WCET-Analyse in Mehrkern-Systemen

Eine Ausarbeitung im Rahmen des KvBK-Seminars

Sebastian Rietsch
Friedrich-Alexander-Universität Erlangen-Nürnberg
sebastian.rietsch@fau.de

Abstract

Diese Ausarbeitung behandelt die Grundlagen der dynamischen und statischen WCET-Analyse. Schwierigkeiten die sich dabei auf Mehrkern-Systemen auftun werden aufgezeigt. Ausserdem wird ein Ansatz vorgestellt, die WCET unter zur Hilfenahme der Single Core Equivalence Technologie auf Mehrkern-Systemen zu bestimmen.

1. EINLEITUNG

Harte Echtzeitsystemen müssen zu fest vorgegebenen Terminen Ergebnisse liefern, da Terminüberschreitungen in deren Anwendungsbereichen zu Katastrophen führen können und somit nicht tolerierbar sind. Die Rechtzeitigkeit solcher Systeme hängt maßgeblich von der maximalen Ausführungsdauer ihrer Aufgaben, der sogenannten Worst Case Execution Time (WCET) ab. Generell gibt es zwei Ansätze mit denen die WCET bestimmt werden kann: durch Messungen oder durch statische Analyse. Die WCET bezieht sich in der Regel auf eine Aufgabe auf einem bestimmten System. Sind die WCETs aller Aufgaben bekannt, so kann durch Planbarkeits-Analyse die Zuverlässigkeit des Echtzeitsystems verifiziert werden.

In der heutigen IT-Branche werden Mehrkern-Prozessoren immer beliebter, da sie unter Anderem eine bessere Perfomanz bieten. Um dies zu bewerkstelligen kommen in Mehrkern-Systemen moderne Komponenten und Mechanismen zum Einsatz, durch welche die Leistung des Prozessors gesteigert werden kann. Dies resultiert oft in einer erhöhten Komplexität des Systems und wirkt sich unter Umständen negativ auf die Vorhersagbarkeit des Systems aus, was wiederum die WCET Ermittlung erschwert. [4]

2. MESSBASIERTE WCET-ANALYSE

Die simpelste und in der Industrie oft angewandte Methode zur Bestimmung der WCET sind Messungen der Aufführungen auf realer Hardware oder Simulatoren. Das Ergebnis der Messungen ist die maximale beobachtete Ausführungszeit (engl. worst observed execution time, WOET). Die Granularität der Messungen kann variieren: entweder es wird die komplette End-zu-End Ausführungszeit einer Aufgabe auf einmal oder Teile bis hin zu einzelnen Instruktionen dieser gemessen. Oft werden letztere Messungen mit Methoden der Kontrollflussanalyse gekoppelt um eine obere Schranke der WCET zu bestimmen. [4] Diese Methoden werden im nächsten Abschnitt näher diskutiert.

Messbasierte WCET Ermittlungen sind generell unsicher. [4] Die Eingabe, welche zur längsten Ausführungszeit führt ist in der Regel nicht bekannt. Auch können die Aufgaben nicht mit allen Eingaben getestet werden, es ist also nicht garantiert, dass die tatsächliche WCET nicht über der WOET liegt. Deshalb wird in der Praxis oft ein Sicherheitsfaktor auf die WOET addiert, was zu einer starken Überschätzung der WCET führen kann. Neben den Eingaben hängt die Ausführungszeit auch vom Zustand mehrer Systemkomponenten (engl. initial-state) ab, z.B. der Belegung des Caches. Normalerweise ist der Anfangszustand, der zur WCET einer Aufgabe führt nicht bekannt, weshalb Messungen mit mehreren Startzuständen durchgeführt werden müssen. Um Messungen überhaupt zu ermöglichen muss außerdem der Quellcode verändert werden, falls keine spezielle Testhardware zur Vefügung steht, da beispielweise Hardware-Timer angesprochen werden müssen. Bei der Validierung sicherheits-kritischen Systeme ist ein solches Vorgehen nicht erlaubt. [1]

Im Hinblick auf Multicore-Systeme werden möglicherweise auftretende Anomalien außer Acht gelassen, falls diese während den Messungen nicht auftreten. Auch der Zustand geteilter Ressourcen wird durch auf anderen Kernen laufende Aufgaben mitbeeinflusst. Der Einfluss dieser auf die Ausführungszeit einer Aufgabe kann bei Messungen nur schwer mitberückstichtigt werden.

