SysWCET: Ende-zu-Ende-Antwortzeiten für OSEK-Systeme

Christian Dietrich
Leibniz Universität Hannover
dietrich@sra.uni-hannover.de

Peter Wägemann
Friedrich-Alexander Universität Erlangen-Nürnberg
waegemann@cs.fau.de

ABSTRACT

Um Rechtzeitigkeit von Aufgaben in Echtzeitsystemen garantieren zu können ist die Bestimmung von schlimmstmöglichen Antwortzeiten unerlässlich. Hierbei ist die Herausforderung die präzise Analyse aller Aktivitäten des Gesamtsystems, wie synchrone Systemaufrufe und asynchrone Interrupts, ohne allzu pessimistische Annahmen zu treffen.

Zur Lösung dieses Problems haben wir den Analyseansatz SysWCET entwickelt, der die erste integrierte Formulierung des Antwortzeitproblems ermöglicht. Diese Formulierung umfasst dabei das vollständige Rechensystem unter Berücksichtigung aller Aufgaben, aller Interrupts, und der Ablaufplanung.

1 ANTWORTZEITANALYSE VON ECHTZEITSYSTEMEN

Die schlimmstmögliche Antwortzeit (engl. worst-case response time, WCRT) für die Abarbeitung einer Aufgabe ist eine entscheidende Größe von Echtzeitsystemen, die innerhalb bestimmter Zeitschranken Ergebnisse bereitstellen müssen. Die WCRT bemisst die Dauer vom Auftreten eines (physikalischen) Ereignisses (z.B. Eintreffen eines Hardwareinterrupts) bis zur Bereitstellung des Ergebnisses (z.B. Setzen eines Motorstellwertes). Sie kennzeichnet somit die Ende-zu-Ende-Antwortzeit einer Aufgabe. Betrachtet man nun ein System das mehrere Aufgaben, mittels mehrerer Fäden (engl. Threads) abarbeitet, so ist die Berechnung der WCRT einer Aufgabe deutlich komplexer als die Berechnung der schlimmstmöglichen Ausführungszeit (engl. worst-case execution time, WCET) der Aufgabe unter Vernachlässigung anderer Threads. Beispielsweise können Hardwareinterrupts einen in Ausführung befindenden Thread asynchron unterbrechen, was zu einer Verlängerung der Antwortzeit des unterbrochenen Threads führt. Außerdem kann die Aktivierung des Echtzeitbetriebssystems (engl. real-time operating system, RTOS), mittels eines Systemaufrufs, zu einer Verdrängung führen, falls ein hochpriorerer Thread lauffähig wird. Beide Fälle, asynchrone und synchrone Verdrängungen, können darüber hinaus auch kombiniert auftreten, wenn das RTOS in der Interruptbehandlung aktiviert wird. Folglich muss in jedem Fall für die korrekte Berechnung der WCRT einer Aufgabe immer das gesamte System (RTOS, Interrupts, andere Threads) berücksichtigt werden.

2 PROBLEME MIT ZUSAMMENGESETZTER ANTWORTZEITANALYSE

Traditionell wird die WCRT Analyse in zwei Schritten vollzogen. Zuerst wird jede Aktivität (Threads und Interruptbehandlungen) in Isolation betrachtet und eine WCET pessimistisch berechnet. Hierbei agiert das RTOS als eine Grenze, welche die Analyse nicht

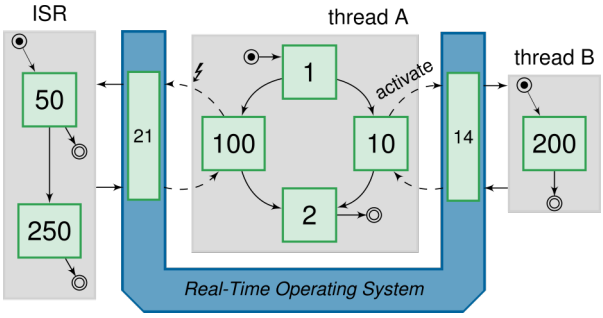

Figure 1: Zusammengesetzte Antwortzeitanalyse

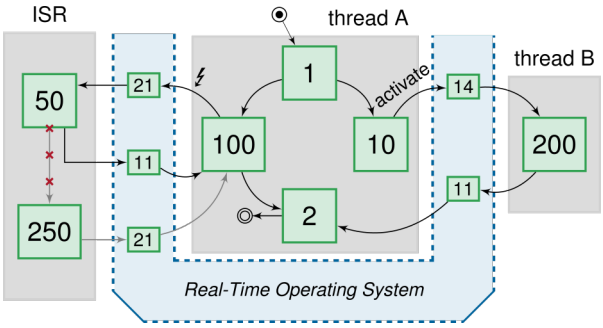

Figure 2: Integrierte Antwortzeitanalyse

überschreiten kann. Im zweiten Schritt werden dann diese WCETs pessimistisch anhand der Schedulingstrategie akkumuliert. So werden, auf die WCET der betrachteten Aufgabe, die WCETs aller höherprioren Aufgaben aufgeschlagen.

