mit der Pipeline-Analyse integriert, um die Effekte des Prefetchings, die durch Pipeline-Analyse bestimmt werden, direkt in der Cache-Analyse berücksichtigen zu können.
- **Pfad-Analyse** zur Berechnung des längsten Ausführungspfades: Die Pfad-Analyse bestimmt aus den worst-case Laufzeiten der Basisblöcke die Gesamtlaufzeit einer Task. Dazu wird der Kontrollflußgraph zusammen mit Benutzereingaben zu maximalen Iterationszahlen von Schleifen und Rekursionstiefen und nicht ausführbaren

Pfaden in ein ganzzahligen Gleichungssystem übersetzt, dessen Lösung die Gesamtlaufzeit darstellt [TF98] [MAWF98].

– **Visualisierung** zur detaillierten graphischen Darstellung verschiedener ausführungszeitrelevanter Aspekte: Zusätzlich zur Laufzeit im schlimmsten Fall (in Maschinenzyklen oder Sekunden) bietet aiT die Visualisierung relevanter Aspekte durch **aiSee** [aiS] an. Die Möglichkeiten reichen vom Aufruf-Graphen bis zu einer detaillierten Visualisierung einzelner Pipeline-Zustände.

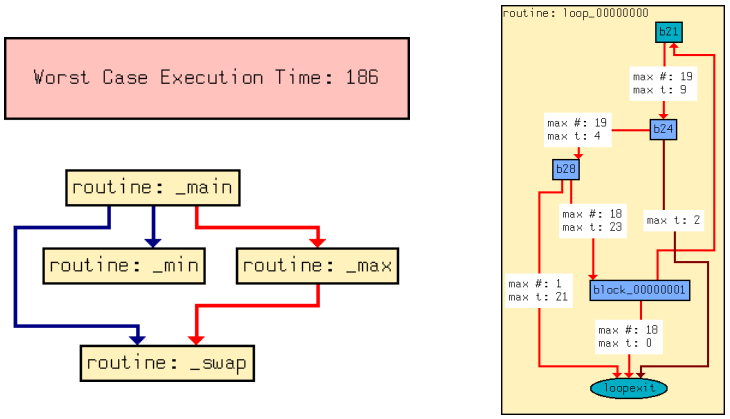

Abbildung 2. Links: Visualisierung der Analyse-Ergebnise: Im Aufrufgraphen sind alle Aufrufe (Kanten), die zur maximalen Laufzeit beitragen, farbig dargestellt. Die Laufzeit im schlimmsten Fall ist hier in Prozessorzyklen angegeben. Rechts: Basisblockgraph einer Schleife. `max #` beschreibt die maximale Durchlaufzahl einer Kante. `max t` beschreibt die maximale Ausführungszeit eines Basisblocks. Der längste Pfad, die Durchlaufzahlen und Zeiten werden automatisch von aiT bestimmt.

3 Zielarchitekturen

Im Augenblick ist aiT für die Architekturen Motorola PowerPC 755, Motorola ColdFire 5307 und ARM7TDMI verfügbar. Aktuell in Entwicklung befindet sich aiT für den Motorola MPC 555/565 und Motorola M68HC12-STAR12.

Die Entwicklung von aiT für PPC755 und CF5307 basierte auf den Anforderungen der Flugzeugindustrie zur Validierung sicherheitskritischer Flugsteuerungsanwendungen. aiT für ARM7TDMI und MPC555 hingegen ist nach den Anforderungen von Automobil-Zulieferern entworfen.

4 Zusammenfassung

Zur Validierung des Zeitverhaltens von sicherheitsrelevanten Anwendungen im Rahmen einer Planbarkeitsanalyse werden zuverlässige Abschätzungen von Task-Laufzeiten benötigt. Das automatische Analysewerkzeug aiT bietet eine Alternative zu den oftmals

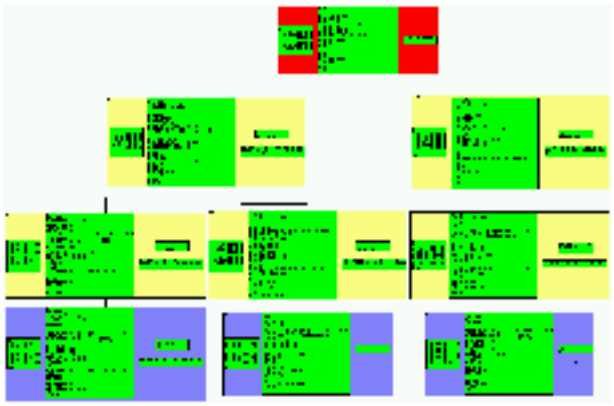

Abbildung 3. aiT für PPC755: Zeitliche Entwicklung der Cache- und Pipeline-Zustände an einem Programmpunkt. Jede Ebene im Baum entspricht einem Maschinenzyklus. Verzweigungen entsprechen unbekannten Optionen, wie beispielsweise ein Cache-Hit oder Cache-Miss bei einem Speicherzugriff, dessen Verhalten nicht präzise vorhergesagt wurde. Dabei überprüft aiT alle Möglichkeiten automatisch.

fehleranfälligen auf direkter Laufzeitmessung basierenden Verfahren und verbessert damit die Sicherheit. Direkte Laufzeitmessung ist in der Regel auch sehr aufwendig in der Durchführung, so daß aiT dazu beitragen kann, den Entwicklungsaufwand von sicherheitsrelevanten Anwendungen zu reduzieren.

Literatur

[aiS] http://www.aisee.com. *aiSee Home Page.*

[Fer97] Christian Ferdinand. *Cache Behavior Prediction for Real-Time Systems.* PhD thesis, Universität des Saarlandes, 1997.

[LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1), 1973.

[LTH02] Marc Langenbach, Stephan Thesing, and Reinhold Heckmann. Pipeline Modeling for Timing Analysis. *Proceedings of the 9th International Static Analysis Symposium*, 2002.

[MAWF98] Florian Martin, Martin Alt, Reinhard Wilhelm, and Christian Ferdinand. Analysis of Loops. In *Proceedings of the International Conference on Compiler Construction (CC'98)*. Springer-Verlag, 1998.

[SF99] Jörn Schneider and Christian Ferdinand. Pipeline Behavior Prediction for Superscalar Processors. Technical report, Universität des Saarlandes, May 1999.

[TF98] Henrik Theiling and Christian Ferdinand. Combining Abstract Interpretation and ILP for Microarchitecture Modelling and Program Path Analysis. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, Madrid, Spain, 1998.

[The00] Henrik Theiling. Extracting Safe and Precise Control Flow from Binaries. In *Proceedings of the 7th Conference on Real-Time Computing Systems and Applications*, Cheju Island, South Korea, 2000.

[The01] Henrik Theiling. Generating Decision Trees for Decoding Binaries. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, Snowbird, Utah, USA, June 2001.