3. STATISCHE WCET-ANALYSE

Das Ziel der statischen WCET-Analyse ist es, die WCET konservativ zu schätzen, also eine garantierte maximale WCET (obere Schranke) zu bestimmen. Die Schätzung erfolgt ohne Ausführung des Programmcodes auf der eigentlichen Hardware und bezieht alle möglichen Eingaben mit in die Berechnung ein. Dafür muss Folgendes analysiert werden: Welche Pfade sind innerhalb des Aufgabencodes möglich und wie beeinflusst die Hardware die Ausführungszeit. Auf Basis dieser Informationen kann die WCET im Anschluss geschätzt werden. [4]

3.1 Kontrollfluss-Analyse

Um die möglichen Pfade durch den Kontrollfluss eines Programms zu bestimmen, ist es sinnvoll diesen in einer geeigneten Repräsentation darzustellen. Besonders geeignet ist hierfür der Kontrollfluss-Graph (CFG). Dieser besteht aus Grundblöcken als Knoten und Übergängen zwischen diesen als Kanten (Figure 1, a). Ein Grundblock besteht dabei aus sequentiell ausgeführten Instruktionen, und verbraucht dadurch eine gewisse Rechenzeit. Der CFG kann auf Basis verschiedener Abstraktionsebenen des Programmcodes erstellt werden, ein Grundblock kann also beispielweise einer Anwei-

sung auf Hochsprachenebene oder einer Menge von Instruktionen auf Maschinencodeebene entsprechen. Bei Verwendung des Hochsprachencodes muss die Abbildung des Compilers auf die Maschinencodeebene mit einbezogen werden, da dieser unter Umständen Optimierungen vornimmt. Es müssen auch Aufrufbeziehungen von Funktionen mitbeachtet werden. Weiter Informationen wie die Menge der möglichen Eingaben oder obere Schranken der Schleifendurchgänge sind für die statische Analyse ebenfalls nötig.

Das Ergebnis der Kontrollfluss-Analyse ist entweder ein annotierter Syntaxbaum für den strukturbasierten Schätzungsansatz oder eine Menge an Flussfakten für den IPET-Ansatz. [4] Diese beiden Methoden werden in einem späteren Abschnitt behandelt.

Es stellt sich nun die Frage, wie die Ausführungszeiten der Basisblöcke des CFG bestimmt werden können, und wie diese durch verschiedene Zustände der Hardware beeinflusst werden.

3.2 Hardware-Analyse

Die Hardware-Analyse stützt sich auf ein abstraktes Modell der tatsächlichen Hardware. Dies umfasst in der Regel den Prozessor (Pipeline), das Speichersystem (DRAM, Caches), Busse und Peripherie. Die Abstraktion muss dabei konservativ im Hinblick auf die Ausführungszeit erfolgen, sodass die vorhergesagt Ausführungszeit keines Falls unter der tatsächlichen liegt. Umso komplexer die Hardware ist, desto schwerer ist die konservative Abstraktion zu realisieren.

Die Ausführungszeit von Instruktionen ist wie schon erwähnt kontextabängig. Beispielsweise kann eine Instruktion um einiges länger dauern, wenn benötigte Daten nicht im Cache liegen. Um die maximale Ausführungszeit einer Instruktion bestimmen zu können müssen deswegen alle Hardwarezustände die bei der Ausführung dieser vorliegen können mit einbezogen werden. Die maximale Ausführungszeit einer Instruktion ergibt sich also aus der Menge der Ausführungshistorien des Prozessors, die zu dieser Instruktion führen können.