Im Beispiel wird zunächst für beide Threads (A, B), die Interruptserviceroutine (ISR), und das RTOS eine WCET berechnet (ISR=300, A=103, B=200, RTOS=21). Will man nun die WCRT für Thread A berechnen, so startet man mit der WCET von A: 103 Zyklen. Hat nun Thread B eine höhere Priorität, so kommen die Kosten für B und die zweifache Aktivierung des RTOS (hin und zurück) hinzu: 103+2x21+200=345 Zyklen. Bei einer Aktivierung der ISR, kommt man auf insgesamt: 345+2x21+300=687 Zyklen.

Das Problem mit dieser zusammengesetzten WCRT Analyse ist zum einen ihre grobe Granularität auf Threadebene, sowie die fehlende Möglichkeit Beeinflussungen zwischen den Threads zu formulieren. In den wenigsten Echtzeitsystemen sind die einzelnen Aufgaben unabhängig voneinander. Der längste Pfad in einem Thread kann unter Umständen nicht zeitgleich mit dem längsten Pfad im anderen Thread auftreten. Dazu wird nun das vorherige Beispiel in größerer Detailtiefe betrachtet.

Aus dem Kontrollflussgraph von Thread A wissen wir, dass Thread B nur dann aktiviert werden kann, wenn der rechte Pfad,

Dieser Artikel ist eine Zusammenfassung der Arbeit von Dietrich et al. [1]. Im Gegensatz zum Orginalartikel, ist das vorliegende Fragment als eingeladener Vortrag *nicht* peer-reviewed.

welcher nicht der längste Pfad durch A ist, abgelaufen wird. Außerdem ist anzunehmen, dass nicht jeder Pfad durch den Kernel die gleiche Ausführungszeit hat. Bereits mit diesem Wissen, das direkt aus der Struktur des Gesamtsystems ableitbar ist, kann eine präzisere WCRT Abschätzung vorgenommen werden.

Weiterhin haben Domänenexperten häufig weiteres Wissen über das Verhalten des Systems. Zum Beispiel kann das Auftreten von Interrupts eingeschränkt sein und nur im linken Pfad von Thread A auftreten. Außerdem ist es möglich, dass die ISR den ersten der beiden Ausgänge nimmt, wenn sie Thread A unterbricht.

Bringt man nun all dieses Wissen zusammen, so ist zu erkennen, dass der Pfad der die längste WCRT für dieses Beispielsystem ausmacht folgenden Verlauf hat: Start in Thread A, rechter Pfad, Aktivierung und Wechsel zu Thread B, Abarbeitung von B, Wechsel zu A, Ende in A. In Summe ergibt das eine WCRT von 1+10+14+200+11+2=238 Zyklen, welche deutlich präziser ist als die durch zusammengesetzte Analyse bestimmte WCRT von 687 Zyklen.

In unserer Forschung haben wir SysWCET[1] entwickelt: eine automatisierte Ende-zu-Ende Antwortzeitanalyse für OSEK Systeme. In einer integrierten Problemformulierung fangen wir nicht nur Anwendungs- und Kernelcode ein, sondern bieten zudem die Möglichkeit threadübergreifende Abhängigkeiten zu formulieren.

3 GANZHEITLICHE SYSTEMANALYSE VON OSEK SYSTEMEN

Die Grundlage von SysWCET bildet eine Analyse[2] für gesamte OSEK Systeme, welche nicht nur die statische Konfiguration in Betracht zieht, sondern auch die dynamische Interaktion zwischen Anwendungscode und Betriebssystem. Zusammengefasst lässt sich diese Interaktion durch den globalen Kontrollfluss beschreiben, der sich sowohl durch die Programmlogik der Anwendung und als auch des Kernels webt: ein Thread setzt einen Systemaufruf ab und verändert damit den Kernzustand; der Kernel reagiert auf diese Veränderung und wechselt zum Kontrollfluss eines anderen Threads.