Ein weit verbreiteter Ansatz um diese zu analysieren ist die Datenflussanalyse. Es werden Invarianten für jede Instruktion bestimmt. Diese beziehen sich jeweils auf bestimmte Flüsse durch das Programm, sie drücken beispielweise Wissen über die Cachebelegung aus. Aus diesen Invarianten können dann Informationen wie die Anzahl an Cache-Fehlern abgeleitet werden. Dies ermöglicht eine genauere Schätztung der oberen WCET Schranke. [4]

Grosse Schwierigkeiten können sich bei der Modellierung von komplexen Systemenfeatures wie Pipelines, out-of-order execution, Spekulationen und Sprungvorhersagen auftun. Diese sind schon auf Singlecore-Systemen schwer zu analysieren, auf Multicore-Systemen wird diese Analyse noch komplexer.[5] Ebenfalls gibt es in Multicore-Systemen meist geteilte Ressourcen, wie zum Beispiel Caches. Auf diesen kann es zu intercore-Interferenzen kommen, sodass beispielweise die Ausführung einer Aufgabe auf einem Kern Instruktionen einer andren Aufgabe eines andren Kerns aus dem Cache verdrängt. Um die WCET zu schätzen kann angenommen werden, dass jeder Cachezugriff zu einem Cache-Fehler führt, was jedoch in einer starken Überschätzung resultiert. [5] In Multicore-Systemen können auch Anomalien auftreten (beispielweise kann die lokale längere Ausführungszeit einer Instruktion zu einer insgesamt kürzeren globalen Ausführungzeit führen). Das hat zur Folge, dass die Schätzung einer globalen WCET durch lokale maximalen Ausführungszeiten der Instruktionen unsicher ist. Der Hardwarezustand bezüglich eines Flusses wird nicht-deterministisch und es müssen unter Umständen verschiedene Nachfolgezustände angenommen werden, die Komplexität der Analyse wächst enorm. [4] Insgesamt kann die WCET-Analyse von Aufgaben also nicht mehr unabhängig voneinander erfolgen.

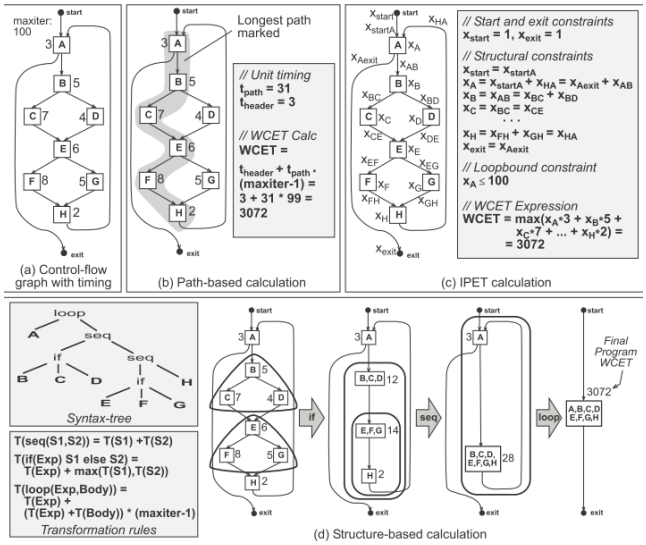

Figure 1: WCET Schätzung [4]

3.3 WCET Schätzung

Nachdem eine Kontrollfluss- und Hardware-Analyse durchgeführt wurde kann die obere Schranke der WCET geschätzt werden. In diesem Abschnitt werden drei dieser Ansätze vorgestellt.