Der OSEK-OS Standard, der zum größten Teil auch in AUTOSAR enthalten ist, bildet dabei die Grundlage unserer Systeme. Der OSEK Standard, welcher in der Automobilindustrie entwickelt wurde, beschreibt statische Echtzeitsysteme die auf Steuergeräten zum Einsatz kommen. Statisch bezieht sich hierbei darauf, dass die Systemkonfiguration bereits zur Übersetzungszeit fest steht. In einer Konfigurationsdatei (OIL File) legt der Entwickler zum Beispiel fest: welche Threads es gibt, wo deren Einstiegspunkte sind, welche Priorität sie haben und ob ein Thread auf ein Software-Ereignis warten kann.

Im ersten Schritt unserer Analyse lesen wir diese Konfiguration ein und extrahieren aus dem Anwendungscode die Kontrollflussgraphen (engl. control-flow graphs, CFGs) aller Funktionen. Um die extrahierte Anwendungslogik leichter zu analysieren, fassen wir größere Regionen innerhalb der CFGs zu atomaren Basisblöcken (engl. atomic basic blocks, ABB) zusammen, die aus der Perspektive des Betriebssystemkerns atomar ausgeführt werden. Ein ABB umfasst also den Code zwischen zwei Systemaufrufen.

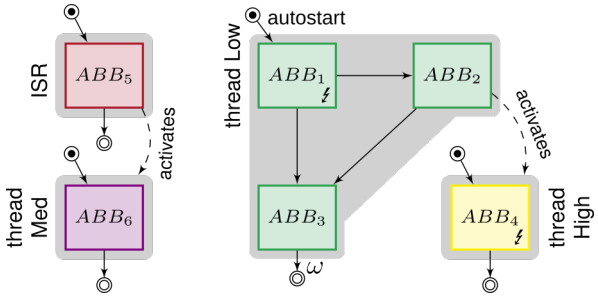

Figure 3: Beispielsystem

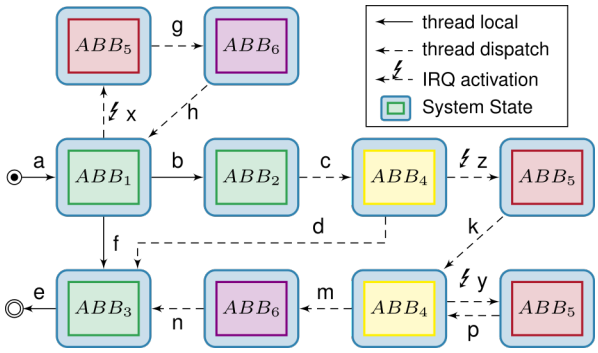

Figure 4: STG für das Beispielsystem

Im nächsten Analyseschritt kombinieren wir nun die CFGs, die Systemkonfiguration und die OSEK Semantik um den Zustandsgraphen des Systems (engl. state-transition graph, STG) zu erhalten. Beginnend beim Zustand des Betriebssystems beim Booten, gehen wir alle Systemzustände explizit ab und erhalten somit alle möglichen Pfade durch das gesamte System in einem Graphen. Dies wird nun am folgenden Beispiel veranschaulicht.

In Figure 3 3 ist ein Beispielsystem mit 3 Threads und einer ISR abgebildet. Der niederpriore Thread Low wird gestartet und aktiviert der hochprioren Thread High in seinem rechten Pfad. Der Interrupt, welcher die ISR aktiviert, kann (um das Beispiel übersichtlich zu halten) nur in ABB 1 und ABB 4 auftreten. Wird die ISR ausgeführt, so aktiviert sie in jedem Fall den mittelprioren Thread Med.

In Figure 4 ist der gesamte Zustandsgraph für die Beispielanwendung abgebildet. Hierbei entspricht jeder Knoten einem einzelnen Systemzustand, welcher unter anderem den Programmzähler des aktuell laufenden Threads und die Liste alle lauffähigen Threads beinhaltet (nicht gezeigt). Die Kanten geben an wie zwischen diesen Zuständen durch Systemaufrufe, Interrupts oder die Anwendungslogik gewechselt werden kann. Im Startzustand (Kante a) wird Thread Low den ABB 1 ausführen, wobei 3 Dinge passieren können: (1, Kante f) der linke Pfad in A wird genommen und wir führen ABB3 als Nächstes aus. (2, Kante b) Wir führen ABB 2 als Nächstes aus, welcher wiederum Thread High aktiviert (Kante c). (3, Kante x) Ein Interrupt tritt auf, Low wird verdrängt und der von der ISR aktivierte Thread Med wird zuerst ausgeführt, bevor Thread Low in ABB 1 fortgesetzt wird (Kante h).