Bei der pfadbasierten Berechnung (Figure 1, b) werden alle Pfade des Programms explizit aufgezählt und der Pfad mit der längsten Ausführungszeit gesucht. Die Menge aller Pfade nimmt exponentiell mit der Menge der Verzweigungen zu, weshalb bei grösseren Programmen diese Methode algorithmisch schwer handhabbar ist. Die strukturbasierte Berechnung (Figure 1, d) arbeitet auf Basis eines annotierten Abstrakten Syntaxbaumes (AST). Die maximale Ausführungszeit wird durch Aggregation der AST Komponenten nach festen Regeln berechnet. Die Ausführungszeit zweier sequentieller Anweisungen ist beispielweise die Summe der beiden einzelnen Ausführungszeiten. [4]

Die Implicit Path Enumeration Tequnique (IPET) (Figure 1, c) arbeitet mit Flussfakten des Programms. Diese werden auf ein lineares Optimierungsproblem mit Flussrestriktionen als Nebenbedigungen abgebildet. Das Optimierungsproblem kann dann mit geeigneten mathematischen Verfahren gelöst werden. In [5] wird ein Ansatz vorgestellt, bei dem die möglichen L2 Instruktions-Cache Fehler auf Multi-Core Systemen bei der Schätzung mittels IPET mit einbezogen werden.

4. WCET(M) SCHÄTZUNG AUF MEHRKERN SYSTEMEN MITTELS SCE TECHNOLOGIE

Ein interessanter Ansatz für die Bestimmung der WCET wird in [2] vorgeschlagen. Mittels der Single Core Equivalence Technologie wird aus einer Multicore-Platform eine Menge von gleichartigen virtuellen Singlecore-Maschinen exportiert. Dadurch kann die Planbarkeit des Systems durch die getrennte Analyse der einzelnen Kerne verifiziert werden. Die WCET einer Aufgabe hängt dabei von der Anzahl der aktiven Kernen m ab und wird als *WCET(m)* bezeichnet. Im Folgenden wird erklrt, wie SCE die Ressourcen eines Multicore-Systems unter den einzelnen Kernen aufteilt. Table 1 zeigt alle Parameter, die im Verlauf der nächsten Abschnitte eingeführt werden.

4.1 Single Core Equivalence

Single Core Equivalence (SCE) ermöglicht die Regulierung geteilter Speicherressourcen eines Systemens auf Betriebssystemebene. Liegt ein System mit m Kernen vor, so erhält jeder Kern *1/m* der geteilten Speicherressourcen. Zu diesen Speicherressourcen gehören Cache- und Arbeitsspeicher.

Geteilter Cachespeicher wird durch die Technik *colored lockdowm* verwaltet. Zuerst werden die Speicherzugriffe der Aufgaben und Arbeitsaufträge in Isolation profiliert, wobei oft benutzte Seiten (*hot pages*) identifiziert werden. Diese bleiben bei der späteren Ausführung fest im Cachespeicher und können nicht mehr verdrängt werden (*lockdown*). Ausserdem werden die Seiten im Speicher neu positioniert, da in der Regel viele Seiten auf die gleiche Cacheadresse abgebildet werden (*page coloring*). Das ermöglicht eine grössere Flexibilität bei der Allokierung. Im Anschluss wird für jede Aufgabe eine *progressive lockdown curve* bestimmt. Diese besteht aus mehreren WCETs C der Aufgabe bei Ausführung in Isolation in Abängigkeit von der Anzahl der fest im Cache gehaltenen Seiten. Die WCETs werden durch Messungen ermittelt, der vorgestellte Ansatz kann aber auch für statisch ermittelte Werte angeändert werden. Für jeden Datenpunkt dieser Kurve kann eine maximale Anzahl an Cache-Fehlern μ aus der Anzahl der festen Seiten abgeleitet werden.

Neben Cache stellt der DRAM eine Ressource dar, bei der es auf Multi-Core Systemen zu Wettstreitigkeiten kommen kann. Ein Arbeitsspeicher ist in Bänken organisiert, welche aus Zeilen und Spalten bestehen. Am effektivsten kann auf Daten innerhalb einer Bank zugegriffen werden, wenn diese in der gleichen Zeile stehen. Greifen mehrere Prozesse auf die gleiche Bank zu ist dies unter Umständen nicht mehr gegeben und die Anfragen dauern länger. Auch kann es zur Umordnung von Speicheranfragen auf Bankebene durch die Speicherverwaltung kommen, wodurch zuerst gestelle Speicheranfragen nicht immer als erstes bearbeitet werden. Mittels PALLOC, einem Programm das auf Betriebssystemebene arbeitet, werden deshalb Kernen oder Aufgaben private DRAM-Bänke zugewiesen.

Neben dem DRAM-Speicher an sich muss auch dessen Bandbreite verwaltet werden. MemGuard teilt diese gleichmässig unter allen Kernen auf. Es wird ein festes Zeitintervall P (Regulationsperiode) festgelegt, in dem jeder Kern eine maximale Anzahl an Speicheranfragen $K_q = \frac{P}{mL_{max}}$ stellen darf. L_{max} ist hierbei die maximale Dauer einer Speicheranfragen. Diese kann ermittelt werden, indem die Dauer untereinander abhängiger Speicheranfragen gemessen werden, wobei die angeforderten Daten in verschienenen DRAM-Zeilen liegen. Die minimale Dauer eine Speicheranfrage L_{min} wird durch die Messung von nicht abhängigen Speicheranfragen ermittelt, wobei die Daten in der gleichen DRAM-Zeile liegen. Die Grösse einer Cachezeile in Bytes wird als L_{size} bezeichnet. Führt ein Kern zu viele Speicherzugriffe durch, so wird er in den Leerlauf überführt, bis die nächste Periode beginnt. Die Regulationseriode sollte dabei um ein vielfaches kleiner sein als die Perioden der Aufgaben.

m	Aktive Kerne
C	WCET einer Aufgabe in Isolation
μ	Anzahl an residualen Cache-Fehlern
P	Dauer einer Regulations-Periode
K_q	Schranke der Speicheranfragen pro Periode P
L_{max}	Maximale Dauer einer Speicheranfrage
L_{min}	Minimale Dauer einer Speicheranfrage
L_{size}	Grösse einer Cachezeile in Bytes

Table 1: Parameter

4.2 Methodik

In diesem Abschnitt wird das Vorgehen der Implementierung des Systems näher erläutert.

Im ersten Schritt wird die *progressive lockdown curve* und die dazugehörigen Cache-Fehler für alle Aufgaben bestimmt. Dafür werden alle Aufgaben in Isolation ausgeführt, während alle andren Kerne abgeschaltet werden.

Danach wird der DRAM unter allen Kernen mittels PALLOC und MemGuard gleichmässig verteilt, in Folge dessen jeder Kern einen m-mal langsameren Arbeitsspeicher zur Verfügung hat.

Nun müssen die Aufgaben den einzelnen Kernen zugewiesen werden und im Anschluss die Cachezuteilungen dieser bestimmt werden. Für diesen Zweck wird die *progressive lockdown curve* verwendet. Es kann eine gewünschte WCET der Aufgabe gewählt werden woraus sich die Menge des benötigten Cachespeichers ergibt oder umgekehrt. Dabei muss darauf geachtet werden, dass pro Kern höchstens $\frac{Cachegrösse}{m}$ Speicher allokiert wird.

Im Anschluss können die WCET(m)s der Aufgaben bestimmt und das System auf Planbarkeit überprüft werden. Wie mittels SCE eine Abschätzung dieser bestimmt werden kann wird in den folgenden Abschnitten gezeigt.

4.3 Systemmodell

Um Schranken der Ausführungszeiten von Aufgaben bestimmen zu können muss das zugrunde liegende Systeme bestimmt Anforderungen erfüllen.

Die Vorhersagbarkeit des Zustands der CPU wird durch Mechanismen wie Prefetching und Spekulation enorm erschwert, da diese heuristische Verfahren anwenden. Deshalb müssen diese deaktiviert werden.

Auch das Verhalten des Schedulers sollte möglichst vorhersagbar sein, weshalb ein System mit Rate Monotonic Scheduling angenommen wird. Die Prioritäten der Aufgaben werden dabei im Hinblick auf die Dauer ihrer Perioden gewählt.