4 ANTWORTZEITANALYSE ALS ILP PROBLEM

Der STG ist die Vereinigung aller möglichen Instruktionsfolgen (Ausführungspfade) durch das Gesamtsystem (inkl. Interrupts und RTOS). Um die schlimmstmögliche Antwortzeit für eine Aufgabe zu errechnen, müssen wir den längsten Pfad auf dem STG zwischen erster und letzter Instruktion der Aufgabe finden. Zu diesem Zwecke wenden wir die Technik der impliziten Pfadaufzählung (engl. implicit path-enumeration technique, IPET) aus Systemebene an.

Die IPET wird auf der Programmebene bereits genutzt um die WCET eines Programms zu bestimmen. Dabei wird der Kontrollflussgraph eines Programms als Optimierungsproblem formuliert und hierfür in ein ganzzahlig lineares Programm (engl. integer linear program, ILP) übersetzt. Während der Optimierung wird die Ausführungsfrequenz für jeden Basisblock bestimmt und mit seiner Ausführungsdauer gewichtet.

Da der STG ein Kontrollflussgraph ist, können wir mittels IPET ein ILP formulieren, welches für jeden Systemzustand eine Zustandsfrequenz enthält, die angibt wie häufig dieser Zustand im schlimmstmöglichen Fall besucht wird. Weiterhin betten wir für jeden ABB ein, mittels IPET formuliertes, ILP Fragment ein und leiten seine Aktivierungsfrequenz aus den Zustandsfrequenzen ab. Auf diese Weise erhalten ein integriertes ILP, welches sowohl die möglichen globalen Kontrollflüsse, die sich aus der Ablaufplanung ergeben, als auch die Instruktionsebene enthält.

Um die Skalierbarkeit der SysWCET Methodik zu ermitteln, haben wir die Kontrollsoftware eines Quadrocopters untersucht, welche aus 11 Threads, 3 Alarmen und einer ISR besteht. Innerhalb weniger Minuten konnten wir die WCRT für verschiedene Aufgaben in dieser Anwendung untersuchen, obwohl diese sich gegenseitig mittels Interrupts unterbrechen und das RTOS zur Synchronisation verwenden.

5 ZUSAMMENFASSUNG

Mit SysWCET haben wir einen Analyseansatz für systemweite Antwortzeitanalysen entwickelt, der alle relevanten Aktivitäten des Echtzeitsystems berücksichtigt und in einer einheitlichen Problemformulierung zusammenfasst. Unter vertretbarem Analyseaufwand von wenigen Minuten können vollständige Systeme automatisch analysiert werden, um verlässliche Schranken der Antwortzeiten zu bestimmen. Die Implementierung von SysWCET ist unter einer Open-Source-Lizenz verfügbar: https://gitlab.cs.fau.de/syswcet

6 LITERATUR

[1] C. Dietrich, P. Wägemann, P. Ulbrich, D. Lohmann; SysWCET: Whole-System Response-Time Analysis for Fixed-Priority Real-Time Systems; in Proceedings of the 23nd Real-Time and Embedded Technology and Applications Symposium (RTAS '17), 2017

[2] C. Dietrich, M. Hoffmann, D. Lohmann; Global Optimization of Fixed-Priority Real-Time Systems by RTOS-Aware Control-Flow Analysis; ACM Transactions on Embedded Computing Systems (ACM TECS), no. 16, 2017

7 AUTORENPROFILE

Christian Dietrich (M.Sc.) arbeitet als Wissenschaftlicher Mitarbeiter an der Leibniz Universität Hannover im Fachbereich System- und Rechnerarchitektur. Sein Hauptinteresse gilt dabei der Interaktionsanalyse zwischen Anwendung und Echtzeitbetriebssystem, sowie der Möglichkeit daraus maßgeschneiderte Systeme automatisch abzuleiten.

Peter Wägemann (M.Sc.) arbeitet als Wissenschaftlicher Mitarbeiter am Lehrstuhl für Verteilte Systeme und Betriebssysteme der Friedrich-Alexander-Universität Erlangen-Nürnberg. In seiner Forschung beschäftigt er sich mit energiebeschränkten Echtzeitsystemen sowie deren statischer Analyse.