Da durch MemGuard und PALLOC garantiert werden soll, dass jeder Kern pro Regulationsperiode eine gewisse Anzahl an Speicheranfragen durchführen kann, sollten DRAM-

Controller und Bus Arbiter nach dem Rundlauf Verfahren (engl. round-robin) arbeiten.

Eine sehr wichtige Annahme die für die Abschätzung der WCET(m) getroffen werden kann ist, dass mögliche Anomalien bei dieser nicht beachtet werden müssen. Jede Aufgabe bekommt durch SCE eine Menge an fest im Cache liegenden Speicherseiten, weshalb diejenigen Speicheranfragen die zu einem Cache-Treffer führen bekannt sind und sich nach mehrmaliger Ausführung nicht ändern. In Folge dessen müssen möglicherweise existierende Anomalien während der Bestimmung der *progressive lockdown curve* schon aufgetreten und mitgemessen worden sein.

4.4 Rechenverzögerung durch Regulation

Die Ausführung von Aufgaben und Arbeitsaufträgen kann auf Grund von benötigten Daten aus dem Hauptspeicher verzögert werden. Einerseits ist es möglich, dass der Speicher zu einem Zeitpunkt stark ausgelastet ist und deshalb nicht alle Speicheranfragen sofort bearbeitet werden können (Verzögerung durch Wettkampf). Andererseits hat jeder Prozessor unter SCE nur eine beschränkte Anzahl an Speicherzugriffen pro Regulationsperiode zur Verfügung. Wird diese Anzahl überschritten greift MemGuard ein und der Prozessor muss bis zum Beginn der nächsten Periode warten, bis er wieder vom Speichersystem bedient wird (Verzögerung durch Regulation). Im Folgenden wird erklärt, wie durch die Verwaltung der geteilten Ressourcen unter SCE eine Schranke der Verzögerung durch Speicheranfragen angegeben werden kann. Es muss dabei die maximale Verzögerung im Falle des Wettkampfs und im Falle der Regulation betrachtet werden.

Die maximale Verzögerung durch Wettkampf um die Bandbreite des Hauptspeicher entsteht, falls alle Kerne K_q Speicheranfragen mit einer jeweiligen Dauer von L_{max} durchführen. Ein Kern ist dabei eine Dauer von $(m-1)K_qL_{max}$ blockiert. Dies lässt sich auflösen zu $P - K_qL_{max}$.

Die maximale Verzögerung durch Regulation ensteht, falls ein Kerne K_q Speicheranfragen stellt, während alle anderen Kerne keine Speicherzugriffe durchführen, und jede dieser in der minmal möglichen Zeit L_{min} beantwortet werden. Die Dauer, die der Kern in Folge dessen im Leerlauf verbringt ist deshalb $P - K_qL_{min}$.

Man sieht, dass $P - K_qL_{min} > P - K_qL_{max}$. Die grösste Verzögerung durch Speicheranfragen entsteht also durch Regulation. Daraus folgt ebenfalls derjenige Zustand eines Kerns, der zur grössten Verzögerung durch Regulation führt. Sie wird am grössten, wenn ein Prozessor zu Beginn einer neuen Periode mindestens eine Menge K_q Speicheranfragen stellt.

Durch die hergeleiteten Schranken kann ein Verzögerungsterm einer Aufgabe in Abhängigkeit von den Cache-Fehlern dieser angegeben werden.

THEOREM 1.

$$stall(\mu) = \lceil \frac{\mu}{K_q} \rceil (P - K_qL_{min}) + (m-1)L_{max}(\mu - (\lceil \frac{\mu}{K_q} \rceil - 1)K_q)$$

Der erste Term beschreibt die maximale Verögerung einer Aufgabe durch Regulation. $\lceil \frac{\mu}{K_q} \rceil$ ist dabei die Anzahl an Perioden, in denen die Aufgabe reguliert wird. Sie wird pro Periode um maximal $P - K_qL_{min}$ verzögert.

Der zweite Term beschreibt die maximale Verzögerung durch Wettkampf der nicht unter Regulation fallenden Speicheranfragen. Die Anzahl dieser Anfragen ist $\mu - (\lceil \frac{\mu}{K_q} \rceil - 1)K_q$. Jede dieser Anfragen dauert maximal $(m-1)L_{max}$.

4.5 WCET(m) und Antwortzeit

Mit Hilfe des im letzten Abschnitt ermittelten Verzögerungsterms kann die Antwortzeit einer Aufgabe bestimmt werden. Die WCET einer Aufgabe ist nicht die eigentliche Zeit die zwischen Beginn und Beendigung der Ausführung vergeht, da sie durch höherpriore Aufgaben von der CPU verdrängt werden kann. Durch die Analyse der Antwortzeit ergibt sich gleichzeitig die Formel der WCET(m).

Bei dem zu analysierenden System wird vorausgesetzt, dass Aufgaben nach dem Rate Monotonic Scheduling Verfahren eingeplant werden. Dadurch ergibt sich die maximale Antwortzeit einer Aufgabe aus der WCET dieser und der Zeit die vergeht, bis die höherprioren Aufgaben, die auf diesem Kern laufen, fertig ausgeführt wurden. Zusätzlich muss die Verzögerung aller Aufgaben durch ihre Speicheranfragen mitbeachtet werden, da diese die Antwortzeit ebenfalls mitbeeinflussen.

Wie die letztendliche Formel der Antwortzeit und WCET(m) hergeleitet wird ist in Appendix A zu sehen. Mit dieser Formel kann das System direkt auf Planbarkeit überprüft werden.

5. SCHLUSS

Die WCET-Analyse ist ein essenzieller Bestandteil bei der Validierung von Echtzeit-Systemen. Auf Singlecore-Systemen kann sie relativ einfach durchgeführt werden, während die Analyse von Multicore-Systemen sehr komplex sein kann. Dabei bereiten geteilte Ressourcen oft grosse Schwierigkeiten, da verschieden Kerne gleichzeitig den Zustand dieser beeinflussen. In Folge dessen verändern sich auch die Ausführungszeiten der Aufgaben, die auf den Kernen laufen. Deshalb muss oft ein stark abstrahiertes Modell der Hardware verwendet werden, weshalb die WCET-Schätzung nicht immer genau ist. Da aber besonders Multicore-Platformen in der heutigen Zeit immmer beliebter werden sind Forschungen auf diesem Gebiet sehr wichtig. Der vorgestellte Ansatz der WCET(m) Schätzung mittels SCE ist dabei definitiv ein Schritt in die richtige Richtung, wobei bei diesem im Hinblick auf Effizienz und Genauigkeit noch Spielraum besteht. Dies wurde von den Autoren ebenfalls betont und der Ansatz soll in Zukunft noch weiter verbessert werden.

6. REFERENCES

[1] P. Lokuciejewski and P. Marwedel. Wcet analysis techniques. In *Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems*, Embedded Systems, pages 13–22. Springer Netherlands, 2011.

[2] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. Wcet(m) estimation in multi-core systems using single core equivalence. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 174–183, July 2015.

[3] F. Singhoff. Real-time scheduling analysis. Web page, 2013. Accessed: 29.11.2015.

[4] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand,

R. Heckmann, T. Mitra, et al. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

[5] J. Yan and W. Zhang. Wcet analysis for multi-core processors with shared l2 instruction caches. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 80–89. IEEE, 2008.

APPENDIX

A. WCET(M)-HERLEITUNG

Für die Herleitung der WCET(m) nehmen wir an, dass μ durch K_q ohne Rest teilbar ist. Ist dies nicht gegeben, so muss eine sichere Abschätzung $\hat{\mu} = \lceil \frac{\mu}{K_q} \rceil K_q$ von μ verwendet werden. Der Verzögerungsterm lautet dann wie folgt:

$$\begin{aligned} stall(\hat{\mu}) &= \frac{\hat{\mu}}{K_q}(P - K_q L_{min}) + (m-1)L_{max}K_q \\ &= \frac{\hat{\mu}}{K_q}(P - \frac{PL_{min}}{mL_{max}}) + (m-1)L_{max}K_q \\ &= \frac{\hat{\mu}mL_{max}}{P}(P - \frac{PL_{min}}{mL_{max}}) + (m-1)L_{max}K_q \\ &= \hat{\mu}L_{max}(m - \frac{L_{min}}{L_{max}}) + (m-1)L_{max}K_q \end{aligned}$$

Die allgemeine Antwortzeit einer Aufgabe lässt sich wie folgt berechnen:

$$R_i = C_i + \sum_{\tau_j \in hp(i)} WaitingTime_j$$

τ_j ist hierbei eine Aufgabe *j*. *hp(i)* ist die Menge alle Aufgaben, die eine höhere Priorität als die Aufgabe i besitzen. Durch einsetzen der Ausführungsdauer höherpriorer Aufgaben und des Verzögerungsterms ergibt sich obiges zu:

$$\begin{aligned} R_i^{(k+1)} &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j + stall(\mu_i^{(k)}) \\ \mu_i^{(k)} &= \sum_{\tau_j \in hep(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil \hat{\mu}_j \end{aligned}$$

T_j ist die Periode der Aufgabe *j*. Diese Formel wird folgendermassen verwendet: Es wird $R_i^{(0)} = C_i$ gesetzt. Danach wird die Formel iterativ angewandt. Ist nach einer gewissen Iteration $R_i^{(k)} > T_i$ so ist die Aufgabe nicht einplanbar. Tritt der Fall $R_i^{(k)} = R_i^{(k-1)}$ ein, so wird der Termin der Aufgabe eingehalten und sie ist einplanbar. [3]

Als nächstes wird die Definition von $stall(\mu_i^{(k)})$ eingesetzt:

$$\begin{aligned} &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j + \mu_i^{(k)} L_{max}(m - \frac{L_{min}}{L_{max}}) \\ &\quad + K_q L_{max}(m-1) \end{aligned}$$

Im nächsten Schritt erweitern wir $\mu_i^{(k)}$:

$$\begin{aligned} &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j + (\sum_{\tau_j \in hep(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil \hat{\mu}_j) \\ &\quad L_{max}(m - \frac{L_{min}}{L_{max}}) + K_q L_{max}(m-1) \\ &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j + (\lceil \frac{R_i^{(k)}}{T_i} \rceil \hat{\mu}_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil \hat{\mu}_j) \\ &\quad L_{max}(m - \frac{L_{min}}{L_{max}}) + K_q L_{max}(m-1) \end{aligned}$$

hep(i) ist die Menge aller Aufgaben mit höherer oder gleicher Priorität.

Da wir auf Planbarkeit von τ_i prüfen gilt $\lceil \frac{R_i^{(k)}}{T_i} \rceil = 1$:

$$\begin{aligned} &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_j + (\hat{\mu}_i + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil \hat{\mu}_j) \\ &\quad L_{max}(m - \frac{L_{min}}{L_{max}}) + K_q L_{max}(m-1) \\ &= C_i + \hat{\mu}_i L_{max}(m - \frac{L_{min}}{L_{max}}) + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil \\ &\quad (C_j + \hat{\mu}_j L_{max}(m - \frac{L_{min}}{L_{max}}) + K_q Lmax(m-1) \end{aligned}$$

Durch eine letzte Umformulierung erhalten wir:

$$R_i^{(k+1)} = C_{sce_i} + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i^{(k)}}{T_j} \rceil C_{sce_j} + K_q Lmax(m-1)$$

$$C_{sce} = C + \hat{\mu} L_{max}(m - \frac{L_{min}}{L_{max}})$$

C_{sce} ist die WCET(m) der Aufgabe.